ҚАЗАҚСТАН РЕСПУБЛИКАСЫ
БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ

Т. А. Павловская

C/C++
ЖОҒАРҒЫ ДЕҢГЕЙЛІ ТІЛДЕ ПРОГРАММАЛАУ

Оқулық

Алматы, 2012

ӘОЖ 004.43(075.8)
КБЖ 32.973-018 я 73
П 12

Қазақстан Республикасы Білім және ғылым министрлігінің «Оқулық» республикалық ғылыми-практикалық орталығы бекіткен

Қазақшаға аударғандар: БӨРІБАЕВ Б., АБДРАХМАНОВА М. Б.

Павловская Т. А.

П 12 С/С++. Жоғарғы деңгейлі тілде программалау: Оқулық.-Алматы: «Дәуір» баспасы, 2012. - 504 бет.

ISBN 978-601-217-347-5

Бұл кітаптың мақсаты – ISO/IEC 14882 стандартына сәйкес С++ тіліне қысқа әрі нақты түрде сипаттама беру. Кітап тілді оның әліпбиінен бастап игеруге бел буған студенттерге арналған, алайда ол программалауды жетік меңгерген мамандар үшін де пайдалы болады. Кітапты жазуда алға қойылған мақсат – С++ тілінің мүмкіндіктерін және оларды пайдалану тәсілдерін дұрыс көрсету, осы және басқа да программалау тілдерін үйренуге түрткі болу және қазіргі заманғы объектіге бағытталған технологияларды түсінуге жол ашу.

Өзекті тақырыптар бойынша бақылау тапсырмалары 20 нұсқада берілген және автор өзінің оқытушыларға көрсетіп отырған көмегін олар дұрыс бағалайды деп сенім артады.

ӘОЖ 004.43(075.8)
КБЖ 32.973-018 я 73

© ЗАО Издательство «Питер», 2012
© ҚР жоғары оқу орындарының қауымдастығы, 2012

ISBN 978-601-217-347-5

Алғы сөз

Бұл кітаптың негізгі мақсаты – C++ тілін «Для чайников» сериясының үзінділігі мен үстіртін баяндауынан қорғай отырып, маңызы тек программалау тәжірибесін игергеннен кейін айқындалатын ерекшеліктер туралы шексіз пікірлерсіз, ISO/IEC 14882 (1998) стандартына сәйкес, нақты әрі қысқа түрде сипаттау болып табылады. Бұл кітап тілді жаңадан оқып үйрене бастаған студенттерге арналған, алайда ол программалауда бірсыпыра білім жинақтағандар үшін де қажеттілігін жоймайды. Оны дәрістердің қысқаша конспектісі немесе қосымша анықтамалық құрал ретінде қарастыруға болады.

Назарларыңызға ұсынылып отырған кітап жалпы материал мазмұнын толыққанды ашуға ұмтылмайды, ол үшін анықтамалық құралдар, құжаттамалар және мәтінге тәуелді көмекті қолдану мүмкіндіктері бар. Мұны жазуда алға қойылған мақсат – C++ тілінің мүмкіндіктері мен оларды қолдану туралы дұрыс көзқарас қалыптастыру, сонымен қатар болашақта осы және басқа да программалау тілдерін үйренуге жол көрсету.

Кітапта тілдің негізгі конструкцияларынан басқа, программаларды жазу барысында қолданылатын негізгі мәліметтер құрылымдары, программалардың сапалық көрсеткіштері, сонымен қатар программалау стилі және технологиясы жөнінде бірсыпыра ұсыныстар қарастырылады.

Осы кітапқа қосымша ретінде 2002 жылы Т. А. Павловская мен Ю. А. Щупактың «C/C++. Жоғарғы деңгейлі тілде программалау. Құрылымдық программалау» атты практикумы «Питер» баспасынан жарық көрді. Онда құрылымдық парадигма шеңберінде қолданылатын C++ тілінің құралдары сипатталып, оқулықта берілген зертханалық тапсырмаларды түсіндіре отырып талдау жүргізіледі, сонымен қатар программаларды құру және жөндеп түзету технологиялары түбегейлі қарастырылып, тапсырмаларды шешу нұсқалары оның тиімділігін бағалау жолдары да ұсынылады. 2004 жылы практикумның объектіге бағытталған программалауға арналған екінші бөлімі басылып шықты. Практикумда қарастырылатын негізгі сұрақтар ішінде UML, программалық инженерияның базалық қағидалары, жобалау үлгілері келтіріліп, олар оқулықта қамтылған тақырыптар шеңберінен шығып, оқырманға кәсіби программалау әлеміне жылдам қадам басуға мүмкіндік береді.

Кітаптың үштен бірінен астам бөлігі стандартты кітапханаға, яғни программаларды құру мерзімін қысқарту және олардың сенімділігін арттыруға, бір ортадан екінші бір ортаға көшірілу мүмкіндіктері мен қолданылу әмбебаптылығын жоғарылату мақсатында әрбір кәсіби программалаушы меңгеруі тиіс құралдарға шолу жасауға арналған.

Кітапта Windows жүйесіне және басқа да орталарға қатысты программалау ісі қарастырылмайды, сондықтан барлық мысалдар «консольдік қосымшалар» ретінде көрсетіледі. Мысалдардың синтаксисі C++ стандартына сәйкес келеді, сондықтан олардың кейбіреулері осы аталған стандарттың бекітілуіне дейін шыққан программалық қоршауларда орындалмауы мүмкін.

Кітап негізгі үш бөлімнен тұрады, олар: «Құрылымдық программалау», «Объектіге бағытталған программалау» және «Стандартты кітапхана».

Бірінші бөлімде C++ тілінің негізгі конструкциялары және қазіргі кездегі сапа мен сенімділік талаптарына жауап беретін программа құрудың базалық технологиясы қарастырылады.

Екінші бөлім көлемді программаларды дұрыс құруға мүмкіндік беретін тіл құралдарын – кластарды, мұралауды, шаблондарды және айрықша жағдайларды (ерекше жағдайларды) – сипаттауға арналған. Бұл бөлім материалдары стандартты кітапхананы игеру үшін де қажет болады.

Үшінші бөлімде C++ тілінің стандартты кітапханасы құралдарына сипаттама берілген. Ағымдар, контейнерлер, итераторлар, векторлар, стандартты алгоритмдер және олардың арасындағы байланыстар қарастырылады.

Кітаптағы тілдің синтаксистік элементтерінің анықтамалары қарайтылған қаріппен ерекшеленіп берілген.

Қосымшаларда кез келген программаны жазу барысында қажет болатын анықтамалық ақпараттың жинақталған көлемі берілген.

Кітапта сипаттама берілген барлық түйінді сөздерді, типтерді, тұрақтыларды, функцияларды, макростар мен кластарды пәндік нұсқауыш арқылы табуға болады, осы мүмкіндік кітапты анықтамалық ретінде де қолдануға жол ашады. Қолдануды жеңілдету үшін нұсқауыш жолында элемент атынан кейін үтір арқылы осы элемент кіретін класс аты немесе түсіндірме берілген.

Осы мүмкіндікті пайдалана отырып, автор өзінің алғашқы программалау тілінен берген ұстаздары – П. А. Шипилов пен Г. Л. Голованевскийге, «Питер» баспасының қызметкері А. В. Васильевке, әріптестері – А. В. Лаздин, Ю. А. Щупак және И. Яковлевке, әдебиеттер тізімінде есімі аталған барлық корифейлерге, сонымен қатар ата-анасына және суреттерді салған ұлы Евгенийге үлкен алғысын білдіреді.

Кітаптың негізіне автордың Санкт-Петербург ақпараттық технологиялар, механика және оптика университетінде (СПбГУ-ИТМО) оқитын дәрістер курсы алынған.

Кітапқа қатысты айтар сыни пікірлеріңіз бен ұсыныстарыңызды *mux@tp2055.spb.edu* немесе *2:5030/1016.20@fidonet.org* адрестеріне хабарлауыңызға болады.

Баспадан

Сіздердің ескертулеріңіз, ұсыныстарыңыз, сауалдарыңыз болып жатса, онда мына электрондық пошта адресіне жіберіңіздер: comp@piter.com («Питер» баспасы, компьютерлік редакция).

Біз сіздердің пікірлеріңізді білуге әрдайым қуаныштымыз!

Біздің кітаптар туралы толық ақпаратты баспаның Web-сайтынан таба аласыздар: *http://www.piter.com.*

I БӨЛІМ

Құрылымдық программалау

Жақсы программаларды жазу ісі
ақыл, талғам және сабырлылықты талап етеді.
Б. Страуструп

Дәстүрлі программалау технологиясы программалардың негізгі тұтынушылары ғылыми мекемелер ғана болған кезде, есептеу ресурстары шектеулі болып, ал программаны сүйемелдеу мәселелері әлі белгісіз болған жағдайда қалыптасты. Программа сапасының негізгі көрсеткіштері ретінде оның тар мағынасындағы тиімділігі мен жинақылығы есептелді. Уақыт өте келе, программалар күрделілігінің артуына байланысты, оларды құруға үлкен ұжымның бірнеше жыл еңбегі кететін болды, осылардың нәтижесінде программалық жүйелер мерзімінен кешігіп және көптеген қателермен шығарылып отырды.

Программалық жабдықтамалар жасаудың дағдарысы программаның өмірлік циклі барысында – оны жазу жоспары пайда болғаннан бастап қолданудан алып тастағанға дейінгі кезеңде – программалар құрудың жалпы шығындарын азайтатын жаңа тәсіл табу қажеттілігіне алып келді. Мұндай технология өткен ғасырдың 70-ші ж. басында пайда болып, құрылымдық программалау деген атауға ие болды. Оның негізін программалау теориясы мен жоғары білікті мамандардың жеке тәжірибесінің дамуын ескеру, сонымен қатар программаларға қойылатын соңғы талаптар мен оларды жасаудың өндірістік сипат алуын есепке алу құрайды.

Программаларға қойылатын негізгі талап – олардың спецификацияларымен толық үйлесімді түрде жұмыс істеуі және қолданушының кез келген іс-әрекеттеріне дұрыс жауап қайтаруы болып табылады. Сонымен қатар, программа көрсетілген мерзімде шығуы керек және қажетті өзгертулер мен толықтыруларды жедел енгізуге мүмкіндік беруі тиіс. Өкінішке орай, мұндай жағдайда пайдаланылатын жады көлемі мен алгоритмдер тиімділігі кейінгі орындарға қойылады. Басқаша айтқанда, *программалар сапасының қазіргі кездегі критерийлері – бұл,* алдымен, *сенімділік, сонымен қатар программаны жазып бітіру жұмысы мен оны сүйемелдеуді дұрыс жоспарлау.* Осындай мақсаттарға қол жеткізу үшін программа құрылымы қарапайым болып, ол жеңіл оқылуы және жылдам өзгертілуі қажет.

Құрылымдық программалау – бұл белгілі бір ережелерге сүйене отырып, программаның жасалу уақыты мен қателер санын азайтуға, сондай-ақ оны түрлендіруді жеңілдетуге мүмкіндік беретін программалар құру технологиясы. Құрылымдық тәсіл жоба құрудың барлық сатыларын қамтиды, олар: спецификациялау, жобалау, программалаудың өзі және оны тесттен өткізу.

C++ тілінде программалау барысында белгілі бір тәртіпке жүгінудің маңызы аса зор. Бұл тілдің икемділігі мен мүмкіндіктерінің кеңдігі соншалықты, егер бастан-аяқ өзіңізді белгіленген бір шекаралық аймақта ұстамасаңыз, программа тез арада түзету мен басқаруға келмейтін бір ірі «құбыжыққа» айналып кетеді.

Программалаудың құрылымдық тәсілі айтарлықтай ірі жобаларды сәтті құруға мүмкіндік берді, бірақ программалық жабдықтамалар күрделілігі күннен-күнге арта түсті, сол себепті осы қиындықты шешетін құралдарды үздіксіз дамыту қажеттілігі туындады. Құрылымдық программалау идеялары ары қарай объектіге бағытталған программалау (ОБП) ісіне жалғасып, өз шешімін тапты. ОБП – бұл өте көлемді программалық жүйелер құрылымын қарапайым түрде жасап, олардың басқарылуын жеңілдетіп құруға мүмкіндік беретін технологиялар.

Программа құрудың ең жақсы жалғыз ғана тәсілі болмайды. Әртекті және күрделілік деңгейі әртүрлі есептерді шығару үшін программалау технологияларының да бірнеше түрін қолдану қажет. Қарапайым жағдайларда программаларды құрылымдық түрде жазу негіздерін меңгеру жеткілікті. Ал күрделі жобалар құру үшін тілдің толық көлемін еркін меңгеру жеткіліксіз, оған қоса программаларды жобалау мен түзету қағидаларын, стандартты және басқа да кітапханалар мүмкіндіктерін, т.б. білген жөн. Көбінесе есеп неғұрлым күрделенген сайын, оны шешуге қажетті құралдарды меңгеруге соғұрлым көп уақыт керек болады.

Біз тілдің құрамы мен негізгі құрылымын – программаларды жазуға қажетті, бірақ оған толық жеткілікті бола қоймайтын саланы оқып үйренуден бастаймыз.

1-ТАРАУ

C++ тілінің базалық құралдары

Тілдің құрамы

Кез келген табиғи тілдегі мәтінде 4 негізгі элементті бөліп қарастыруға болады, олар: символдар, сөздер, сөз тіркестері және сөйлемдер. Осындай элементтер алгоритмдік тілде де бар, алайда мұнда сөздерді лексемалар (элементар конструкциялар), сөз тіркестерін өрнектер, ал сөйлемдерді операторлар деп атайды. Лексемалар символдардан, өрнектер лексемалар мен символдардан, ал операторлар символдар, өрнектер және лексемалардан құралады (1.1-сурет):

- Тіл алфавиті немесе символдары – бұл тілдегі барлық мәтіндер жазылатын, ары қарай бөлінбейтін негізгі таңбалар;
- Лексема немесе элементар конструкция – өзіндік мағынасы бар тілдің ең кіші бірлігі;
- Өрнек белгілі бір мәннің есептелу ережесін береді;
- Оператор кез келген бір әрекеттің аяқталған спатталуын береді.

1.1-сурет. Алгоритмдік тілдің құрамы

Күрделі іс-әрекетті бейнелеу үшін операторлар тізбегі қажет. Операторлар құрама оператор немесе блок[1] түрінде бірігуі мүмкін. Мұндайда олар бір оператор ретінде қарастырылады.

[1]C++ тілінде блок болып жүйелі жақшаның { } ішіне алынған операторлар тізбегі есептеледі.

Операторлар *орындалатын* және *орындалмайтын* болып бөлінеді. Орындалатын операторлар мәліметтермен атқарылатын әрекеттерді білдіреді. Орындалмайтын операторлар мәліметтерді сипаттау үшін қажет, сондықтан оларды көбінесе сипаттау операторлары немесе жай ғана сипаттауыштар деп те атайды.

Тілдің әрбір элементі синтаксис және семантика арқылы анықталады. Синтаксистік анықтаулар тіл элементтерінің құрылу ережесін білдіреді, ал семантика олардың мағынасы мен қолданылу ережелерін анықтайды.

Біртұтас алгоритммен біріктірілген сипаттаулар жиыны мен операторлар алгоритмдік тілдің программасы болып табылады. Программаны орындау үшін, оны процессорға түсінікті машиналық кодқа аудару керек. Бұл процесс бірнеше сатылардан тұрады. 1.2-суретте осы сатылар C++ тілі үшін көрсетілген.

Программа алдымен *препроцессорға* беріледі, ол мәтін құрамындағы *директиваларды* (мысалы, мәтінге тақырыптық файлдарды – программада қолданылатын элементтердің сипаттамасы сақталған мәтіндік файлдарды қосуды) орындайды.

Алынған программаның толық мәтіні *компилятордың* кіру нүктесіне беріледі де, компилятор лексемаларды тауып алып, тіл грамматикасының негізінде, осы лексемалардан тұратын өрнектер мен операторларды анықтайды. Сонымен қатар, компилятор синтаксистік қателерді табады, егер олар жоқ болса, *объектілік модуль* құрады.

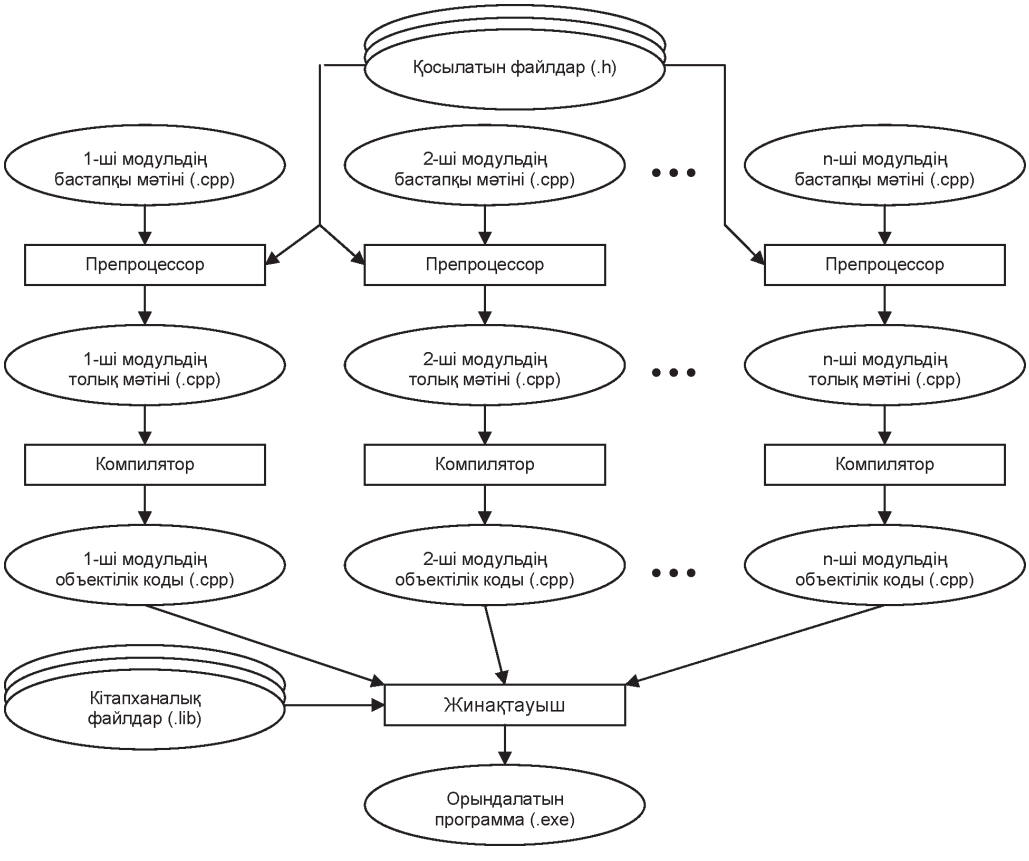
Жинақтауыш (компоновщик) немесе байланыстар редакторы бір объектілік модульге басқа объектілік модульдерді және де кез келген программада болатын (мысалы, экранға мәлімет шығаруды жүзеге асыру үшін) кітапхана функцияларын қамтитын модульдерді біріктіре отырып, программаның *орындалатын модулін* қалыптастырады. Егер программа бірнеше бастапқы файлдардан құралатын болса, олар жеке-жеке түрлендіріліп, осы жинақтау сатысында біріктіріледі. Осының нәтижесінде шығатын атқарылатын модульдің кеңейтілуі `.exe` болады және ол әдеттегі тәсілмен орындауға жіберіледі.

Құжаттамаларда, кітаптарда **тіл ережелерін сипаттау** үшін көбінесе бір формальді метатіл қолданылады, мысалы, Бэкус-Наур формулалары немесе синтаксистік диаграммалар. Бұл кітапта баяндау тілінің қарапайымдылығы мен көрнекілігін сақтау үшін кең таралған сипаттаудың формальді емес түрі қолданылған, мұнда синтаксистік конструкциялардың міндетті түрде қажет етілмейтін бөліктері тік жақшаға алынады да, белгілі бір нақты мәнмен ауыстырылуы тиіс мәтін қазақ тілінде жазылған, ал бірнеше элементтің бірін таңдап алу шарты *вертикаль сызықпен* белгіленген. Мысалы,

```
[ void | int ] атауы();
```

түріндегі жазба C++ тілінің ережелеріне сәйкес **`атауы`** конструкциясының орнына нақты бір басқа ат қойылуы тиіс екенін, ал оның алдында **`void`** не-

месе **int** сөздерінің бірі болуы немесе ешбір түйінді сөз болмауы мүмкін екендігін білдіреді. *Жүйелі жақшалар* бірнеше элементтің ішінен біреуі ғана таңдап алынуы тиіс элементтерді топтастырады. Ал тік жақшалардың өздері синтаксистік элемент ретінде қарастырылатын жағдайда ол ерекше түрде айтылады.



1.2-сурет. Орындалатын программаны құру сатылары

C++ тілін үйренуді ең қарапайым түсініктен, яғни алфавитінен бастаймыз, одан кейін күрделірек элементтерді меңгеріп, объектіге бағытталған программалау ерекшеліктерін оқып, білімімізді тереңдетуге тырысамыз. Осы кітапты оқып шыққан соң оқырман «айқын қатынасу хаттамасы бар инкапсуляцияланған абстракциялар» туралы пікір айтуға, «объект көшірмесінің конструкторы жалпыға қолжетерлік болмай қалған кездегі ерекше жағдайды генерациялаудың» мүмкін болмау себебі және де басқа қызықты мәселелер жөнінде өз ойын қорыта алады.

C++ тілінің алфавиті

C++ тілінің алфавиті келесі таңбаларды қамтиды:

- латын тілінің бас және кіші әріптері және астын сызу белгісі;
- 0-ден 9-ға дейінгі араб цифрлары, A және F арасындағы оналтылық цифрлар;
- арнайы белгілер:
  " { } , | [ ] ( ) + - / % *. \ ` : ? < = > ! & # ~ ; ^
- бос орын символдары: бос орын, табуляция символы, жаңа жолға көшу символы.

Алфавит символдарынан тіл *лексемалары* құралады. Олар:

- идентификаторлар;
- түйінді (резервтегі) сөздер;
- операциялар белгілері;
- тұрақтылар;
- ажыратқыштар (жақшалар, нүкте, үтір, бос орын симводары).

Лексемалардың шекараларық символдары ретінде ажыратқыштар немесе операциялар белгілері сияқты басқа лексемалар қолданылады.

Идентификаторлар

Идентификатор – бұл программалық объектінің атауы. Оны құру үшін латын әріптері, цифрлар және астын сызу белгісін қолдануға болады. Бас әріптер мен кіші әріптер әртүрлі символдар болып саналады, мысалы, **sysop, SySoP** және **SYSOP** – үш түрлі атау. Идентификатордың алғашқы символы әріп немесе астын сызу белгісі болуы тиіс, алғашқы символ ретінде цифрды қолдануға болмайды. Атау символдарының ішіне бос орын қойылмайды.

КЕҢЕС

Программа түсінікті түрде дұрыс оқылуы үшін объектілерге мағынасы бар атаулар берген жөн. Атауларды беру ережелері туралы венгрлік нотация деп аталатын (себебі, бұл нотацияны Microsoft компаниясының ұлты венгр қызметкері ұсынған) келісім бар, осы нотацияға сәйкес идентификатор құрамындағы әрбір сөз бас әріптен басталады да, оның алдына шаманың типіне сәйкес келетін префикс қойылады, мысалы, **iMaxLength, lpfnSetFirstDialog**. Келесі бір қалыптасқан дәстүр – атауды құрайтын сөздерді астын сызу белгілерімен ажыратып жазу, мысалы: **zat_bagasy, max_length, number_of_galosh**.

Стандарт бойынша идентификатордың ұзындығы шектелмеген, бірақ кейбір компиляторлар мен жинақтауыштар (компоновщиктер) оған шектеу қояды. Идентификатор айнымалыларды, функцияларды, типтерді және т.с.с. жариялау кезеңінде құрылады, осыдан кейін оны программаның келесі операторларында қолдануға болады. Идентификаторды таңдауда төмендегі шарттарды ескерген жөн:

- идентификатор түйінді сөздермен және тілдің қолданылатын стандартты объектілерінің атауларымен сәйкес келмеуі тиіс;
- идентификаторларды астын сызу символымен бастамаған жөн, себебі олар жүйелік функциялардың немесе айнымалылардың атауларымен сәйкес келуі мүмкін, сонымен қатар бұл программаның орындалу жылдамдығын төмендетеді;
- сыртқы айнымалыларды анықтау үшін қолданылатын идентификаторларға жинақтауыштар шектеу қояды (әртүрлі жинақтауыштарды немесе жинақтауыш нұсқаларын қолдану сыртқы айнымалылар атауларына да әртүрлі талаптар қояды).

Түйінді сөздер

Түйінді сөздер – бұл компилятор үшін арнайы мағынасы бар алдын ала анықталған (резервтегі) идентификаторлар. Оларды тек өз анықталған мағынасында ғана қолдануға болады. C++ тілінің түйінді сөздерінің тізімі 1.1-кестеде берілген.

1.1-кесте. C++ тілінің түйінді сөздері тізімі

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

Операциялар таңбалары

Операциялар таңбалары, яғни белгілері – бұл операндтармен орындалатын әрекеттерді анықтайтын бір немесе бірнеше символдар тізбегі. Операция таңбаларының ішіне бос орын белгісі қойылмайды. Құрамындағы операндтар

санына байланысты операциялар унарлы, бинарлы, тернарлы болып бөлінеді. Мәтін ішіндегі орнына, яғни қолданылатын ортасына қарай бір таңбаның қызметі әртүрлі болуы мүмкін. Мынадай операциялар **[ ], ( )** және **?** таңбаларынан басқа белгілер жеке лексема қызметін атқарады.

С++ тіліндегі операциялар таңбалары «Операциялар» бөлімінде, 25 б. және де бірінші қосымшада берілген. Стандартты операциялардың басым бөлігін қайта анықтауға (асыра жүктеуге) болады. Операцияларды асыра жүктеу әрекеті 205 б. қарастырылған.

Тұрақтылар

Тұрақтылар деп программадағы мәні өзгермейтін шамаларды айтады. Олар бүтін, нақты, символдық және тіркестік тұрақтылар болып бөлінеді. Компилятор тұрақтыны лексема ретінде белгілеп алып, оны сыртқы бейнесіне қарай белгілі бір типке жатқызады[1].

Әрбір типке сәйкес келетін тұрақтылардың форматтары 1.2-кестеде көрсетілген.

1.2-кесте. С++ тіліндегі тұрақтылар

Тұрақты	**Форматы**	**Мысалдар**
Бүтін	Ондық: санның өзі 0 болмаса, 0-ден басталмайтын ондық цифрлар тізбегі	8, 0, 199226
	Сегіздік: 0-ден басталатын сегіздік цифрлар (0, 1, 2, 3, 4, 5, 6, 7)	01, 020, 07155
	Он алтылық: 0x немесе 0X таңбаларынан басталатын он алтылық цифрлар (0, l, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)	0xA, 0x1B8, 0X00FF
Нақты	Ондық: [цифрлар].[цифрлар][2]	5.7, .001, 35.
	Экспоненциалдық: [цифрлар][.][цифрлар]{E|e}[+|-][цифрлар][3]	0.2E6, .11e-3, 5E10
Символдық	Апострофқа алынған бір немесе екі символ	'A', 'ю', '*', 'db', '\0', '\n\', \012', '\x07\x07'
Тіркестік	Тырнақшаға алынған символдар тізбегі	**"Мұнда Vasia болды", "\tМәні r=\0xF5\n"**

Бүтін және нақты типті тұрақтылардың мүмкін болатын мәндер ауқымдылығы (диапазоны) 1.4-кестеде көрсетілген.

---

[1]Программалаушы тұрақтының типін өзі бере алады.

[2]Бүтін немесе бөлшек бөлігінің біреуі берілмеуі мүмкін, алайда бірден екеуі емес.

[3]Бүтін немесе бөлшек бөлігінің біреуі берілмеуі мүмкін, алайда бірден екеуі емес. Егер екі бөлігі де көрсетілсе, нүкте белгісі міндетті.

Егер теріс таңбалы бүтін немесе нақты тұрақтыны қалыптастыру қажет болса, онда тұрақтының алдына таңбаны өзгертетін унарлы операция белгісі **(-)** қойылады. Мысалы, -218, -022, -0x3C, -4.8, -0.1e4.

Экспоненциалдық форматтағы нақты тұрақты *мантисса* және *дәреже* түрінде беріледі. Мантисса экспонента таңбасының (**E** немесе **e**) сол жағына, дәрежесі – таңбаның оң жағына жазылады. Тұрақтының мәні 10 санының көрсетілген дәрежесі мен мантиссаның көбейтіндісі ретінде анықталады. Сан таңбаларының ішіне бос орын қойылмайтынын, ал санның бүтін бөлігін оның бөлшегінен ажырату үшін үтір емес, нүкте қолданылатынын есте сақтаған жөн.

Бір символдан тұратын символдық тұрақтылар компьютер жадында бір байт орын алады және стандартты **char** типіне жатады. Қос символды тұрақтылар екі байт орын қажет етеді және **int** типіне жатады, мұндағы бірінші символ адресі кіші байтта орналастырылады (мәліметтер типтері туралы келесі бөлімде жазылған).

Кері қиғаш сызық белгісі келесі мақсаттарда қолданылады:

- графикалық кескіні жоқ кодтарды (мысалы, **\a** – дыбыстық сигнал, **\n** – курсорды келесі жолдың басына көшіру) бейнелеу үшін;
- апостроф **(')**, кері қиғаш сызық **(\)**, сұрақ белгісі **(?)** және тырнақша **(")** символдарын бейнелеу үшін;
- кез келген символды оналтылық немесе сегіздік кодтардың көмегімен бейнелеу үшін, мысалы, **\073, \0xF5**. Сандық мәндер 0 мен 255 аралығында болу керек.

1.3-кесте. C++ тілінің басқару тізбектері

Бейнесі	Он алтылық коды	Аталуы
\a	**7**	Дыбыстық сигнал
\b	**8**	Бір адым артқа
\f	**C**	Парақты (форматты) ауыстыру
\n	**A**	Келесі жолға көшу
\r	**D**	Каретканы қайтару
\t	**9**	Горизонталь табуляция
\v	**B**	Вертикаль табуляция
****	**5C**	Кері қиғаш сызық
\'	**27**	Апостроф
\″	**22**	Тырнақша
\?	**3F**	Сұрақ белгісі
\0ddd	**-**	Символдың сегіздік коды
\0xddd	**ddd**	Символдың он алтылық коды

Кері қиғаш сызықтан басталатын символдар тізбектерін *басқару тізбектері* немесе *escape-тізбектер* деп атайды. 1.3-кестеде олардың мүмкін болатын мәндері берілген. Басқару тізбегі жеке бір символ ретінде қабылданады. Егер кері қиғаш сызық белгісінен кейін 1.3-кестеде көрсетілмеген символ кездесетін болса, онда оның нәтижесі белгісіз болып саналады. Ал егер сандар тізбегінде цифрдан өзге символ кездессе, ол сандық кодтың соңы болып есептеледі.

Басқарушы тізбектер тіркестік литералдар деп аталатын тіркестік тұрақтыларда қолданылуы мүмкін. Мысалы, егер тіркес ішіне тырнақша жазу керек болса, оның алдына кері қиғаш сызық қойылады, сол арқылы компилятор бұл тырнақшаның тіркесті шектейтін тырнақшалардан басқа екенін ажырата алады:

```
"\"Питер\" баспа үйі"
```

Барлық тіркестік литералдарды компиляторлар әртүрлі объект ретінде қарастырады.

Программада бос орын символдарымен ғана бөлінген тіркестік тұрақтылар компиляция кезінде біртұтас тұрақты болып біріктіріледі. Бір жолға симайтын ұзын тіркестік тұрақтыны кері қиғаш сызық символы арқылы бөле отырып, бірнеше жолдарға орналастыруға болады. Мұндай бөлу символдарын компилятор жұмыс барысында ескермейді де, келесі жол алдыңғысының жалғасы ретінде қабылданады. Мысалы:

```
"Ешбір адамға өзінің сыртқы \
келбеті ұнамайды,алайда әркімнің \
өз ақылы өз көңілінен шығады"
```

тіркесі келесі тіркеспен толықтай эквивалентті болып саналады:

```
"Ешбір адамға өзінің сыртқы келбеті ұнамайды, алайда әркімнің өз ақылы өз көңілінен шығады"
```

Әрбір тіркестік литералдың соңына компилятор `\0` басқару тізбегімен берілетін *нөлдік символды* қосып қояды. Сондықтан тіркес ұзындығы оның жазуындағы символдар санынан әрқашанда бір символға артық болады. Осылайша, `" "` бос тіркестің ұзындығы 1 байтқа тең болады. Бір символдық тіркес (`"A"`) пен символдық тұрақты (`'A'`) арасындағы айырмашылыққа назар салыңыз.

Бос символдық тұрақты қолданылмайды.

Түсініктемелер

Жол соңындағы түсініктемелер екі «тура қиғаш сызық» символдарынан (`//`) басталып, жаңа жолға көшу символымен аяқталады, ал түсініктеменің екінші түрі `/*` және `*/` жақшалық символдар ішіне жазылады. Түсініктемелер ішінде тек C++ тілі алфавитінің символдарын ғана емес, компьютерде теруге болатын кез келген символдарды енгізе беруге болады. Қабатталған

түсініктеме-жақшаларды (`/*`) пайдалану стандарт бойынша рұқсат етілмейді, бірақ кейбір компиляторда оларды пайдалануға болады.

КЕҢЕС

Түсініктемелер беру үшін `//` таңбаларын қолданған жөн, ал `/*...*/` сияқты жақшаларды программаны түзету кезінде код блоктарының әсерін уақытша алып тастау үшін қолдану ұсынылады. Түсініктемелерді жалпы пайдалану бойынша ұсынылатын кеңестерді «Программаны кодтау және құжаттау» бөлімінен қараңыз, 109 б.

C++ тіліндегі мәліметтер типтері

Мәліметтер типтерінің тұжырымдамасы

Кез келген программаның негізгі мақсаты мәліметтерді өңдеу болып табылады. Әр типтегі мәліметтер әртүрлі жолмен өңделеді және сақталады. Кез келген алгоритмдік тілдегі әрбір тұрақтының, айнымалының, өрнектің немесе функциялардың есептеу нәтижелері белгілі бір типке сәйкес келуі керек.

Мәліметтер типі бойынша төмендегілер анықталады:

- мәліметтердің компьютер жадындағы *ішкі бейнесі*;
- осы типтегі шамалар қабылдай алатын *мәндер жиыны*;
- осы типтегі шамаларға қолдануға болатын *операциялар мен функциялар*;

Осындай сипаттамаларға байланысты программадағы нақты объектілерді бейнелеу үшін қолданылатын әрбір шаманың типі алдын ала таңдалады. Типті міндетті түрде сипаттау арқылы компилятор программаның әртүрлі конструкцияларының жұмыс істеу мүмкіндігін тексере алады. Мәліметтерді өңдеу үшін қолданылатын машиналық командалар шамалардың типтеріне байланысты жұмыс атқарады.

C++ тілінің барлық типтері *негізгі* және *құрама* болып екіге бөлінеді. C++ тілінде бүтін, нақты, символдық және тіркестік шамаларды бейнелеу үшін алты негізгі мәліметтер типі анықталған. Осы типтерді негізге ала отырып, программалаушы құрама типтер сипаттамасын енгізе алады. Оларға жиымдар, тізбелер, функциялар, құрылымдар, сілтемелер, нұсқауыштар, біріктірмелер және кластар жатады.

Мәліметтердің негізгі типтері

Мәліметтердің негізгі (*стандартты*) типтерін көбінесе *арифметикалық* типтер деп те айтады, өйткені оларды арифметикалық операцияларға қолдануға болады. Негізі типтерді сипаттау үшін келесі түйінді сөздер анықталған:

- **`int`** (бүтін);
- **`char`** (символдық);

- **wchar_t** (кеңейтілген символдық);
- **bool** (логикалық);
- **float** (нақты);
- **double** (екі еселенген дәлдіктегі нақты сандар).

Алғашқы төрт тип бүтін сандық (*бүтін),* соңғы екеуі *жылжымалы нүктелі типтер* (*нақты)* деп аталады. Компилятордың бүтін шамаларды өңдеу кезінде қалыптастыратын коды жылжымалы нүктелі шамаларға арналған кодтан өзгеше болып келеді.

Стандартты типтердің компьютердегі ішкі бейнеленуі мен олардың мәндерінің диапазонын нақтылайтын төрт *тип спецификаторы* бар, олар:

- **short** (қысқа);
- **long** (ұзын);
- **signed** (таңбалы);
- **unsigned** (таңбасыз).

Бүтін тип (int)

Компьютер жадындағы **int** типіндегі шамалардың алатын орны, яғни ұзындығы немесе *өлшемі* стандарт бойынша анықталмайды, ол компьютер мен компиляторға байланысты болып келеді. Осы типтегі шамалар үшін 16 разрядты процессорда 2 байт, ал 32 разрядты процессорда 4 байт бөлінеді.

Тип атауының алдында тұратын **short** спецификаторы компиляторға процессордың разрядтылығына қарамастан, санға 2 байт бөлу қажет екендігін білдіреді. Ал **long** спецификаторы бүтін шамаға компьютер жадынан 4 байт орын берілетінін көрсетеді. Сонымен, 16 разрядты компьютерде **int** және **short int** типтері бірдей, ал 32 разрядты компьютерде **int** және **long int** типтері бірдей (яғни эквивалентті) болып саналады.

Бүтін типтегі шаманың компьютер жадындағы *ішкі бейнесі* – екілік кодпен берілген бүтін сан. **signed** спецификаторын қолданғанда санның алғашқы биті оның таңбасын көрсетеді (0 – оң сан, 1 – теріс сан). **unsigned** спецификаторы тек оң сандарды көрсетуге мүмкіндік береді, өйткені мұнда алғашқы разряд сан кодының бөлігі ретінде қарастырылады. Осылайша, **int** типінің мәндер диапазоны спецификаторларға тәуелді болып келеді. IBM PC-мен үйлесімді компьютерлер үшін спецификаторлары әртүрлі бүтін типті шамалардың мәндер диапазоны 1.4-кестеде көрсетілген.

Алдын ала келісім бойынша барлық бүтін типтер таңбалы сандарды бейнелейді, яғни **signed** спецификаторын жазбауға болады.

Программада кездесетін тұрақтылар сыртқы түріне қарай кез келген бір типке жатқызылады. Егер программалаушыға бұл типтен басқаша тип қажет болып жатса, онда ол **L, l (long)** және **U, u (unsigned)** жалғауларының (суффикстерінің) көмегімен керекті типті көрсете алады. Мысалы, **32L**

тұрақтысына **long** типі беріледі және ол компьютер жадынан 4 байт орын алады. **L** және **U** жалғауларын қатар қолдануға да болады, мысалы, **0x22UL** немесе **05Lu.**

ЕСКЕРТУ

Программаларда **short int, long int, signed int** және **unsigned int** типтерін сәйкесінше **short, long, signed** және **unsigned** түрінде қысқартып жазуға болады.

Символдық тип (char)

Символдық типтегі шамаға компьютердегі символдар жиынының ішінен кез келген таңбаны орналастыруға жеткілікті орын бөлінеді, типтің атауы осыған байланысты түрде берілген. Әдетте, бұл 1 байтқа сәйкес келеді. Басқа да бүтін типтер сияқты, **char** типі де таңбалы немесе таңбасыз болуы мүмкін. Таңбасы бар шамаларда -128 бен +127 аралығындағы мәндерді сақтауға болады. Ал, **unsigned** спецификаторын қолданғанда, мәндердің өзгеруі 0 мен 255 сандары аралығында болады. Бұл 256-символдық ASCII кодтары құрамының кез келген символын сақтауға жеткілікті. **char** типіндегі шамалар осы көрсетілген диапазоннан шықпайтын басқа бүтін сандарды сақтау үшін де қолданыла береді.

Кеңейтілген символдық тип (wchar_t)

wchar_t типі символдарды кодтау кезінде 1 байт жеткіліксіз болатын жағдайларда, мысалы, Unicode ортасында жұмыс істеуге арналған. Бұл типтің ені жүзеге асырылуына тәуелді болады; әдетте ол **short** типіне сәйкес келеді. **wchar_t** типіндегі тіркестік тұрақтылар **L** префиксімен жазылады, мысалы, **L"Gates".**

Логикалық тип (bool)

Логикалық типтегі шамалар тек **true** және **false** мәндерін ғана қабылдай алады. **false** мәнін бейнелеудің ішкі формасы – 0 (нөл). Кез келген басқа мән **true** болып қабылданады. Бүтін типке түрлендірілген жағдайда **true** мәні бірге сәйкес келеді.

Жылжымалы нүктелі типтер (float, double, long double)

C++ тілінің стандарты нақты мәндерді сақтауға арналған мәліметтердің үш типін анықтайды, олар: **float, double** және **long double**.

Жылжымалы нүктелі мәліметтер типтері компьютер жадында бүтін санды типтерден басқаша түрде сақталады. Компьютер жадындағы нақты санның ішкі бейнесі 2 бөліктен – мантисса мен дәрежеден тұрады. IBM PC-мен үйлесімді компьютерлерде **float** типіндегі шамалар 4 байт орын ала-

ды, оның екілік разрядтарының алғашқысы мантисса таңбасына, келесі 8 разряды дәрежеге және қалған 23 разряды мантиссаға беріледі. Мантисса – бұл 1.0-ден үлкен, бірақ 2.0-ден кіші сан. Мантиссаның алғашқы цифры әрқашан 1-ге тең болғандықтан, ол сақталмайды.

Компьютер жадынан 8 байт орын алатын **double** типіндегі шамалар үшін сан дәрежесі мен мантиссаға сәйкесінше 11 және 52 разряд бөлінеді. Мантиссаның ұзындығы санның дәлдігін, ал дәреженің ұзындығы оның өзгеру диапазонын анықтайды. 1.4-кестеге қарасақ, **float** және **long int** типіндегі шамаларға бөлінетін байт саны бірдей болғанымен, олардың ішкі бейнелену формасына байланысты мәндерінің өзгеру диапазоны әртүрлі болып келеді.

double типі атауының алдындағы **long** спецификаторы сандық шамаға компьютер жадынан 10 байт орын бөлінетінін көрсетеді.

Жылжымалы нүктелі тұрақтыларға келісім бойынша **double** типі беріледі. Тұрақтының типін **F, f (float)** және **L, l (long)** жалғауларының көмегімен нақты түрде көрсетуге болады. Мысалы, **2E+6L** тұрақтысының типі **long double**, ал **1.82f** тұрақтысының типі **float** болады.

1.4-кесте. IBM PC үшін қарапайым мәліметтер типтері мәндерінің диапазондары

Типі	Мәндер диапазоны	Мөлшері (байт)
bool	true және false	1
signed char	-128... 127	1
unsigned char	0 ... 255	1
signed short int	-32 768... 32 767	2
unsigned short int	0... 65 535	2
signed long int	-2 147 483 648 ... 2 147 483 647	4
unsigned long int	0 ... 4 294 967 295	4
float	3.4e-38 ... 3.4e+38	4
double	1.7e-308 ... 1.7e+308	8
long double	3.4e-4932 ... 3.4e+4932	10

1.4-кестеде нақты типтер үшін олардың ең кіші және ең үлкен мәндерінің абсолюттік шамалары берілген.

Әртүрлі платформаларға ауыстырылатын программаларды жазу кезінде **int** типінің мөлшері туралы жорамал жасауға болмайды. Оны анықтау үшін **sizeof** операциясын қолдану керек, соның нәтижесінде байтпен берілген типтің мөлшері, яғни ұзындығы табылады. Мысалы, MS-DOS операциялық жүйесі үшін **sizeof(int)** функциясының нәтижесі 2 болса, ол Windows 9x немесе OS/2 жүйесі үшін 4-ке тең болады.

ANSI стандартында негізгі типтер үшін мәндер диапазондары берілмейді, тек олардың ұзындықтары арасындағы қатынас анықталады. Мысалы,

```
sizeof(float) ≤ sizeof(double) ≤ sizeof(long double)
sizeof(char) ≤ sizeof(short) ≤ sizeof(int)≤ sizeof(long)
```

ЕСКЕРТУ

Бүтін типтер үшін олардың ең кіші және ең үлкен мүмкін болатын мәндері типтің жүзеге асырылуына байланысты болып келеді және олар **<limits.h>** (**<climits>**) тақырыптық файлында, ал нақты типтердің сипаттамалары **<float.h>** (**<cfloat>**) файлында, сонымен қатар **numeric_limits** класының шаблонында көрсетілген («Стандартты кітапхананың басқа құралдары» бөлімін және 5-қосымшаны қараңыз, 411 б.).

Мәліметтерді бейнелеу дәлдігімен және олардың өзгеру диапазонымен ерекшеленетін бүтін және нақты типтердің бірнеше түрлері программа құрушыға компьютердің нақты аппаратуралары мүмкіндіктерін барынша тиімді қолдану үшін енгізілген. Өйткені программаны орындау жылдамдығы мен пайдаланылатын компьютер жадының көлемі көбінесе қолданылатын типтерді таңдауға байланысты болып келеді. Алайда компьютердің белгілі бір түрі үшін тиімді түрде жұмыс істейтін программа басқа платформаларға көшіру мүмкіндігінен айырылуы мүмкін, сондықтан, жалпы жағдайда, мәліметтер типтерінің нақты сипаттамаларына тәуелді болмауға тырысу керек.

void типі

Аталған типтерден басқа C++ тілінің негізгі типтеріне **void** типі де жатады, бірақ бұл типтің мәндер жиыны бос болып табылады. Ол мән қайтармайтын функцияларды анықтау үшін, функцияның аргументтерінің жоқ екенін көрсету үшін (функциялар жайында 74 б. айтылған), нұсқауыштардардың базалық типі ретінде (49 б.) және типтерді түрлендіру операциясында (54 б.) қолданылады.

Программа құрылымы

C++ тіліндегі программа *функциялардан, сипаттамалардан* және *препроцессор директиваларынан* тұрады. Функциялардың біреуінің атауы **main** болуы тиіс. Программаның орындалуы осы функцияның бірінші операторынан басталады. Функцияны анықтаудың қарапайым форматы төмендегідей болады:

```
қайтарылатын_мән_типі атауы ([ параметрлер ]){
    функция тұлғасын құратын операторлар
}
```

Көбінесе функция кез келген бір мәнді есептеу үшін қолданылады, сондықтан функция атауының алдында оның типі көрсетіледі. Функциялар туралы 74 б. айтылады, ал төменде ең маңызды болып табылатын мәліметтер келтірілген, олар:

- □ егер функция мән қайтармауы тиіс болса, **void** типі көрсетіледі;
- □ функцияның тұлғасы блок болып табылады, сондықтан ол жүйелі жақшаға алынады;
- □ функциялар қабаттасқан түрде берілмейді;
- □ әрбір оператор нүктелі үтірмен аяқталады (құрама оператордан басқасы).

Құрамында **main, f1** және **f2** сияқты функциялары бар программа мысалы:

```
Препроцессор директивалары
сипаттамалар
int main(){
   негізгі функция операторлары
}
int f1(){
   f1 функциясының операторлары
}
int f2(){
   f2 функциясының операторлары
}
```

Программа бірнеше *модульден* (бастапқы файлдардан) құралуы мүмкін.

Енгізу/шығару туралы бірқатар алдын ала ескертулер. C++ тілінде құрамдас енгізу/шығару құралдары жоқ, ондай операциялар стандартты кітапханалардағы функциялар, типтер және объектілер көмегімен жүзеге асырылады. Көбінесе екі тәсіл қолданылады: C тілінен мұраланған функциялар және C++ тілінің объектілері.

C тілі стиліндегі негізгі *енгізу/шығару функциялары*:

```
int scanf (const char* format, ...)   // енгізу
int printf(const char* format, ...)   // шығару
```

Бұлар шамалардың кез келген санын **format** форматы тіркесіне сәйкес форматталған түрде енгізу және шығару әрекеттерін орындайды. Формат тіркесінде мәлімет шығару кезінде ағымға (экранға) көшірілетін немесе мәлімет енгізу кезінде ағымнан (пернетақтадан) шақырылатын символдар және % белгісінен басталатын, енгізу және шығару кезінде нақты шамалармен алмастырылатын *түрлендіру спецификациялары* болады. Программалау кезінде жиі қолданылатын түрлендіру спецификацияларының тізімі 2-қосымшада берілген.

C тілі стиліндегі енгізу/шығару функцияларын пайдаланатын программаның мысалы:

```
#include <stdio.h>
int main(){
   int i;
   printf("Бүтін сан енгізіңіз\n"):
   scanf("%d", &i);
   printf("Сіз %d санын енгіздіңіз, рахмет!", i);
   return 0;
}
```

Бұл программаның бірінші жолы – препроцессор директивасы, ол программада қолданылған енгізу/шығару функцияларының сипаттамалары орналасқан **<stdio.h>** тақырыптық файлын программа мәтініне кірістіреді (бұл жердегі бұрыштық жақшалар тілдің элементі болып табылады). Препроцессор директиваларының барлығы да **#** белгісінен басталады.

Үшінші жолда бүтін типті **i** айнымалысының сипаттамасы берілген. Айнымалылар 22 б. қарастырылған.

Төртінші жолдағы **printf** функциясы «Бүтін сан енгізіңіз» мәтінін экранға шығарады да, **\n** басқару тізбегіне сәйкес курсор жаңа жолға көшеді. Келесі **scanf** функциясы пернетақтадан енгізілген бүтін санды **i** айнымалысына меншіктейді (**&** белгісі адресті алу операциясын білдіреді), ал келесі оператор берілген тіркес құрамындағы түрлендіру спецификациясын осы санның мәніне алмастырып, осы тіркесті экранға шығарады. C тілі стиліндегі енгізу/шығару операциялары «Енгізу/шығару функциялары» бөлімінде (92 б.) қарастырылған.

Жоғарыда қарастырылған мысал үшін C++ кластар кітапханасын пайдалану программасы келесідей түрде жазылады:

```
#include <iostream.h>
int main(){
   int i;
   cout << "Бүтін сан енгізіңіз\n";
   cin >> i;
   cout << "Сіз " << i << " санын енгіздіңіз, рахмет!";
   return 0;
}
```

Мұндағы **<iostream.h>** тақырыптық файлы енгізу/шығаруды басқаруға арналған кластар жиынының сипаттамасын қамтиды. Мұнда мәліметтерді пернетақтадан енгізуге арналған **cin** және мәндерді экранға шығаруға арналған **cout** объект-ағымдары, сонымен қатар ағымға орналастыру **<<** және ағымнан оқу **>>** операциялары анықталған. Ағымдар «Ағымдық кластар» бөлімінде 283 б. қарастырылған.

Келесі бөлімдерде осы сипатталған екі тәсіл де қатар қолданылады, бірақ оларды бір программада араластыра отырып, қатар қолдануға болмайды.

Айнымалылар және өрнектер

Кез келген программада есептеулер жүргізу қажет болады. Мәндерді есептеу үшін операндтардан, операция белгілері мен жақшалардан тұратын *өрнектер* қолданылады. Операндтар есептеуге қажетті мәліметтерді береді. Операциялар орындалуы тиіс әрекеттерді анықтайды. Әрбір операнд, өз кезегінде, өрнек немесе оның дербес бір түрі, мысалы, тұрақты немесе айнымалы болып табылады. Операциялар приориттеріне (басымдылығына) сәйкес орындалады. Операциялардың орындалу ретін өзгерту үшін жай дөңгелек жақшалар қолданылады.

Өрнектердің құрамдас бөліктерін және оларды есептеу ережелерін қарастырайық.

Айнымалылар

Айнымалы – бұл белгілі бір типтегі мәліметтерді сақтай алатын, атау берілген компьютер жады аймағы. Айнымалылардың аты және мәні болады. Оның аты мәні сақталатын жады аймағын пайдалану үшін қызмет етеді. Программаны орындау барысында айнымалының мәнін өзгертуге болады. Кез келген айнымалыны қолданудың алдында оны сипаттау керек.

Аты **a** болатын бүтін және аты **x** болатын нақты айнымалыларды сипаттау мысалы:

```
int a; float x;
```

Айнымалыларды *сипаттау операторының* жалпы түрі:

```
[жады класы] [const] типі аты [инициалдаушы];
```

Осы оператордың құрамындағы бөліктерді жазу ережелерін қарастырайық:

- Міндетті түрде көрсетілмейтін жады класы **auto**, **extern**, **static** және **register** мәндерінің біреуін қабылдай алады. Олар туралы төменде айтылады.
- **const** модификаторы программада айнымалының мәнін өзгертуге болмайтынын көрсетеді. Мұндай айнымалыны *атаулы тұрақты* немесе жай ғана *тұрақты* деп атайды;
- Сипаттау кезінде айнымалыға бастапқы мән меншіктеуге болады, бұл *инициалдау* деп аталады. Инициализаторды екі формада жазуға болады, алғашқысы – теңдік белгісімен:

  ```
  = мәні
  ```

  немесе дөңгелек жақшада:

  ```
  ( мәні )
  ```

Тұрақты жариялану кезінде инициалдануы керек. Бір операторда бір типтегі бірнеше айнымалыны үтір арқылы бөле отырып, сипаттауға болады.

Мысалдар:

```
short int a = 1;          // a бүтін айнымалы
const char C = 'C';       // C символдық тұрақты
char s, sf = 'f';         // инициалдау тек sf-ке қатысты
char t (54);
float c = 0.22, x(3), sum;
```

Егер инициалдаушы мәннің типі айнымалы типімен сәйкес келмесе, белгілі ережелер бойынша *типтерді түрлендіру* орындалады (3-қосымша және 32 б.).

Тип пен жады класынан басқа айнымалы сипаттамалары айқын түрде немесе келісім бойынша оның *әрекет ету аймағын* береді. Жады класы және әрекет ету аймағы тек сипаттаманың өзіне ғана емес, сонымен қатар оның программа мәтініндегі орналасуына да тәуелді болып келеді.

Идентификатордың әрекет ету аймағы – идентификатормен байланысқан жады аймағын пайдалану (онымен қатынас құру) үшін осы идентификатордың өзін қолдануға болатын программа бөлігі. Әрекет ету аймағына байланысты айнымалы жергілікті (локалды) немесе ауқымды (глобалды) болуы мүмкін.

Егер айнымалы блок ішінде анықталған болса (блок жүйелі жақшамен қоршалып тұрады), ол *жергілікті* айнымалы болып табылады да, оның әрекет ету аймағы сипатталу нүктесінен блок соңына дейінгі аралықты қамтиды. Ал егер айнымалы кез келген блоктан тыс анықталған болса, ол *ауқымды* айнымалы болып саналады да, оның әрекет ету аймағы болып осы айнымалы анықталған сипаттау нүктесінен файл соңына дейінгі аймақ есептеледі.

Жады класы программалық объектінің (жеке жағдайда, айнымалының) өмірлік уақытын, яғни пайдаланылу кезеңін және көріну аймағын анықтайды. Егер жады класы айқын түрде көрсетілмесе, оны компилятор жариялану мәтініне тәуелді түрде (контекстіне байланысты) анықтайды.

Айнымалының *өмірлік кезеңі* (уақыты*)* тұрақты (программаны орындап біткенше) және уақытша (блокты орындап біткенше) болуы мүмкін.

Идентификатордың көріну аймағы деп идентификатормен байланысты жады аймағына қарапайым қатынас құру мүмкін болатын программа мәтінінің бөлігін айтады. Көбінесе идентификатордың көріну аймағы оның әрекет ету аймағымен бірдей болып келеді. Мұндағы ерекше жағдай – ішкі блокта да дәл осындай атпен сипатталатын айнымалы болған кезде туындайды. Мұндайда сыртқы айнымалы ішкі блоктың әрекет ету аймағына кіргенменен, ол онда, яғни ішкі блокта көрінбейді. Дегенмен, егер бұл айнымалы ауқымды болса, онда оның көріну аймағына қатынас құру операциясын `::` пайдалана отырып, оны қолдана беруге болады.

Жады класын беру үшін келесі спецификаторлар қолданылады:

□ **`auto`** – *автоматтық* айнымалы. Ол үшін жады стектен бөлінеді және қажет болғанда айнымалының анықтамасы орналасқан оператордың әрбір орындалуы кезінде ол инициалданып отырады. Жадыны босату айнымалы

сипатталған блоктан шығатын кезде орындалады. Оның өмірлік кезеңі – сол айнымалы сипатталған сәттен блок соңына дейінгі аралық болып табылады. Ауқымды айнымалылар үшін бұл спецификатор қолданылмайды, ал жергілікті айнымалылар үшін ол алдын ала келісім бойынша қабылданады, сондықтан оны айқын түрде берудің онша қажеті жоқ.

□ **extern** – айнымалы программаның басқа жерінде (басқа файлда немесе мәтіннің төменгі бөлігінде) анықталатынын көрсетеді. Олар өздері жарияланған программаның барлық модульдерінде қолдануға болатын айнымалыларды құру үшін қолданылады[1]. Сыртқы айнымалыларды қолдану туралы толығырақ «Сыртқы жарияланымдар» бөлімінде айтылады, 104 б.

□ **static** – *статикалық* айнымалы. Өмірлік кезеңі – тұрақты. Ол айнымалының анықтамасы орналасқан операторлың алғашқы орындалуы кезінде бір рет инициалданады. Сипаттау операторының орналасуына байланысты статикалық айнымалылар жергілікті немесе ауқымды болуы мүмкін. Ауқымды статикалық айнымалылар өздері сипатталған модульде ғана көрінеді (яғни пайдаланылады).

□ **register** – **auto** спецификаторына ұқсас, бірақ мұнда айнымалыға, мүмкіндігінше, процессор регистрлерінен орын (жады) бөлінеді. Егер компиляторда мұндай мүмкіндік жоқ болса, айнымалылар **auto** сияқты өңделеді.

```
int a;                  // 1 a ауқымды айнымалысы
int main() {
   int b;               // 2 b жергілікті айнымалысы
   extern int x;        // 3 x айнымалысы басқа жерде
                        //   анықталған
    static int c;       // 4 c жергілікті статикалық
                        //   айнымалысы
    a = 1;              // 5 ауқымды айнымалыны меншіктеу
    int a;              // 6 a жергілікті айнымалысы
    a = 2;              // 7 жергілікті айнымалыны меншіктеу
    ::a = 3;            // 8 ауқымды айнымалыны меншіктеу
    return 0;
}
int x = 4;              //9 x-ті анықтау және
                        //  инициалдау
```

Бұл мысалда **a** ауқымды айнымалысы барлық блоктан тыс анықталған. Ол үшін жады программа жұмысының басында мәліметтер сегментінде бөлінеді, ал оның әрекет ету аймағы бүкіл программа көлемін қамтиды. Көріну аймағы – 6-8 жолдан басқа барлық программа жолдары, өйткені 6-жолда дәл осындай атпен берілген жергілікті айнымалы анықталады, мұның әрекет ету аймағы сипаттау нүктесінен (6-жолдан) басталып, блоктың соңында аяқталады. **b** және **c** – жергілікті айнымалылар, олардың көріну аймағы – блок іші, бірақ өмірлік

[1]Егер айнымалы дәл сол операторда инициалданса, онда **extern** спецификаторы елеусіз қалады.

кезеңдері әртүрлі: **b** үшін жады стектен блокқа кіру кезінде бөлінеді де, одан шығарда босатылады, ал **c** айнымалысы мәліметтер сегментінде орналасады да, ол программаның жұмысы аяқталғанша сақталады.

Егер айнымалыларды анықтау кезінде бастапқы мән бірден берілмесе, компилятор ауқымды және статикалық айнымалыларға өз типтеріне сәйкес нөл мәнін меншіктейді. Ал автоматты айнымалылар инициалданбайды.

Айнымалының аты өзінің әрекет ету аймағында қайталанбайтын бірегей болуы тиіс (мысалы, бір блокта екі айнымалының аты бірдей болуы мүмкін емес). Әрекет ету аймақтары толығырақ 103 б. айтылады.

КЕҢЕС

Айнымалыларға мағынасына сәйкес лайықты ат ойлап табуға уақыт үнемдемеңіздер. Атау сақталатын шаманың мағынасын көрсетуі тиіс, сонымен қатар ол жеңіл есте қалатындай болғаны дұрыс, олардың аттары бір-бірімен шатастыруға болатын символдардан, мысалы `1, l` (кіші **L**) немесе **I** (үлкен **i**), тұрмауы тиіс. Берілген атаудың бөліктерін ажырату үшін астын сызу белгісін қолдануға болады. Айнымалыларға шетелдік слэнг сөздерден тұратын атау бермеуге тырысыңыз – халықтың «қайыққа қандай ат берсеңіз, солай жүзеді» деген мәтел сөзі есте болсын. Көбінесе әрекет ету аймағы кеңірек айнымалыларға ұзынырақ атаулар беріледі (оның тип префиксі болғаны жөн), ал программаның бірнеше жолында ғана әрекет ететін айнымалылар үшін бір әріптен тұратын ат қойып, оған түсініктеме беру жеткілікті.

Айнымалыны сипаттау *жариялау* немесе *анықтау* түрінде берілуі мүмкін. *Жариялау* компиляторға айнымалы типі мен жады класы туралы мәлімет береді, ал *анықтама* құрамында компиляторға айнымалы типіне сәйкес жады бөлу туралы нұсқау беріледі. C++ тілінде жариялаулардың басым бөлігі анықтамалар да болып табылады. Жоғарыдағы мысалда тек үшінші сипаттама ғана анықтамаға жатпайтын жариялау болып табылады.

Айнымалы бірнеше рет жариялануы мүмкін, бірақ программаның тек бір жерінде ғана анықталады, себебі жариялау тек айнымалының қасиеттерін сипаттайды, ал анықтама оны жадының нақты бір аймағымен байланыстырады.

Операциялар

1.5-кестеде C++ тілінде анықталған негізгі операциялар тізімі өздерінің басымдықтарына сәйкес (басымдықтардың кемуі бойынша, әртүрлі басымдықты операциялар сызықтармен бөлініп берілген) көрсетілген. Басқа операциялар жайлы оларды түсіндіру барысында айтылады. Операциялардың толық тізімі 1-қосымшада көрсетілген.

Құрамындағы операндтар санына сәйкес операциялар унарлы (бір операнд), бинарлы (екі операнд) және тернарлы (үш операнд) болып бөлінеді.

Кестеде келтірілген операциялар, **sizeof** және шартты операциядан басқалары, асыра жүктелуі мүмкін. Операциялардың асыра жүктелуі туралы келесі бөлімдерде (205 б.) ақпарат беріледі.

Енді негізгі операцияларды толығырақ қарастырайық.

1.5-кесте. C++ тілінің негізгі операциялары

Операция	Қысқаша сипаттамасы
Унарлы операциялар	
`++`	1-ге арттыру
`--`	1-ге кеміту[1]
`sizeof`	өлшем (мөлшер, көлем, ен немесе ұзындық)
`~`	разрядтары бойынша терістеу
`!`	логикалық терістеу
`-`	арифметикалық терістеу (унарлық минус)
`+`	унарлық плюс
`&`	адресті алу
`*`	адрессіздендіру (адресті жоғалту)
`new`	жадыны бөлу
`delete`	жадыны босату
`(type)`	типті түрлендіру
Бинарлы және тернарлы операциялар	
`*`	көбейту
`/`	бөлу
`%`	бөлгендегі қалдықты табу
`+`	қосу
`-`	азайту
`<<`	солға ығыстыру
`>>`	оңға ығыстыру
`<`	кіші
`<=`	кіші немесе тең (үлкен емес)
`>`	үлкен
`>=`	үлкен немесе тең (кіші емес)
`==`	тең
`!=`	тең емес
`&`	разрядтары бойынша конъюнкция (ЖӘНЕ)
`^`	разрядтары бойынша аластайтын НЕМЕСЕ
`|`	разрядтары бойынша дизъюнкция (НЕМЕСЕ)
`&&`	логикалық ЖӘНЕ
`||`	логикалық НЕМЕСЕ
`?:`	шартты операция (тернарлық)
`=`	меншіктеу

[1]Операция ішіндегі символдар арасында бос орындар болмауы тиіс.

Операция	Қысқаша сипаттамасы
*=	көбейтіп барып меншіктеу
/=	бөліп барып меншіктеу
%=	бөлгендегі қалдықты тауып барып меншіктеу
+=	қосып барып меншіктеу
-=	азайтып барып меншіктеу
<<=	солға ығыстырып алып меншіктеу
>>=	оңға ығыстырып алып меншіктеу
&=	меншіктеуі бар разрядтары бойынша ЖӘНЕ
|=	меншіктеуі бар разрядтары бойынша НЕМЕСЕ
^=	меншіктеуі бар разрядтар бойынша аластайтын НЕМЕСЕ
.	тізбектей отырып есептеу

1-ге арттыру және кеміту операциялары (++ және --). Инкремент және декремент деп те аталатын бұл операциялардың екі түрлі жазылу формасы бар – *префикстік* нұсқада операция операнд алдында жазылады және *постфикстік* нұсқада – операциядан кейін көрсетіледі. Префикстік формада алдымен операнд өзгереді, содан кейін оның мәні өрнекті есептеуге қатысады, ал постфикстік формада өрнекті есептеуге операндтың бастапқы мәні алынады да, содан соң операнд өзгертіліп, өрнектің нәтижелік мәні есептеледі.

```
#include <stdio.h>
int main(){
   int x = 3, y = 3;
   printf ("Префикстік өрнектің мәні: %d\n", ++x);
   printf ("Постфикстік өрнектің мәні: %d\n", y++);
   printf ("Қосудан кейінгі x мәні: %d\n", x);
   printf ("Қосудан кейінгі y мәні: %d\n", y);
   return 0;
}
```

Программаның орындалу нәтижесі:

```
Префикстік өрнектің мәні: 4
Постфикстік өрнектің мәні: 3
Қосудан кейінгі x мәні: 4
Қосудан кейінгі y мәні: 4
```

Жалпы инкремент операциясының операнды *L-мән* (*L-value*) деп аталады. Мән енгізуге болатын белгілі бір жады бөлігін адрестейтін кез келген өрнек осылай белгіленеді. Бұл атау меншіктеу операциясынан шыққан, себебі операцияның нәтижесі жадының қай аймағына енгізілетінін оның сол жақ бөлігі (Left) анықтайды. Айнымалы L-мәннің дербес бір түрі болып табылады.

Объектінің немесе типтің пайдаланатын жады көлемін анықтайтын **sizeof** операциясы оның мөлшерін байтпен есептеуге арналған, бұл операцияны жазудың екі формасы бар:

```
sizeof өрнек          және
sizeof ( тип )
```

Мысал (C++ тілі стилінде):

```
#include <iostream.h>
int main(){
   float x = 1;
   cout << "sizeof (float):" << sizeof (float);
   cout << "\nsizeof x:" << sizeof x;
   cout << "\nsizeof (x + 1.0):" << sizeof (x + 1.0);
   return 0;
}
```

Программа жұмысының нәтижесі:

```
sizeof (float): 4
sizeof x: 4
sizeof (x + 1.0): 8
```

Соңғы өрнек нәтижесінің 8 байт болып тұрғаны – алдын ала келісім бойынша нақты тұрақтылар ені ұзынырақ **double** типіне келтіріледі, сондықтан **x** айнымалысы мен өрнектің толық мәнінің типі осы **double** типіне сәйкес келеді. Жақшалар оның ішіндегі өрнектің мәні басымдығы қосу амалынан жоғары болып табылатын типтерді түрлендіру операциясынан бұрын есептелуі үшін қажет.

Терістеу операциялары (- , ! және ~). *Арифметикалық терістеу* (унарлық минус -) бүтін немесе нақты типтегі операнд таңбасын қарама-қарсыға ауыстырады. *Логикалық терістеу* (!) операнд мәні ақиқат (нөл емес) болса, нәтиже ретінде 0 мәнін, ал операнд 0-ге тең болса, 1 мәнін қайтарады. Операнд бүтін немесе нақты типте немесе нұсқауыш типінде болуы да мүмкін. *Разрядтар бойынша терістеу* (~) көбінесе биттік терістеу деп аталады, ол бүтін типті операндтың екілік кодындағы әрбір разрядты инверсиялайды, яғни кері кодпен (1 → 0, ал 0 → 1) жазып шығады.

Бөлу (/) және бөлгенде шыққан қалдықты табу (%). *Бөлу операциясы* арифметикалық типтегі операндтарға қолданылады. Егер операндтардың екеуі де бүтін типте болса, онда операция нәтижесі төменгі бүтін санға дейін дөңгелектенеді, кері жағдайда оның нәтижесі типі түрлендіру ережелеріне сәйкес анықталады («Өрнектер» бөлімін, 32 б. және 3-ші қосымшаны қараңыз). *Бөлуден шыққан қалдықты табу операциясы* тек бүтін типтегі операндтарға қоданылады. Нәтиженің таңбасы есептеуге қатынасатын операндтарға байланысты болады.

```
#include <stdio.h>
int main(){
   int x = 11, y = 4;
   float z = 4;
   printf("Бөлу нәтижелері: %d %f\n", x/y, x/z);
   printf("Қалдық: %d\n", x%y);
   return 0;
}
```

Программа орындалуының нәтижесі:

```
Бөлу нәтижелері: 2 2.750000
Қалдық: 3
```

Ығыстыру операциялары (<< және >>) бүтін типті опернадтарға қолданылады. Олар бірінші операндтың екілік кодын екінші операндта берілген мәнге сәйкес солға немесе оңға ығыстырады. *Солға ығыстыру* (<<) кезінде босатылған разрядтар нөлмен толтырылады. Ал *оңға ығыстыруда* (>>) бірінші операнд таңбасыз типте болса, босаған биттер нөлмен, кері жағдайда таңбалық разрядпен толтырылады. Ығыстыру операциялары разрядтардың толып кетуі мен дәлдікті жоғалтуды есепке алмайды.

Қатынас операциялары (<, <=, >, >=, ==, !=) бірінші операндты екінші операндпен салыстырады. Операндтар арифметикалық типте немесе нұсқауыштар түрінде болуы мүмкін. Операция нәтижесі **true** немесе **false** болады (кез келген нөлге тең емес мән **true** болып есептеледі). Тең және тең емес сияқты салыстыру операцияларының басымдықтары басқа салыстыру операцияларына қарағанда төмен болады.

НАЗАР АУДАРЫҢЫЗ

Екі мәннің тең екендігін тексеру (==) мен меншіктеу операциясының (=) арасындағы айырмашылыққа назар аударыңыз, меншіктеу опрециясының нәтижесі сол жақ операндқа меншіктелген мән болып табылады.

Разрядтар бойынша орындалатын операциялар (&, |, ^) тек бүтін типтегі операндтарға қолданылады және олар тек екілік кодтармен жұмыс істейді. Операцияларды орындау кезінде операндтар әрбір бит бойынша (бірінші операндтың алғашқы биті екінші операндтың алғашқы битімен, бірінші операндтың екінші биті екінші операндтың екінші битімен, т.с.с.) бір-бірімен салыстырылады.

Разрядтар бойынша конъюнкция немесе *разрядтар бойынша ЖӘНЕ* (операция белгісі &) операциясының орындалуы кезінде екі операндтың да сәйкес биттері бірге тең болғанда ғана нәтижелік бит бірге тең болуы тиіс.

Разрядтар бойынша дизъюнкция немесе *разрядтар бойынша НЕМЕСЕ* (операция белгісі |) операциясы кезінде салыстырылатын екі биттің біреуінің немесе екеуінің де мәні бірге тең болса, нәтижелік бит 1-ге тең болады.

Разрядтар бойынша аластамалы НЕМЕСЕ (операция белгісі ^) екі биттің тек біреуінің ғана мәні 1-ге тең болғанда нәтижелік бит бірге тең болып келеді.

```
#include <iostream.h>
int main(){
   cout << "\n 6 & 5 = " << (6 & 5);
   cout << "\n 6 | 5 = " << (6 | 5);
   cout << "\n 6 ^ 5 = " << (6 ^ 5);
   return 0;
}
```

Программа жұмысының нәтижесі:

```
6 & 5 = 4
6 | 5 = 7
6 ^ 5 = 3
```

Логикалық операциялар (&& және ||). ЖӘНЕ (&&) мен НЕМЕСЕ (||) логикалық операцияларының операндтары арифметикалық типте нұсқауыштар түрінде болуы мүмкін, оның үстіне әрбір операциядағы операндтар әр түрлі типте болуы мүмкін. Мұнда типтерді түрлендіру орындалмайды, әрбір операндтың нөлге эквиваленттілігі бағаланады (нөлге тең операнд **false**, нөлге тең емес операнд **true** ретінде қарастырылады).

Логикалық операцияның нәтижесі **true** немесе **false** болады. *Логикалық ЖӘНЕ* операциясының нәтижесі оның екі операндының да мәні **true** болса ғана **true** мәніне ие болады. Ал *логикалық НЕМЕСЕ* операциясының нәтижесі операндтардың кем дегенде біреуінің мәні **true** болса, **true** мәнін қабылдайды. Логикалық операциялар солдан оңға қарай орындалады. Егер операция нәтижесін анықтау үшін бірінші операнд мәні жеткілікті болса, онда екінші операнд есептелмейді.

Меншіктеу операциялары (=, +=, -=, *= және т.б.). Меншіктеу операциялары программада аяқталған операторлар ретінде қолданылуы мүмкін.

Қарапайым меншіктеу операциясының (=) форматы:

```
операнд_1 = операнд_2
```

Мұнда бірінші операнд L-мәнді, екіншісі өрнек болуы тиіс. Алдымен операцияның оң жағында орналасқан өрнек есептеледі, содан кейін оның нәтижесі сол жақ бөлігінде көрсетілген жады аймағына жазылады (мнемоникалық ереже: «меншіктеу – бұл мәліметтерді “сол жақ бөлікке” беру»). Осы жады аймағында бұрын сақталған мәлімет, әрине, жоғалады.

```
#include <iostream.h>
int main(){
   int a = 3, b = 5, c = 7;
   a = b; b = a; c = c + 1;
   cout << "a = " << a;
   cout << "\t b = " << b;
```

```
    cout << "\t c = " << c;
    return 0;
}
```

Программа нәтижесі:

```
a = 5   b = 5   c = 8
```

НАЗАР АУДАРЫҢЫЗ

Меншіктеу кезінде өрнек типін L-мән типіне түрлендіру орындалады, бұл ақпараттың жоғалуына әкеліп соғуы мүмкін.

Күрделі меншіктеу операцияларында (+=, *=, /=, т.б) оның оң жақ бөлігінде орналасқан өрнекті есептеу кезінде сол жақ бөліктегі L-мән де қолданылады. Мысалы, *қоса отырып меншіктеуде* екінші операндқа біріншісі қосылып, нәтижесі бірінші операндқа жазылады, демек **a += b** өрнегі **a = a + b** өрнегінің жинақы түрде қысқаша жазылуы болып табылады.

Шартты операция (?:). Бұл операция тернарлы, яғни үш операндтан тұрады. Оның жазылу форматы:

```
операнд_1 ? операнд_2 : операнд_3
```

Бірінші операнд арифметикалық типте немесе нұсқауыш болуы мүмкін. Ол нөлге эквиваленттілігі тұрғысынан бағаланады (нөлге тең операнд **false**, нөлге тең емес операнд **true** ретінде қарастырылады). Егер 1-операндты есептеу нәтижесі **true** болса, онда шартты операцияның нәтижесі екінші операнд мәніне, кері жағдайда үшінші операнд мәніне тең болады. Әрқашан екінші немесе үшінші операнд есептеледі. Олардың типтері әртүрлі болуы мүмкін. Шартты операция **if** шартты операторының қысқартылған жазылу формасы болып табылады (ол 36 б. көрсетілген).

```
#include <stdio.h>
int main(){
    int a = 11, b = 4, max;
    max = (b > a) ? b : a;
    printf("Үлкен сан: %d", max);
    return 0;
}
```

Программа жұмысының нәтижесі:

```
Үлкен сан: 11
```

Шартты операцияны қолданудың тағы бір мысалы: егер кездейсоқ бүтін шаманың мәні берілген **n** санынан кем болса, осы кездейсоқ шаманы бірге арттыру керек, кері жағдайда оның мәні бірге тең болуы талап етіледі:

```
i = (i < n) ? i + 1 : 1;
```

Бұл бөлімде қарастырылмаған операциялар кейінірек сипатталады.

Өрнектер

Жоғарыда айтылып кеткендей, өрнектер операндтардан, операция таңбалары мен жақшалардан тұрады да, белгілі бір типтегі кез келген мәнді есептеу үшін қолданылады. Өз кезегінде, әрбір операнд өрнек немесе оның дербес түрлерінің бірі – тұрақты немесе айнымалы болып табылады.

Өрнектер мысалдары:

```
(a + 0.12)/6
x && y || !z
(t * sin(x)-1.05e4)/((2 * k + 2) * (2 * k + 3))
```

Операциялар өз *приоритеттеріне* сәйкес орындалады. Операциялардың орындалу ретін өзгерту үшін жай жақшалар қолданылады. Егер бір өрнекте басымдықтары бірдей бірнеше операция қатар жазылған болса, онда унарлы операциялар, шартты операция және меншіктеу операциялары *оңнан солға* қарай, ал қалғандары *солдан оңға* қарай орындалады. Мысалы, **a = b = c** өрнегі **a = (b = c)** екенін білдіреді, ал **a + b + c** өрнегі **(a + b) + c** болатынын көрсетеді. Өрнектер құрамындағы ішкі өрнектерді есептеу реттілігі анықталмаған: мысалы, **(sin(x + 2) + cos(y + 1))** өрнегінде синус косинустан ерте есептеледі деп, ал **x + 2** өрнегі **y + 1** өрнегіне қарағанда ерте есептеледі деп айтуға болмайды.

Өрнекті есептеу нәтижесі өз мәні және типімен сипатталады. Мысалы, егер **a** және **b** – бүтін типті айнымалылар болып, олар төмендегідей түрде сипатталса:

```
int a = 2, b = 5;
```

онда **a + b** өрнегінің мәні 7 және типі **int** болады, ал **a = b** өрнегі **a** айнымалысына орналастырылған шамаға тең (мұнда 5) мәнді және осы айнымалның типіне сәйкес келетін типті қабылдайды. Осылайша, C++ тілінде **a = b = c** түріндегі өрнекте алдымен **b = c** өрнегі есептеледі, содан кейін оның мәні **a** айнымалысына меншіктелетін оң жақта орналасқан операнд болып саналады.

Өрнек құрамында әртүрлі типтегі операндтар болуы мүмкін. Егер операндтардың типтері бірдей болса, онда операция нәтижесі де сол типте болады. Егер операндтардың типтері әртүрлі болса, онда есептеу алдында белгілі бір ережелер бойынша типтерді түрлендіру атқарылады, яғни шамалар дәлдігін қамтамасыз ету үшін ендері қысқа типтегі мәндер ендері ұзынырақ типтерге түрлендіріледі.

Түрлендірулер екі түрде орындалады:

- шамалардың ішкі бейнесін өзгерту (дәлдікті жоғалту арқылы немесе дәлдікті жоғалтпайтын);
- шамалардың тек ішкі бейнесінің интерпретациялануын өзгерту.

Бірінші түрге, мысалы, бүтін санды нақты санға (дәлдікті жоғалтпайтын) және керісінше (мүмкін, дәлдікті жоғалта отырып) түрлендіру, ал екінші түрге – таңбалы бүтін санды таңбасыз бүтін санға түрлендіру жатады.

Егер **int** типі **char, signed char, unsigned char, short int** және **unsigned short int** типіндегі барлық мәндерді бейнелей алатындай болса, онда шамалар бүтін типке, ал кері жағдайда олар **unsigned int** типіне түрлендіріледі.

Осыдан кейін операндтар солардың ішіндегі ең ені ұзын типке түрлендіріледі де, ол нәтиже типі ретінде қолданылады. Түрлендіру ережелері 3-қосымшада келтірілген.

Программалаушы типтердің түрлендірілуін өзі анықтай алатын айқын түрде де бере алады (бұл туралы «Типтерді түрлендіру» бөлімінде, 249 б. айтылады).

Сонымен, біз программалау тілін құрастыратын конструкциялардың ең төменгі деңгейін егжей-тегжейлі түрде ұзақ қарастырып шықтық, енді олардан маңыздырақ құрылымдарды – алдымен жеке операторларды, сонан соң программаларды құруды бастау керек. Тіпті ең қарапайым программаның өзі де белгілі бір ережелерге сәйкес құрылуы тиіс, сонда ғана ол сенімді, нәтижелі әрі көрікті болады. Осы ережелерді келесі бөлімде қарастырамыз.

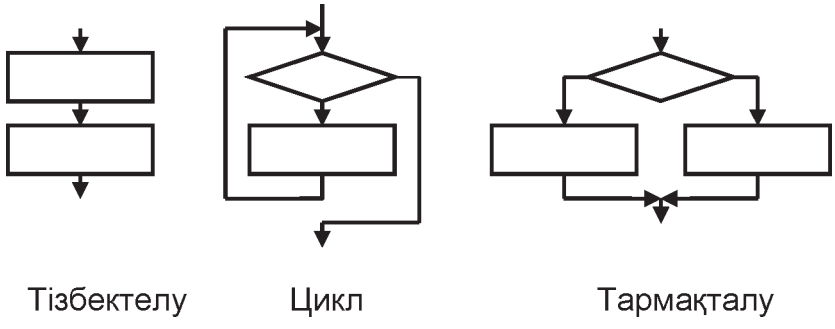
Қарапайым программаларды құру практикумның [11] 13-бетіндегі бірінші семинарда толығырақ қарастырылады.

Құрылымдық программалаудың базалық конструкциялары

Программалау теориясында кез келген күрделі есептерді шешуге арналған программаны тізбектелу, тармақталу және цикл деп аталатын үш құрылым көмегімен құруға болатыны көрсетілген. Бұл қағиданы Бойм және Якопини 1966 жылы анықтаған, олар кез келген программаны тек осы құрылымдар мен солардың комбинацияларынан тұратын эквивалентті программаға түрлендіруге болатынын дәлелдеген.

Тізбектелу, тармақталу және цикл құрылымдық программалаудың *базалық конструкциялары* деп аталады. Тізбектелу, яғни *сызықтық құрылым* екі немесе одан да көп операторлардың (қарапайым немесе құрама) тізбектеле орындалуын бейнелейтін конструкция болып табылады. *Тармақталу құрылымы кез келген* бір шарттың орындалуына байланысты бірнеше операторлардың *ішіндегі біреуінің орындалуын білдіреді. Цикл* операторлардың бірнеше рет қайталанып орындалуын көрсетеді (1.3-сурет). Бұл базалық конструкциялардың ерекшелігі – олардың кез келгенінің бір кіріс және бір

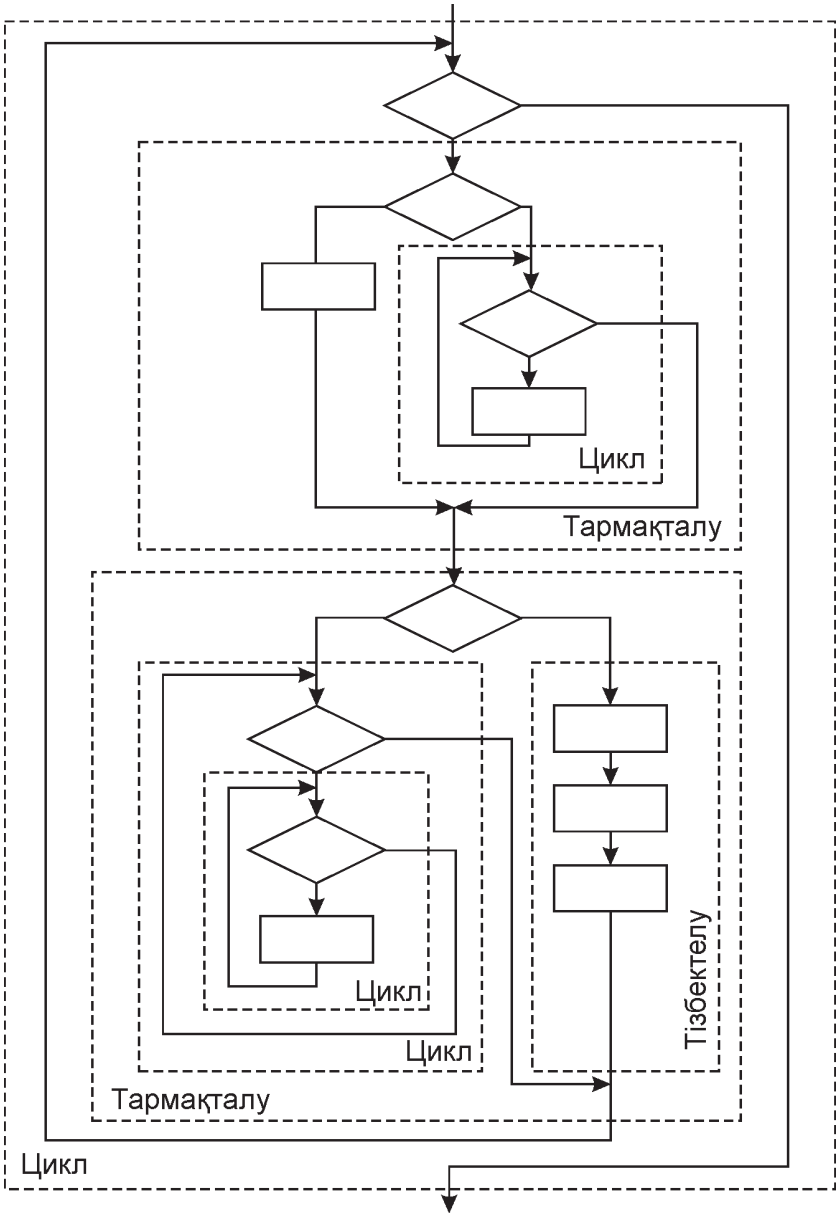
шығыс нүктесінің болуында, сондықтан конструкциялар бір-бірімен кез келген тәртіпте байланысуы мүмкін, мысалы цикл ішінде екі тармақталудан құралатын сызықтық құрылым болып, ал тармақталудың әрқайсысы басқа бір циклдерден тұратын болуы мүмкін (1.4-сурет).



1.3-сурет. Құрылымдық программалаудың базалық конструкциялары

Базалық конструкцияларды қолданудың мақсаты – қарапайым құрылымдағы программа алу. Мұндай программаны оқу (программаларды жазуға қарағанда жиірек оқуға тура келеді), түзету және қажет болғанда, оған өзгерістер енгізу жеңіл орындалады. Құрылымдық программалауды көбінесе «goto бұйрығынсыз программалау» деп атайды, бұл шындыққа жанасады: программаның кез келген бір нүктелерінде – басқаруды басқа жаққа беру операторын жиі қолдану – оның жұмыс істеу логикасын бақылауды қиындатады. Басқа жағынан алғанда, ешбір қағиданы абсолютті түрде нәтижелі деп санауға болмайды, **`goto`** операторын қолдану тиімді болып, программа құрылымын қарапайым ететін жағдайлар да туындап жатады. Олар туралы «goto операторы» бөлімінде айтылады (46 б.).

Жоғарғы деңгейдегі тілдердің көпшілігінде базалық конструкциялардың бірнеше нұсқалары қолданылады; C++ тілінде циклдің үш түрі, тармақталудың екі түрі (екі тармаққа және одан артық тармақтарға) бар. Олар программалау ыңғайлы болуы үшін енгізілген және әртүрлі жағдайларда олардың керектісін таңдай білу керек. Қарапайым программалардың өзін жазу кезінде де ең негізгі есте сақтау құралы – кез келген программа алдын ала анықталған құрылымдағы блоктардың бір-бірімен байланыстырылған айқын тізбегінен тұруы тиіс. «Айқын ойлайтын адам ойын да анық жеткізеді» деген ұғым арқылы құрылған «саналық түйсік» стилінде жазылған программалардың практикада онша өміршең бола алмайтыны және көркемдік жағынан да ұтылатыны дәлелденген.



1.4-сурет. Базалық конструкциялардың қабаттаса орналасуы

Енді құрылымдық программалаудың базалық конструкцияларын жүзеге асыратын тіл операторларын қарастырайық.

«Өрнек» операторы

Нүктелі үтірмен аяқталатын кез келген өрнек оператор ретінде қарастырылып, оны орындау белгілі бір есептеу ісін жүзеге асырады. Өрнектің

дербес бір түрі *бос оператор* (*;*) болып табылады (ол синтаксис бойынша оператор керек болғанымен, ал мән-мағынасы бойынша ол қажет етілмейтін жағдайда қолданылады). Мысалдар:

```
i++;           // инкремент операциясы орындалады
a *= b + c;    // көбейту арқылы меншіктеу орындалады
fun(i, k);     // функцияны шақыру орындалады
```

Тармақталу операторлары

If шартты операторы

If шартты операторы есептеу процесін екі бағытта тармақтау үшін қолданылады. Операторлың құрылымдық сызбасы 1.5-суретте көрсетілген. Операторлың жазылу форматы:

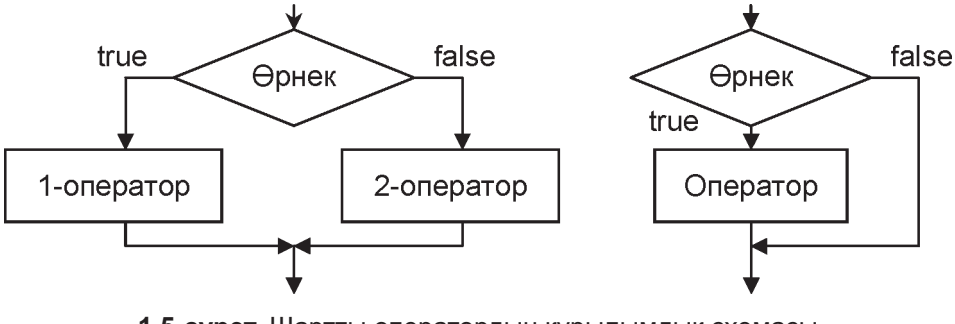
```
if ( өрнек ) 1-оператор; [else 2-оператор;]
```

Алдымен арифметикалық типте немесе нұсқауыш типте болатын өрнек есептеледі. Егер ол нөлге тең болмаса (мәні **true** болса), бірінші оператор, кері жағдайда – екінші оператор орындалады. Осыдан кейін шартты операторлардан кейін орналасқан келесі операторлар атқарылады.

Тармақтардың **else** түйінді сөзінен басталатын бөлігі болмауы да мүмкін. Егер кез келген бір тармақта бірнеше оператор орындалуы керек болса, оларды блок ішіне енгізу қажет, әйтпесе компилятор бұл тармақтың қай жерде аяқталатынын біле алмайды. Блок ішінде кез келген операторлар орналасуы мүмкін, мысалы, сипаттамалар немесе басқа шартты операторлар (бірақ ол тек сипаттамалардан ғана тұра алмайды). Блок ішінде сипатталған айнымалы блоктан тыс жұмыс істемейтінін ескеру қажет.

Мысалдар:

```
if (a<0) b = 1;                                          // 1
if (a<b && (a>d || a==0)) b++; else {b *= a; a = 0;} // 2
if (a<b) {if (a<c) m = a; else m = c;}
    else {if (b<c) m = b; else m = c;}                  // 3
if (a++) b++;                                            // 4
if (b>a) max = b; else max = a;                          // 5
```



1.5-сурет. Шартты операторлың құрылымдық схемасы

1-мысалда **else** тармағы жоқ. Мұндай конструкция "операторды аттап өту" деп аталады, өйткені шарттың орындалуына байланысты меншіктеу операторы орындалады немесе оны аттап өту ісі жүзеге асырылады.

Егер бірнеше шартты қатар тексеру қажет болса, олар логикалық операциялар белгілерімен біріктіріледі. Мысалы, *2-ші мысалда* **a<b** шарты және жақша ішіндегі шарттардың бірі қатарласа орындалатын болған жағдайда ғана өрнек толығынан ақиқат болады. Егер ішкі жақшаларды алып тастасақ, алдымен логикалық ЖӘНЕ, сонан соң барып НЕМЕСЕ орындалады.

3-мысалдағы оператор үш айнымалының ішінен ең кіші мәнді іздеп табады. Мұнда компилятор **else** бөлігін оған ең жақын **if** операторымен байланыстыратындықтан, жүйелі жақшаның болуы міндетті емес.

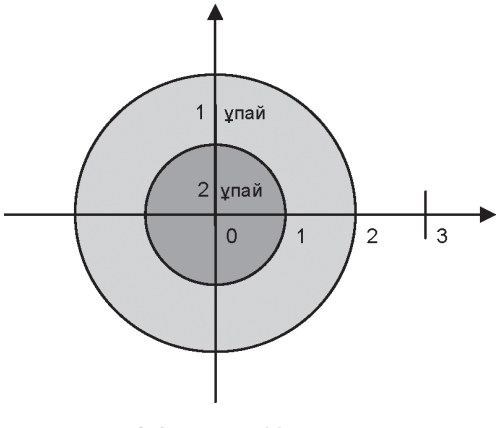
4-мысал **if** операторында шартты өрнектер ретінде көбінесе қатынас операциялары қолданылатынына қарамастан, бұлай істеу міндетті емес екендігін білдіреді.

5-мысалдағы операторларға ұқсас конструкцияларды шартты операция түрінде жазған ыңғайлы және көрнекірек болады, мұндағы мысал үшін:

```
max = (b > a) ? b : a;
```

Мысалы, 1.6-шы суретте, көрсетілген нысанаға жебе атылған кезде алынған ұпайлар санын анықтау керек.

```
#include <iostream.h>
int main(){
   float x, y; int kol;
   cout << "Ату координаталарын енгізіңіз\n";
   cin >> x >> y;
   if (x*x + y*y < 1) kol = 2;
   else if (x*x + y*y < 4) kol = 1;
   else kol = 0;
   cout << "\n Ұпайлар саны: " << kol;
   return 0;
}
```



1.6-сурет. Нысана

НАЗАР АУДАРЫҢЫЗ

Шартты операторларды жазуда кең таралған қателіктердің бірі – өрнектердегі теңдікке тексеру (==) орнына (=) қарапайым меншіктеуді қолдану, мысалы **if (a=1) b=0;** Мұнда синтаксистік қате жоқ, өйткені меншіктеу операциясы нөлге тең/тең емес болып бағаланатын нәтиже қалыптастырады. Бұл мысалда **b** айнымалысына меншіктеу **a** айнымалысының мәнінен тәуелсіз түрде орындалады. Сондықтан айнымалының тұрақтыға тең екендігін тексеру өрнектерінде тұрақтыны салыстыру операциясының сол жағына жазған жөн: **if (1==a) b=0;**. Екінші қателік – *өрнек мәнінің берілген диапазонға тиісті екендігін тексеру* кезінде жиі кездеседі. Мысалы, **0<x<1** шартын тексеру үшін, оны шартты операторда дәл осындай түрде тікелей жазуға болмайды, себебі алдымен **0<x** жеке салыстырылады да, оның нәтижесі (**int** типіне түрлендірілген **true** немесе **false**) 1мен салыстырылады. Мұны дұрыс жазу жолы: **if (0<x && x<1)...**

Айнымалылар типі олардың атқаратын қызметіне қарай таңдалады. Жоғарыдағы жебе ату координаталарын бүтін шамалармен бейнелеуге болмайды, өйткені бұл нәтиже дәлдігінің жоғалуына алып келеді, ал ұпайлар санын нақты шама ретінде сипаттау спортта қолданылмайды. Осындай қарапайым программаның өзін аралық (қосымша) айнымалы енгізу арқылы және шартты екі қабаттасқан **if** операторының орнына тізбектелген операторлар түрінде жазу арқылы одан әрі оңайлатуға болады (бірінші нұсқада **kol** айнымалысының мәні бір рет қана меншіктелсе, екінші нұсқада шарттың орындалуына байланысты ол бір реттен үш ретке дейін меншіктелуі мүмкін):

```
#include <iostream.h>
int main(){
    float x, y, temp; int kol;
    cout << "Нысана координаталарын енгізіңіз \n";
    cin >> x >> y;
    temp = x * x + y * y;
    kol = 0;
    if (temp < 4) kol = 1;
    if (temp < 1) kol = 2;
    cout << "\n Ұпайлар саны: " << kol;
    return 0;
 }
```

Егер кез келген бір айнымалы тек шартты оператор ішінде ғана қолданылатын болса, оны жақшалардың ішінде хабарлау ұсынылады:

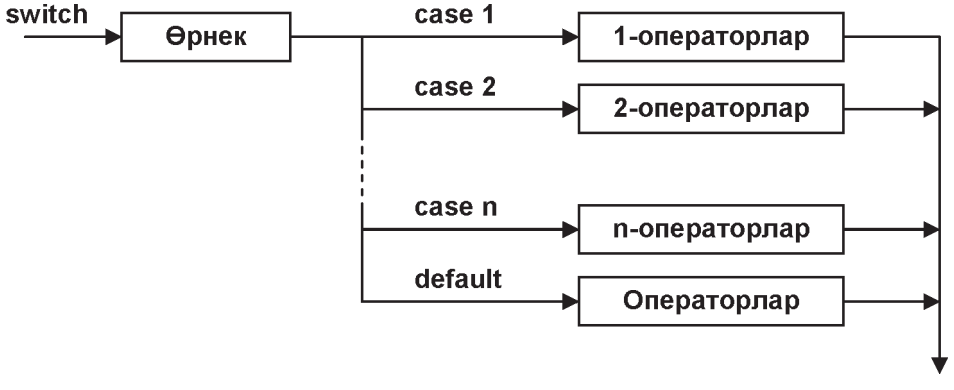
```
if (int i = fun(t)) a -= i; else a += i;
```

Айнымалыны қажет болғанда ғана, яғни оған мән беру кезінде жариялау – жақсы программалау стилінің белгісі болып табылады және айнымалыны оны инициалдауға дейін кездейсоқ қолдану қаупінен қорғайды. **if** операторының ішінде бір ғана айнымалыны жариялауға болады. Оның көріну аймағы жариялау нүктесінен басталады да, оператордың екі тармағынан соң аяқталады.

switch операторы

switch (ауыстырғыш) операторы есептеу процесін бірнеше бағытта тармақтауға арналған. Операторлың құрылымдық схемасы 1.7-суретте көрсетілген. Операторлың жазылу форматы:

```
switch ( өрнек ) {
   case 1-тұрақтылық_өрнек: [1-операторлар_тізімі]
   case 2-тұрақтылық_өрнек: [2-операторлар_тізімі]
   ...
   case n-тұрақтылық_өрнек: [n-операторлар_тізімі]
   [default: операторлар]
}
```



1.7-сурет. switch операторының құрылымдық схемасы

Операторлың орындалуы **switch** сөзінен кейін тұрған өрнекті есептеуден басталады (ол бүтін сан түрінде болуы керек), сонан соң есептелген өрнек мәніне сәйкес келетін тұрақтылық_өрнекпен белгіленген жолдағы операторлар орындалады. Осы *операторлар соңында ауыстырғыштан шығатын оператор (break) айқын көрсетілмесе, қалған барлық тармақтар кезектесіп орындала береді.*

Ауыстырғыштан шығу **break** немесе **return** операторларының көмегімен орындалады. **break** операторы **switch, for, while** және **do** сияқты операторлардың ішкі блогынан сыртқа шығу ісін атқарады. Ал **return** операторы өзі анықталған функция тұлғасынан сыртқа шығуды орындайды.

Ауыстырғыш құрамындағы тұрақтылық_өрнектер мәндері әртүрлі болуы тиіс, бірақ олардың барлығы да бүтін сандық типте болуы керек. Бірнеше тұрақтылық өрнектер бір жолда қатарласа орналасуы мүмкін. Егер өрнек мәні тұрақтылар мәнінің ешқайсысына сәйкес келмесе, **default** сөзінен кейін орналасқан операторлар орындалады (олар жоқ болған жағдайда басқару **switch** тұлғасынан кейін орналасқан операторға беріледі).

Мысал (мұнда 4 арифметикалық амал орындайтын қарапайым калькулятор жұмысы жүзеге асырылады):

```
#include <iostream.h>
int main(){
   int a, b, res;
   char op;
   cout << "\nБірінші операндты енгізіңіз: "; cin >> a;
   cout << "\nОперация белгісін енгізіңіз: "; cin >> op;
   cout << "\nЕкінші операндты енгізіңіз: "; cin >> b;
   bool  f = true;
   switch (op){
      case '+': res = a + b; break;
      case '-': res = a - b; break;
      case '*': res = a * b; break;
      case '/': res = a / b; break;
      default: cout << "\nБелгісіз операция"; f = false;
   }
   if (f) cout << "\nНәтиже: " << res;
   return 0;
}
```

ЕСКЕРТУ

---

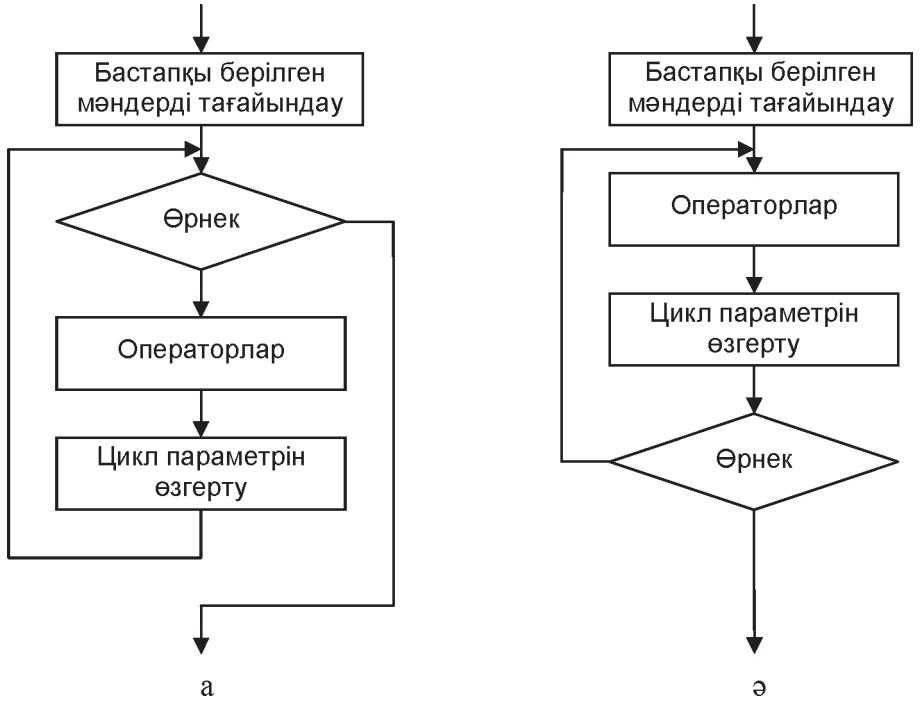
default сөзінде синтаксистік қате кеткен жағдайда ол туралы хабарлама шығарылмайды, өйткені онда компилятор бұл сөзді оператор алдындағы белгі ретінде қабылдайды.

Шарттарды тексеру әрекеттері көрсетілген басқа программалар мысалдары практикумның [11] 30 б. берілген.

Цикл операторлары

Цикл операторлары бірнеше рет қайталанып орындалатын есептеулерді ұйымдастыру үшін қолданылады. Кез келген цикл оның тұлғасынан, яғни бірнеше рет қайталанатын оператор тізбегінен, бастапқы берілген мәндерді тағайындаудан, цикл *параметрінің* өзгеруінен және цикл қайталануын жалғастыру шартын тексеруден тұрады (1.8-сурет).

Циклдің бір рет орындалуы итерация (қадам) деп аталады. Шарттың тексерілуі әрбір итерация сайын кейде цикл тұлғасына дейін (алғы шартты цикл), кейде цикл тұлғасынан кейін (соңғы шартты цикл) атқарылады. Олардың арасындағы айырмашылық: соңғы шартты цикл кем дегенде бір рет орындалады да, содан кейін барып оны қайталап орындау қажеттілігі тексеріледі. Ал алғы шартты циклде оны қайталау шартын тексеру цикл тұлғасына дейін жүргізілетіндіктен, оның бір рет те орындалмауы мүмкін.



1.8-сурет. Цикл операторларының құрылымдық сызбасы: а – шарты алдын ала тексерілетін цикл; ә – шарты соңынан тексерілетін цикл

Цикл тұлғасында өзгертілетін және оны қайталау шартын тексеруде қолданылатын айнымалылар цикл параметрлері деп аталады. Әрбір итерация сайын тұрақты қадаммен өзгертілетін бүтін сан типіндегі цикл параметрлері *цикл санауыштары* болып табылады.

Бастапқы берілгендер программаға айқын түрде қатыспауы да мүмкін, олардың атқаратын қызметі – циклге кіруден бұрын онда қолданылатын айнымалыларға мән беру болып саналады.

Егер циклді жалғастыру шарты орындалмаса, ол бірден аяқталады. Орындалып жатқан ағымдағы итерацияны да, циклдің өзін толығымен де мәжбүрлі түрде аяқтауға болады. Ол үшін **`break`**, **`continue`**, **`return`** және **`goto`** операторлары қолданылады («Басқаруды беру операторлары» бөлімін қара, 46 б.). Цикл сыртынан оның ішіне кіру мүмкіндігін пайдалану ұсынылмайды.

C++ тілінде программа құру ыңғайлы болуы үшін, онда үш түрлі цикл операторлары – **`while`**, **`do while`** және **`for`** қолданылады.

Алғы шартты цикл (while)

Шарты алдын ала тексерілетін алғы шартты цикл 1.8, а-суретте көрсетілген және ол төмендегідей түрде жазылады:

```
while (өрнек) оператор
```

Өрнек қарапайым немесе құрама операторомен берілген цикл тұлғасының қайталану шартын анықтайды. Оператордың орындалуы өрнекті есептеуден басталады. Егер ол ақиқат болса (**false** емес), цикл операторы орындалады. Ал егер алғашқы тексеру кезінде өрнек мәні **false** болса, цикл бір рет те орындалмайды. Өрнектің типі арифметикалық немесе соған келтірілетін нұсқада болуы тиіс, ол циклдің әрбір итерациясы алдында есептеледі.

Мысал (программа $y = x^2 + 1$ функциясының мәндері көрсетілген диапазонда кесте түрінде экранға шығарылады):

```
#include <stdio.h>
int main(){
   float Xn, Xk, Dx;
   printf("Аргументтің өзгеру аралығы мен қадамын
          енгізіңіз: ");
   scanf("%f%f%f", &Xn, &Xk, &Dx);
   printf("|  X  |  Y  |\n");     // кесте тақырыбы
   float X = Xn;            // цикл параметрін тағайындау
   while (X <= Xk){         // жалғастыру шартын тексеру
      printf("| %5.2f | %5.2f |\n", X, X*X + 1);
                            // цикл тұлғасы X += Dx;
      x+=Dx                 // параметрді өзгерту
   }
   return 0;
}
```

Мысал (программа бүтін оң санның барлық бөлгіштерін анықтайды):

```
#include <iostream.h>
int main(){
   int num;
   cout << "\nСан енгізіңіз: "; cin >> num;
   int half = num / 2;          // санның жартысы
   int div = 2;                 // бөлгіш болуы мүмкін сан
   while (div <= half){
      if (!(num % div)) cout << div << "\n";
      div++;
   }
   return 0;
}
```

Программалаудағы кең таралған тәсілдердің бірі – **while (true)** немесе **while (1)** тәрізді тақырыптар арқылы шексіз цикл ұйымдастыру, олардан шығу кез келген бір шарттың орындалуы бойынша цикл тұлғасынан мәжбүрлі түрде сыртқа шығу жолымен орындалады.

while түйінді сөзінен кейінгі дөңгелек жақшаларда айнымалының сипаттамасын[1] енгізуге болады.

Оның әрекет ету аймағы цикл іші болып табылады:

```
while(int x = 0){ ... /* x-тің әрекет ету аймағы */ }
```

Соңғы шартты цикл (do while)

Шарты соңынан тексерілетін цикл 1.8, ә-суретте көрсетілген құрылымдық сызбаны жүзеге асырады және ол мынадай түрде жазылады:

```
do оператор while өрнек;
```

Алдымен цикл тұлғасын құрайтын жай немесе құрама оператор орындалады да, содан кейін өрнек есептеледі. Егер өрнек ақиқат болса (**false** емес), цикл тұлғасы тағы қайталанып орындалады. Өрнек мәні **false** болғанда немесе цикл тұлғасында кез келген бір басқаруды беру операторы орындалған кезде, цикл жұмысы аяқталады. Өрнектің типі арифметикалық немесе соған келтірілетін түрде болуы керек.

Мысал (программа мәлімет енгізуді тексеру ісін жүзеге асырады):

```
#include <iostream.h>
int main(){
   char answer;
   do {
      cout << "\nПіл сатып ал! "; cin >> answer;
   } while (answer != 'y');
   return 0;
}
```

Мысал. Программа берілген **Eps** дәлдігімен нақты сан болып табылатын **x** аргументінің квадрат түбірін төмендегі итерациялық формула бойынша есептейді:

$$y_n = \frac{1}{2}(y_{n-1} + x / y_{n-1})$$

мұндағы y_{n-1} – түбірдің бұдан бұрынғы жуық мәні (есептеу басында кездейсоқ түрде таңдалады), y_n – түбірдің келесі жуық шамасы. Есептеу процесі түбірдің екі жуық мәндері айырмасының абсолюттік шамасы берілген дәлдіктен төмен болған кезде, тоқтатылады. Абсолюттік шаманы есептеу **fabs()** стандартты функциясы арқылы орындалады, оны пайдалану үшін **<math.h>** тақырыптық файлы қолданылады.

[1]Компиляторлардың ескі нұсқаларында стандарттың бұл талабы орындалмауы мүмкін.

```
#include <stdio.h>
#include <math.h>
int main(){
   double X, Eps;        // аргумент және дәлдік
   double Yp, Y = 1;     //алдыңғы және келесі жуықтаулар
   printf("Аргумент пен дәлдікті енгізіңіз: ");
   scanf("%lf%lf", &X, &Eps);
   do {
      Yp = Y;
      Y = (Yp + X/Yp)/2;
   } while (fabs(Y - Yp) >= Eps);
   printf("\n%lf санының түбірі = %lf", X, Y);
   return 0;
}
```

Параметрлі цикл (for)

Параметрлі циклдің жазылу форматы:

```
for (инициалдау; өрнек; модификациялар) оператор;
```

Инициалдау циклде қолданылатын шамаларды жариялау мен оларға бастапқы мәндерді тағайындау үшін пайдаланылады. Бұл бөлімде бірнеше операторларды («тізбектей орындау» операциясымен) бір-бірінен үтірлер арқылы ажырата отырып жазып қоюға болады, мысалы төмендегідей түрде:

```
for (int i = 0, j = 2;...
int k, m;
for (k = 1, m = 0;...
```

Циклдің инициалдау бөлімінде жарияланған айнымалылардың әрекет ету аймағы цикл іші ғана болып табылады. Инициалдау цикл орындалуының басында бір-ақ рет атқарылады.

Өрнек циклдің қайталану шартын анықтайды: егер оның **bool** типіне келтірілген нәтижесі **true** болатын болса, цикл орындалады. Бұл параметрлі цикл шарты алдын ала тексерілетін цикл ретінде жүзеге асырылған.

Модификациялар циклдің әрбір итерациясынан (қадамынан) кейін орындалады және цикл параметрлерін өзгерту үшін қолданылады. Модификациялар бөлімінде де үтір арқылы бөле отырып, бірнеше операторларды қатар жазуға болады. Қарапайым немесе құрама *оператор* цикл тұлғасын құрайды. **for** операторының кез келген бөлігін алып тастауға болады (бірақ нүктелі үтірлерді өз орындарында қалдыру керек!).

Мысал (1 мен 100 аралығындағы бүтін сандардың қосындысын есептейтін оператор):

```
for (int i = 1, s = 0; i <= 100; i++) s += i;
```

Мысал (программа $y = x^2 + 1$ функциясының енгізілген диапазондағы мәндері кестесін экранға шығарады):

```
#include <stdio.h>      // C тілі стиліндегі программа
int main(){
   float Xn, Xk, Dx, X;
   printf("Аргументтің өзгеру диапазоны мен қадамын
           енгізіңіз: ");
   scanf("%f%f%f", &Xn, &Xk, &Dx);
   printf("|  X  |  Y  |\n"):
   for (X = Xn; X <= Xk; X += Dx)
      printf("| %5.2f | %5.2f |\n", X, X*X + 1);
   return 0;
}
```

Мысал (программа бүтін оң санның барлық бөлгіштерін анықтайды):

```
#include <iostream.h>    // C++ тілі стилінде
int main(){
   int num, half, div;
   cout << "\nСан енгізіңіз: "; cin >> num;
   for (half = num / 2, div = 2; div <= half; div++)
      if (!(num % div)) cout << div << "\n";
   return 0;
}
```

Соңғы екі мысал алғы шартты циклдер мысалдарындағыдай әрекеттер атқарады, бірақ мұнда бұлар ықшам және көрнекі түрде жазылған: циклді басқаруға байланысты барлық іс-әрекеттер оның тақырыбында орналасқан.

Кез келген **while** циклі оған эквивалентті **for** цикліне және кері бағытта келесі сұлба бойынша келтірілуі мүмкін:

```
for(b1; b2; b3)
   оператор b1;
while (b2){
   оператор b3;}
```

Циклдерді программалауда жиі кездесетін қателер – цикл тұлғасында инициалданбаған айнымалыларды қолдану және циклден шығу шартының дұрыс жазылмауы.

Қателіктерге жол бермеу үшін төмендегі ережелерді бұлжытпай орындау ұсынылады:

- цикл тұлғасындағы меншіктеу операторларының оң жағында тұратын барлық айнымалыларға осыған дейін бастапқы мән берілгендігін (сонымен қатар, басқа операторлардың орындалу мүмкіндігін) тексеру;
- циклден шығу шартына енген айнымалылардың кем дегенде біреуі цикл ішінде өзгеретінін немесе өзгермейтінін тексеру;

- итерациялардың белгілі бір санына жеткенде циклден мәжбүрлі түрде шығуды қарастыру (келесі бөлімдегі мысалды қара);
- цикл тұлғасында бірден артық операторлар санын орындау қажет болса, оларды жүйелі жақшаға алып жазу керек екендігін ұмытпау қажет.

Цикл операторларын өзара алмастырып та орналастыруға болады, бірақ әрбір нақты жағдайда ең тиімді түрде жұмыс істейтін операторды таңдауды қамтамасыз ете алатын бірнеше *ұсыныстар* келтіруге болады, олар:

- **do while** операторы, әдетте, цикл кем дегенде бір рет орындалуы тиіс болатын жағдайларда қолданылады (мысалы, цикл ішінде мәліметтерді енгізу орындалатын кездерде);
- **for** операторы өзге жағдайлардың басым бөлігінде қолданылады (әсіресе, санауыштар арқылы ұйымдастырылатын циклдер құру үшін);
- **while** операторын итерациялар саны алдын ала белгісіз болып, циклдің нақты параметрлері әлі анықталмаған кездерде немесе параметрлер мәндерін өзгертуді цикл тұлғасының соңында жазған ыңғайсыз болатын жағдайларда қолданған дұрыс болып табылады.

Басқаруды беру операторлары

C++ тілінде программа операторларының бекітілген табиғи орындалу реттілігін өзгертетін төрт оператор бар:

- **goto** – шартсыз көшу операторы;
- **break** – циклден шығу операторы;
- **continue** – циклдің келесі итерациясына көшу операторы;
- **return –** функциядан негізгі программаға қайту оралу операторы.

goto операторы

Шартсыз көшу операторы – **goto** келесі форматта жазылады:

```
goto белгі;
```

Мұндайда осы программа немесе функция тұлғасында төмендегідей түрдегі бір конструкция міндетті түрде болуы керек:

```
белгі: оператор;
```

goto операторы басқаруды белгіленген операторға береді. *Белгі* – бұл қарапайым идентификатор, оның көріну аймағы өзі анықталған функция іші болып табылады.

Шартсыз көшу операторын қолдану төмендегі екі жағдайда тиімді:

- бірнеше қабаттасқан циклдерден немесе ауыстырғыштан тұратын программа ішінен одан кейінгі бөліктерге (төмен қарай) мәжбүрлі түрде шығу керек болған жағдайларда;

- функцияның бірнеше жерінен көрсетілген бір орынға көшу (мысалы, функциядан шығу алдында әрқашанда кез келген бір әрекеттер тізбегін орындау қажет болса).

Қалған жағдайларда кез келген алгоритмді жазу үшін тиімдірек болатын басқа құралдар бар, ал **goto** операторын пайдалану программа құрылымының күрделенуіне және оны түзетудің қиындауына алып келеді[1]. **goto** операторын қолдану құрылымдық және модульдік программалау қағидаларын бұзады, олар бойынша программаны құрамындағы барлық блоктардың бір ғана кіріс және бір ғана шығыс нүктелері болуы тиіс.

Кез келген жағдайларда басқаруды **if, switch** және цикл операторларының ішіне бермеген жөн. Айнымалыларды инициалдау орындалатын блоктардың ішіне және одан кейін орналасқан операторларға сырттан кіруге (көшуге) болмайды, себебі мұндайда керекті инициалдау ісі орындалмай қалады, мысалы:

```
int к; ...
goto belgi; ...
{  int a = 3, b = 4;
   k = a + b;
   belgi: int m = к + 1; ...
}
```

Программаның belgi орналасқан жолына сырттан ауысып келген кезде **m** айнымалысының мәні анықталмаған болып қалады.

break операторы

break операторы цикл тұлғасынан немесе **if**, **switch** операторларының ішінен программаның одан кейінгі бөлігіне бірден көшуді жүзеге асыру үшін қолданылады.

Мысал. Программа нақты **x** аргументі бар синус функциясының мәнін оны шексіз қатарға жіктеу арқылы берілген **eps** дәлдігімен есептейді.

$$\sin x = x - x^3/3! + x^5/5! - x^7/7! + \ldots$$

Қажетті дәлдікке **|Rn| < eps** болғанда қол жеткізуге болады, мұндағы **Rn** – қатардың қалдық мүшесі, оны осы қатардың біртіндеп қосылып отыратын кезекті келесі мүшесімен алмастыруға болады.

```
#include <iostream.h>
#include <math.h>
int main(){
   const int MaxIter = 500;       // итерация санының
                                  // шектеуіші
```

[1]Көрсетілген жағдайларда да код бөліктерінде локалды объектілер құрылмаған кезде ғана **goto** операторын қолданған дұрыс болады. Кері жағдайда, конструктордың қатысуынсыз деструкторды қолдануға болады, бұл программада қателіктердің туындауына алып келеді.

```
    double x, eps;
    cout << "\nАргумент пен дәлдікті енгізіңіз: ";
    cin >> x >> eps;
    bool ok = true;          // есептеу сәттілігі көрсеткіші
    double y = x, ch = x;    // қатар қосындысы және
                             // оның алғашқы мүшесі
    for (int n = 0; fabs(ch) > eps; n++) {
       ch *= - x * x /(2 * n + 2)/(2 * n + 3);
                             // қатардың кезекті мүшесі
       y += ch;
       if (n > MaxIter){
           ok = false; break; }
    }
    if (ok) cout << "\nФункцияның мәні: " << y;
    else cout << "\nҚатар жинақталмайды!";
    return 0;
}
```

continue операторы

Циклдің келесі итерациясына көшу операторы – **continue** цикл тұлғасындағы (оның соңына дейінгі) қалған барлық операторларды аттап өтіп, басқаруды циклдің келесі итерациясының басына береді.

return операторы

Функциядан қайтып оралу операторы – **return** функцияның орындалуын аяқтап, басқаруды оның шақырылу нүктесіне береді. Оператордың жазылуы:

```
return [ өрнек ];
```

Өрнек скалярлық типте болуы керек. Егер функцияның қайтаратын мәнінің типі **void** ретінде сипатталса, өрнек болмауы тиіс.

Нұсқауыштар мен жиымдар

Нұсқауыштар

Компилятор айнымалыны анықтау операторын өңдеген кезде, мысалы, **int i = 10;**, ол (**int**) типіне сәйкес жады бөледі және оны көрсетілген мәнмен (**10**) инициалдайды. Программада айнымалының атауы (**i**) көрсетілген барлық сәттерде компилятор айнымалы мәні сақталған жады аймағының адресін пайдаланады. Программалаушы жады аймақтарының адрестерін сақтау үшін өз айнымалыларын анықтап қолдана алады. Мұндай айнымалылар *нұсқауыштар* деп аталады.

Сонымен, *нұсқауыштар жады аймақтарының адрестерін сақтауға арналған* айнымалылар болып табылады. C++ тілінде нұсқауыштардың үш түрі бар: объектіге, функцияға және **void** типіне арналған нұсқауыштар, оларды бір-бірінен қасиеттеріне және орындай алатын операциялар жиынына қарай отырып ажыратуға болады. Нұсқауыш өз алдына дербес тип емес, ол әрқашан қандай да бір нақты типпен байланысады.

Функцияға нұсқауыш код сегментінде функцияның атқарылатын коды орналасқан адресті, яғни функцияны шақырған кезде басқару берілетін адресті есте сақтайды. Функцияға нұсқауыштар функцияны жанамалы түрде шақыру үшін (функция аты арқылы емес, оның адресін сақтайтын айнымалыны пайдалану арқылы), сонымен қатар берілген функцияның атын басқа функцияға параметр ретінде беру үшін қолданылады. Функция нұсқауышы «берілген типтегі аргументтерден тұратын және берілген типтегі мәнді қайтаратын функция нұсқауышы» типін қабылдайды:

```
типі (*аты) (аргументтер_типтерінің_тізімі);
```

Мысалы,

```
int (*fun) (double, double);
```

түріндегі жариялау **int** типті мән қайтаратын және **double** типті екі аргументі бар **fun** атты функцияға нұсқауышты көрсетеді.

Объектіге нұсқауышта белгілі бір типтегі (негізгі немесе құрама) мәліметтер сақталатын жады аймағының адресі сақталады. Объектіге нұсқауышты (бұдан әрі жай ғана нұсқауыш деп атаймыз) қарапайым түрде жариялаудың форматы төмендегідей болады:

```
типі *аты;
```

мұндағы **типі** ретінде сілтеме (56 б. қараңыз) мен биттік өрістен (70 б. қараңыз) басқа кез келген мәліметтер типі берілуі мүмкін, сонымен қатар бұл сәтке дейін тип тек қана жарияланып, әлі анықталмаған болуы мүмкін (демек, мысалы, құрылымда дәл осы типтегі құрылымға нұсқауыш кездесуі мүмкін).

Жұлдызша белгісі тікелей нұсқауыш атына қатысты болып саналады, сондықтан бірнеше нұсқауышты жариялау үшін олардың әрқайсысының аты алдына жеке-жеке жұлдызша қою керек. Мысалы:

```
int *a, b, *c;
```

операторында **a** және **c** атты бүтін айнымалыларға екі нұсқауыш және жай **b** бүтін айнымалысы сипатталған.

Нұсқауыштың мөлшері (ені) жады үлгісіне байланысты. Нұсқауышқа нұсқауышты да, т.б. анықтауға болады.

void типіне нұсқауыш адресі сақталуы тиіс объектінің нақты типі анықталмаған жағдайда (мысалы, бір айнымалы арқылы әртүрлі уақыт аралықтарында әртүрлі типтегі объектілердің адрестерін сақтау қажет болғанда) қолданылады.

void типіне нұсқауышқа кез келген типтегі нұсқауыштың мәнін меншіктеуге және оны кез келген басқа нұсқауыштармен салыстыруға болады, алайда ол сілтеме жасап тұрған жады аймағымен белгілі бір әрекеттер орындау алдында нұсқауышты нақты бір типке айқын түрде түрлендіріп алу қажет (249 б. қараңыз).

Нұсқауыш тұрақты немесе айнымалы болуы мүмкін, сонымен қатар тұрақтыға немесе айнымалыға сілтеме жасауы мүмкін. Мысалдар қарастырайық:

```
int i;                  // бүтін айнымалы
const int ci = 1;       // бүтін тұрақты
int * pi;               // бүтін айнымалыға нұсқауыш
const int * pci;        // бүтін тұрақтыға нұсқауыш
int * const cp = &i;  //бүтін айнымалыға нұсқауыш-тұрақты
const int * const cpc = &ci;  /* бүтін тұрақтыға
                                 нұсқауыш-тұрақты */
```

Мысалдардан байқағанымыздай, нұсқауыштың аты мен жұлдызшаның арасында орналасқан **const** модификаторы нұсқауыштың өзіне қатысты болады және оның өзгеруіне мүмкіндік бермейді, ал жұлдызшаның сол жағындағы **const** өзі нұсқап тұрған мәннің тұрақтылығын көрсетеді. Нұсқауыштарды инициалдау үшін **&** – адресті алу операциясы қолданылған.

Нұсқауыш типті шамалар өздерінің әрекет ету аймағын, көріну аймағын және өмірлік кезеңін анықтаудың жалпы ережелеріне бағынады.

Нұсқауыштарды инициалдау

Нұсқауыштарды көбінесе динамикалық жадымен жұмыс істеу кезінде қолданады, оларды кейбір білгірлер үйінді (ағылшын тіліндегі *heap* сөзінің аудармасы) деп атап жүр. Бұл программаның орындалуы кезінде қажеттілігіне қарай орын бөлуге болатын бос жады аймағы. *Динамикалық айнымалылар* деп аталатын динамикалық жадының бөлінген аймақтарын пайдалану тек нұсқауыштар арқылы орындалады. Динамикалық айнымалылардың өмірлік кезеңі оларды құру нүктесінен программа соңына дейін немесе сол бөлінген жадыны айқын түрде босатып алуға дейін созылады. C++ тілінде динамикалық жадымен жұмыс істеудің екі тәсілі бар. Бірінші тәсілде – C тілінен келген **malloc** функциялары мүмкіндіктері пайдаланылса, ал екіншісінде – **new** және **delete** операциялары қолданылады.

Нұсқауышты анықтау кезінде оны инициалдауға, яғни оған бастапқы мән беруге тырысу керек. Инициалданбаған нұсқауыштарды байқамай жоспарсыз пайдалану – программалардағы кең таралған қателіктер көзі. Инициалдау кезінде берілетін мән (инициализатор) нұсқауыштың атынан кейін дөңгелек жақшалар ішінде немесе теңдік белгісінен кейін жазылады.

Нұсқауышты инициалдаудың төмендегідей әдістері бар:

1. Нұсқауышқа объектінің адресін меншіктеу, ол келесідей жолдармен орындалады:

□ адресті алу операциясының көмегімен:

```
int a = 5;          // бүтін айнымалы
int* p = &a;        /* нұсқауышқа a шамасының
                       адресі жазылады */
int* p (&a);        /* жоғарыдағы операция
                       орындалуының басқа тәсілі */
```

□ басқа инициалданған нұсқауыш мәнінің көмегімен:

```
int* r = p;
```

□ адрес ретінде қолданылатын жиымның немесе функцияның аты арқылы («Жиымдар» және «Функциялардың атауларын параметр ретінде беру» бөлімдерін қараңыз, 57 б., 82 б.):

```
int b[10];      // жиым
int* t = b;     // жиым басының адресін меншіктеу
…
void f(int a){ /*... */ }     // функцияны анықтау
void (*pf)(int);              // функцияға нұсқауыш
pf = f;         // функция адресін меншіктеу
```

2. *Нұсқауышқа жады аймағының адресін айқын түрде меншіктеу:*

```
char* vp = (char *)0xB8 000 000;
```

Мұндағы **0xB8 000 000** – он алтылық түрдегі тұрақты, **(char *)** – типті түрлендіру операциясы; тұрақты «**char**-ға нұсқауыш» типіне түрлендіріледі.

3. *Бос мәнді меншіктеу:*

```
int* suxx = NULL;
int* rulez = 0;
```

Алғашқы жолда **NULL** тұрақтысы қолданылады, ол C тілінің кейбір тақырыптық файлдарында нөлге тең нұсқауыш ретінде анықталған. Жай ғана 0 түріндегі белгілеуді қолдану ұсынылады, себебі **int** типіндегі бұл мән стандартты тәсілдер көмегімен мәтінге (контекстке) сәйкес дұрыс түрлендіріледі. Нөлдік адресі бар объектілердің болмауына кепілдік берілетіндіктен, нұсқауыштың нақты объектіге сілтеме жасау шартын тексеру үшін бос нұсқауышты қолдануға болады.

4. Динамикалық жады аймағын бөліп беру және нұсқауышқа оның адресін меншіктеу:

□ **new** операциясының көмегімен:

```
int* n = new int;                    // 1
```

```
int* m = new int (10);                    // 2
int* q = new int [10];                    // 3
```

□ **malloc**[1] функциясының көмегімен:

```
int* u = (int *)malloc(sizeof(int));    // 4
```

Жоғарыдағы мысалдың **1** операторындағы **new** операциясы **int** типіндегі шаманы орналастыруға жеткілікті болатын динамикалық жады аймағын бөліп, осы аймақтың бастапқы адресін **n** айнымалысында сақтауды орындайды. Ал **n** айнымалысының өзіне жады бөлу (нұсқауышты орналастыруға жеткілікті көлемде орын беру) программаны компиляциядан өткізу кезеңінде орындалады.

2 операторында (жоғарыда аталған әрекеттерден басқа) қосымша бөлінген динамикалық жадыны 10 мәнімен инициалдау жүзеге асырылады.

3 операторындағы **new** операциясы **int** типті 10 санға (10 элементтен тұратын жиым) арналған жады бөлу ісін орындайды және осы жады аймағының бастапқы адресін жиымның атауы ретінде қарастыруға болатын **q** айнымалысына жазып қояды. Жиымның атын қолдану арқылы оның кез келген элементін пайдалануға болады. Жиымдар туралы 57 б. қараңыз.

Егер жады бөлу әрекетін орындау мүмкін болмаса, онда стандарт бойынша **bad_alloc** ерекше операциясы туындауы тиіс (олар туралы «Аластамаларды өңдеу» бөлімінде – 240 б., ал жады бөлу кезіндегі қателіктер туралы 412 б. мәліметтер берілген).

4 операторда дәл **1** операторындағы сияқты әрекеттер орындалады, бірақ олар C кітапханасынан мұраланған **malloc** жады бөлу функциясының көмегімен атқарылады. Функциға тек бір параметр – бөлінетін жады көлемі байтпен беріледі. **(int*)** конструкциясы нұсқауыштың функциядан қайтарылатын типін қажетті типке түрлендіру үшін қолданылады (типтерді айқын түрде түрлендіру туралы 249 б. қараңыз). Егер жады бөлу мүмкін болмаса, онда функция 0 мәнін қайтарады.

malloc функциясына қарағанда, **new** операциясын қолданудың артықшылықтары бар, әсіресе, ол объектілермен жұмыс істегенде ыңғайлы болып табылады.

Егер жады **new** операциясының көмегімен бөлінген болса, оны **delete** арқылы, ал **malloc** функциясымен бөлінген жадыны – **free** функциясының көмегімен босату керек. Мұндайда нұсқауыш-айнымалы сақталады және оны қайталап инициалдауға болады. Жоғарыда көрсетілген динамикалық айнымалылар келесідей түрде жойылады:

```
delete n; delete m; delete [] q; free (u);
```

Егер жады **new[]** көмегімен бөлінген болса, оны босату үшін **delete[]** функциясын қолдану керек. Мұндайда жиымтың өлшемі көрсетілмейді. Егер

[1]malloc функциясын қолдану үшін программаға <malloc.h> тақырыптық файлын қосу керек.

тік жақша қойылмаса, қате туралы хабарлама берілмейді, бірақ жиымның тек алғашқы элементі бос болып белгіленеді де, қалғандарын кейінгі операцияларда пайдалануға болмайды. Мұндай қол жеткізуге болмайтын жады ұяшықтары «қоқыс» деп аталады.

НАЗАР АУДАРЫҢЫЗ

Егер нұсқауыш-айнымалы өзінің әрекет ету аймағынан шығып кетсе, оған бөлінген жады аймағы босатылады, яғни нұсқауыш сілтеме жасап тұрған динамикалық айнымалыға қол жеткізу мүмкін болмай қалады. Мұндайда динамикалық айнымалының өзіне бөлінген жады босатылмайды. «Қоқыстың» пайда болуының басқа бір жағдайы – инициалданған нұсқауышқа басқа нұсқауыштың мәнін меншіктеу. Мұндай кездерде оның бастапқы мәні мүлдем жойылады.

Жұлдызшалар, жай және тік жақшалар комбинациясы арқылы құрама типтерді және құрама типтерге нұсқауыштарды сипаттауға болады, мысалы,

```
int *(*p[10])();
```

операторында **int** типіне нұсқауыштарды қайтаратын параметрсіз функцияларға деген 10 нұсқауыштан тұратын жиым жарияланған.

Келісім бойынша тік және жай жақшалардың басымдықтары (басымдылықтары) бірдей, бірақ олар жұлдызшадан гөрі басымырақ болып табылады да, солдан оңға қарай қарастырылады. Қарастыру реттілігін өзгерту үшін қосымша жай жақшалар қолданылады.

Күрделі сипаттамаларды қарастыру кезінде «іштен сыртқа қарай» ережесін ұстану керек, оның әрекеттері:

1) егер атаудың оң жағында тік жақшалар бар болса, онда – бұл жиым, ал жай жақшалар болған жағдайда – бұл функция;

2) егер атаудың сол жағында жұлдызша бар болса, бұл бұрынырақ сипатталған конструкцияға нұсқауыш;

3) егер оң жақта жабылған жай жақша кездессе, онда жақша ішінде жоғарыда көрсетілген ережелерді пайдаланып алып, содан соң сыртқы өрнектерді қарастыру керек;

4) ең соңғы кезекте тип спецификаторы қарастырылады.

Жоғарыда берілген сипаттама үшін қарастырылатын приоритеттер реттілігі цифрлер көмегімен көрсетілгенде, төмендегідей болады:

```
int *(*p[10])();
5  4 2  1    3  // сипаттаманы қарастыру реттілігі
```

Нұсқауыштармен орындалатын операциялар

Нұсқауыштармен келесі операцияларды орындауға болады: адрессіздендіру немесе объектіні жанамалы түрде пайдалану **(*)**, меншіктеу, тұрақтыны қосу, азайту, инкремент **(++)**, декремент **(--)**, салыстыру, типтерге келтіру.

Нұсқауыштармен жұмыс істеу кезінде адресті алу **(&)** операциясы жиі қолданылады.

Адрессіздендіру операциясы, нұсқауышта адресі сақталған шаманы пайдалану үшін қолданылады. Бұл операцияны шаманың мәнін алу үшін де және оны өзгерту үшін (егер ол тұрақты ретінде жарияланбаса) де қолдануға болады:

```
char a;                  // char типті айнымалы
char *p = new char;      /* нұсқауышқа және char типті
                         динамикалық айнымалыға жады бөлу */
*p = 'Ю'; a = *p;        // екі айнымалыға да мән беру
```

Мысалдан көріп отырғанымыздай, ***нұсқауыш_аты*** конструкциясын меншіктеу операторының сол жағында қолдануға болады, себебі ол L-мән (27 б.) болып табылады, яғни ол жады аймағының адресін анықтайды. Жеңілдік үшін бұл конструкцияны нұсқауыш сілтеме жасап тұрған айнымалының аты ретінде қабылдауға болады. Осы айнымалыға сәйкес типтегі шамалар үшін анықталған барлық әрекеттерді нұсқауышқа да қолдануға болады (егер нұсқауыш инициалданған болса). Жадының бір аймағына әртүрлі типтегі бірнеше нұсқауыш сілтеме жасауы мүмкін. Оларға қолданылатын адрессіздендіру операциясы әртүрлі нәтиже береді. Мысалы, төмендегі программаны IBM PC-мен үйлесімді компьютерде орындағанда:

```
#include <stdio.h>
int main(){
   unsigned long int A = 0Xcc77ffaa;
   unsigned short int* pint = (unsigned short int*) &A;
   unsigned char* pchar = (unsigned char *) &A;
   printf("  | %x | %x | %x |", A, *pint, *pchar);
   return 0;
}
```

оның экранға шығарылатын нәтижесі:

```
| cc77ffaa | ffaa | aa |
```

pint және **pchar** нұсқауыштарының мәндері бірдей, бірақ адрессіздендіру нәтижесінде **pchar** осы адрес бойынша бір кіші байтты, ал **pint** – екі кіші байтты береді.

Жоғарыда келтірілген мысалда нұсқауыштарды инициалдау кезінде *типтерді түрлендіру операциясы* қолданылған. Типтерді айқын түрде түрлендіру операциясының синтаксисі қарапайым: керекті тип айнымалы атының алдында жақша ішінде көрсетіледі. Мұндайда ақпараттардың осы күйде сақталуына кепілдік берілмейді, сондықтан типтерді осындай айқын түрдегі түрлендірулерді қолданбаған жөн. Бұл туралы толығырақ мәлімет 249 б. келтірілген.

Бір өрнекте әртүрлі типтегі нұсқауыштарды араластыра қолданған кезде, **void*** типінен басқа, барлық нұсқауыштардың типтерін айқын түрде

түрлендіру қажет етіледі. Нұсқауыш айқын түрде көрсетілмей-ақ, **bool** типті мәнге түрлендірілуі мүмкін (мысалы, шартты оператор өрнегінде), мұндайда нөлге тең болмайтын нұсқауыш **true** мәніне, ал нөлге тең нұсқауыш **false** мәніне ие болады.

Айқын түрдегі түрлендірусіз орындалатын меншіктеу операциясы төменде көрсетілген екі жағдайда мүмкін болады:

- **void*** типті нұсқауыштар үшін;
- егер меншіктеу операциясының сол және оң жақтарындағы нұсқауыштардың типтері бірдей болса.

Осылайша, айқын түрде көрсетілмеген түрлендіру тек **void*** типі үшін орындалады. 0 мәні айқын емес түрде кез келген тип нұсқауышына түрлендіріледі. Объектілерге нұсқауыштарды функцияларға нұсқауыштарға (және керісінше) меншіктеу мүмкін емес. Кез келген типтегі нұсқауыштұрақтыларға және кез келген типтегі тұрақтыларға да мән меншіктеуге тыйым салынған (тұрақтыларға нұсқауыштарға және нұсқауыш-тұрақты сілтеме жасайтын айнымалыларға мән меншіктеуге рұқсат етілген).

Нұсқауыштармен орындалатын арифметикалық операциялар (тұрақтымен қосу, азайту, инкремент және декремент) нұсқауыштар адрестейтін шамалардың типтері мөлшерін (енін) автоматты түрде ескереді. Мұндай операциялар тек бір типтегі нұсқауыштарға қолданылады және олар компьютер жадында тізбектеле орналасқан мәліметтер құрылымдарымен, мысалы, жиымдармен жұмыс істеу кезінде кең пайдаланылады.

Инкремент операциясы нұсқауышты жиымның келесі элементіне, ал *декремент* – алдыңғы элементіне көшіреді. Нұсқауыштың мәні іс жүзінде **sizeof(тип)** шамасына өзгеріп отырады. Егер нақты бір типке нұсқауыш тұрақтыға артса немесе кемісе, оның мәні берілген типтегі объект енін осы тұрақтыға көбейтілген шамаға өзгереді, мысалы:

```
short * p = new short [5];
p++;                          // p мәні 2-ге артады
long * q = new long [5];
q++;                          // q мәні 4-ке артады
```

Екі нұсқауыштың *айырмасы* – бұл нұсқауыштар мәндерінің айырмасын байтпен берілген тип көлеміне бөлу болып табылады (мысалы, жиымдарға қатысты үшінші және алтыншы элементтерге нұсқауыштардың айырмасы үшке тең). Екі нұсқауышты қосуға болмайды.

Нұсқауыштары бар өрнектерді жазған кезде операциялар басымдылықтарына назар аудару керек. Мысал ретінде

```
*p++ = 10;
```

операторында берілген әрекеттер тізбегін қарастырайық.

Инкремент және адрессіздендіру операцияларының басымдылықтары бірдей болады және олар солдан оңға қарай орындалады, алайда инкремент постфиксті болғандықтан, ол меншіктеу операциясынан кейін орындалады. Осылайша, **p** нұсқауышында сақталған адрес бойынша 10 мәні жазылады, содан кейін нұсқауыш оның типіне сәйкес байт санына артады. Осы айтылғандарды төмендегідей түрде реттеп жазуға да болады:

```
*p = 10;  p++;
```

(*p)++ өрнегі, керісінше, нұсқауыш сілтеме жасап тұрған мәнді инкременттейді.

Адресті алудың унарлық операциясы – **&** аты бар және жедел жадыда орналастырылған шамаларға қолданылады. Осылайша, скалярлық өрнектің, ат берілмеген тұрақтының немесе регистрлік айнымалының адресін алу мүмкін емес. Операция мысалдары жоғарыда келтірілді.

Сілтемелер

Сілтеме сілтемені инициалдау кезінде көрсетілген атаудың синонимі болып табылады. Сілтемені әрқашанда атаусызданатын (аты жойылып кететін) нұсқауыш ретінде қарастыруға болады. Сілтемені жариялаудың форматы:

```
типі & аты;
```

мұндағы тип – сілтеме нұсқап тұратын шаманың типі, **&** – сілтеме операторы, ол өзінен кейін жазылатын атау сілтемелік типті айнымалының аты екенін көрсетеді, мысалы:

```
int kol;
int& pal = kol;          /* pal сілтемесі – kol үшін
                            альтернативті атау */
const char& CR = '\n';   // тұрақтыға сілтеме
```

Төмендегі ережелерді есте сақтаңыз.

- Айнымалы-сілтеме сипатталу кезінде айқын түрде инициалдануы тиіс, бұл ереже келесі жағдайларда: сілтеме функцияның параметрі болғанда (80 б.), ол **extern** ретінде сипатталған кезде немесе кластың мәліметтер өрісіне (193 б.) сілтеме жасаған кездерде орындалмайды.
- Инициалданғаннан кейін сілтемеге басқа айнымалыны меншіктеуге болмайды.
- Сілтеменің типі сол сілтеме жасалған шама типімен сәйкес келуі керек.
- Сілтемелерге нұсқауыштарды анықтауға, сілтемелер жиымдарын құруға және сілтемелерге сілтеме жасауға тыйым салынған.

Сілтемелер көбінесе функциялар параметрлері ретінде және функцияның қайтаратын мәндерінің типтері түрінде қолданылады. Сілтемелер функциялардың құрамында белгілі бір адреске берілетін айнымалыларды адрессіздендіру операциясының қатысуынсыз қолдануға мүмкіндік береді,

бұл программаны оқуды жеңілдетеді (толығырақ «Функцияның параметрлері» бөлімінде берілген, 79 б.).

Сілтеменің нұсқауыштан айырмашылығы – ол компьютер жадында қосымша орын алмайды және ол сол шаманың басқа атауы болып табылады. Сілтемеге операциялар қолдану сілтеме жасалған шаманың өзгеруіне алып келеді.

Жиымдар

Қарапайым айнымалыларды пайдаланған кезде мәліметтерді сақтауға арналған әрбір жады аймағына өзіндік жеке атау беріледі. Егер бір типтегі шамалар жиынымен бірдей әрекеттер орындау керек болса, оларға бір ғана атау беріледі де, олар бір-бірінен реттік нөмірлері бойынша ажыратылады. Осылайша көптеген операцияларды циклдер көмегімен жинақы түрде жазуға мүмкіндік туады. Біртиптегі шамалардың ат беріліп, шектелген тізбегі *жиым* деп аталады. Программадағы жиымды сипаттаудың қарапайым айнымалыны сипаттаудан айырмашылығы: жиым атауынан кейін тік жақшаның ішінде жиымтегі элементтер саны (*өлшемі*) көрсетіледі:

```
float a [10]; // 10 нақты саннан тұратын жиымды сипаттау
```

НАЗАР АУДАРЫҢЫЗ

Жиымдарды сипаттау кезіндегі тік жақшалар конструкцияның міндетті еместігінің көрсеткіші емес, синтаксис элементі болып табылады.

Жиым элементтері нөлден бастап нөмірленеді. Жиымды сипаттау кезінде қарапайым айнымалыларды сипаттауда жазылатын модификаторлар (жады класы, **const** және инициализатор) қолданылады. Жиымдарды инициалдайтын мәндер жүйелі жақшалардың ішіне жазылады. Әрбір элементке берілетін мәндер реті бойынша меншіктеледі. Егер жиым элементтерінің саны инициалдауға берілген мәндер санынан артық болса, мәндері көрсетілмеген элементтер нөлге теңестіріледі:

```
int b[5] = {3, 2, 1};  /* b[0]=3, b[1]=2, b[2]=1,
                          b[3]=0, b[4]=0 */
```

Жиымның өлшемі және элементтерінің типі жиымды орналастыруға қажетті жады көлемін анықтайды, оларға бөлінетін жады программаны компиляциядан өткізу кезеңінде беріледі, сондықтан жиым өлшемі тек бүтін оң *тұрақтымен* немесе *тұрақтылық өрнекпен* берілуі тиіс. *Егер жиымды* сипаттау кезінде оның өлшемі көрсетілмеген болса, онда инициализатор болуы керек, мұндайда компилятор жадыны инициалдаушы мәндер санына байланысты бөледі. Кейін жиым өлшемі формалды параметрлер тізімінде де кездеспеуі мүмкін екендігін көре аламыз («Функцияның параметрлері» бөлімін қараңыз, 79 б.).

Жиым элементіне қол жеткізу үшін оның атынан кейін тік жақшалар ішінде элемент нөмірі *(индекс)* көрсетіледі. Келесі мысалда жиым элементтерінің қосындысы есептеледі.

```
#include <iostream.h>
int main(){
   const int n = 10;
   int i, sum;
   int marks[n] = {3, 4, 5, 4, 4};
   for (i = 0, sum = 0; i<n; i++) sum += marks[i];
   cout << "Элементтер қосындысы:  " << sum;
   return 0;
}
```

Жиымдардың өлшемдерін, осы мысалда көрсетілгендей түрде, атауы бар тұрақтылар көмегімен берген жөн, себебі мұндайда оны өзгерту үшін программаның тек бір бөлігіндегі тұрақтының мәнін түзету жеткілікті. Жиымның соңғы элементінің нөмірі оны сипаттау кезінде көрсетілген өлшемінен бірге кем болатынына назар аударыңыз.

НАЗАР АУДАРЫҢЫЗ

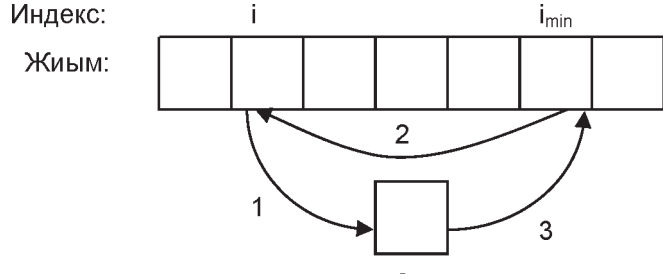
Жиым элементтерін пайдалану кезінде элемент индексінің жиым шекарасынан шығып кетуі автоматты түрде қадағаланбайды, бұл қателіктердің туындауына алып келеді.

Мысал. Бүтін сандар түрінде берілген жиымды *таңдау әдісімен* сұрыптау (реттеу) керек. Алгоритмнің жұмысы келесідей: жиымның ең кіші элементі таңдалып, ол жиымтың алғашқы элементімен орын ауыстырады, содан кейін жиым екінші элементтен бастап қарастырылып, оның да ішіндегі ең кішісі мен екінші элементтің орны ауыстырылады және осылайша n-1 рет орын ауыстырулар атқарылады (циклдің ең соңғы орындалуында, қажет болса, ең соңғы элемент өзінің алдындағы элементпен алмасады).

```
#include <iostream.h>
int main(){
   const int n = 20;    // жиым элементтерінің саны
   int b[n];             // жиымды сипаттау
   int i;
   for (i = 0; i<n; i++) cin >> b[i]; // жиымды енгізу
   for (i = 0; i<n-1; i++){   /* n-1 рет ең кіші
                              элементті іздейміз */
      /* қарастырылып отырған элементтердің алғашқысын
      ең кіші элемент ретінде қабылдаймыз: */
      int imin = i;
      /* реттелмеген элементтер ішінен ең кіші элемент
      нөмірін іздеу: */
      for (int j = i + 1; j<n; j++)
```

```
        //одан кіші элемент тапсақ, оның нөмірін сақтаймыз
           if (b[j] < b[imin]) imin = j;
        int a = b[i];          // элементтерді алмастыру
        b[i] = b[imin];        // нөмірлері i
        b[imin] = a;           // және imin
    }
    // реттелген жиымды экранға шығару:
    for (i = 0; i<n; i++)  cout << b[i] << ' ';
    return 0;
}
```

Циклдің **i**-ші айналымында **a** буферлік айнымалысының көмегімен жиымның **i** және **imin** нөмірлі элементтерінің орнын алмастыру процесі 1.9-суретте көрсетілген. Бағыттауыш сызықтардың жанындағы цифрлар әрекеттердің орындалу ретін көрсетеді.



1.9-сурет. Екі айнымалының мәндерін алмастыру

Жиым идентификаторы оның нөлінші элементіне тұрақтылық нұсқауыш болып табылады. Мысалы, жоғарыда қарастырылған листингтегі жиым үшін **b** атауы **&b[0]** белгілеуімен бірдей болып саналады, ал жиымның **i**-ші элементін ***(b+i)** өрнегі арқылы пайдалануға болады. Нұсқауышты сипаттап, оған жиымтың бастапқы адресін меншіктей отырып, жиыммен нұсқауыш арқылы жұмыс істеу мүмкіндігі қарастырылған. Келесі программа үзіндісі **a** жиымының барлық элементтерін **b** жиымына көшіріп жазады:

```
int a[100], b[100];
int *pa = a;              // немесе int *p = &a[0];
int *pb = b;
for(int i = 0; i<100; i++)
   *pb++ = *pa++;         // немесе pb[i] = pa[i];
```

Динамикалық жиымдар new операциясының көмегімен құрылады, мұнда оның типі мен өлшемін көрсету керек, мысалы:

```
int n = 100;
float *p = new float [n];
```

Бұл жолда **float** типіне нұсқауыш-айнымалы құрылады, динамикалық жадыда нақты типтегі 100 элемент орналастыруға жеткілікті үздіксіз аймақ бөлінеді және оның бастапқы адресі **p** нұсқауышына жазылады. Динамикалық жиымдарды құру кезінде инициалдауға болмайды және олар нөлге теңестірілмейді.

Динамикалық жиымдардың артықшылығы: олардың өлшемдері, яғни элементтерінің саны айнымалы болуы мүмкін, яғни жиымға бөлінетін жады көлемі программаның орындалу барысында анықталады. Динамикалық жиым элементтерін пайдалану дәл статикалық жиым элементтерін пайдалану сияқты орындалады, мысалы, жоғарыда көрсетілген жиымның 5-ші нөмірлі элементіне **p[5]** немесе ***(p+5)** өрнектері арқылы қол жеткізуге болады.

Динамикалық жиымды құрудың альтернативті (баламалы) тәсілі – С тілі кітапханасының **malloc** функциясын қолдану:

```
int n = 100;
float *q = (float *) malloc(n * sizeof(float));
```

malloc функциясын пайдалану алдында жазылған типті түрлендіру операциясының қажеттілігі келесі себеппен түсіндіріледі: функция **void*** типті нұсқауыш мәнін қайтарады, ал инициалданған нұсқауыштың типі – **float.**

Динамикалық жиым үшін **new[]** операциясы арқылы бөлінген жады **delete[]** операциясы арқылы босатылады, ал **malloc** функциясы арқылы бөлінген жадыны **free** функциясының көмегімен босату керек, мысалы:

```
delete [] p; free (q);
```

Компьютер жадын бөлу және босату тәсілдері сәйкес келмеген жағдайда туындайтын нәтиже белгісіз болып табылады. **delete** операциясында жиым өлшемі көрсетілмейді, бірақ тік жақшалар міндетті түрде жазылуы тиіс.

Көпөлшемді жиымдар оның әрбір өлшемін тік жақшалар ішіне жазу арқылы беріледі, мысалы

```
int matr [6][8];
```

операторы 6 жолдан және 8 бағанадан тұратынын екіөлшемді жиымды сипаттайды. Мұндай жиым компьютер жадында тізбектелген ұяшықтарда бірбіріне жалғаса отырып, *жолдар бойынша қатарласа* орналасады. *Көпөлшемді жиымдар* олардың құрамындағы бір элементтен келесісіне көшу кезінде алдымен соңғы индекс өзгеретіндей етіп орналастырылады. Көпөлшемді *жиым* элементіне қол жеткізу (пайдалану) үшін оның барлық индекстері көрсетіледі, мысалы, **matr[i][j]**, немесе басқаша жолдармен: ***(matr[i]+j)** немесе ***(*(matr+i)+j)** түрінде жазылады. **matr[i]** жиымның **i**-ші жолының бастапқы адресі болғандықтан, осындай жазу түрлерін пайдалану мүмкін болып табылады.

Көпөлшемді жиымды инициалдау кезінде ол жиымдардан құралған жиым ретінде беріледі де, оның ішкі әрбір жиымы өз жүйелі жақшаларымен

қоршалып тұрады (мұндағы сипаттау кезінде өрнектің сол жағында жиым өлшемдерін көрсету міндетті емес):

```
int mass2 [][2] = { {1, 1}, {0, 2}, {1, 0} };
int mass2 [3][2] = {1, 1, 0, 2, 1, 0};
```

Мысал. Программа бүтін санды матрицадағы нөлге тең элементтерінің саны ең көп болатын жол (қатар) нөмірін анықтайды.

```
#include <stdio.h>                  // C тілі стилінде
int main(){
   const int nstr = 4, nstb = 5;  // жиым өлшемдері
   int b[nstr][nstb];             // жиым сипаттамасы
   int i, j;
   for (i = 0; i<nstr; i++)       // жиымды енгізу
   for (j = 0; j<nstb; j++) scanf("%d", &b[i][j]);
   int istr = -1, MaxKol = 0;
   // жиымды жолдар бойынша қарап шығу:
   for (i = 0; i<nstr; i++){
      int Kol = 0;
      for (j = 0; j<nstb; j++) if (b[i][j] == 0) Kol++;
      if (Kol > MaxKol){istr = i; MaxKol = Kol;}
   }
   printf(" Бастапқы жиым:\n");
   for (i = 0; i<nstr; i++){
      for (j = 0; j<nstb; j++) printf("%d ", b[i][j]);
      printf("\n");}
   if (istr == -1) printf("Нөлге тең элементтер жоқ");
   else printf("Жол нөмірі: %d", istr);
   return 0;}
```

Қажетті жол нөмірі **istr** айнымалысында, ағымдағы (**i**-ші) жолдағы нөлге тең элементтердің саны – **Kol** айнымалысында, нөлге тең элементтердің ең үлкен саны – **MaxKol** айнымалысында сақталады. Жиым жолдар бойынша тексеріледі, олардың әрқайсысындағы нөлге тең элементтер саны есептеледі (әрбір жолды қарастыру алдында **Kol** айнымалысы нөлге теңестірілетініне назар аударыңыз). Нөлдердің ең көп саны және соған сәйкес жол нөмірі есте сақталады.

Динамикалық көпөлшемді жиымды құру үшін **new** операциясында жиымның барлық өлшемдерін көрсету керек (ең сол жақтағы өлшем айнымалы болуы мүмкін), мысалы:

```
int nstr =5;
int ** m = (int **) new int [nstr][10];
```

Екіөлшемді жиымға компьютер жадынан орын бөлудің әмбебап әрі қауіпсіз тәсілі – оның екі өлшемін де программаның орындалуы барысында беру, мысалы:

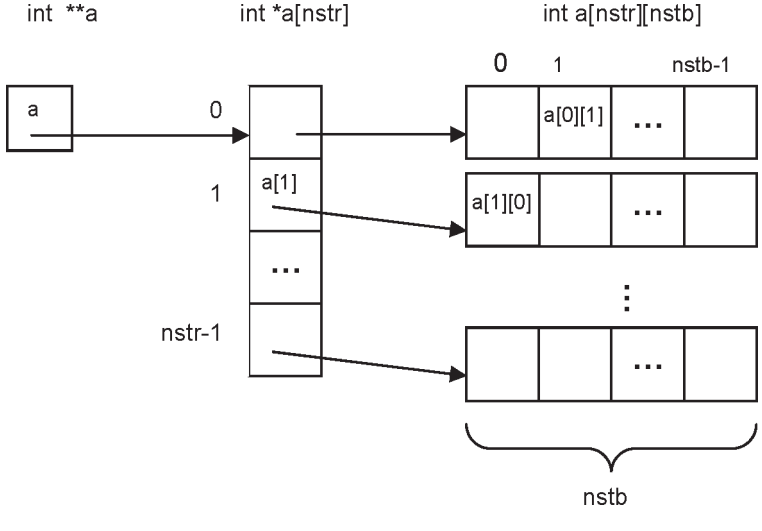
```
int nstr, nstb;
cout << " Жолдар және бағаналар санын енгізіңіз:";
cin >> nstr >> nstb;
int **a = new int *[nstr];      //1
for(int i = 0; i<nstr; i++)     // 2
   a[i] = new int [nstb];       // 3
...
```

1 операторында «**int** типіне нұсқауышқа нұсқауыш» типіндегі айнымалы жарияланады және жиым жолдарына (жолдар саны – **nstr**) нұсқауыштардан құралған жиым үшін жады бөлінеді. **2** операторында жиымның әрбір жолына жады бөлу үшін цикл ұйымдастырылады. **3** операторында жолдарға нұсқауыштардан құралған жиымның әрбір элементіне екіөлшемді жиымның жолы үшін бөлінген жады аймағының бастапқы адресі меншіктеледі. Әрбір жол **int** типті элементтердің **nstb** санынан тұрады (1.10-сурет).

Кез келген өлшемдердегі жиымнан компьютер жадын босату **delete[]** операциясы арқылы орындалады. Тұрақтыға нұсқауышты жою мүмкін емес.

ЕСКЕРТУ

Сипаттауларды дұрыс түсіну үшін келесі мнемоникалық ережені есте сақтаған жөн: «суффикс префикстен берік бекітілген». Егер айнымалыны сипаттау кезінде бір мезгілде ***** (нұсқауыш) префиксі және **[]** (жиым) суффиксі қатар қолданылса, онда айнымалы жиымға нұсқауыш ретінде емес, нұсқауыштар жиымы ретінде қарастырылады: мысалы, **int *p[10]; – int** типіне арналған 10 нұсқауыштан құралған жиым.



1.10-сурет. Екіөлшемді жиымға жады бөлу

Жиымтар қолданылатын басқа программалар практикумның [11] 55-88 беттерінде қарастырылған.

Тіркестер

Сөз тіркесі немесе тіркес нөл-символмен аяқталатын, символдардан құралған жиым болып табылады. Нөл-символ – бұл коды нөлге тең болатын символ, ол `'\0'` басқару тізбегі түрінде жазылады. Нөл-символдың орналасуы бойынша тіркестің ұзындығы анықталады. Тіркесті тіркестік литералмен инициалдауға болады (13 б. қараңыз):

```
char str[10] = "Vasia";
// 0 мен 9 аралығында нөмірленген 10 элемент бөлінген
// алғашқы элементтер - 'V', 'a', 's', 'i', 'a', '\0'
```

Бұл мысалда тіркеске 10 байт бөлінген, олардың бесеуі тіркес символдарына, ал алтыншысы нөл-символға жұмсалған. Егер тіркес анықталу кезінде инициалданса, оның өлшемін жазу міндетті емес (компилятор қажетті байт санын өзі бөледі):

```
char str[] = "Vasia"; // 6 байт бөлінді және толтырылды
```

Келесі оператор:

```
char *str = "Vasia"
```

тіркестік айнымалы емес, өзгертілуі мүмкін емес тіркестік тұрақтыға (мысалы, `str[1]='o'` түріндегі операторды жазуға болмайды) нұсқауыш жасайды. Тіркестік литерал алдындағы теңдік белгісі меншіктеуді емес, инициалдауды білдіреді. Бір тіркесті екіншісіне меншіктеу операциясы анықталмаған (себебі тіркес жиым болып табылады), ол цикл арқылы немесе стандартты кітапхана функцияларының көмегімен орындалуы тиіс. Кітапхана көшіру, салыстыру, тіркестерді біріктіру, ішкі тіркесті іздеу, тіркес ұзындығын анықтау және т.с.с. мүмкіндіктерді орындауды қамтамасыз етеді (кітапхана мүмкіндіктері «Тіркестермен және символдармен жұмыс істеу функциялары» бөлімінде (97 б.) және 6-қосымшада көрсетілген), сонымен қатар кітапхана құрамында тіркестер мен жеке символдарды пернетақтадан және файлдардан енгізудің арнайы функциялары қарастырылған.

Мысал. Программа құпия сөзді (пароль) үш реттен артық сұрамайды.

```
#include <stdio.h>
#include <string.h>
int main(){
   char s[5], passw[] = "kuku";
   // passw – эталондық  пароль.
   // *passw = "kuku"; түрінде де сипаттауға болады
   int i, k = 0;
   for (i = 0; !k && i<3; i++){
      printf("\nПароль енгізіңіз (4 символ):\n");
      gets(s);              // тіркесті енгізу функциясы
```

```
      if (strcmp(s.passw) == 0) k = 1;  /* тіркестерді
                                        салыстыру функциясы */
   }
   if (k) printf("\nпароль қабылданды");
   else printf("\nпароль қабылданбады");
   return 0;
}
```

Тіркестермен жұмыс істеу кезінде нұсқауыштар жиі қолданылады.

НАЗАР АУДАРЫҢЫЗ

Тіркестермен жұмыс істеу кезінде жиі кездесетін қателіктер – нөл-символдың жоқ болуы және тіркесті қарастыру кезінде нұсқауыштың тіркес шекарасынан шығып кетуі.

Енді **src** тіркесін **dest** тіркесіне көшіру процесін қарастырайық. Оның алгоритмі төмендегідей болады:

```
char src[10], dest[10];
...
for (int i = 0; i<=strlen(src); i++) dest[i] = src[i];
```

Тіркес ұзындығы **strlen** функциясы арқылы анықталады, бұл функция тіркес ұзындығын нөл-символдың тұрған орнын іздеу жолымен есептеп шығарады. Осылайша, тіркес екі рет қарастырылады. Нөл-символға тексеруді тікелей программаның орындалу барысында анықтаған айтарлықтай тиімді болып табылады. Индексті арттыруды нұсқауыштар инкрементімен алмастыруға болады (ол үшін **src** тіркесіне жады динамикалық түрде бөлінуі тиіс, сонымен қатар, қосымша нұсқауышты анықтап, оны **dest** тіркесінің бастапқы адресімен инициалдау керек):

```
#include <iostream.h>
int main(){
   char *src = new char [10];
   char *dest = new char [10], *d = dest;
   cin >> src;
   while (*src != 0) *d++ = *src++;
   *d = 0;                          // аяқтаушы нөл
   cout << dest;
   return 0;
}
```

Цикл ішінде тіркестер элементтерін символдап меншіктеумен бірге, нұсқауыштарды инкременттеу қоса орындалады. Меншіктеу операциясының нәтижесі – берілетін мән, ол цикл шартында тексеріледі, сондықтан меншіктеуді шарттың орнына қойып, нөлге тең емес екенін тексеру шартын алып тастауға болады (мұнда аяқтаушы нөл циклде көшіріледі, оны меншіктеу үшін жеке

оператордың қажеті жоқ). Осылардың нәтижесінде тіркесті көшіру циклінің шарты келесідей түрде болады:

```
while (*d++ = *src++);
```

Тіркестермен жұмыс істеудің екі тәсілі де (жиымдар және нұсқауыштар арқылы) қолайлы, олардың әрқайсысының жағымды және жағымсыз жақтары бар, бірақ жалпы жағдайда, кітапхана функцияларын пайдалануды немесе C++ тілінің стандартты кітапханасында анықталған **string** класын қолданған жөн. **string** класы тіркестерді индекстеуді, меншіктеуді, салыстыруды, толықтыруды, біріктіруді және ішкі тіркестерді іздеуді, сонымен қатар Стіркестерді түрлендіруді, яғни **char** типті жиымдарды string типіне және керісінше түрлендіруді қамтамасыз етеді (C++ тіліндегі тіркестер «Тіркестер» бөлімінде, 307 б. сипатталған).

ЕСКЕРТУ

String тіркестерінің **char** типті символдардан тұруы міндетті емес. Бұл кез келген символдар жиынын (тек ASCII ғана емес) қолдануға мүмкіндік береді, алайда кездейсоқ жиын үшін көшіру операциясын өзіңіз анықтап алуыңыз керек, ал бұл жұмыстың нәтижелілігін төмендетуі мүмкін.

Тіркестерді қолданатын программалар құру практикумның [11] бесінші семинарында көрсетілген.

Қолданушылар анықтайтын мәліметтер типтері

Өмірдегі нақты есептерді шығару кезінде өңделуі тиіс ақпараттың құрылымы айтарлықтай күрделі болуы мүмкін. Оларды түпнұсқасына сәйкес түрде бейнелеу үшін қарапайым мәліметтер типтерінің – жиымдар мен нұсқауыштардың негізінде құрылған күрделірек мәліметтер типтері қолданылады. C++ тілі программалаушыға өзіндік мәліметтер типтерін құруға және олармен жұмыс істеу ережелерін анықтауға мүмкіндік береді. Оларды тарихи түрде *қолданушылар анықтайтын мәліметтер типтері* деп атау қабылданған, алайда оларды программалаушы анықтайтын мәліметтер типтері деп атаған дұрысырақ болар еді.

Типтердің атын ауыстыру (typedef)

Программаның айқындылығын жоғарылату үшін типке **typedef** түйінді сөзінің көмегімен жаңа ат беру мүмкіндігі қарастырылған:

```
typedef тип жаңа_атау [ өлшемі ];
```

мұндағы тік жақшалар синтаксис элементі болып табылады. Тип өлшемін көрсетпеуге де болады, яғни ол міндетті емес. Мысалдар:

```
typedef unsigned int UINT;
typedef char Msg[100];
typedef struct{
   char fio[30];
   int date, code;
   double salary;} Worker;
```

Осындай жолмен енгізілген жаңа атауды стандартты типтердің атаулары сияқты қолдануға болады:

```
UINT i, j;          // unsigned int типті екі айнымалы
Msg str[10];        // 100 символдық 10 жолдан құралған жиым
Worker stuff[100]; // 100 құрылымнан тұратын жиым
```

Осы **typedef** түйінді сөзі сипаттамалары ұзақ болып келетін типтер үшін қысқартылған атаулар берумен қатар, программаларды бір платформадан екіншісіне көшіру мүмкіндігін жеңілдету үшін де қолданылады: егер машинаға тәуелді типтер **typedef** операторларының көмегімен жарияланған болса, программаны басқа платформаға ауыстырған кезде тек осы операторларға ғана өзгеріс енгізу жеткілікті болып табылады.

Тізбелер (enum)

Программаларды жазу кезінде кейде барлығының мәндері әртүрлі болуы талап етілетін, атау берілген бірнеше тұрақтыларды анықтау қажеттілігі туады (мұндағы олардың нақты мәндері маңызды болмауы да мүмкін). Осындай мақсатта барлық мүмкін мәндері бүтін сандық тұрақтылармен берілетін тізбелік мәліметтер типтерін қолданған ыңғайлы болып табылады. Оның форматы:

```
enum [ тип_атауы ] { тұрақтылар_тізімі };
```

Тип атауы программада айнымалының жаңа бір типін анықтау керек болған жағдайда ғана беріледі. Компилятор осы айнымалылар мәндерінің тек берілген тұрақтылар тізімінен қабылдануын қамтамасыз етеді. Тұрақтылар бүтін сандық типте болуы керек және әдеттегі жолмен инициалданатын болуы тиіс. Инициализатор жоқ болса, бірінші тұрақты нөлге теңестіріледі де, ал әрбір келесі тұрақты алдыңғысымен салыстырғанда бірге артық мән қабылдап отырады:

```
enum Err {ERR_READ, ERR_WRITE, ERR_CONVERT};
Err error;
...
switch (error){
   case ERR_READ:     /* операторлар */ break;
   case ERR_WRITE:    /* операторлар */ break;
```

```
   case ERR_CONVERT;  /* операторлар */ break;
}
```

ERR_READ, ERR_WRITE, ERR_CONVERT тұрақтыларына сәйкесінше 0, 1 және 2 мәндері меншіктеледі.

Басқа мысал:

```
enum {two = 2, three, four, ten = 10,
   eleven, fifty = ten + 40};
```

Мұндағы **three** және **four** тұрақтыларына 3 және 4 мәндері, **eleven** тұрақтысына 11 мәні беріледі.

Тізбелік тұрақтылардың атаулары қайталанбайтын бірегей болуы тиіс, алайда олардың мәндері бір-бірімен сәйкес келе беруі мүмкін. Тізбелерді қолданудың атаулы тұрақтыларды сипаттаудан немесе **#define** директивасынан («#define директивасы» бөлімін қараңыз, 100 б.) артықшылығы – байланысқан тұрақтылардың көрнектілігінде, сонымен қатар, компилятор тұрақтыларды инициалдау кезінде типтерді тексеруді орындай алады.

Арифметикалық операцияларды орындау кезінде тізбелер бүтін типке түрлендіріледі. Тізбелер қолданушылар анықтайтын типке жататындықтан, олар үшін өзіндік операцияларды да енгізуге болады («Операциялардың асыра жүктелуі» тақырыбын қараңыз, 205 б.).

ЕСКЕРТУ

Тізбенің мәндер диапазоны оның барлық мәндерін бейнелеуге қажетті биттер санымен анықталады. Бүтін типті кез келген мәнді айқын түрде тізбелік типке келтіруге болады, бірақ оның диапазонынан шыққан жағдайдағы нәтиже анықталмаған.

Құрылымдар (struct)

Элементтерінің бәрі бір типте болып табылатын жиыммен салыстырғанда, құрылымдар әртүрлі типтегі элементтерден тұруы мүмкін. C++ тілінде құрылым кластың бір түрі болып табылады және оған кластың барлық қасиеттері тән, алайда құрылымды көбінесе C тілінде анықталғандай түрде қолдану жеткілікті болып саналады:

```
struct [ тип_аты ] {
   тип_1 элемент_1;
   тип_2 элемент_2;
   ...
   тип_n элемент_n;
} [ сипаттауыштар_тізімі ];
```

Құрылым элементтері *құрылымның өрістері* деп аталады және олар осы құрылым типінен басқа кез келген типте болуы мүмкін немесе осы

құрылымға нұсқауыш та болуы мүмкін. Егер типтің атауы берілмеген болса, онда айнымалылардың, нұсқауыштардың немесе жиымдардың сипаттауыштары тізімі көрсетілуі керек. Мұндайда құрылымның сипаттамасы осы тізім элементтерінің анықтамасы ретінде қолданылады:

```
// Құрылымдар жиымын және құрылымға нұсқауышты анықтау:
struct {
   char fio[30];
   int date, code;
   double salary;
} stuff[100], *ps;
```

Егер тізім берілмесе, құрылымның сипаттамасы жаңа типті анықтайды, оның атын кейіннен стандартты типтермен қатар қолдануға болады, мысалы:

```
struct Worker{     // Worker жаңа типінің сипаттамасы
   char fio[30];
   int date, code;
   double salary;
};                 // сипаттама нүктелі үтірмен аяқталады
/* Worker типті жиымды және Worker типіне нұсқауышты
анықтау: */
Worker stuff[100], *ps;
```

Компилятордың құрылым мөлшерін білуі міндетті болмаған кезде құрылымды жариялағаннан кейін бірден оның атын қолдана беруге болады (анықтамасын кейінірек беруге рұқсат етіледі), мысалы:

```
struct List;               // List құрылымын жариялау
struct Link{
   List *p;               // List құрылымына нұсқауыш
   Link *prev, *succ;     // Link құрылымына нұсқауыштар
};
struct List { /* List құрылымының анықтамасы */};
```

Бұлар бір-бірімен байланысқан құрылымдар тізімдерін құруға мүмкіндік береді.

Құрылымды инициалдау үшін оның элементтерінің мәндерін жүйелі жақшаның ішінде сипатталу реті бойынша тізіп жазу керек:

```
struct{
   char fio[30];
   int date, code;
   double salary;
} worker = {"Страусенко", 31, 215, 3400.55};
```

Құрылымдардан тұратын жиымдарды инициалдау кезінде жиымның әрбір элементін жүйелі жақшаның ішіне жазу керек (көпөлшемді жиым – бұл жиымдардан құралған жиым екенін ескеру қажет):

```
struct complex{
   float real, im;
} compl [2][3] = {
   {{1, 1}, {1, 1}, {1, 1}},
   // 1 жол, яғни comp1[0] жиымы
   {{2, 2},  {2, 2},  {2, 2}}
   // 2 жол, яғни comp1[1] жиымы
};
```

Құрылымдық типтері бірдей болып келетін айнымалылар үшін **меншіктеу** операциясы анықталған, мұнда әр элементті жеке-жеке көшіріп жазу жүзеге асырылады. Құрылымды функцияға беруге және функция мәні ретінде қайтаруға болады. Құрылымдарға қолданылатын басқа операцияларды қолданушылардың өздері анықтайды («Операциялардың асыра жүктелуі» тақырыбын қараңыз, 205 б.). Құрылымның өлшемі оның элементтерінің өлшемдерінің қосындысына тең болмауы мүмкін, себебі олар сөздің шекараларына байланысты туралanады.

Құрылымның өрістеріне қол жеткізу (пайдалану) – құрылым атауы арқылы орындалғанда **.** (нүкте) және нұсқауыш арқылы орындалғанда **->** таңдау операциялары көмегімен жүзеге асырылады, мысалы:

```
Worker worker, stuff[100], *ps;
...
worker.fio = "Страусенко";
stuff[8].code = 215;
ps->salary = 0.12;
```

Егер құрылым элементі ретінде басқа құрылым қолданылса, онда оның элементтеріне қол жеткізу екі таңдау операциясы арқылы орындалады:

```
struct A {int a; double x;};
struct B {A a; double x;} x[2];
x[0].a.a = 1;
x[1].x = 0.1;
```

Осы мысалдан байқайтынымыздай, әртүрлі құрылымдардың өріс атаулары бірдей болуы мүмкін, себебі олардың көріну аймағы әртүрлі. Егер құрылымдық айнымалыны анықтау кезінде **struct** сөзі қолданылса, онда бір көріну аймағында бірдей атау берілген құрылым мен басқа объектіні (мысалы, айнымалыны немесе жиымты) жариялауға болады, алайда бұлай жасамаған жөн – компилятор программаны дұрыс орындағанмен, өзіңіздің қателесуіңіз әбден мүмкін.

Биттік өрістер

Биттік өрістер – бұл құрылым өрістерінің ерекше бір түрі. Олар мәліметтерді тығыздап жинақтау үшін, мысалы, «иә/жоқ» типіндегі жалаушаларды қысқартып жазуда қолданылады. Программадағы ең кіші адрестелетін жады ұяшығы – 1 байт болғанмен, жалаушаны сақтау үшін бір бит жеткілікті. Биттік өрісті сипаттау кезінде оның атауынан кейін қос нүкте арқылы өрістің ұзындығы биттермен (бүтін оң тұрақты) көрсетіледі:

```
struct Options{
   bool centerX:1;
   bool centerY:1;
   unsigned int shadow:2;
   unsigned int palette:4;
};
```

Биттік өрістер кез келген бүтін типте бола алады. Өрістің атауы көрсетілмеуі мүмкін, мұндай өрістер аппараттық шекараға туралау үшін қолданылады. Өріске қол жеткізу қарапайым тәсілмен – оның атауы арқылы орындалады. Өрістің адресін алу мүмкін емес, алайда басқа қасиеттері бойынша биттік өрістерді карапайым құрылымның өрістері сияқты пайдалана беруге болады. Жеке биттермен орындалатын операциялар байттар мен сөздерге қарағанда аса тиімді орындалмайтынын ескерген жөн, өйткені мұнда компилятор арнайы кодтарды генерациялауы керек, сондықтан айнымалылар үшін жады үнемдеу ісі программа коды көлемінің артуына алып келеді. Биттік өрістерді компьютер жадында орналастыру компилятор мен аппаратураға тәуелді болып саналады.

Құрылымдармен жұмыс практикумның [11] алтыншы семинарында толық қарастырылған.

Біріктірмелер (union)

Біріктірме (union) барлық өрістері бір ғана адрес бойынша орналасатын құрылымның жеке бір түрі болып табылады. Біріктірмелердің сипатталу форматы құрылымның форматымен бірдей, тек **struct** түйінді сөзінің орнына **union** сөзі қолданылады. Біріктірме ұзындығы оның ең үлкен өрісінің ұзындығына тең. Әрбір уақыт мезетінде біріктірме типіндегі айнымалыда тек бір ғана мән сақталады және оның дұрыс қолданылуына программалаушы жауапты болып саналады.

Біріктірмелерді программаны орындау барысында бір мезгілде бір өрістен артық қажет етілмейтіні белгілі болған жағдайларда компьютер жадын үнемдеу үшін қолданады:

```
#include <iostream.h>
int main(){
```

```
   enum paytype {CARD, CHECK};
   paytype ptype;
   union payment{
      char card[25];
      long check;
   } info;
   /* info және ptype мәндерін беру */
   switch (ptype){
      case CARD: cout << "Карта арқылы төлеу: "
                      << info.card; break;
      case CHECK: cout << "Чек арқылы төлеу: "
                       << info.check; break;
   }
   return 0;
}
```

Біріктірмелерді құрылым өрістері ретінде жиі қолданады, мұндайда құрылымға әрбір мезетте біріктірменің қай элементі қолданылатынын анықтайтын қосымша өріс енгізген ыңғайлы. Біріктірменің атын көрсетпеуге болады, бұл мүмкіндік оның өрістеріне тікелей қол жеткізуге (пайдалануға) болатынын білдіреді:

```
#include <iostream.h>
int main(){
   enum paytype {CARD, CHECK};
   struct{
      paytype ptype;
      union{
         char card[25];
         long check;
      };
   } info;
   ... /* info мәнін беру */
   switch (info.ptype){
      case CARD: cout << "Карта арқылы төлеу: "
                      << info.card; break;
      case CHECK: cout << "Чек арқылы төлеу: "
                       << info.check; break;
   }
   return 0;
}
```

Сонымен қатар, біріктірмелер бір биттік бейнелеуді әртүрлі интерпретациялау үшін де қолданылады (бірақ, мұндайда типтерді айқын түрде түрлендіруді қолданған жөн). Мысал ретінде биттік өрістерден тұратын құрылымды қарастырамыз:

```
struct Options{
   bool centerX:1;
   bool centerY:1;
   unsigned int shadow: 2;
   unsigned int palette: 4;
};
union {
   unsigned char ch;
   Options bit;
}option = {0xC4};
cout << option.bit.palette;
option.ch & = 0xF0; // маска қолдану
```

Құрылымдармен салыстырғанда біріктірмелерге бірқатар шектеулер қойылады. Олардың кейбіреуінің мағынасы кейін түсінкті болады:

- біріктірме тек алғашқы элементінің мәні арқылы инициалдана алады;
- біріктірме құрамында биттік өрістердің болуы мүмкін емес;
- біріктірменің құрамында виртуалды әдістердің, конструкторлардың, деструкторлардың және меншіктеу операциясының болуы да мүмкін емес;
- біріктірме кластар иерархиясына кіре алмайды.

2-ТАРАУ

Модульдік программалау

Модульділік – барлық нәтижелі жұмыс
істейтін ірі жүйелердің іргелі аспектісі.
Б. Страуструп

Программаның көлемі ұлғайған сайын оның барлық ұсақ-түйек мәселелерін есте сақтау мүмкіндігі азаяды. Кез келген есептің күрделілігін төмендетудің табиғи тәсілі оны шағын бөліктектерге бөлу болып табылады. C++ тілінде кез келген программаны *функциялар* арқылы қарапайым және түсінікті бөліктерге бөлуге болады, осыдан кейін программаны функциялардың өзара әрекеттесуі деңгейінде ірілендірілген түрде қарастыру мүмкіндігі туады. Адам фактілердің шектеулі көлемін ғана есте сақтауға қабілетті болғандықтан, осы мәселенің өзі өте маңызды болып саналады. Функцияларды пайдалану программаның абстрактылық дәрежесін арттырудың алғашқы қадамы болғандықтан, ол программа құрылымын қарапайым етуге мүмкіндік береді.

Сонымен қатар, программаны функцияларға бөле отырып жазу оның ішіндегі ұқсас кодтарды қайталаудан құтқарады, өйткені функция тек бір рет жазылады, ал оны программаның әртүрлі орындарында қажеттілігіне қарай бірнеше рет қайталап орындауға болады. Функциялардан тұратын программаны түзету процесі де жеңіл атқарылады. Жиі қолданылатын функцияларды кітапханаларға орналастыруға болады. Осылайша, оңай жөнделіп, жеңіл сүйемелденетін программалар құрылады.

Программаның абстрактылық дәрежесін арттыратын келесі қадам – функцияларды және олармен байланысқан мәліметтерді жеке файлдарға (модульдерге) топтастырып жазу, олардың әрқайсысы компиляциядан жеке-жеке өткізіледі. Компиляция нәтижесінде пайда болған объектілік модульдер жинақтауыш көмегімен атқарылатын программаға біріктіріледі. Программаны модульдерге бөлу қайта компиляциялау уақытын азайтады және программаны түзету процесін жеңілдетеді, себебі ол маңызды емес іс-әрекеттерді модуль интерфейсінен тыс орналастырады да, программаны бөлшектей отырып түзетуге (немесе әртүрлі программалаушылардың көмегімен түзетуіне) мүмкіндік береді.

Модуль мәліметтер мен оларды өңдейтін функциялардан тұрады. Басқа модульдердің осы мәліметтерді өңдейтін өзіндік құралдарының болмағаны дұрыс, олар бұл үшін алғашқы модуль функцияларын қолдануы тиіс. Модульді қолдану үшін оның жүзеге асырылуының барлық ерекшеліктерін білудің қажеті жоқ, тек оның интерфейсін білу жеткілікті. Модульдер бір-бірінен

неғұрлым тәуелсіз түрде жұмыс істейтін болса, программаны түзету соғұрлым жеңіл болады. Мұндай тәсіл программаны жөндеу кезінде бір мезгілде есте сақталуы қажет ақпараттың жалпы көлемін азайтады. Программаны барынша тәуелсіз бөліктерге бөлу күрделі мәселе болып табылады, ол программаны жобалау кезеңінде жүзеге асырылуы тиіс.

Жүзеге асыру ерекшеліктерін жасыру инкапсуляция деп аталады. Инкапсуляция тек құрылымдық емес, объектіге бағытталған программалаудың да негізгі идеясы болып табылады. Инкапсуляция мысалы – программалық код бөлігін функцияға жазып орналастыру және оған қажетті барлық мәліметтерді параметрлер ретінде беру. Мұндай функцияны пайдалану үшін оның тақырыбымен анықталатын (аты, қайтаратын мәнінің типі және параметрлерінің типі) интерфейсін білу жеткілікті. Барлық функциялардың тақырыптары және сыртқа қол жеткізуге болатын типтердің, айнымалылардың және тұрақтылардың сипаттамалары *интерфейстің модулі* болып табылады. Ауқымды (глобалды) программалық объектілердің сипаттамалары программаның барлық модульдеріне сәйкес келуі тиіс.

C++ тіліндегі программаның модульділігі препроцессор директиваларының, атаулар кеңістіктерінің, жады кластарының, аластаулар мен жеке компиляциялаудың (анығын айтқанда, программаны бөлшектей отырып компиляциялау тілдің элементі болып табылмайды, ол тілдің жүзеге асырылуына байланысты орындалады) көмегімен сүйемелденеді.

Функциялар

Функциялар күрделі алгоритмдерде
тәртіп орнату үшін қолданылады.
Б. Страуструп

Функцияларды жариялау және анықтау

Функция – бұл белгілі бір аяқталған әрекетті орындайтын сипаттамалар мен операторлардың атау берілген тізбегі. Функция параметрлерді қабылдап, мән қайтара алады.

C++ тіліндегі кез келген программа функциялардан тұрады, олардың бірі міндетті түрде **`main`** деп аталуы тиіс (программаның орындалуы осы функциядан басталады). Функцияны шақырған кезде ол орындала бастайды. Кез келген функция жариялануы және анықталуы тиіс. Функция да басқа шамалар сияқты бірнеше рет жариялануы мүмкін, бірақ ол бір-ақ рет анықталуы тиіс. Компилятор функцияны шақырудың дұрыстығын тексеруі үшін программа мәтінінде функцияны жариялау оны шақырудан ерте орналасуы тиіс.

Функцияны жариялау (прототип, тақырып, сигнатура) оның атын, қайтарылатын мәнінің типін және оған берілетін параметрлер тізімін

тағайындайды. Функцияны анықтауда, оны жариялаумен қатар, функция *тұлғасы* қамтылады, ол жүйелі жақшалар ішіндегі операторлар мен сипаттамалар тізбегінен тұрады:

```
[ класс ] типі атауы([ параметрлер_тізімі ]) [throw
( аластамалар )] { функция тұлғасы }
```

Фукнкцияны анықтаудың құрама бөліктерін қарастырайық.

□ Міндетті түрде қажет етілмейтін **класс** модификаторының көмегімен, **extern** және **static** түйінді сөздерін қолдана отырып, функцияның көріну аймағын айқын түрде көрсетуге болады:

- **extern** – функцияның программаның барлық модульдерінде ауқымды, яғни глобалды (келісім бойынша) түрде көрінуі;
- **static** – функцияның тек өзі анықталған модуль шеңберінде көрінуі.

□ Функцияның қайтаратын мәнінің **типі** жиым мен функциядан (бірақ жиымға немесе функцияға нұсқауыш бола алады) басқа кез келген тип түрінде болуы мүмкін. Егер функция оны шақырған программаға ешқандай мән қайтармайтын болса, онда **void** типі көрсетіледі.

□ **Параметрлер тізімі** функцияға оны шақыру кезінде берілуі тиіс болатын шамаларды анықтайды. Параметрлер тізімінің элементтері бір-бірінен үтір арқылы ажыратылады. Функцияға берілетін әрбір параметрдің аты мен типі көрсетіледі (жариялау кезінде оның атын көрсетпеуге де болады).

□ Функция арқылы өңделетін *аластаулар* туралы «Функцияның аластамалар тізімі» бөлімінде 245 б. жазылған.

Бір функцияның анықталуындағы, жариялануындағы және шақырылуындағы параметрлердің типтері мен жазылу реттілігі бір-бірімен сәйкес келу керек. Параметрлер аттарына олардың сәйкестігі бойынша шек қойылмайды, себебі функцияны әртүрлі аргументтермен шақыруға болады, ал прототиптердегі атауларды компилятор ескермейді (олар тек программаның оқылуын жеңілдету үшін қолданылады).

Функцияны **inline** модификаторының көмегімен *құрамдас* функция ретінде анықтауға болады, бұл модификатор компиляторға функцияны пайдалану орнына оның кодын әрбір шақыру нүктесіне тікелей орналастыру керек екенін көрсетеді. **inline** модификаторы функция типінің алдына қойылады. Ол қысқа функцияларды шақыруға кететін қосымша шығындарды азайту үшін (регистрлерді сақтау және қалпына келтіру, басқаруды беру) қолданылады. i**nline** директивасы ұсынылатын әрекет ретінде беріледі, сондықтан компилятор оны мүмкіндігіне қарай орындайды. **inline** функцияларды пайдалану атқарылатын программаның көлемін ұлғайтып жіберуі мүмкін. Функцияны анықтау оны шақырулардан бұрын орындалуы тиіс,

әйтпесе компилятор **inline** кеңейтілудің орнына қарапайым функцияны шақыру әрекетін атқарады.

Қайтарылатын мәннің типі және параметрлердің типтері біріге отырып, *функцияның типін* анықтайды.

Қарапайым жағдайда функцияны шақыру үшін оның атын, содан кейін жай жақша ішінде үтірлер арқылы бөлініп берілетін аргументтер аттарын көрсету керек. Функцияны шақыру программаның кез келген бөлігінде орындалуы мүмкін, ол тек синтаксис бойынша функцияның қайтаратын типі тұра алатын орында тұрса болғаны. Егер функцияның қайтаратын мәнінің типі **void** болмаса, онда ол кез келген өрнектің құрамына енуі мүмкін немесе жалпы меншіктеу операторының оң жақ бөлігінде орналаса алады.

Екі бүтін шаманың қосындысын қайтаратын функция мысалы:

```
#include <iostream.h>
int sum(int a, int b);              // функцияны жариялау
int main(){
   int a = 2, b = 3, c, d;
   c = sum(a, b);                   // функцияны шақыру
   cin >> d;
   cout << sum(c, d);               // функцияны шақыру
   return 0;
}
int sum(int a, int b){              // функцияны анықтау
   return (a + b);
}
```

Экранға функцияға берілген құрылым өрістерін шығару мысалы:

```
#include <iostream.h>
struct Worker{
   char fio[30];
   int date, code;
   double salary;
};
void print_worker(Worker);          // функцияны жариялау
int main(){
   Worker stuff[100];
   ... /*stuff жиымын қалыптастыру */
   for (int i = 0; i<100; i++)
   print_worker(stuff[i]);          // функцияның шақыру
   return 0;
}
void print_worker(Worker w){        // функцияны анықтау
   cout << w.fio << ' ' << w.date << ' ' << w.code
        << ' ' << w.salary << endl;
}
```

Функцияның ішінде сипатталған барлық шамалар және оның параметрлері жергілікті (локалды) болып табылады. Олардың әрекет ету аймағы – функция іші (тұлғасы). Функцияны шақыру кезінде, кез келген блокка енген кездегі сияқты, стекте жергілікті автоматты айнымалылар үшін жады бөлінеді. Сонымен қатар, стекте функцияны шақырудың алдыңғы сәтіндегі процессор регистрлерінің мәндері және функциядан кері қайту адресі сақталады, ол осы функциядан шыққаннан кейін оны шақырған программаның (басқа функцияның) орындалуын ары қарай жалғастыру үшін қажет болып табылады.

Функциядан шыққан кезде ол орналасқан стек аймағы босатылады, сондықтан бір функцияның бірнеше шақырылулары арасындағы жергілікті (локалды) айнымалылардың мәндері сақталмайды. Егер мұндай жағдайдан құтылу қажет болса, жергілікті айнымалыларды жариялау кезінде **static** модификаторы қолданылады:

```
#include <iostream.h>
void f(int a){
   int m = 0;
   cout << "n m p\n";
   while (a--){
      static int n = 0;
      int p = 0;
      cout << n++ <<''<< m++<<''<< p++ <<'\n';
   }
}
int main(){ f(3); f(2); return 0;
}
```

Жоғарыдағы мысалда **n** статикалық айнымалысы мәліметтер сегментінде орналасады және құрамында оның анықтамасы бар оператордың алғашқы орындалуы кезінде бір-ақ рет инициалданады. Келесі **m** автоматты айнымалысы функцияға кірген кез сайын, ал **p** автоматты айнымалысы цикл блогына кірген сайын инициалданады. Программа экранға келесі нәтижені шығарады:

```
n m p
0 0 0
1 1 0
2 2 0
n m p
3 0 0
4 1 0
```

Функциялар бірігіп жұмыс істеген кезде өзара ақпарат алмасуы тиіс. Ол үшін ауқымды (глобалды) айнымалыларды, параметрлерді және функцияның қайтаратын мәнін қолдануға болады.

Ауқымды айнымалылар

Ауқымды (глобалды) айнымалылар өздерімен аттас жергілікті айнымалылар сипатталмаған барлық функцияларда көрінеді, сондықтан оларды функциялар арасында мәліметтер алмасу үшін қолданған өте жеңіл. Дегенмен, бұл тәсілді пайдаланбаған жөн, себебі мұндайда программаны түзету қиындайды және функцияларды ортақ пайдаланылатын кітапханалар ішіне орналастыруға кедергі келтіреді. Функциялар барынша тәуелсіз болып, олардың интерфейсі толығымен функция прототипі арқылы анықталуына ұмтылу керек.

Қайтарылатын мән

Функциядан оны шақырған функцияға қайту механизмі келесі оператор арқылы жүзеге асырылады:

```
return [ өрнек ];
```

Функцияның құрамында бірнеше **return** операторлары болуы мүмкін (бұл алгоритм қажеттіліктерімен анықталады). Егер функция **void** ретінде сипатталса, өрнек көрсетілмейді. Егер функциядан қайту жүйелі жақшаны жабу алдында орындалса, **void** типті функция үшін **return** операторын жазу міндетті емес. **main** функциясы үшін де **return** операторын жазбауға болады. Бұл кітапта орынды үнемдеу мақсатында **main** функциясында **return** операторы көрсетілмеген, сондықтан мысалдарды компиляциялау кезінде экранға ескерту шығарылады. **return** операторынан кейін көрсетілген өрнек функцияның қайтаратын мәні типіне түрлендіріліп, функцияның шақырылу нүктесіне беріледі.

Мысалдар:

```
int f1 (){ return 1;}      // дұрыс
void f2(){ return 1;}      /* дұрыс емес, f2 мән
                                қайтармауы тиіс */
double f3(){ return 1;}  /* дұрыс, 1 double типіне
                                түрлендіріледі */
```

НАЗАР АУДАРЫҢЫЗ

Функциядан жергілікті айнымалыға нұсқауышты қайтаруға болмайды, себебі функцияға ішінде жергілікті айнымалыларға бөлінген жады функция жұмысынан соң босатылады.

Мысалы:

```
int* f(){
   int a = 5;
   return &a ;           // бұлай жазуға болмайды!
}
```

Функцияның параметрлері

Параметрлер механизі шақыратын және шақырылатын функциялар арасындағы ақпарат алмасудың негізгі жолы болып табылады. Функция сипаттамасының тақырыбында көрсетілген параметрлер формалды параметрлер немесе *жай ғана параметрлер* деп аталады, ал функцияны шақыру операторында жазылған параметрлер нақты параметрлер немесе аргументтер деп аталады.

Функцияны шақыру кезінде алдымен аргументтердің орнында тұрған өрнектер есептеледі; содан кейін стекте функцияның формалды параметрлері үшін солардың типіне сәйкес жады бөлінеді де, олардың әрқайсысына өздеріне сәйкес аргументтің мәні меншіктеледі. Мұнда типтердің бір-біріне сәйкестігі тексеріледі, егер қажеттілік туса, оларды түрлендіру орындалады. Типтердің сәйкессіздігі байқалса, диагностикалық хабарлама беріледі.

Функцияға параметрлер берудің екі тәсілі бар, олар: мәні арқылы және адресі арқылы.

Параметрлерлерді *мән арқылы беру кезінде* стекке аргументтер мәндерінің көшірмелері енгізіледі де, функциялардың операторлары осы көшірмелермен жұмыс істейді. Функция оған берілген параметрлердің бастапқы мәндеріне қол жеткізе алмайды, сондықтан оларды өзгерту мүмкіндігі де жоқ.

Параметрлерлерді *адрес арқылы беру кезінде* стекке аргументтер адрестерінің көшірмелері енгізіледі, ал функция осы адрестер бойынша жады ұяшықтарындағы аргументтерді пайдаланады да, олардың бастапқы мәндерін өзгерте алады:

```
#include <iostream.h>
void f(int i, int* j, int& k);
int main(){
   int i = 1, j = 2, k = 3;
   cout << "i j k\n";
   cout << i << ' ' << j << ' ' << k << '\n';
   f(i, &j, k);
   cout << i << ' ' << j << ' ' << k;
   return 0;
}
void f(int i, int* j, int& k){
   i++; (*j)++; k++;
}
```

Программа жұмысының нәтижесі:

```
i j k
1 2 3
1 3 4
```

Бірінші параметр (**i**) мәні арқылы беріледі. Оның функциядағы өзгерісі бастапқы мәніне әсер етпейді. Екінші параметр (**j**) нұсқауыштың көмегімен адресі арқылы беріледі, функцияға нақты параметрдің адресін беру үшін адресті алу операциясы қолданылады, ал оның мәнін алу үшін функцияда оның атын ауыстыру операциясы қажет болады. Үшінші параметр (**k**) адресі арқылы сілтеме көмегімен беріледі.

Сілтеме арқылы параметр беруде функцияға шақыру кезінде көрсетілген параметр адресі беріледі, ал функцияның ішінде параметрлерді пайдалану кезінде олардың барлығы да атаусыздандырылады. Сондықтан, нұсқауыштардың орнына сілтемелерді қолдану программаның оқылуын жақсартады, өйткені ол адрес алу және атаусыздандыру операцияларын қолдану қажеттілігінен босатады. Мәні арқылы параметр берудің орнына сілтемелерді қолданудың тиімділігі жоғары, себебі параметрлерді көшіруді талап етпейді, мұның көлемді мәліметтер құрылымдарын функцияға беру кезінде маңызы зор болып табылады.

Егер функцияның ішінде параметрді өзгертуге тыйым салу керек болса, **const** модификаторы қолданылады:

```
int f(const char*);
char* t(char* a, const int* b);
```

КЕҢЕС

Функцияның ішінде өзгеруі қарастырылмаған барлық параметрлердің алдында **const** модификаторын жазған жөн. Бұл көлемді программаларды түзетуді жеңілдетеді, себебі функцияның тақырыбы бойынша оның қандай шамалары өзгеретіндігі, қандайлары өзгермейтіндігі туралы қорытынды жасауға болады. Сонымен қатар, **const&** типті параметрдің орнына тұрақты берілуі мүмкін, ал айнымалы үшін, қажет болған жағдайда, типтерді түрлендіру операциясы орындалады.

Сонымен, функция ішінде өзгермеуі тиіс бастапқы мәліметтерді тұрақтылық сілтемелер арқылы берген жөн.

Келісім бойынша жиым мен функциядан басқа кез келген типтегі параметрлер (мысалы, нақты, құрылымдық тип, тізбе, біріктірме, нұсқауыш) функцияға мәні арқылы беріледі.

Жиымдарды параметрлер ретінде беру

Функция параметрі ретінде жиымды қолданған кезде функцияға жиымның алғашқы элементіне нұсқауыш беріледі, басқаша айтқанда, жиым әрқашанда адрес арқылы беріледі. Мұнда жиым элементтерінің саны туралы мәлімет жоғалады, сондықтан оның өлшемін жеке параметр арқылы берген жөн[1] (символдар жиымын бергенде, яғни сөз тіркестерін

[1]Егер жиымның өлшемі тұрақты болса, қиындықтар туындамайды, себебі оны формалды параметрді сипаттау кезінде де, жиымды функция ішінде өңдеу кезінде циклдер шекарасы ретінде де көрсетуге болады.

пайдаланғанда, оның нақты ұзындығын нөл-символдың орны арқылы анықтауға болады):

```
#include <iostream.h>
int sum(const int* mas, const int n);
int const n = 10;
int main(){

   int marks[n] = {3, 4, 5, 4, 4};
   cout << "Жиым элементтерінің қосындысы: "
        << sum(marks, n);
   return 0;
}
int sun(const int* mas, const int n){
   // нұсқалар: int sum(int mas[], int n)
   // немесе  int sum(int mas[n], int n)
   // (n шамасы тұрақты болу керек)
   int s = 0;
   for (int i = 0; i<n; i++) s += mas[i];
   return s;
}
```

Көпөлшемді жиымдарды беру кезінде компиляция сатысында белгісіз болған оның барлық өлшемдері параметрлер ретінде берілуі керек. Функцияның ішінде жиым бір өлшемді ретінде қабылданады, ал оның индексі программа барысында қайта есептеледі. Төменгі келтірілген мысалда функциялар көмегімен екі екіөлшемді жиым элементерінің қосындысы есептеледі. Мұнда **b** жиымының өлшемі компиляция кезеңінде белгілі, ал **a** жиымына жады динамикалық түрде бөлінеді:

```
#include <stdio.h>
#include <stdlib.h>
int sum(const int *a, const int nstr, const int nstb);
int main(){
   int b[2][2] = {{2, 2}, {4, 3}};
   printf("b элементтерінің қосындысы: %d\n",
           sum(&b[0][0], 2, 2));
   /* типтері сәйкес болмағандықтан, жиым атын sum функция-
   сына беруге болмайды */
   int i, j, nstr, nstb, *a;
   printf("Бағаналар мен жолдар санын енгізіңіз: \n");
   scanf("%d%d", &nstr, &nstb);
   a = (int *)malloc(nstr * nstb * sizeof(int));
   for (i = 0; i<nstr; i++)
      for (j = 0; j<nstb; j++) scanf("%d", &a[i * nstb + j]);
   print("a элементтерінің қомындысы: %d\n",
          sum(a, nstr, nstb));
```

```
   return 0;}
int sum(const int *a, const int nstr, const int nstb){
   int i, j, s = 0;
   for (i = 0; i<nstr; i++)
      for (j = 0; j<nstb; j++) s += a[i * nstb + j];
   return s;
}
```

Екіөлшемді жиыммен табиғи түрде жұмыс істеу үшін жады бөлудің альтернативті тәсілін қолдануға болады:

```
#include <iostream.h>
int sum(int **a, const int nstr, const int nstb);
int main(){
   int nstr, nstb;
   cin >> nstr >> nstb;
   int **a, i, j;
   // a матрицасын қалыптастыру:
   a = new int* [nstr];
   for (i =0; i<nstr; i++)
      a[i] = new int [nstb];
   for (i = 0; i<nstr; i++)
      for (j = 0; j<nstb; j++) cin >> a[i][j];
   cout << sum(a, nstr, nstb);
   return 0;
}
int sum(int **a, const int nstr, const int nstb){
   int i, j, s = 0;
   for (i = 0; i<nstr; i++)
      for (j = 0; j<nstb; j++) s += a[i][j];
   return s;
}
```

Мұнда жады екі кезеңде бөлінеді: алдымен нұсқауыштардан тұратын бағана үшін матрицаның жолдарына, содан кейін циклде әрбір жол үшін 1.10-суретте көрсетілгендей түрде беріледі. Жадының босатылуы кері реттілікпен орындалуы тиіс.

Функциялардың атауларын параметрлер ретінде беру

Функцияны оған нұсқауыш арқылы шақыруға болады. Бұл үшін сәйкес типтегі нұсқауыш жарияланады және оған адресті алу операциясының көмегімен функцияның адресі меншіктеледі:

```
void f(int a){ /*...*/ }        // функцияны анықтау
void (*pf)(int);                // функцияға нұсқауыш
...
pf = &f;  // нұсқауышқа функцияның адресі меншіктеледі
```

```
            // (pf = f деп жазуға болады;)
pf(10);     // f функциясы pf нұсқауышы арқылы шақырылады
            // ((*pf)(10) деп жазуға болады)
```

Программаның оңай оқылуы үшін функцияларға нұсқауыштарды сипаттау кезінде типтердің аттарын өзгерту (**typedef**) қолданылады. Функцияларға нұсқауыштардың жиымдарын жариялауға болады (бұл, мысалы, меню жасағанда тиімді болуы мүмкін):

```
// PF типін int типті бір параметрі бар
// функцияға нұсқауыш ретінде сипаттау:
typedef void (*PF)(int);
// Нұсқауыштар жиымын сипаттау және инициалдау;
PF menu[] = {&new, &open, &save};
menu[1](10); // open функциясын шақыру
```

Бұл жерде **new**, **open** және **save** – алдын ала жариялануы тиіс функциялардың атаулары.

Функцияларға нұсқауыштар ішкі программаға басқа типтегі параметрлер сияқты беріледі:

```
#include <iostream.h>
typedef void (*PF)(int);
void f1(PF pf){ // f1 функциясы параметр ретінде
                // PF типіндегі нұсқауышты алады
   pf(5);       // нұсқауыш арқылы берілген
                // функцияны шақыру
}
void f(int i){ cout << i; }
int main(){
   f1(f);
   return 0;
}
```

Нұсқауыштың типі мен осы нұсқауыш арқылы шақырылатын функцияның типі бір-бірімен дәлме-дәл сәйкес келуі керек.

Келісім бойынша берілген мәндері бар параметрлер

Функцияны шақыруды жеңілдету үшін оның тақырыбында параметрлердің келісім бойынша берілген мәндерін көрсетуге болады. Олар параметрлер тізімнің соңында болуы тиіс және функцияны шақыру кезінде олардың жазылуы міндетті емес. Егер шақыру кезінде параметр көрсетілмесе, одан кейін орналасқан барлық параметрлер де жазылмайды. Параметрлердің келісім бойынша берілген мәндері ретінде тұрақтылар, ауқымды айнымалылар және өрнектер қолданылуы мүмкін:

```
int f(int a, int b = 0);
void f1(int, int = 100, char* = 0);
/* мұндағы * және = таңбалары арасындағы бос орынға назар
аударыңыз (оның қатысуынсыз бұл күрделі меншіктеу операциясы
*= болып шығатын еді) */
void err(int errValue = errno);     // errno – ауқымды
                                    // айнымалы
...
// f функциясын шақыру нұсқалары:
f(100); f(a, 1);
// f1 функциясын шақыру нұсқалары:
f1(a); f1(a, 10); f1(a, 10, "Vasia");
f1(a,,"Vasia")          // бұл дұрыс емес!
```

Параметрлерінің саны айнымалы функциялар

Егер функцияның формалды параметрлерінің тізімі көп нүктемен аяқталса, оны шақырған кезде осы жерде тағы бірнеше параметр көрсетуге болатынын білдіреді. Бұл параметрлер үшін типтердің сәйкестігін тексеру орындалмайды, **char** және **short** типтері **int** ретінде, ал **float** типі **double** ретінде беріледі. Мысал ретінде **printf** функциясын келтіруге болады, оның прототипі келесі түрде беріледі:

```
int printf (const char*, ...);
```

Жоғарыдағы мысал функцияны шақыру кезінде кем дегенде бір **char*** типті параметр болуын және басқа параметрлердің болуы немесе болмауы мүмкін екендігін көрсетеді:

```
printf("Бастапқы мәліметтерді енгізіңіз");
                                    // бір параметр
printf("Қосынды: %5.2f теңге", sum);
                                    // екі параметр
printf("%d %d %d %d", a, b, c, d);// бес параметр
```

Функция ішіндегі міндетті емес параметрлерге қол жеткізу үшін **<stdarg.h>** тақырыптық файлында орналасқан **va_start**, **va_arg** және **va_ end** кітапхана макростары қолданылады. Бұл макростар 5-ші қосымшада сипатталған.

Компиляторда типтерді бақылауға қажетті ақпарат жоқ болғандықтан, параметрлерінің саны айнымалы функциялардың орнына келісім бойынша берілген параметрлерді немесе асыра жүктелген функцияларды («Функцияларды асыра жүктеу», 86 б.) пайдаланған жөн, бірақ саны айнымалы параметрлерді қолдану жақсы нәтиже беретін жағдайларды да көрсетуге болады.

Рекурсивті функциялар

Өзін өзі шақыратын функция рекурсивті деп аталады. Мұндай рекурсияны *тура* деп атайды. Екі немесе одан артық функциялар бір-бірін шақырған кезде туындайтын рекурсияны *жанама* рекурсия деп атаймыз. Егер функция өзін шақырса, қарапайым функцияны шақырған кездегі сияқты, стекте осы функция параметрлері мәндерінің көшірмесі жасалады, содан кейін басқару функцияның алғашқы атқарылушы операторына беріледі. Қайталап шақыру жағдайында бұл процесс қайталанады. Есептеулер аяқталуы үшін әрбір рекурсивті функцияда қайтару операторымен аяқталатын кем дегенде бір рекурсивті емес алгоритм тармағы болу керек. Функция аяқталған кезде стектің сәйкес бөлігі босатылады және басқару рекурсивті шақырудан кейінгі нүктеден атқарылуы жалғасатын шақырушы функцияға беріледі.

Рекурсивті функцияның классикалық мысалы ретінде факториалды есептеуді (бұл факториалды дәл осылай есептеу керек екендігін білдірмейді) қарастыруға болады. Мұнда **n** санының факториал мәнін алу үшін, (**n-1**) санының факториалын **n** санына көбейту қажет. Сонымен қатар, 0!=1 және 1!=1 болатыны белгілі.

```
long fact(long n){
   if (n==0 || n==1) return 1;
   return (n * fact(n-1));
}
```

Осы программа үзіндісін қысқартылған түрде былайша жазуға болады:

```
long fact(long n){
   return (n > 1) ? n * fact (n-1) : 1;
}
```

Рекурсивті функцияларды көп жағдайда рекурсивті алгоритмдерді ықшамды түрде жүзеге асыру үшін, сонымен қатар, рекурсивті сипатталған мәліметтердің құрылымдарымен, мысалы, екілік бұтақтармен жұмыс істеу үшін қолданады (132 б.). Кез келген рекурсивті функцияны рекурсияны қолданбай жүзеге асыруға болады, ол үшін программалаушы барлық қажетті мәліметтердің сақталуын өзі қамтамасыз ету керек. Рекурсияның артықшылығы ықшам түрде қысқаша жазылуында болып есептеледі, ал рекурсияның кемшіліктеріне функцияны қайта шақыруларға және оған параметрлер көшірмелерін беруге жұмсалатын уақыт пен жадының шығыны және, ең негізгісі, стектің толып кету қаупі жатады.

Функцияларды қолдана отырып, программаларды құру технологиясы практикумның [11] жетінші семинарында қарастырылған.

Функцияларды асыра жүктеу

Көбінесе мәліметтердің әртүрлі типтері үшін бірдей алгоритмді жүзеге асыратын функциялар атауларының бірдей болғаны ыңғайлы болады. Егер бұл атау қажетті ақпаратты беретін болса, онда программаны түсіну жеңілдейді, себебі әрбір әрекет үшін бір ғана атауды есте сақтау керек болады. Атаулары бірдей, бірақ параметрлерінің типтері әртүрлі бірнеше функцияны қолдану *функцияларды асыра жүктеу* деп аталады.

Қандай функцияны шақыру қажеттігін компилятор нақты параметрлер типіне байланысты анықтайды. Бұл процесс асыра жүктеудің рұқсатнамасы деп аталады (resolution ағылшын сөзінің «нақтылау» мағынасындағы аудармасы). Функцияның қайтаратын мәнінің типі рұқсатнамаға әсер етпейді. Рұқсатнама механизмі айтарлықтай күрделі ережелер жиынына негізделген, олардың мағынасын қысқаша түрде былай қорытындылауға болады: функцияны барынша лайықты аргументтермен қолданып, ондай мүмкіндік табылмаса, қате туындағаны туралы хабарлама шығару қажет. Ең үлкен мәнді (максимумды) анықтайтын функцияның төрт нұсқасы бар делік:

```
// Екі бүтін санның үлкенін қайтарады:
int max (int, int);
// Ең ұзақ ішкі тіркесті қайтарады:
char* max(char*, char*);
// Екінші параметр ұзындығын және бірінші параметрді
// салыстырып, үлкенін қайтарады:
int max (int, char*);
// Бірінші параметр ұзындығы мен екінші параметрді
// салыстырып, үлкенін қайтарады:
int max (char*, int);
void f(int a, int b, char* c, char* d){
   cout << max(a, b) << max(c, d) << max(a, c) << max(c, b);
}
```

Компилятор осы **max** функциясын шақырған кезде нақты параметрлер типіне сәйкес келетін функция нұсқасын таңдайды (келтірілген мысалда функцияның төрт нұсқасы бірінен соң бірі тізбекті түрде шақырылады).

Егер дәл сәйкестік табылмаса, онда жалпы ережелерге сәйкес типтерді түрлендіру орындалады (32 б. пен 3-қосымшаны қараңыз), мысалы **bool** және **char** типтері **int** типіне, **float** типі **double** типіне және т.с.с. түрлендіріледі. Осыдан кейін типтерді стандартты түрлендірулер, мысалы, **int** типін **double** типіне немесе нұсқауыштарды **void*** типіне түрлендіру орындалады. Келесі қадамда қолданушылар тағайындаған типтерді түрлендіру (бұл түрлендірулер туралы кітаптың екінші бөлімінде, 210-211 б. жазылады), сонымен қатар функциялардың аргументтер саны ай-

нымалы болғандықтан, олардың арасындағы сәйкестіктерді іздеу жүзеге асырылады. Егер бір қадамның өзінде-ақ сәйкестікті бірнеше тәсілмен анықтау мүмкін болса, шақыру тиянақсыз болып есептеледі де, қате туралы хабарлама беріледі.

Тиянақсыздық, яғни аргументтердің көрсетілген типтермен нақты түрде сәйкес келмеуі келесі жағдайларда:

- типті түрлендіру кезінде;
- сілтеме-параметрлерді қолдануда;
- келісім бойынша берілген аргументтерді пайдалану кезінде туындауы мүмкін.

Типті түрлендіру кезіндегі сәйкессіздікке мысал:

```
#include <iostream.h>
float f(float i){
   cout << "function float f(float i)" << endl;
   return i;
}
double f(double i){
   cout << "function double f(double i)" << endl;
   return i*2;
}
int main(){
   float x = 10.09;
   double y = 10.09;
   cout << f(x) << endl; // f(float) шақырылады
   cout << f(y) << endl; // f(double) шақырылады
   /* cout << f(10) << endl; бірмәнсіздік – 10 санын қалай
   түрлендіру керек: float типіне немесе double типіне ме?
   */
   return 0;
 }
```

Бұл сәйкессіздікті жою үшін 10 тұрақтысының типін айқын түрде түрлендіру керек.

Сілтеме-параметрлерді қолдану кезіндегі сәйкессіздік мысалы: егер асыра жүктелетін функциялардың бірі **int f(int a, int b)** түрінде, ал екіншісі – **int f(int a, int &b)** түрінде жарияланған болса, компилятор осы функциялардың қайсысы шақырылатынын анық біле алмайды, себебі параметрді мәні арқылы қабылдайтын функцияны шақыру мен параметрді сілтеме бойынша қабылдайтын функцияны шақыру арасында синтаксистік айырмашылық жоқ.

Келісім бойынша берілген аргументтерді қолдану кезіндегі сәйкессіздік мысалы:

```
#include <iostream.h>
int f(int a){return a;}
int f(int a, int b = 1){return a * b;}
int main(){
   cout << f(10, 2); // f(int, int) шақырылады
   /* cout << f(10);  Сәйкессіздік – қайсысы шақырылады:
   f(int, int) немесе f(int) ? */
   return 0;
}
```

Төменде асыра жүктелген функцияларды сипаттау ережелері келтірілген.

□ Асыра жүктелген функциялар *бір көріну аймағында* орналасуы тиіс, әйтпесе қабаттасқан блоктардағы айнымалалар атаулары бірдей болғандықтан, бір-бірінің мәндерін өзгертіп жібереді.

□ Асыра жүктелген функциялардың келісім бойынша берілген *параметрлері* болуы мүмкін, мұндайда бір параметрдің әртүрлі функциялардағы мәні бірдей болуы керек. Асыра жүктелген функциялардың әртүрлі нұсқаларында келісім бойынша берілген параметрлер саны әртүрлі болуы мүмкін.

□ Егер функциялар параметрлерінің сипаттамалары тек **const** модификаторымен немесе *сілтеме қолданумен* (мысалы, **int** және **const int** немесе **int** және **int&)** ерекшеленетін болса, функциялардың асыра жүктелуі мүмкін емес.

Функциялардың шаблондары

Көптеген алгоритмдер өздері жұмыс істейтін мәліметтер типтеріне тәуелді болмайды (классикалық мысал – сұрыптау). Мұндай кездерде алгоритмді мәліметтердің әртүрлі типтері үшін қолдануға болатындай етіп параметрлер енгізуге талпыну – табиғи ұмтылыс. Бұл мәселені шешудің бірінші жолы – тип туралы ақпаратты параметр ретінде беру (мысалы, функцияға бір параметр арқылы мәліметтерге нұсқауышты, екінші параметр арқылы мәліметтер элементінің ұзындығын байтпен беруге болады). Қосымша параметр қолдану қосымша код генерациялау қажеттігін білдіреді, әсіресе рекурсивті шақырулар және ішкі циклдердегі шақырулар кезінде бұл программаның тиімділігін төмендетеді; оның үстіне, типтерді бақылау мүмкіндігі де болмайды. Осының басқа шешімі ретінде мәліметтердің әртүрлі типтерімен жұмыс істеу үшін бірнеше асыра жүктелген функцияларды қолдануға болады, бірақ мұндай программада логикасы бойынша бірдей бірнеше функция пайда болады және әрбір жаңа тип үшін жаңа функция енгізу қажеттігі туады.

C++ тілінде параметрлеудің қуатты құралы – шаблондар болып табылады. Функциялардың шаблондары және кластардың шаблондары болады (кластардың шаблондары туралы 228 б. қараңыз). Функциялар шаблондарының көмегімен әртүрлі типтегі мәліметтер үшін қолданылатын алгоритмді анықтауға болады, ал белгілі бір мәліметтер типі функцияға компиляция кезінде параметр ретінде беріледі. Компилятор берілген типке сәйкес келетін дұрыс кодты автоматты түрде генерациялайды. Осылайша, автоматты түрде өзін-өзі асыра жүктейтін және параметрлеумен байланысты қосымша шығындары жоқ функция құрылады.

Қарапайым шаблон-функцияның форматы:

```
template <class Type> тақырыбы{
   /*функция тұлғасы */
}
```

Type сөзінің орнына кез келген атау қолданылуы мүмкін.

Жалпы жағдайда функция шаблонында бірнеше параметр болуы мүмкін, олардың әрқайсысы тип қана емес, қарапайым айнымалы болуы да ықтимал, мысалы:

```
template <class A, class B, int i> void f(){ ... }
```

Мысалы, таңдау әдісімен (бұл әдіс 58 б. қарастырылған) кез келген типтегі **n** элементтен тұратын жиымды сұрыптайтын функцияны шаблон ретінде төмендегідей түрде сипаттауға болады:

```
template <class Type>
   void sort_vybor(Type *b, int n){
   Type a;                // элементтерді алмастыруға
                           //арналған буферлік айнымалы
   for (int i = 0; i<n-1; i++){
      int imin = i;
      for (int j = i + 1; j<n; j++)
         if (b[j] < b[imin]) imin = j;
      a = b[i]; b[i] = b[imin]; b[imin] = a;
   }
}
```

Программаның жоғарыдағы шаблон-функцияны шақыратын негізгі функциясы келесі түрде болуы мүмкін:

```
#include <iostream.h>
template <class Type> void sort_vybor(Type *b, int n);
int main(){
   const int n = 20;
   int i, b[n];
   for (i = 0; i<n; i++) cin >> b[i];
```

```
    sort_vybor(b, n);// Бүтін сандар жиымын сұрыптау
    for (i = 0; i<n; i++) cout << b[i] << ' ';
    cout << endl;
    double a[] = {0.22, 117, -0.08, 0.21, 42.5};
    sort_vybor(a, 5);// Нақты сандар жиымын сұрыптау
    for (i = 0; i<5; i++) cout << a[i] << ' ';
    return 0;
}
```

Мәліметтердің белгілі бір типін пайдаланатын функцияны алғашқы рет шақыру кезінде компилятор функцияның соған сәйкес нұсқасы үшін код құрайды. Бұл процесс шаблонды инстанциялау деп аталады (instantiation). Инстанциялануы тиіс белгілі бір типті компилятор функцияны шақыру кезіндегі параметрлер типтеріне байланысты автоматты түрде анықтайды немесе ол айқын түрде беріледі. Дәл осы мәліметтер типімен функцияны қайталап шақыру кезінде код қайта генерацияланбайды. Тип емес айнымалы болып табылатын шаблон параметрінің орнында тұрақтылық өрнек көрсетілуі тиіс[1].

Функцияны шақыру кезінде шаблон аргументтерін айқын түрде беру мысалы:

```
template<class X, class Y, class Z> void f(Y, Z);
void g(){
    f<int, char*, double>("Vasia", 3.0);
    f<int, char*>("Vasia", 3.0); /* Z класы double
                                    ретінде анықталады */
    f<int>("Vasia", 3.0); /* Y класы char* ретінде, ал Z
    класы double ретінде анықталады */
    /* f("Vasia", 3.0); қате: X класын анықтау мүмкін
                                              емес */
}
```

Шаблон-функцияны қолданушы анықтайтын мәліметтер типіне (құрылымға немесе класқа) пайдалану үшін функцияда қолданылатын осы тип үшін анықталған операцияларды асыра жүктеу қажет болады (операциялардың асыра жүктелуі туралы 205 б. қараңыз).

Қарапайым функциялар сияқты, функциялардың шаблондары да *шаблондар көмегімен немесе қарапайым функциялар арқылы асыра жүктелуі мүмкін*.

Жеке параметрлер мен типтер үшін *функция шаблонының специализациясы* арқылы арнайы өңдеуді енгізуге болады. Бүтін сандар үшін жалпы сұрыптау алгоритмін тиімдірек түрде жүзеге асыру қажет болсын делік. Мұндайда бүтін сандармен жұмыс істейтін функция шаблонының нұсқасын өзіміз «қолдан» бере аламыз:

[1]Шаблондардың параметрлері туралы «Класс шаблондары» бөлімінде, 228 б. толығырақ айтылған.

```
void sort_vibor<int>(int *b, int n){
   ...  // Функцияның арнайы нұсқасының тұлғасы
}
```

Функция шаблоны сигнатурасының құрамында тек оның типі мен параметрлерінің типі ғана емес, шаблонның нақты аргументі де болады. Қарапайым функцияның атауы және қайтаратын мәнінің типі шаблонмен бірдей болуына қарамастан, ол ешқашанда шаблон специализациясы болып есептелмейді.

Асыра жүктеу мен функциялар шаблондарын практикумның [11] сегізінші семинарынан қараңыз.

main() функциясы

Программаны іске қосқаннан кейін басқару берілетін функцияның аты **main** болуы тиіс. Ол өзін шақырған жүйеге мән қайтара алады және сыртқы ортадан параметрлер қабылдай алады. Қайтарылатын мән бүтін типті болу керек. Стандарт бойынша функцияның екі форматы қарастырылған:

```
// параметрлерсіз:
тип main(){ /*...*/ }
// екі параметрмен:
тип main(int argc, char* argv[]){ /*... */}
```

Программаны іске қосқанда параметрлер бір-бірінен бос орын белгісі арқылы ажыратылады. Программада параметрлердің атаулары кез келген сөз болуы мүмкін, бірақ **argc** және **argv** атауларын қолдану қабылданған. Бірінші параметр **(argc)** функцияға берілетін параметрлердің санын анықтайды, осы параметрлерге программаның аты да кіреді. Екінші параметр **(argv) char*** типті нұсқауыштар жиымына нұсқауыш болып табылады. Жиымның әрбір элементінде нөл-символмен аяқталатын С-тіркес ретінде сақталған командалық жолдың жеке параметріне нұсқауыш болады. Жиымның алғашқы элементі (**argv[0]**) атқарылуға жіберілетін файлдың толық атына сілтеме жасайды, келесі элемент (**argv[1]**) бірінші параметрге, **argv[2]** екінші параметрге, т.с.с. нұсқайды. Келесі **argv[argc]** параметрі нөлге тең болуы тиіс.

Егер **main()** функциясы мән қайтармаса, оны шақырған жүйе сәтті аяқталуды білдіретін мән қабылдайды. Нөлдік емес мән программаның қателікпен, яғни апаттық аяқталуын білдіреді. **main()** функциясынан қайту операторын жазу міндетті емес.

```
#include <iostream.h>
void main(int argc, char* argv[]){
   for (int i = 0; i<argc; i++) cout << argv[i] << '\n';
}
```

Программаның орындалатын файлының аты **main.exe** болсын, сонда ол командалық жолдан төмендегідей түрде шақырылады:

```
d:\cpp\main.exe one two three
```

Экранға келесі нәтиже шығарылады:

```
D:\CPP\MAIN.EXE
one
two
three
```

Стандартты кітапхана функциялары

C++ тіліндегі кез келген программада стандартты кітапхананы пайдалану мүмкіндігі бар, кітапханада типтердің, тұрақтылардың, макростардың, функциялар мен кластардың анықтамалары орналасады. Оларды программада қолдану үшін **#include** директивасының көмегімен программаның бастапқы мәтініне сәйкес жарияланымдар орналасқан тақырыптық файлдарды қосу керек. Кітапханалық функциялардың өздері компиляцияланған түрде сақталады және олар программаға жинақтау кезеңінде қосылады. C++ тіліндегі программаларда C кітапханасынан мұраланған функцияларды қолдануға болады. Стандартты кітапхана кластарының қолданылуы кітаптың үшінші бөлімінде қарастырылады.

Кітапхана функцияларын атқаратын қызметіне байланысты топтарға бөлуге болады, олар: енгізу/шығару, сөз тіркестерін өңдеу, математикалық функциялар, динамикалық жадымен жұмыс істеу, іздеу және сұрыптау, т.с.с. Кітапхана функцияларының тізімі 6-қосымшада келтірілген. Төменде C тілі стиліндегі енгізу/шығару функцияларына, тіркестермен және символдармен жұмыс істеу функцияларына және математикалық функцияларға қысқаша шолу жасалған.

Енгізу/шығару функциялары

C++ тілінде енгізу/шығару әрекеттері C кітапханасынан мұраланған функциялар көмегімен немесе C++ тіліндегі ағымдар арқылы жүзеге асырылады. Бір программа ішінде бұл екі тәсілді қатарластыра қолдану үшін **sync_with_stdio()** функциясының көмегімен енгізуді синхронизациялау қажет. Әр тәсілдің өзіндік артықшылықтары бар. Ағымдарды қолданудың артықшылығы – оларды форматтауды қажет етпейтін қарапайым енгізу/шығару әрекеттері үшін қолдану жеңіл, сонымен қатар, ағымдық операцияларды өзіндік кластар үшін қайта анықтауға болатыны өте маңызды. C тілі стиліндегі енгізу/шығару операцияларын объектіге бағытталған техника қолданылмайтын программаларда мәліметтерді форматталған шығару кезінде пайдаланған ыңғайлы.

Сонымен қатар, программалаушылардың C тілінде жазылып, C++ тіліне ауыстырылған миллиондаған программалар қатарларын кездестірулеріне тура келеді. C++ тіліндегі ағымдар «Ағымдық кластар» бөлімінде, 283 б. қарастырылады.

C стиліндегі енгізу/шығару функцияларын қолдану үшін программада **<stdio.h>** немесе **<cstdio>** тақырыптық файлын іске қосу керек. Енгізу/шығару кезінде мәліметтер байттар ағымы ретінде қарастырылады. Физикалық тұрғыдан алғанда ағым файл немесе құрылғы (мысалы, файлдың дербес жағдайы ретінде қарастырылатын пернетақта немесе дисплей) болып табылады.

Ағымды ашу

Ағыммен жұмыс істеу оны ашудан басталады. Мәлімет оқу және/немесе жазу үшін ағымды екілік немесе мәтіндік режимде (тәртіпте) ашуға болады. Ағымды ашу функциясының форматы төмендегідей болады:

```
FILE* fopen(const char* filename, const char* mode);
```

Ағымды ашу сәтті орындалған жағдайда, функция ағыммен жұмыс істеуге қажетті барлық ақпаратты қамтитын **FILE** типті алдын ала анықталған құрылымға нұсқауышты, ал кері жағдайда **NULL** мәнін қайтарады. Бірінші параметр – C-тіркес түріндегі ашылатын файлдың аты, екіншісі – файлды ашу режимі. Ашу режимдері былай белгіленеді:

"r" – файл оқу үшін ашылады;

"w" – мәлімет жазуға арналған бос файл ашылады (мұндай файл бар болса, ол жойылады);

"a" – файл соңына ақпарат қосу үшін ашылады;

"r+" – файл оқу және жазу үшін ашылады (мұндай файл бар болуы тиіс);

"w+" – мәлімет оқуға және жазуға арналған бос файл ашылады (мұндай файл бар болса, ол жойылады);

"a+" – файлды оқып, соңына ақпарат қосу үшін ашамыз.

Файл ашу режимінің жазылуында **t** (мәтіндік режим) немесе **b** (екілік режим) символдары болуы мүмкін, олар жаңа жолға көшу символдарын өңдеуге байланысты ажыратылады. Келісім бойынша файл мәтіндік режимде ашылады, бұл режимде «каретканы қайтару» және «жолды ауыстыру» (0x13 0x10) символдарының комбинациялары мәлімет енгізу кезінде келесі жолға көшуді жүзеге асыратын жалғыз символға түрлендіріледі (мәлімет шығару кезінде кері түрлендіру орындалады). Екілік режимде бұл түрлендірулер жүзеге асырылмайды.

Мысал:

```
FILE *f = fopen ("d:\\cpp\\data", "rb+");
```

Мұндағы **f** нұсқауышы кейін кездесетін ағымдармен орындалатын операцияларда қолданылады. Олар енгізу/шығару функцияларына параметр ретінде беріледі.

Ағымды ашу кезінде онымен *буфер* деп аталатын жады аймағы байланыстырылады. Мәлімет шығару кезінде барлық ақпарат буферге жіберіледі және онда буфер толғанша немесе ағым жабылғанша сақталады. Мәлімет оқу буфер көлеміне тең блоктар арқылы орындалады және мәліметтер буферден оқылады. Буферді пайдалану сыртқы құрылғылармен айтарлықтай жылдам әрі тиімді түрде ақпарат алмасу мүмкіндігін береді. Программада оқыс жағдайлар туындап, оны апаттық аяқтау жағдайында алдыңғы буферге шығарылып жинақталған мәліметтер жоғалып кетуі мүмкін екендігін ескерген жөн. Буферлер көлемін және олардың құрылуын **setbuf** және **setvbuf** функцияларының көмегімен басқаруға болады.

Программа жұмысының басында ашылатын, алдын ала анықталған бес ағым бар, олар: **stdin** стандартты түрде мәлімет енгізу, **stdout** стандартты түрде мәлімет шығару, **stderr** қателер туралы стандартты хабарламалар шығару ағымдары, **stdaux** қосымша стандартты ағым және **stdprn** баспаға шығарудың стандартты ағымы. Келісім бойынша алғашқы үш ағым консольге қатысты болып саналады. Бұларды ағым нұсқауышы қажет болғанда, кез келген енгізу/шығару функциясында қолдануға болады.

Ағымға мәлімет енгізу/шығару

Ағымға мәлімет енгізу/шығару әрекеттерін әртүрлі тәсілдермен: байттар тізбегі түрінде, символдар мен тіркестер түрінде немесе форматтық түрлендірулерді қолдану арқылы жүзеге асыруға болады. Әрбір операция түрі үшін өзіндік функциялар жиыны анықталған. Олар 6-қосымшада, 439 б. сипатталған.

Енгізу/шығару операциялары *ағым нұсқауышының* орны арқылы анықталатын ағымның қазіргі сәттегі позициясынан бастап орындалады. Файлды ашу кезінде нұсқауыш оның басына немесе соңына (ашу режиміне байланысты) орналастырылады және әрбір енгізу/шығару операциясынан кейін автоматты түрде өзгеріп отырады. Нұсқауыштың ағымдағы орнын **ftell** және **fgetpos** функцияларының көмегімен анықтауға және **fseek**, **fsetpos** функцияларының көмегімен тікелей тағайындауға болады. Бұл функцияларды стандартты ағымдар үшін қолдануға болмайды. Төменде ағымның негізгі енгізу/шығару функциялары көрсетілген.

- Байттар ағымын оқу мен жазуды **fread** және **fwrite** функциялары орындайды.
- Ағымнан символды оқу – **getc**, **fgetc**, стандартты ағымнан символды оқу – **stdin – getchar** функциялары арқылы атқарылады.

- Символды ағымға жазу – **putc**, **fputc**, оны стандартты ағымға жазу – **stdout – putchar** функциялары арқылы.
- Тіркесті ағымнан оқу – **fgets**, оны стандартты ағымнан оқу – **stdin – gets** функциялары арқылы.
- Тіркесті ағымға жазу – **fputs**, оны стандартты ағымға жазу – **stdout – puts**.
- Форматталған түрде ағымнан енгізу – **fscanf**, стандартты ағымнан енгізу – **stdin – scanf**, тіркестен енгізу – **sscanf**.
- Форматталған түрде ағымға шығару – **fprintf**, стандартты ағымға шығару – **stdout – printf**, тіркеске шығару – **sprintf** функциялары арқылы атқарылады..

Ағымды жабу

Программа аяқталған кезде ағым автоматты түрде немесе **fclose** функциясының көмегімен тікелей жабылады:

```
int fclose(FILE*);
```

Ағымды жабу алдында онымен байланысқан буферлерде сақталып тұрған ақпарат дискіге жүктеледі. Мәліметтерді жоғалтып алмас үшін оларды сақтап қою мақсатында жазып алу үшін ашылған ағымдарды тікелей жабу ұсынылады.

Қателерді өңдеу

Ағыммен жұмыс істеуге арналған функциялардың қайтаратын мәндерін программа барысында талдап, файлдарды ашу немесе ағымнан оқу сияқты жағдайларда туындайтын қателерді өңдеу ұсынылады. Файлдармен жұмыс істеу кезінде **feof** және **ferror** функциялары жиі қолданылады:

- **int feof(FILE*)** – файл соңына жеткен кезде нөлге тең емес мәнді, кері жағдайда 0 мәнін қайтарады;
- **int ferror(FILE*)** – енгізу/шығару кезінде қателік туындаса нөлге тең емес мәнді, кері жағдайда 0 мәнін қайтарады.

Ағымдармен жұмыс істеу мысалы

Файлда мониторлар туралы ақпарат сақталған болсын делік. Мәліметтердің әрбір жолында олардың типі, көтерме және бөлшек сауда бағалары және ескерту жазылған. Жеңіл болу үшін әрбір жолда мәліметтер бірдей сипатта берілген: алғашқы 20 символ монитор типіне, содан кейін көтерме және бөлшек сауда бағасын білдіретін бүтін сандардың әрқайсысына 5 символдан, соңынан ескертуге 40 символдан артпайтын орын бөлінген.

Төменде берілген программа мәтіндік файлдан мәліметтерді s буферлік айнымалысына жеке жолдар бойынша көшіреді, содан кейін олардан **mon**

құрылымын қалыптастырып, оны екілік режимде нәтижелік (шығыс) файлға жазады. Төменде осы файлдан кез келген жазбаны оқу мысалы көрсетілген.

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(){
   FILE *fi, *fo;
   if((fi = fopen("d:\\c\\file.txt", "r"))==0){
      cout << "Кіріс файлды ашуда қате"; return 1;}
   if((fo = fopen("d:\\c\\binfile.out", "w+b"))==0){
      cout << "Шығыс файлды ашуда қате"; return 1;}
   const int dl = 80;
   char  s[dl];
   struct {
      char type[20];
      int  opt, rozn;
      char comm [40];
   } mon;
   int kol = 0;             // Файлдағы жазбалар саны
   while (fgets(s, dl, fi)){
      //Тіркесті құрылымға түрлендіру:
      strncpy (mon.type, s, 19);
      // strncpy сипаттамасын 6-қосымшадан қараңыз
      mon.type[19]='\0';
      mon.opt = atoi(&s[20]);
      // atoi сипаттамасын келесі бөлімнен қараңыз
      mon.rozn = atoi(&s[25]);
      strncpy(mon.comm, &s[30], 40);
      fwrite(&mon, sizeof mon, 1, fo);
      kol++;
   }
   fclose(fi);
   int i; cin >> i;        // Жазба нөмірі
   if (i >= kol){ cout << "Мұндай жазба жоқ";
   return 1;}
   /* Файлдың ағымдағы позициясы нұсқауышын i жазбасына
      орнату:      */
   fseek(fo, (sizeof mon)*i, SEEK_SET);
   fread(&mon, sizeof mon, 1, fo);
   cout << "mon.type "<< mon.type << " opt " << mon.opt
        << " rozn " << mon.rozn << endl;
```

```
    fclose(fo);
    return 0;
}
```

Тіркестермен және символдармен жұмыс істеу функциялары

Тіркес нөл-символмен аяқталатын символдар жиымы болып табылады. C++ тілінде тіркестермен жұмыс істеудің екі мүмкіндігі: C кітапханасынан мұраланған функциялар (**<string.h>** немесе **<cstring>** тақырыптық файлы) және тіркестерді бейнелеудің, өңдеудің, бақылаудың кеңірек мүмкіндіктерін қамтамасыз ететін C++ тілінің **string** кітапханалық класы (ол «Тіркестер» бөлімінде, 307 б. қарастырылады) бар.

C кітапханасында тіркестерді көшіру (**strcpy, strncpy**), салыстыру (**strcmp, strncmp**), біріктіру (**strcat, strncat**), ішкі тіркесті іздеу (**strstr**), ішкі символды іздеу (**strchr, strrchr, strpbrk**), тіркес ұзындығын анықтау (**strlen**) және басқа да функциялар бар. Функциялардың толық тізімі 6-қосымшада берілген.

Тіркестерді сандарға түрлендіру функциялары **<stdlib.h>** және **<cstdlib>** тақырыптық файлдарында сақталған (кері түрлендірулерді **sprintf** функциясының көмегімен орындауға болады):

- **double atof (const char* p)** – берілген тіркесті **double** типіне түрлендіреді;
- **int atoi (const char* p)** – берілген тіркесті **int** типіне түрлендіреді;
- **long atol (const char* p)** – берілген тіркесті **long** типіне түрлендіреді.

Тіркес басындағы бос орын және табуляция белгілері ескерілмейді. Түрлендіру операциясы мағынасы болмайтын символ коды кездескенде немесе тіркес соңына жеткенде тоқтатылады. Егер тіркесті санға түрлендіру мүмкін болмаса, 0 мәні қайтарылады. Ал егер сан берілген тип диапазонының шекарасынан шығып кетсе, **errno** айнымалысына **(<cerrno>** тақырыптық файлы) **ERANGE** мәні меншіктеледі және мүмкін болатын сан қайтарылады.

Мысал (программа тіркестегі **double** типті жиымды толтырады):

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main (){
   char s[] = "2, 38.5, 70, 0, 0, 1", *p = s;
   double m[10];
   int i = 0;
   do{
      m[i++] = atof(p);
      if (i >9) break;
```

```
    } while(p = strchr(p, '.'), p++);
    for(int k = 0; k < i; k++) printf("%5.2f ", m[k]);
    return 0;
}
```

Стандартты кітапханада символдармен жұмыс істеуге арналған (**<ctype.h>** және **<cctype>** тақырыптық файлдары) келесі функциялар бар:

Аты	Символдың жиынға тиістілігін тексеру
isalnum	әріптер мен цифрлар (A–Z, a–z, 0–9)
isalfa	әріптер (A–Z, a–z)
iscntrl	басқарушы символдар (0..31 және 127 кодтарымен)
isdigit	цифрлар (0–9)
isgraph	бос орын белгісінен басқа экранда бейнеленетін символ дар **(isalfa | isdigit | ispunct)**
islower	төменгі регистр әріптері (a–z)
isprint	экранда бейнеленетін символдар
ispunct	пунктуация белгілері
isspace	ажыратқыш символдар
isupper	жоғарғы регистр әріптері (A–Z)
isxdigit	он алтылық цифрлар (A–F, a–f, 0–9)

Функциялар **int** типіндегі шамаларды қабылдайды және шарт орындалса, **true** мәнін қайтарады. Өзіндік тексеру циклдерін жазудың орнына стандартты функцияларды қолдану ұсынылады, бұл программадағы қателер санын төмендетеді.

Жоғарыда аталғандардан басқа, кітапханада латын алфавитінің символын сәйкесінше төменгі және жоғарғы регистрге ауыстыратын **tolower** және **toupper** функциялары бар.

Аталған функциялардың әрқайсысы үшін **wchar_t** типіндегі көпбайттық символдарға арналған, атауында **w** әрпі болатын функциялар аналогы анықталған.

Математикалық функциялар

C++ тілі C тілінің стандартты математикалық функцияларын мұралаған, олардың сипаттамалары **<math.h>** (**<cmath>**) тақырыптық файлдарында орналасқан. Олар санның абсолюттік мәнін (**abs, fabs**), дөңгелектенген санды (**ceil, floor**), санның квадрат түбірін (**sqrt**), дәрежесін (**pow**), тригонометриялық функциялардың мәндерін (**sin, cos, tan, sinh, cosh, tanh, asin, acos, atan, atan2**), экспонентаны (**exp**), логарифмді (**log, log10**), санның бүтін және бөлшек бөліктерін (**modf**), бөлуден қалған қалдықты (**fmod**) және т.б. есептеуге мүмкіндік береді.

Препроцессор директивалары

Компилятордың алғашқы іс-әрекеттерді орындау кезеңі (фазасы) препроцессор деп аталады. Препроцессор нұсқаулары директивалар деп аталады. Олар **#** символынан басталуы тиіс, бұл символдың алдында тек бос орын символдары орналасуы мүмкін (10 б.).

#include директивасы

#include <файл_аты> директивасы көрсетілген файлдың құрамындағы ішкі мәліметтерді бастапқы файлдағы өзі жазылған орынға кірістіреді. Қосылатын файлдың құрамында да **#include** директивалары болуы мүмкін. Егер файлдың орналасқан каталогының жолы толық көрсетілмесе, ол кіріктірілетін файлдардың стандартты каталогтарынан ізделеді. Бұрыштық жақшалардың орнына тырнақшаларды (**" "**) қолдануға болады – мұндайда файлды іздеу бастапқы файл орналасқан каталогта, содан кейін барып стандартты каталогта жүргізіледі.

#include директивасы әртүрлі файлдардағы жарияланымдардың үйлесімділігін қамтамасыз етудің қарапайым құралы болып табылады, бұл директива файлдарға тақырыптық файлдардың интерфейсі туралы ақпарат береді.

Тақырыптық файлдардың кеңейтілуі әдетте **.h** болады және олардың құрамында келесі элементтер болуы мүмкін:

- типтердің, тұрақтылардың, құрамдас функциялардың, шаблондар мен тізбелердің анықтамалары;
- функциялардың, мәліметтердің, атаулар мен шаблондардың жарияланымдары;
- атаулар кеңістіктері;
- препроцессор директивалары;
- түсініктемелер.

Тақырыптық файлда функциялар мен мәліметтердің анықтамалары болмауы тиіс. Мұндай ережелер тілдің талабы бойынша емес, олар директиваны қолданудың ыңғайлы тәсілін бейнелейді.

Стандартты кітапхананың тақырыптық файлдарын жазу кезінде **.h** кеңейтілуін жазу міндетті емес[1]. Бұл оларды сақтау тәсілдеріне шек қоймау мақсатында жасалған. C кітапханасының әрбір **<name.h>** атты файлы үшін C++ кітапханасында соған сәйкес **<cname>** файлы бар, бұл файлда C кітапханасының құралдары **std** атаулар кеңістігінде сипатталады. Мысалы, **#include <cstdio>** директивасы **#include <stdio.h>** директивасымен бірдей қызмет атқарады, бірақ стандартты функцияларды пайдалану кезінде **std** атаулар кеңістігінің атын көрсету талап етіледі («Атау берілген аймақтар» бөлімін, 106 б. қараңыздар).

[1]Компиляторлардың ескі нұсқалары стандарттың бұл жаңа талабын қанағаттандырмауы мүмкін.

#define директивасы

#define директивасы программа мәтініне алмастыру арқылы енгізілетін қосымша мәліметтерді анықтайды. Ол төмендегі элементтерді анықтау үшін қолданылады:

□ *символдық тұрақтыларды:*

```
#define атауы алмастыру_мәтіні
```

(атаудың барлық нұсқалары алмастыру мәтініне ауыстырылады);

□ *макростарды*, олардың түрі функцияларға ұқсас болады, бірақ олар программадағы мәтінін өздерінің мәтініне алмастыру арқылы жүзеге асырылады:

```
#define атауы( параметрлер ) алмастыру_мәтіні
```

□ *шартты компиляцияны басқаратын символдарды,* олар **#ifdef** және **#ifndef** директиваларымен бірге қолданылады. Форматы:

```
#define атауы
```

Мысалдар

```
#define VERSION 1
#define  VASIA "Василий Иванович"
#define MAX(x,y) ((x)>(y)?(x):(y))
#define MUX
```

Атауларды айнымалылар мен функциялар аттарынан ажырату үшін бас әріптермен жазу ұсынылады. Макрос параметрлері макроалмастыру кезінде қолданылады, мысалы, егер программа мәтінінде `y = MAX(suml, sum2);` макросын шақыру орындалса, ол `y = ((suml)>(sum2)?(suml):(sum2));` өрнегіне алмастырылады. Дөңгелек жақшалардың жазылмауы есептеу реттілігіндегі қателіктерге алып келуі мүмкін, себебі препроцессор кірістірілетін мәтінді синтаксис тұрғысынан бағаламайды. Мысалы, егер `#define sqr(x) (x*x)` макросын `sqr(y+l)` өрнегі арқылы пайдаланатын болсақ, алмастыру нәтижесінде `(y+1*y+1)` өрнегі есептеледі.

Макростар мен символдық тұрақтылар С тілінен мұраланған, С++ тілінде программа жазу кезінде оларды қолданбаған жөн. Символдық тұрақтылар орнына **const** немесе **enum**, ал макростар орнына құрамдас функцияларды немесе шаблондарды қолдану дұрысырақ болып саналады.

Шартты компиляция директивалары

Шартты компиляция директивалары – **#if**, **#ifdef** және **#ifndef** – программаның кейбір жеке бөліктерін компиляциядан алып тастау үшін қолданылады. Бұл түзету кезінде немесе, мысалы, әртүрлі платформалар үшін программаның әртүрлі нұсқаларын сүйемелдеу керек болған кездерде пайдалы болады.

#if директивасының форматы:

```
#if тұрақтылық_өрнек
…
[ #elif тұрақтылық_өрнек
… ]
[ #elif тұрақтылық_өрнек
… ]
[ #else
… ]
#endif
```

#elif директиваларының саны бірнешеу бола береді. Компиляциядан шығарылатын код блоктарында сипаттамалар да, атқарылатын операторлар да болуы мүмкін. Тақырыптық файлдың әртүрлі нұсқаларын шартты енгізу мысалы:

```
#if VERSION == 1
   #define INCFILE "vers1.h"
#elif VERSION == 2
   #define INCFILE "vers2.h" /* тағы сол сияқты */
#else
   #define INCFILE "versN.h"
#endif
#include INCFILE
```

Тұрақтылық өрнектерде **defined(тұрақтының_аты)** операторы арқылы тұрақтының анықталуын тексеру қолданылуы мүмкін, мысалы:

```
#if defined(_BORLANDC_) && _BORLANDC_== 0x530  //BC5.3:
typedef istream_iterator<int, char, char_traits<char>,
   ptrdiff_t> istream_iter;
#elif defined(_BORLAND_)                          // BC5.2:
typedef istream_iterator<int, ptrdiff_t> istream_iter;
#else                                              // VC5.0:
typedef istream_iterator<int> istream_iter;
#endif
```

Директиваның басқа бір атқаратын қызметі – код үзінділерін уақытша түсініктемеге (комментарийге) айналдыру, мысалы:

```
#if 0
   int i, j;
   double x, y;
#endif
```

Директивалардың қабаттасуы рұқсат етілгендіктен, мұндай тәсіл өте ыңғайлы болып келеді.

Программада **#ifdef** және **#ifndef** директивалары барынша жиі қолданылады, олардың құрамында көрсетілген символ **#define** директивасының көмегімен анықталғанына (немесе анықталмағанына) байланысты (ең болмағанда бос символ ретінде мысалы,

```
#define 32_BIT_SUPPORT)
```

жолы компиляцияны басқаруға мүмкіндік береді:

```
#ifdef символ
/* Символ анықталған болса, төменде орналасқан код
компиляцияға жіберіледі */
#ifndef символ
/* Символ анықталмаған болса, төменде орналасқан код
компиляцияға жіберіледі */
```

Бұл директивалар келесі **#elif, #else** немесе **#endif** сөздері кездескенге дейін әсер етеді.

#ifndef директивасы тақырыптық файлды тек бір рет енгізуді қамтамасыз ету үшін жиі қолданылады:

```
#ifndef HEADER_INCLUDED
#include "myheader.h"
#define HEADER_INCLUDED
#endif
```

#undef директивасы

#undef атауы директивасы символдың анықтамасын жояды. Ол сирек қолданылады, мысалы, компилятордың қандай да бір қызметін болдырмау үшін пайдаланылады.

Алдын ала анықталған макростар

C++ тілінде, программаның нұсқасын көрсетуге немесе қатенің пайда болған жері туралы ақпарат беруге арналған бірнеше макрос анықталған.

Егер программа C++ тілінің файлы ретінде компиляцияланса, онда **_cplusplus** анықталған болып табылады. Көптеген ксмпиляторлар **.c** кеңейтілуі бар файлдарды өңдеу кезінде оларды C тілінде жазылған программа деп есептейді. Осы макросты қолдану C++ тілінің мүмкіндіктерін пайдалануға болатындығын көрсетеді:

```
#ifdef _cplusplus
// C++ тіліне тән әрекеттер
#endif
```

Мұндай макростар программалық кодты С тілінен С++ тіліне немесе кері бағытта ауыстыру кездерінде қолданылады.

DATE - ағымдағы мерзімді *ай күн жыл* форматында сақтайтын тіркесті береді, мысалы:

```
printf(" Компиляция мерзімі – %s \n",   _DATE_);
```

FILE- ағымдағы файлдың толық аты сақталған тіркесті береді.

LINE- бастапқы мәтіннің ағымдағы жолы.

_TIME _- ағымдағы уақыт, мысалы:

```
printf("%s файлында қате бар \n Компиляция уақыты:
%s\n", _FILE_, _TIME_);
```

Идентификаторлардың әрекет ету аймақтары

Әрбір программалық объектінің түрі мен жариялану орнына байланысты өзіндік әрекет ету аймығы болады. Келесідей әрекет ету аймақтары болуы мүмкін: блок, файл, функция, функцияның прототипі, класс және атау берілген аймақ.

Блок. Блок ішінде сипатталған идентификаторлар жергілікті (локалды) болып табылады. Идентификатордың әрекет ету аймағы оның анықталу нүктесінен басталып, блок соңында аяқталады, көріну аймағы – блок және оның ішкі блоктары, өмірлік кезеңі – осы блоктан шыққанға дейінгі мерзім болып саналады. Блоктан шыққаннан кейін жады босатылады.

Файл. Кез келген блоктан, функциядан, кластан немесе атаулар кеңістігінен тыс сипатталған идентификаторлардың көріну аймағы ауқымды (глобалды) болып саналады, өмірлік кезеңі тұрақты болады және олар анықталу уақытынан бастап қолданылуы мүмкін.

Функция. Әрекет ету аймағы функцияның іші ғана болып табылатын идентификатор – ол оператор белгілері (label) ғана болып табылады. Бір функция құрамындағы белгілер әртүрлі болуы тиіс, бірақ олар басқа функциялар ішіндегі белгілермен сәйкес келуі мүмкін.

Функцияның прототипі. Функция прототипінің (жарияланымының) параметрлер тізімінде көрсетілген идентификаторлардың әрекет ету аймағы тек функция прототипі болады.

Класс. Құрылымдардың, біріктірмелердің және класс элементтерінің (статикалық элементтерді қоспағанда) көріну аймағы тек класс іші ғана болып саналады. Олар көрсетілген типтегі айнымалыны құру кезінде қалыптасып, айнымалыны жойғанда жоғалады.

Атау берілген аймақ. С++ тілі атаулардың анықталу аймағын ауқымды аймақтың бір бөлігі ретінде **namespace** операторының көмегімен тікелей бекітуге мүмкіндік береді.

Сыртқы блокта анықталып, программаның ішкі блогында аты қайталанатын айнымалыны анықтаудан басқа барлық жағдайларда объектінің көріну аймағы әрекет ету аймағымен бірдей болатынын еске салайық. Мұндайда атауы қайталанатын сыртқы айнымалының әрекет ету аймағы ішкі блокқа енгенімен, сыртқы айнымалы бұл блокта көрінбейді. Осыған қарамастан, сыртқы айнымалы ауқымды (глобалды) болса, оны көріну аймағына ену операциясын `::` қолдана отырып ішкі блокта пайдалануға болады. Жасырын жергілікті (локалды) айнымалыны пайдалану тәсілі жоқ.

Әрбір әрекет ету аймағында атаулар кеңістіктерін айқын көрсетуге болады. Атаулар кеңістігі – идентификатор қайталанбай бірегей болуы тиіс аймақ. Әртүрлі кеңістіктердегі атаулар бірдей болуы мүмкін, себебі сілтемелерді таңдау идентификатордың программадағы контекстіне байланысты жүзеге асырылады, мысалы:

```
struct Node{
   int Node;
   int i;
}Node;
```

Мұнда ешқандай қайшылық жоқ, себебі типтің, айнымалының және құрылым элементінің атаулары әртүрлі кеңістіктерге жатады. C++ тілінде құрамындағы атаулар қайталанбауы тиіс төрт түрлі идентификаторлар класы анықталған:

- Бір атаулар кеңістігіне айнымалылардың, функциялардың, қолданушылар анықтаған типтердің (**`typedef`**) және тізбелер тұрақтыларының (65 б. қараңыз) бір көріну аймағындағы атаулары жатады. Функция атауларынан басқасы кіріктірілген ішкі блоктарда қайта анықталуы мүмкін.
- Тізбелер типтерінің, құрылымдардың, кластардың және біріктірмелердің атаулары басқа бір атаулар кеңістігін құрайды. Бір көріну аймағындағы әрбір атау басқа типтер атауларынан өзгеше болуы керек.
- Әрбір құрылым, класс және біріктірме элементтері жеке кеңістікті құрайды. Элемент атауы құрылым ішінде қайталанбауы тиіс, бірақ олар басқа құрылымдар элементтерінің атауларымен бірдей болуы мүмкін.
- Белгілер (label) жеке атаулар кеңістігін құрайды.

Сыртқы жарияланымдар

Кез келген функция автоматты түрде программаның барлық модульдерінде көрінеді. Егер функцияның әрекет ету аймағын өзі сипатталған файлмен шектеу қажет болса, **`static`** модификаторы қолданылады.

Айнымалыны немесе тұрақтыны бірнеше модульде қолданып, қолжетерлік ету үшін келесі әрекеттерді орындау қажет:

- оны тек бір модульде ғана ауқымды (глобалды) объект ретінде анықтау;
- басқа модульдерде оны **extern** модификаторының көмегімен сыртқы объект ретінде жариялау.

Осы мақсатқа жетудің басқа жолы – бұл жарияланымды тақырыптық файлға орналастырып, оны қажетті модульдерге қосу[1].

Бір айнымалының барлық сипаттамалары бір-бірімен үйлесімді болуы керек.

Екі ауқымды (глобалды) айнымалыны **one.cpp** және **two.cpp** файлдарында **my_header.h** тақырыптық файлының көмегімен сипаттау мысалы:

```
// my_header.h – сыртқы жарияланымдар
extern int a;
extern double b;
...
// ---------------------------------
// one.cpp
#include "my_header.h"
int a;
...
// ---------------------------------
// two.cpp
#include "my_header.h"
double b;
...
```

Екі айнымалыны да **one.cpp** және **two.cpp** файлдарында қолдануға болады.

Егер айнымалы **static** ретінде сипатталса, оның әрекет ету аймағы өзі анықталған файлмен шектеледі.

Типті сипаттау кезінде *бір анықтама* ережесін ұстанған жөн, яғни программада қолданылатын тип дәл бір рет анықталуы керек. Көбінесе бұл тақырыптық файлда орындалады, содан кейін файл осы типті қолданатын модульдерге қосылады. Бұл ереженің сақталмауы оңайлықпен табыла қоймайтын қателіктердің пайда болуына алып келеді, себебі компиляторлар бір типтің әртүрлі файлдардағы анықталуын бір-бірінен ажырата алмайды.

---

[1]Жариялау кезінде, анықтаумен салыстырғанда, айнымалы құрылмайтынын еске алайық. **extern** модификаторымен жариялау кезінде инициалдау болмауы тиіс: егер ол кездессе, онда **extern** модификаторы ескерілмейді.

Атау берілген аймақтар

Атау берілген аймақтар жарияланымдарды логикалық түрде топтау және оларды пайдалану мүмкіндігін шектеу үшін қолданылады. Программа неғұрлым көлемді болған сайын, атау берілген аймақтарды қолдану да соғұрлым өзекті болады. Оны қолданудың қарапайым мысалы ретінде бір адамның жазған кодын басқа адамның жазған кодынан ажыратуды келтіруге болады. Жалғыз ауқымды көріну аймағын қолданған жағдайда жеке бөліктерден программа қалыптастыру атаулар сәйкестігі мен қайшылығына байланысты күрделі жұмыс болып табылады. Атау берілген аймақтарды қолдану қажет емес құралдарды пайдалану кезінде жұмысқа кедергі болады.

Атау берілген аймақты (оны атаулар кеңістігі деп те атайды) жариялау форматы:

```
namespace [ аймақ_атауы ] { /* Жарияланымдар */
```

Атау берілген аймақ бірнеше рет жариялануы мүмкін, мұндайда келесі жарияланымдар алдыңғыларының кеңейтілуі ретінде қарастырылады. Осылайша, атау берілген аймақты бір файл шекарасынан тыс жариялауға және өзгертуге болады.

Егер аймақ атауы берілмесе, компилятор оны әрбір модуль үшін әртүрлі болып келетін, қайталанбайтын бірегей идентификатор арқылы анықтайды. Объектіні атау берілмеген аймақта жариялау оны **static** модификаторының көмегімен ауқымды объект ретінде сипаттаумен бірдей болып саналады. Жарияланымдарды осындай аймаққа орналастыру код жергіліктілігін сақтау үшін тиімді болады. Бір файл ішінен басқа файлдың атау берілген аймағының элементіне қол жеткізу мүмкін емес.

Мысал.

```
namespace demo{
   int i = 1;
   int k = 0;
   void func1(int);
   void func2(int) { /* ... */ }
}
namespace demo{          // Кеңейту
   // int i = 2;      Дұрыс емес – екі рет анықталған
   void func1(double); // Асыра жүктеу
   void func2(int);    // Дұрыс (қайта жариялау)
}
```

Атау берілген аймақты жариялауда жарияланымдармен қатар анықтамалар да болуы мүмкін. Оның құрамына тек жарияланымдарды орналастырып, кейіннен аймақ атауы мен көріну аймағына қол жеткізуді `::` операторының көмегімен анықтаудың да өз логикасы бар, мысалы:

```
void demo::func1(int) { /* ... */ }
```

Бұл интерфейс пен оны жүзеге асыруды бір-бірінен бөлу үшін қолданылады. Мұндай тәсілмен атаулар кеңістігінің жаңа элементін жариялауға болмайды.

Аймақ ішінде жарияланған объектілер жариялау сәтінен бастап көрінетін болады. Аймақ атауы мен көріну аймағына қол жеткізу `::` операторының көмегімен оларды тікелей пайдалануға болады, мысалы:

```
demo::i = 100; demo::func2(10);
```

Егер атау өз кеңістігінен тысқары аймақта жиі қолданылатын болса, оны **using** операторының көмегімен қолжетерлік етіп жариялауға болады:

```
using demo::i;
```

Осыдан кейін аймақты тікелей көрсетпей-ақ, атауды қолдануға болады.

Егер кез келген бір аймақтағы барлық атауларға қол жеткізу керек болса, онда **using namespace** операторы қолданылады:

```
using namespace demo;
```

Басқа аймақтың жарияланымдарына қол жеткізу үшін **using** және **using namespace** операторларын атау берілген аймақ жарияланымының ішінде қолдануға болады:

```
namespace Department_of_Applied_Mathematics{
   using demo::i;
   // ...
}
```

Атау берілген аймақта тікелей немесе **using** операторының көмегімен жарияланған атаулар **using namespace** операторының көмегімен жарияланған атауларға қарағанда басымырақ болып саналады (бұл бірдей атаулары бар бірнеше атау берілген аймақтарды қосу кезінде маңызды).

Атаулар кеңістіктерінің қысқа аттары бір-бірімен қайшылыққа келуі мүмкін, ал ұзақ атаулар программа жазуда ыңғайсыз, сондықтан атаулардың синонимдерін енгізуге рұқсат берілген:

```
namespace DAM = Department_of_Applied_Mathematics;
```

Стандартты кітапхананың атаулар кеңістіктері. Стандартты кітапхана объектілері **std** атаулар кеңістігінде анықталған. Мысалы, С тілінің стандартты енгізу/шығару құралдарының **<stdio.h>** тақырыптық файлындағы жарияланымдары атаулар кеңістігіне төмендегідей түрде орналастырылған:

```
// stdio.h
namespace std{
   int feof(FILE *f);
   ...
}
using namespace std;
```

Бұл жоғарыдан төменге қарай үйлесімділікті қамтамасыз етеді. Тікелей қол жеткізуге болмайтын атаулардың болғанын қаламайтындар үшін жаңа **<cstdio>** тақырыптық файлы анықталған:

```
// cstdio
namespace std{
int feof(FILE *f);
...
}
```

Егер программаға **<cstdio>** файлы қосылған болса, атаулар кеңістігінің атын айқын түрде көрсету керек:

```
std::feof(j);
```

Атаулар кеңістіктерінің механизмі **#include** директивасымен біріге отырып, байланысқан шамаларды логикалық топтастыру мен олармен қатынасуды шектеуді үйлестіру арқылы көлемді программалар жазу кезінде қажетті икемділікті қамтамасыз етеді.

Көбінесе программаның кез келген функционалдық тұрғыдан аяқталған үзіндісінде осы үзіндіні қолдану үшін қажетті интерфейстік бөлікті (мысалы, функциялардың тақырыптары, типтердің сипаттамалары) және оны жүзеге асыру бөлігін, яғни қосалқы айнымалыларды, функцияларды және сырттан қол жеткізуге болмайтын басқа да құралдар бөлігін жекелеп көрсетуге болады. Атаулар кеңістіктері жүзеге асыру ерекшеліктерін көрсетпеуге, яғни программа құрылымын жеңілдетуге және қателер санын азайтуға мүмкіндік береді. Программаны модульдерге жоспарлы түрде бөлу, интерфейстерді айқын түрде спецификациялау және оларға қол жеткізуді шектеу программалаушылар тобының жоба бойынша тиімді жұмыс істеуіне мүмкіндік береді.

3-ТАРАУ

Программаларды құру технологиясы

Біз C++ тілінің бірсыпыра құралдарын қарастырдық (олардың басым бөлігі C тілінің ішкі жиынына жатады). Енді программа жазу нәтижесінде көркем әрі сенімді конструкцияға қол жеткізу үшін басшылыққа алынуы тиіс қағидаларды талқылайтын мезгіл келді, әрине ол тиіп кетсең, сынып кететін күркешік тәрізді болмауы тиіс.

Бұл тарауда үш бөлім қарастырылған. Бірінші бөлімде ең қарапайым программа құруда пайдаланылатын ұсыныстар көрсетілген. Олар, негізінен, C++ тілінде алгоритмдер жазуға және программаларды құжаттауға қатысты материалдар. Көлемді программалар құру және оларды тесттен өткізу технологиясы екінші бөлімде қарастырылған (117 б.). Үшінші бөлім мәліметтерді өңдеудің тиімді процедураларын құру үшін кең қолданылатын мәліметтердің динамикалық құрылымдарын карастыруға арналған.

Программаны кодтау және құжаттау

Бірсыпыра тәжірибе жинақтағаннан кейін программалаушы өз ережелері мен стилін қалыптастырады. Бұл үшін тек кездестірген сәтсіздіктеріңнен ғана сабақ алмай, төменде көрсетілген ұсыныстарға да саналы түрде жүгіну көптеген кең таралған қателіктерді болдырмауға көмектеседі. Программалаудың өнерге жатқызылуы (әсіресе дамуының бастапқы кезеңінде) да себепсіз емес қой, сондықтан, барлық туындайтын жағдайларға арналған кеңестер беру мүмкін емес.

Біздің қол жеткізетін негізгі мақсатымыз – жеңіл оқылатын, мүмкіндігінше қарапайым құрылымды программа жазу. Көбінесе дұрыс нәтиже беретін “Better simpler than clever” («ақылға салып созғаннан, оңайлатып озған жақсы») ұстанымы көпшілікке әйгілі. Барлық программалау технологиялары да осы мақсатқа жетуге бағытталған, өйткені тек осындай жолмен жеңіл әрі сенімді түрде өзгертілетін программа[1] жазуға болады. Жоғарыдағы ұстанымның мағынасын былайша түсіндіруге болады: егер кез келген бір әрекетті әртүрлі тәсілдермен программалуға болатын болса, онда олардың ішінен ең ықшамды немесе ең тиімдісін емес, оқуға, түсінуге жеңіл болатын жолын таңдап алу керек. Бұл, әсіресе, программаны біреулер жазып, ал оны жұмыс барысында пайдаланып сүйемелдеу басқаларға жүктелетін жағдайларда өте маңызды болып табылады.

[1]Басынан бастап программадағы өзгерістер туралы ойлау қажеттігі түсініксіз болуы мүмкін. Алайда кез келген сәл де болса сәтті шығарылған коммерциялық өнім үшін жаңа нұсқаларды шығару – қолданушылар арасында өзектілікті сақтаудың жалғыз тәсілі.

Егер сіз жоғарыда көрсетілген тезисті әлі де жеткілікті түрде мойындамайтын болсаңыз, онда төменде көрсетілген программаны талдап көріңіз. Бір қарағанда[1] программа түсініксіз болып көрінгенмен, ол C тілінде жазылған және нәтижелі түрде жұмыс істейді:

```
#include <stdio.h>
main (int t,  int _, char *a) {return!0<t?t<3?main(-
79,-13,a+main(-87,1-_,main(-86,0,a+1)+a)): 1,t<_ ?
main(t+1,_,a):3,main(-94,-27+t,a)&&t==2\
?_<13?main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-72?
main(_,t,"@n'+,#'/*s{}w+/w#cdnr/+,{}r/*de}+,/*{*+,
/w{%+,/w#q#n+,/#{1+,/n\{n+,/+#n+,/# ;#q#n+,/+k#;*+,/'r
:'d*'3,}{w+K w'K:'+}e#';dq#'l q#'+d'K#!\
/+k#;q#'r} eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){)
{nl]'/+#n' ;d} rw' i;# ){nl]!\ /n{n#'; r{#w'r
nc{nl]'/#{l,+'K {rw'
iK{;[{nl]'/w#q#n'wk nw' iwk{KK{nl]!/\ w{%'l##w#' i;
:{nl]'/*{q#'ld;r'}{nlwb!/*de}'c ;;{nl'-{}
rw]'/+,}##'*}\ #nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ')
}+}{rl#'{n' ')# }'+}#(!!/") :t<- 50?_==*a?putchar(31[a]):
main(-65,_,a+1):
main((*a=='/')+t,_,a+1):
0<t?main(2,2,"%s")
:*a=='/'||main(0,main(-61,*a,"!ek;dc i@bK'(q)-[w]*%n
+r3#l,{}:\nuwloca-0;m.vpbks.fxntdCeghiry" ),a+1);}
```

Осы мысалды пернетақтада теруге, компиляциялауға және орындауға ерінгендер үшін одан шығатын мәліметтер үзіндісі келтірілген:

```
On the first day of Christmas my true love gave to me
a partridge in a pear tree.

On the second day of Christmas my true love gave to me
two turtle doves
and a partridge in a pear tree.

On the third day of Christmas my true love gave to me
three french hens, two turtle doves
and a partridge in a pear tree.
...
On the twelfth day of Christmas my true love gave to me
twelve drummers drumming, eleven pipers piping, ten lords
a-leaping, nine ladies dancing, eight maids a-milking, seven
swans a-swimming, six geese a-laying,  five gold rings;
```

[1]Сонымен қатар, екі рет, үш рет және он бес рет қарағанда да.

four calling birds, three french hens, two turtle doves and a partridge in a pear tree.

Программа жазудағы бірінші қадам – оның нені және қалай орындауы қажет екендігін *табиғи тілде жазу* (мүмкін, жалпыланған блок-схемаларды қолдану арқылы). Егер сіз алгоритмді қазақша немесе орысша құра алмасаңыз, онда оның нашар ойластырылған болуы ықтимал (әрине, барлығын жеке операторлар деңгейінде, мысалы, «индексті 1 мен 100 аралығында 1 қадамымен ...» деген сияқты «атап көрсету» туралы сөз болып тұрған жоқ). Алгоритимді осылайша сипаттау бірнеше себептен ыңғайлы болады: ол алгоритмді жанжақты ойластыруға, кейбір қателерді ерте кезеңдерде табуға, программаны блоктардың логикалық тізбектеріне бөлуге, сонымен қатар программаға түсініктемелер қамтамасыз етуге көмектеседі.

Егер алгоритмді аяқталған әрекеттер тізбегіне келтіруге болса (осыған ұмтылу қажет!), онда әрбір аяқталған әрекет *функция түрінде жазылады.* Әрбір функция тек бір ғана есепті шешуі тиіс (тәуелсіз, қысқа екі үзіндіні бір функцияға біріктірмеген жөн). Функция көлемінің өзгеру аралығы кең болады, барлығы функция үшін бөлінетін аяқталған код үзіндісі көлемінің қандай болғанына байланысты. Функция тұлғасы 1-2 экранға сыйып орналасқаны жөн: бірнеше кең функцияны қамтитын программаны түсіну тәрізді, әрқайсысы бірнеше жолдан тұратын, шашыраңқы орналасқан жүздеген ішкі программаларды да түсіну қиын болады.

Егер кейбір әрекеттер программа тұлғасында кем дегенде екі рет кездесетін болса, оларды да функция түрінде жазған жөн. Біртипті әрекеттер асыра жүктелген функциялар түрінде немесе параметрлі функциялар түрінде бейнеленеді. Қысқа функцияларды **inline** директивасы арқылы жариялаған жөн.

Айнымалылар атауларын таңдағанда мұқият болу керек. Дұрыс таңдалған атаулар программаны кез келген бір дәрежеде өзінен өзі құжатталғандай ете алады. Сәтсіз таңдалған атаулар қиындықтар туғызуы мүмкін. Қысқартылған атаулар қолданбаңыз, олар программаның оқылуын қиындатады және көбінесе кез келген бір сөздің қалайша қысқартылғанын ұмыту мүмкіндігі бар. Жалпы тенденция: айнымалының көріну аймағы неғұрлым үлкен болса, оның атауы соғұрлым ұзақ болады. Мұндай атаудың алдында жиі тип префиксі (айнымалының типін анықтауға болатын бір немесе бірнеше әріп) жазылады. Қысқа циклдердің санауыштары үшін, керісінше, **i, j** немесе **k** сияқты бір әріптік атауларды қолданған жөн. Макростарды программаның басқа обьектілерінен ажырату үшін олардың атауларын бас әріптермен жазған жөн. Астын сызу символынан басталатын атауларды, **_t** символдарымен аяқталатын типтер атауларын және C++ тілінің станадартты кітапхана ресурстарының атауларымен сәйкес келетін индификаторларды қолдану дұрыс болып саналмайды.

Айнымалыларды жариялау кезінде инициалдаған жөн, ал жариялауды оларды тікелей қолданылу орнына жақын орындаған дұрыс. Басқа жағынан алғанда, функцияның жергілікті (локалды) айнымалыларының барлық жарияланымдарын табуға жеңіл болатындай етіп, блок басында орналастырған ыңғайлы болады. Функциялар көлемдері үлкен болмаған жағдайда жоғарыдағы екі ұсынысты да біріктіру мүмкіндігі бар.

Глобалды (ауқымды) айнымалыларға қарағанда локалды (жергілікті) айнымалыларды қолданған жөн. Егер бәрі бір ауқымды айнымалыны қолдану қажет болса, оны статикалық айнымалы ретінде жариялаған жөн, бұл оның әрекет ету аймағын бастапқы бір файлмен шектейді.

Функцияға қажетті барлық ақпаратты параметрлер ретінде беруге ұмтылу қажет, өзгерістерін бақылау қиынырақ болатын ауқымды айнымалылар арқылы ақпаратты бермеген жөн.

Функцияда өзгермеуге тиісті *кіріс параметрлерді* мәні бойынша емес, *тұрақтылық сілтемелер ретінде берген жөн.* Бұл тәсіл программаның оқылуын жеңілдету және кездейсоқ қателердің туындау мүмкіндігін азайтумен қатар, әлдеқайда тиімді болып саналады, әсіресе күрделі объектілерді беру жағдайында оның тиімділігі басымырақ болады. Аталған артықшылық көлемдері нұсқауыш көлемінен кіші параметрлер үшін орындалмайды – оларды мәні бойынша берген тиімді (**const** түйінді сөзін жазуды ұмытпаңыз) болып табылады.

Функцияның *шығыс параметрлерін* сілтеме арқылы емес, адрес арқылы берген жөн, осылайша функцияны шақыру семантикасынан оның ішінде параметр өзгеріске ұшырайтынын түсіну мүмкін болады.

Функциядан жергілікті (локалды) айнымалыға сілтеме қайтаруға болмайды, себебі ол айнымалының әрекет ету аймағы болып табылатын функциядан шығу кезінде автоматты түрде жойылады. Функция ішінде динамикалық жады аймағында **malloc** функциясының немесе **new** операциясының көмегімен құрылған айнымалыға сілтеме қайтару ұсынылмайды, себебі бұл бақылануы қиын болатын жадының жұмсалуына алып келеді.

Программада сандарды тікелей қолданудан аулақ болу қажет. Тұрақтылар **const** немесе **enum** арқылы берілген (соңғысын қолданған дұрыс, себебі тізбелер үшін жады бөлінбейді), мағыналы атауларға ие болуы керек. Сөз тізбегі түрінде берілген атаулар программаны түсінуді жеңілдетеді, сонымен қатар тұрақтының мәнін өзгерту қажет болған жағдайда, оны программаның бір жерінде ғана орындауға болады.

Алгоритмнің әрбір үзіндісін жазу үшін *тілдің барынша қолайлы құралдарын қолдану керек.* Кез келген циклді **goto** және **if** операторларының көмегімен жүзеге асыруға болады, бірақ бұл қиынырақ болатын еді, себебі цикл операторлары көмегімен мұндай әрекеттер жеңіл оқылады, ал компилятор олар үшін барынша тиімді кодты генерациялайды. Бірнеше бағытқа тармақталуды қабаттасқан **if** операторлары арқылы емес, **switch** операторының көмегімен

жүзеге асырған жөн, топтағы бір функцияға басқаруды берудің ыңғайлы тәсілі – функцияларға нұсқауыштар жиымын пайдалану.

Шарттардың шамадан тыс артық тексерілуінен аулақ болу керек. Мысалы,

```
if      (strstr(a, b) > 0){...}
else if (strstr(a, b) < 0){...}
else if (strstr(a, b) == 0){...}
```

операторларының орнына төмендегі кодты жазған жөн:

```
int is_equal = strstr(a, b);
if      (is_equal > 0) {... }
else if (is_equal < 0) {... }
else    { ... }   // мұнда is_equal == 0
```

Егер **if** операторының бірінші тармағында басқару берілетін болса, **else** операторын қолданудың қажеті жоқ:

```
if (is_equal > 0) {... break;}
if (is_equal < 0) {... return;}
   {... }       // мұнда is_equal == 0
```

Теңсіздікті нөлмен салыстыру қажет емес (немесе, бұдан да сорақысы, теңдікті **true** немесе **false** мәндерімен салыстыру):

```
bool  is_busy;
...
if (is_busy == true) {... }    // нашар! if (is_busy)
                               // түрінде жазған дұрыс
if (is_busy == false){... }    // нашар! if (!is_busy)
                               // түрінде жазған дұрыс
char s[80];
while (fgets(s) != NULL) {... } // нашар!
               // while (fgets(s)) түрінде жазған дұрыс
while(a == 0) {... } // while(!a) түрінде жазуға болады
```

Егер шартты оператор тармақтардың бірі басқасына қарағанда айтарлықтай қысқа болса, онда қысқарақ **if** тармағын алдымен орналастырған жөн, әйтпесе басқарушы құрылым толығымен экранға сыймауы мүмкін, бұл оны түзету істерін қиындатады.

Кейбір жағдайларда шартты операцияны қолдану шартты операторға қарағанда ыңғайлырақ болады:

```
if (z) i = j; else i = k; /* i = z ? j : k; түрінде
                             жазған дұрыс */
```

Циклдерді қолдану кезінде инициализациялауды, циклден шығу шартын тексеруді және цикл параметрін өзгертуді бір орынға біріктіріп жазуға талпыну керек. Барынша ыңғайлы болып табылатын цикл операторын таңдау

жөніндегі ұсыныстар 46 б. көрсетілген. Итеративті циклдерді жазу кезінде (мұнда шығу шартын тексеру үшін цикл ішінде өзгеріп отыратын айнымалы қатынастары қолданылады) итерацияның ең үлкен қадамына жеткенде одан мәжбүрлі түрде *шығуды қарастырып қою керек.*

Қателердің қайтару кодтарын тексеріп отыру керек және программаның қалыпты жұмысы жағдайында басқару берілмеуі тиіс орындарда ол туралы хабарламалар шығарылуын қарастырып отыру қажет (программаны алғаш іске қосу кезінде сіздің дәл осы хабарламаны алуыңыздың ықтималдығы жоғары). Мысалы, **`switch`** операторында келісім бойынша анықталған жағдайды қарастыратын **`default`** сөзі болуы керек, әсіресе ол оператор құрамында ажыратқыштың барлық мүмкін мәндері көрсетілген жағдайда өте маңызды болып табылады.

Қате кеткен кездерде хабарлама беретін мүмкіндік болуы тиіс және ондайда қолданушыға қатені түзету жолдары туралы түсініктемелер берілгені дұрыс. Мысалы, қате мән енгізілген жағдайда, хабарламада сол мәннің өзгеру диапазоны көрсетілуі керек.

Динамикалық түрде жадыны бөлу мен босату операцияларын бір функция ішіне қарастыру керек. Жадыны бөліп, оны кейіннен босатуды ұмытқан жағдайда туындайтын жадының артық жұмсалуы – жұмыс істеу ұзақтығы шектелмеген программаларда – мәлімет базалары серверлері мен операциялық жүйелер жұмысында үлкен қиындықтар тудырады.

Программаны жазғаннан кейін оны мұқият тексеріп түзетіп шығу керек (ол үшін программаны білімі аздау сіздің дұшпаныңыз жазған деп есептеңіз) – қажет емес жерлерді алып тастау, сипаттамаларды бір орынға жинақтау, шарттарды тексеру мен циклдерді тиімді түрде ұйымдастыру, функцияларға бөлу тиімділігін тексеру, т.с.с. әрекеттерді орындау қажет. Программаны кез келген уақытта басқа программалаушыға беретіндей етіп жазып шығу керек. Жазған программаңызды достарыңыздың немесе әріптестеріңіздің біріне оқуға берген дұрыс (дұшпандарыңызға немесе бәсекелестеріңізге берген тіптен жақсы) және оларға ауызша айтпасаңыз түсініксіз болатын жерлерге қосымша түсініктемелер енгізген дұрыс.

Түсініктелер жазудың маңызы өте зор, себебі программалаушы көбінесе жазушыдан гөрі оқушыға көбірек ұқсайды. Программаны сүйемелдеуші сол программаның авторы болса да, бір жылдан кейін дұрыс құжатталмаған мәтінмен жұмыс істеу қиынға түседі, өйткені көп нәрсе ұмытылып кетеді.

Енді түсініктемелер мен программа мәтінін форматтауға байланысты кеңестерге тоқталайық. C++ тіліндегі программаларды оқып, жылдам түсіну оңай емес (олар Perl тіліндегі программаға қарағанда жеңіл оқылғанмен, Pascal тіліндегі программалардан нашар оқылады), сондықтан оларды форматтау мен құжаттау кезінде жақсы стильді ұстану маңызды жұмыс болып табылады.

«Түсініктемелерсіз программаны сүйемелдеу оңай емес. Ол ... компьютерлік программа емес, сағаттық бомба секілді» [9]. Егер программа пайдала-

нылатын болса, онда ол бір жыл ғана өмір сүрмейді, оған белгілі бір жаңа қасиеттер қосу қажеттілігі оны пайдалануға енгізген бойда бірден пайда болады және программаны сүйемелдеу оны жазуға қарағанда, әлдеқайда көп уақыт алады. Құжаттаманың негізгі бөлігі программа мәтініне енгізілуі керек. Жақсы түсініктемелер жазу жақсы программа жазған сияқты оңай іс емес.

Түсініктемелер қысқартылмай, тыныс белгілері дұрыс қойылған *сөйлемдерден*[1] *құралуы керек және ол айқын мәселелер туралы мәлімет бермеуі тиіс* (бұл кітаптағы түсініктемелерді үлгі ретінде ұсынуға болмайды, өйткені олар программаларды сүйемелдеу үшін емес, студенттерді оқытып үйретуге арналған). Мысалы, «**f** функциясын шақыру» немесе «айнымалыларды сипаттау» деген сияқты түсініктемелерді келтірудің ешқандай қажеттілігі жоқ. Егер программа үзіндісіне қатысты түсініктеме бірнеше жолдан тұратын болса, оны программа кодының оң жағына жазудың орнына, сол үзіндіге дейін орналастырып, оны тік бағытта туралаған жөн. Түсініктеменің абзацтық (азат жолдық) шегінісі түсініктеме жасалатын блоктың шегінісіне сәйкес келуі керек.

```
// Төменде көрсетілген программа
// блогында не орындалатынын
// сипаттайтын түсініктеме.
{ /* Түсініксіз
программа
блогы */ }
```

Функциялар мен басқа да логикалық тұрғыдан алғанда біртұтас үзінділерді бір-бірінен ажырату үшін бос жолдарды немесе келесі түрдегі түсініктемелерді қолданыңыз:

```
// ----------------------------------------
```

Қабаттасқан блоктарда 3-4 символдық шегініс болуы керек, қабаттасудың бір деңгейінде орналасқан блоктар тік бағытта туралануы тиіс. Жабылатын жүйелі жақша ашылатын жақшаның дәл астында немесе блоктың қабаттасу деңгейіне сәйкес бағанда орналасқаны жөн. Мүмкін болған жерлерде мәтінді бағана бойынша форматтаңыз:

```
#include <string.h>
#include <stdlib.h>
int main(){
   double     m[10];                // түсініктеме
   const char s[] = "2,38.5,1";     //түсініктеме
    char       *p = s;              // түсініктеме
   int        i = 0;                // түсініктеме
```

[1]Егер онда орфографиялық қателер болмаса тіпті жақсы.

```
// ----------------------------------------
   do{
      m[i++] = atof(p);
      if (i > 9) break;
   }while(p = strchr(p, '.'), p++);
   for(int k = 0; k<i; k++)
      printf("%5.2f ", m[k]);
   return 0;
}
```

Программа айқын болуы үшін шегіністер ережесін қолданудан бас тартуға да болады:

```
if       ( is_best        ) best();
else if  ( is_bad         ) worse();
else if  ( is_vovochka    ) worst();
```

Көлемді құрама операторлардың соңын белгілеп отырыңыз:

```
while(1){
   while(gets(s)){
      for(i = 0; i<10; i++){
         for(j = 0; j<10; j++){
            // екі бет код
         } // for(j = 0; j<10; j++)
      } // for(i = 0; i<10; i++)
   } // while(gets(s))
} // while(1)
```

Бір жолға бірнеше операторларды орналастырмаған жөн. Орыс немесе қазақ тіліндегі сияқты, тыныс белгілерінен кейін бос орындар қойылуы керек:

```
f=a+b; // нашар!  f = a + b; болғаны дұрыс
```

C++ тілінде программалау және программаларды жобалау бойынша көптеген нақты ұсыныстар Ален И. Голубтың [9] кітабында келтірілген. Б. Страуструптың 1000 беттік кітабын [17] оқи отырып, одан көптеген қызықты мәліметтер алуға болады. Қорытындылай келе, осы кітаптан бір цитат келтіре кетейік: «“C++ тілінде жақсы программаларды қалай жазуға болады?” деген сұрақ “Жақсы ағылшын прозасын қалай жазуға болады?” деген сұрақпен үндеседі. Мұнда екі кеңес бар: “Не айтқың келетінін біл” және “Жаттыға түс. Жақсы стильге ұмтыл”. Осы екі кеңесті де C++ үшін де, ағылшын прозасы үшін де қолдануға болады және екеуін де ұстану қиын іс».

Программаны жобалау және тесттен өткізу

Мэрфи заңдары
Бәрі де оңай болып көрінгенмен, оңай істелмейді.
Бәрі де күтілген мерзімнен ұзаққа созылады.
Егер бір нәрсенің бүлінуі мүмкін болса, ол міндетті түрде бүлінеді.
Каллаганның Мэрфи заңдарына берген түсініктері
Мэрфи оптимист болған.

Программалау ісіне жаңадан бастама алған азаматтар, әсіресе студенттер, программаны көп жағдайда былай жазады: тапсырманы алған бойда компьютерге отырады және бірден ойларына келген алгоритм үзінділерін кодтауды бастайды. Айнымалыларға бірінші кезекте ойға келетін `a, b, c` сияқты немесе басқа да, шамалардың мазмұнына қарағанда, автордың сөздік қорын көбірек сипаттайтын атаулар береді. Талапкер құлшынысының алғашқы екпінін өшіріп, компьютер тоқтап қалғанда барып программалаушы үзіліс жасайды да, содан кейін жазылған кодтар түгел өшіріліп, жұмыс қайтадан басынан басталады.

Жұмыс барысында мәліметтер құрылымдары бірнеше рет өзгеріске ұшырайды, функциялар күйгелектікпен өшіріліп, қайтадан басынан кодталады, ал оны модульдерге бөлу тек программа көлемін қарап шығу қиындаған кезде жасалады. Программаға түсініктемелер жазылмайды, ал мәтінді форматтау жайлы тіпті айтудың да қажеті жоқ. Жиі-жиі компилятордың, компьютердің және операциялық жүйенің дұрыс жұмыс істеп тұрғанына күмән келтіріледі[1].

Программа алғаш рет орындалу кезеңіне жеткенде, оған кездейсоқ мәндер енгізіледі, содан кейін экран бірқатар уақытқа шейін таңданып бақыланатын объектіге айналады. Мұндай программа, әдетте, автордың басқаруымен бастапқы мәліметтердің белгілі бір жиынында ғана «жұмыс істейді», ал программаға тіпті аз да болса өзгерістердің енгізілуі автордың өзіне деген сенімін жоғалтып, оның программалау процесін жек көріп кетуіне де алып келеді.

Сіздің мақсатыңыз программалауға кәсіби түрде кірісуді үйрену болып табылады. Кәсіби маманның осындай программалауды жаңа бастаушы адамнан айырмашылығы оның алдына қойылған тапсырмаға толықтай жауап беретін программа жазуға қанша уақыт жұмсалатынын дәл бағалай алу мүмкіндігінде болады. Ол үшін «ақыл, талғам және сабырлылыққа»[2] қоса, тәжірибе қажет, сонымен қатар программалаушылар осы пәннің жарты ғасырдан астам даму мерзімінде қалыптастырған негізгі қағидаларын білу керек. Тіпті ең қарапайым программалар жазуға да белгілі бір тәртіпті ұстана отырып, ұйымдасқан түрде ден қою керек.

[1]Міндетті түрде Билл Гейтс еске алынады.
[2]Авторға Б. Страуструптың осы тұжырымы өте ұнайды, сондықтан ол кітапта екінші рет қайталанып тұр.

ЕСКЕРТУ

Программаны құрудың негізгі кезеңдері туралы айтудың алдында, оған қарағанда абстрактылы ұғымдар жайлы – әрбір программалаушы басшылыққа алуы тиіс қағидалар туралы – айтып кеткен жөн. Көбінесе программалаушылар өздерінің басты мақсаты пайдалы жұмыс істеу емес, жаңадан ерекше жұмыс істейтін алгоритмдерді ойлап табу деп есептейді, сол арқылы «дүкен алушыға емес, сатушыға арналған» деген қайғылы көзқарасты еске салады. Программалаушы өзінің оны пайдаланушы адам үшін жұмыс істейтіні жөнінде және белгіленген мерзімде сенімді программа құруды камтамасыз ететін топтың мүшесі екенін ұмытпауы тиіс.

Программалаудың құрылымдық тәсілі, жоғарыда айтылып өткендей, жобаны құрудың барлық кезеңдерін қамтиды, олар: спецификациялау, жобалау, нақты түрде программалау және оны тесттен өткізу, яғни тестілеу. Мұндайда алға қойылатын міндеттер – тек пайдалануға болатын құрылымдарды қолдану арқылы туындайтын қателер санын азайту, қателерді мүмкіндігінше ерте анықтау және оларды түзету процесін жеңілдету.

Төменде келтірілген программаны құру кезеңдері программалаушылар тобы арқылы жүзеге асырылатын көлемді жобалар жасауға есептелген. Көлемі шағын қысқа программалар жазу үшін әрбір кезең қысқарып жеңілдейді, бірақ кезеңдердің мазмұны мен реттілігі өзгермейді.

І кезең. Кез келген программаның құрылуы *есептің қойылымынан* басталады. Бастапқыда есеп пәндік аймақ терминдерімен сипатталып жазылады, бірақ оны кейіннен программалауға жақын терминдерге ауыстыру керек. Программалаушы пәндік аймақты, ал тапсырыс беруші программалауды түбегейлі әрі тиянақты түсінетін жағдайлар сирек кездесетіндіктен (қарапайым мысал, бухгалтерлік программа жазу қажет), есептің қойылымы күрделі итерациялық процеске айналып кетуі мүмкін. Сонымен қатар, көбінесе есептің қойылымында тапсырыс беруші өзінің талаптары мен критерийлерін анық және толық жеткізе де алмайды. Мысал ретінде, 1973 жылы Лондон университетінің есептеу орталығының ақпараттық бюллетенінде жарияланған, программалау процесін дәл көрсететіндіктен бірден кең таралып кеткен «Әткеншектер» карикатурасын келтіруге болады (3.1-сурет).

Есептің қойылымы техникалық тапсырма жазып, содан кейін программаның сыртқы спецификациясын құрумен аяқталады. Соңғысының құрамына төмендегі элементтер кіреді:

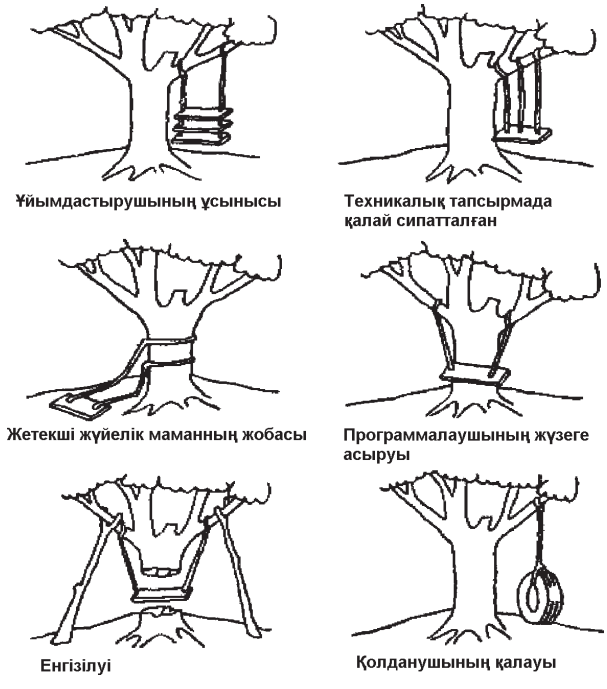
- бастапқы мәліметтер мен нәтижелер сипаттамасы (типтері, форматтары, дәлдігі, берілу тәсілі, шектеулер)[1];
- программа арқылы жүзеге асырылатын есептің сипаттамасы;
- программаны қолдану тәсілі;
- мүмкін болатын төтенше (апаттық) жағдайлар мен программаны қолданушы қателіктерінің сипаттамасы.

[1]Мұндағы типтер мен форматтар программалау тілінің типтерін білдірмейді.

Осылайша, программа функция мен кіріс және нәтижелік (шығыс) мәліметтер анықталған «қара жәшік» ретінде қарастырылады.

II кезең. *Ішкі мәліметтер құрылымдарын қалыптастыру.* Көптеген алгоритмдер мәліметтердің қалай ұйымдастырылғанына тәуелді болады, сондықтан программаны жобалауды алгоритмнен емес, шығыс, кіріс және аралық мәліметтерді бейнелейтін қажетті құрылымдарды қалыптастырудан бастау керектігі белгілі. Мұндайда көптеген факторлар, мысалы, мәліметтер көлемдеріне қойылатын шектеулер, қажетті дәлдігі, программаның жылдамдығына қойылатын талаптар назарға алынады. Мәліметтер құрылымдары статикалық немесе динамикалық түрде болуы мүмкін (динамикалық құрылымдар келесі бөлімде қарастырылады).

III кезең. Жобалау (жалпы құрылымды және модульдердің өзара әрекеттесуін анықтау). Бұл кезеңде программаны *төмен қарай жобалау технологиясы* қолданылады, оның негізгі мәні теориялық тұрғыдан алғанда қарапайым: есепті жеке-жеке қарастыруға болатын, күрделілігі төмен ішкі шағын есептерге бөлшектеу. Мұндайда программаны біртіндеп байыптау (қадамға бөле отырып қарастыру) тәсілі қолданылады. Бұл процесті көзге былайша елестетуге болады: алдымен программа жалпыланған әрекеттерді түсіне алатын гипотетикалық машина тілінде жазылады, содан соң олардың әрқайсысы төменгі абстракциялық деңгейде сипатталады, ары қарай осы үлгіде жалғаса береді. Бұл кезеңде *интерфейстер спецификациясы,* яғни ішкі есептердің бір-бірімен әрекеттесу тәсілдері өте маңызды болып табылады.



3.1-сурет. «Әткеншекті» жобалау кезеңдері

Әрбір ішкі есеп үшін жоғарыда көрсетілгенге ұқсас сыртқы спецификация құрылады. Бұл кезеңде программаны модульдерге бөлу мәселелері де шешіледі, негізгі критерий – олардың өзара әрекеттесуін барынша азайту. Бір есеп бірнеше модульдер көмегімен орындалуы және керісінше, бір модульде бірнеше есеп шығарылуы мүмкін. Жоғарғы деңгейлі жобалауды аяқтағаннан кейін ғана одан төменгі жобалау деңгейіне көшу жүргізіледі. Алгоритмдерді жалпылама түрде – мысалы, сөздермен жазу, жалпыланған блок-схемалар түрінде немесе басқаша да тәсілдер арқылы жазу атқарылады. Егер алгоритм жазуда қиындықтар туындаса, онда ол алгоритмнің нашар құрылғандығының белгісі болып табылады.

Жобалау кезеңінде болашақта программаға өзгерістер енгізу мүмкіндіктерін ескеріп, өзгерістерді енгізу барынша жеңіл болатындай етіп жобалауға ұмтылған жөн. Қандай өзгерістерді енгізу қажеттігі туындайтыны белгісіз болғандықтан, бұл талап «барлығының ортақ теориясын» құруға ұқсас болады; практика жүзінде нұсқаларды саналы түрде таңдаумен шектелу керек. Программалаушы өзінің тәжірибесі мен санасына сүйене отырып, болашақта программның қандай қасиеттерін өзгерту керек немесе жетілдіру қажет болатынын шешеді.

Нақты көлемдегі программалардың барлық егжей-тегжейлерін бірден ойластыру мүмкін емес болғандықтан, жобалау үрдісі итерациялық процесс болып келеді.

IV кезең. *Құрылымдық программалау*. Программалау процесі де «жоғарыдан төмен қарай» принципі бойынша жүзеге асырылады: алдымен ең жоғарғы деңгейдің модульдері кодталады және оларды тексеріп түзетуге арналған тестілік мысалдар құрылады, бұл кезде әлі жазылмаған келесі деңгей модульдерінің орнына «жапсырмалар» – уақытша программалар қойылады. Көбінесе «жапсырмалар» қарапайым жағдайда өздеріне басқару берілгені туралы қарапайым хабарламалар ғана береді, содан кейін басқару шақырушы модульге қайтарылады. Басқа жағдайларда «жапсырма» алдын ала берілген немесе жеңілдетілген қарапайым алгоритм бойынша есептелген мәндерді ғана экранға шығара алады.

Осылайша, алдымен программаның логикалық қаңқасы құрылып, кейіннен ол код түріндегі мәліметтермен толықтырылады.

Программалау үрдісінде «төменнен жоғарыға» бағытталған технологияны қолдану ісі – алдымен төменгі деңгей модульдерін жазып алып түзету, содан кейін барып оларды көлемдірек фрагменттерге біріктіру дұрыс шешім сияқты көрінгенмен, оның да бірқатар кемшіліктері бар.

Біріншіден, жоғарғы деңгейді кодтау үрдісінде программаның төменгі деңгейлерін жобалаудың кейбір қиындықтары ашылуы мүмкін (себебі, программаны жазуда оның логикасы жобалау процесіне қарағанда мұқият ойластырылады). Егер мұндай қателік ең соңында анықталса, дайын

болған төменгі деңгей модульдерін өзгертуге қосымша шығындар қажет болады.

Екіншіден, әрбір модульді, содан кейін барып программаның көлемдірек кодтарын жазу үшін әр жолы жекелеген тест мысалдарын қарастыру қажет болады, осылайша программалаушының кейін модуль жұмыс істейтін ортаны жиі модельдеуіне тура келеді. Программалаудың төмен қарай бағытталған технологиясы тесттерді құрудың табиғи реттілігін – оның төменге қарай бағытталған түзету мүмкіндігін қамтамасыз етеді, оны кейінірек қарастыратын боламыз.

С++ тіліндегі алгоритмдер жазуға арналған ұсыныстар (осы ұсыныстардың басым бөлігі жоғарғы деңгейдегі басқа программалау тілдері үшін де дұрыс болып табылады) алдыңғы бөлімде көрсетілген. Мұндағы негізгі мақсатымыз – жалпы программаның және оны құрамындағы кез келген функция құрылымының қарапайым болуы және кейіннен жеңіл оқылуы екенін еске саламыз. *Программалау барысында интерфейсті (функцияның, модульдің, кластың интерфейсі) оның жүзеге асырылуынан бөле отырып, қажетсіз ақпаратқа қол жеткізуді шектеу керек.* Тіпті программалаудағы ұсақ-түйек ұқыпсыздықтың өзі түзету кезеңінде қателерді іздеуге кететін шығындардың артуына алып келуі мүмкін.

Жобалау және программалау кезеңдері қатарласа жүргізілуі тиіс: ең дұрысы – алдымен жоғарғы деңгей, содан кейін келесісі, т.с.с. сатылы түрде жобаланады және кодталады. Мұндай стратегияның қолданылу себебі –кодтау кезінде төменгі деңгей модульдеріне әсер ететін өзгерістер енгізу қажеттілігі туындауы мүмкін.

V кезең. *Төменге қарай тестілеу.* Бұл кезеңді соңғы етіп жазып отырғанымыз бастапқы кезеңдерде тесттен өткізуді орындау қажет емес дегенді білдірмейді. Жобалау мен программалау міндетті түрде тесттер жынтығын – тексерілетін бастапқы мәліметтер мен оларға сәйкес эталондық нәтижелер жиынын анықтаумен қатар жүруі керек.

Программаны тестілеу және түзету процестерін бір-бірінен ажырата білу керек. *Тестілеу* – программаның дұрыстығын тексеруге арналған процесс. Тестілеу тұрақты түрде жүргізіледі, оның мақсаты – программаның дұрыс жұмыс істейтінін және барлық жоба спецификацияларына сәйкес келетінін көрсету. Программаны *түзетіп жөндеу (debug)* – программадағы қателерді анықтап, соларды жөндеу процесі, мұнда барлық қателерді түзету мақсаты қойылмайды. Тесттен өткізу кезінде анықталған қателер түзетіледі. Қателерді табу процесі қанығу заңына бағынатынын, яғни қателердің көпшілігі тестілеудің бастапқы кезеңдерінде анықталатынын және программада неғұрлым қате аз болса, оларды табу соғұрлым ұзаққа созылатынын жоспарлау кезінде ескеріп отыру керек.

Программаны толық тесттен өткізу үшін *алгоритмнің әрбір тармағын жеке-жеке тексеру* керек. Тармақтардың жалпы саны әрбір кезеңдегі

барлық баламалы (альтернатива) жолдар комбинациясымен анықталады. Ол нақты шектеулі сан, бірақ ол өте үлкен болуы мүмкін, сондықтан программа үзінділерге бөлінеді, солардың әрқайсысын толық тесттен өткізгеннен кейін әрбір фрагмент көлемді тармақтардың элементар түйіндері ретінде қарастырылады. Операторлардың талап етілген реттілікпен орындалуын қадағалайтын мәліметтерден басқа, тесттерде *шекаралық шарттарды тексеру* (мысалы, x>10 шарты бойынша көшу 10 санына тең, одан үлкен және кіші мәндер үшін де тексерілуі керек) де қамтылуы тиіс. *Бастапқы мәліметтердің соларға сәйкес келмейтін қате мәндерін тексеру және оған программаның жауап қайтаруы жеке тексеріледі.*

Төмен қарай бағытталған тестілеу идеясы бойынша программаны жобалау аяқталмай тұрып, оны тесттен өткізу қарастырылады. Бұл негізгі модуларалық интерфейстерді ертерек тексеруге, сонымен қатар программа қолданушының талаптарын қанағаттандыратынына көз жеткізуге мүмкіндік береді. Логикалық ядро негізгі интерфейстердің жүзеге асырылуының дұрыстығына күмән болмайтындай дәрежеде сыналғаннан кейін ғана программаның келесі деңгейін кодтау мен тестілеуге кірісу керек.

Әрине, программа қаңқа түрінде көрсетілетін кезде оны толық тесттен өткізу мүмкін емес, алайда әрбір келесі деңгейді қосу тестілеу аймағын біртіндеп кеңейтуге мүмкіндік береді.

Төмен қарай бағытталған жобалауда жүйе деңгейіндегі кешенді түзету кезеңі жоғары қарай бағытталған тәсілмен салыстырғанда, аз уақыт алады және мұнда тосын жағдайлар да аз болады, себебі жүйенің көлемді бөлігіне әсер ететін үлкен қателердің пайда болу ықтималдығы едәуір аз болады. Сонымен қатар, жүйеге қосылатын әрбір модуль үшін оның қоршаған ортасы қалыптасқан, түзетілген модульдердің нәтижелік (шығыс) мәліметтерін басқа модульдерді тестілеуде кіріс мәліметтер ретінде қолдануға болады, бұл тестілеу үрдісін жеңілдетеді. Бұл жүйеге модульді «шикілей» қосуға болады деген сөз емес – тестілеудің бір бөлігін жеке, яғни автономды түрде өткізу ыңғайлы болып табылады, себебі жүйе кірісінде жеке модульді тестілеу үшін барлық қажетті нұсқаларды генерациялау қиын болады.

Программаларды жөндеп түзету әдістерін қарастыру бұл кітаптың құрамына енгізілмеген, себебі түзету кезінде нақты программалау қоршауы құралдары (мысалы, компиляторлар) оларды түзетуге екпінді түрде әсер етеді.

Мәліметтердің динамикалық құрылымдары

Кез келген программа мәліметтерді өңдеуге арналған, ал жұмыс алгоритмдері мәліметтердің ұйымдастырылу тәсілдеріне тәуелді болып келеді, сондықтан мәліметтер құрылымдарын таңдау алгоритмдерді құрудан

бұрын орындалуы керек. Жоғарыда мәліметтерді ұйымдастырудың C++ тілі қамтамасыз ететін стандартты тәсілдері – негізгі және құрама типтер қарастырылды. Программаларда жиымдар, құрылымдар және олардың араласуы, мысалы, өрістері жиымдар мен құрылымдар болып келетін құрылымдар жиымдары жиі қолданылады.

Мәліметтерге жады бөлу компиляция кезеңінде (мұндағы жадының қажетті көлемі программаның орындалуынан бұрын белгілі болуы тиіс, яғни тұрақты түрінде берілген болуы керек) немесе программаның орындалуы кезінде **new** операциясының немесе **malloc** функциясының (қажетті жады көлемі жадыны бөлуге дейін белгілі болу керек) көмегімен орындалады. Осы екі жағдайда да мәліметтерге жадының үздіксіз аймағы бөлінеді.

Егер мәліметтермен жұмыс істеуді бастағанға дейін оларды сақтауға қанша жады көлемі керек екенін анықтау мүмкін болмаса, онда жады қажеттілігіне қарай бір-бірімен нұсқауыштар көмегімен байланысқан жеке блоктар бойынша бөлінеді. Мәліметтерді ұйымдастырудың мұндай тәсілі *мәліметтердің динамикалық құрылымы* деп аталады, себебі олардың көлемі программаның орындалу барысында өзгеріп отырады. Программада динамикалық құрылымдардан көбінесе *сызықтық тізімдер, стектер, кезектер және бинарлы бұтақтар* жиі қолданылады. Олар жеке элементтердің бір-бірімен байланысу тәсілдері және орындала алатын операциялары арқылы ажыратылады. Динамикалық құрылымдар жедел жадының сыбайлас емес аймақтарын да пайдалануы мүмкін.

Динамикалық құрылымды көлемдері белгілі мәліметтермен тиімді түрде жұмыс істеу үшін де пайдалануға болады, әсіресе мұны сұрыптау есептерін шешу үшін кең қолданады, өйткені динамикалық құрылымдарды реттеу элеметтердің орындарын ауыстыруды қажет етпейді, ол осы элементтерге нұсқауыштарды өзгертуге негізделген. Мысалы, программаны орындау барысында мәліметтердің көлемді жиымын бірнеше рет сұрыптау керек болса, оны сызықтық тізім түрінде ұйымдастыру тиімді болып табылады. Мәліметтерді іздеу кезінде жылдамдық маңыздырақ болатын жағдайларда, мәліметтерді бинарлы бұтақтар түрінде бейнелеген жөн.

Кез келген динамикалық мәліметтер құрылымының *элементі*, кем дегенде екі өрісі бар (мәліметтерді сақтау үшін және нұсқауыш үшін) құрылым (struct мағынасында) болып табылады. Мәліметтер мен нұсқауыштар өрістері бірнешеу болуы мүмкін. Мәліметтер өрістері кез келген типте: негізгі, құрама немесе нұсқауыш типті болуы мүмкін. Қарапайым элементтің (компоненттер, түйіндер) сипаттамасы келесідей түрде болады:

```
struct Node{
   Data d;// Data мәліметтер типі бұрын анықталуы керек
   Node *p;
};
```

Мәліметтердің динамикалық құрылымдарымен орындалатын негізгі операциялардың жүзеге асырылуын қарастырайық (кейіннен тізімнің класс шаблоны түрінде жүзеге асырылуының мысалы көрсетіледі, 228 б.).

Сызықтық тізімдер

Элементтер жиынын байланыстырудың ең жеңіл тәсілі – әрбір элементтің өзінен кейін орналасқан элементке сілтеме жасауын жүзеге асыру. Мұндай тізім *бірбағытты (бірбайланысты)* деп аталады. Егер әрбір элемент үшін өзінен бұрын орналасқан элементке бағытталған тағы бір сілтеме қосатын болсақ, онда *қосбағытты (қосбайланысты) тізім* пайда болады, егер соңғы элементті нұсқауыш арқылы алғашқы элементпен байланыстырса, *сақиналық* тізім құрылады.

Тізімнің әрбір элементінде осы элементті анықтайтын кілт болады. Кілт, әдетте, бүтін сан немесе тіркес болады және ол мәліметтер өрісінің бір бөлігі болып табылады. Тізіммен жұмыс істеу барысында кілт ретінде мәліметтер өрісінің түрлі бөліктерін қолдануға болады. Мысалы, тегі, туылған жылы, жұмыс өтілі және жынысы толтырылатын сызықтық тізім құрылатын болса, жазбаның кез келген бөлігін кілт ретінде алуға болады: алфавит бойынша сұрыптау кезінде кілт тегі болады, ал іздеу кезінде, мысал ретінде еңбек ардагерлерін алсақ, кілт жұмыс өтілі болады. Тізімнің әртүрлі элементтерінің кілттері сәйкес келуі мүмкін.

Тізімдерге келесі операцияларды қолдануға болады:

- тізімді бастапқы қалыптастыру (алғашқы элементін құру);
- тізімнің соңына элемент қосу;
- кілті көрсетілген элементті оқу;
- элементті тізімнің көрсетілген позициясына кірістіру (кілті берілген элементтің соңына немесе алдына);
- кілті көрсетілген элементті жою;
- тізімді кілт бойынша реттеу.

Қосбағытты сызықтық тізімді қарастырайық. Тізімді қалыптастыру және онымен жұмыс жасау үшін кем дегенде бір нұсқауыш болуы керек – ол тізім басына сілтеме жасайды. Тізім соңына бағытталған қосымша бір нұсқауыш енгізген ыңғайлы болады. Жеңілдік үшін тізім бүтін сандардан тұрады деп есептейік, онда тізім элементінің сипаттамасы төмендегідей болады:

```
struct Node{
   int d;
   Node *next;
   Node *prev;
};
```

Төмендегі программа 5 саннан тұратын тізім қалыптастырады, тізімге сан қосады, тізімнен санды өшіреді және тізімді экранға шығарады. Тізім басына нұсқауыш **pbeg**, соңына нұсқауыш **pend**, ал қосымша нұсқауыштар **pv** және **pkey** атауларымен белгіленген.

```
#include <iostream.h>
struct Node{
   int d;
   Node *next;
   Node *prev;
};
//-------------------------------------------------
Node * first(int d);
void add(Node **pend, int d);
Node * find(Node * const pbeg, int i);
bool remove(Node **pbeg, Node **pend, int key);
Node * insert(Node * const pbeg, Node **pend, int key, int d);
//-------------------------------------------------
int main(){
   Node *pbeg = first(1);  // Тізімнің алғашқы
                           //элементін қалыптастыру
   Node *pend = pbeg;      // Тізім басталып,
                           // бірден аяқталады
   // Тізім соңына 4 элемент – 2, 3, 4 және 5 қосу:
   for (int i = 2; i<6; i++) add(&pend, i);
   // 2 элементінен кейін 200 элементін кірістіру:
   insert(pbeg, &pend, 2, 200);
   // 5 элементін жою:
   if(!remove (&pbeg, &pend, 5)) cout << "табылмады";
   Node *pv = pbeg;
   while (pv){              // тізімді экранға шығару
      cout << pv->d << ' ';
      pv = pv->next;
   }
   return 0;
}
//-------------------------------------------------
// Алғашқы элементті қалыптастыру
Node * first(int d){
   Node *pv = new Node;
   pv->d = d; pv->next = 0; pv->prev = 0;
   return pv;
}
```

```
//-------------------------------------------------
// Тізімнің соңына қосу
void add(Node **pend, int d){
   Node *pv = new Node;
   pv->d = d;  pv->next = 0;
   pv->prev = *pend;
   (*pend)->next = pv;
   *pend = pv;
}
//-------------------------------------------------
// Элементті кілті бойынша іздеу
Node * find(Node * const pbeg, int d){
   Node *pv = pbeg;
   while (pv){
      if(pv->d == d)break;
      pv = pv->next;
   }
   return pv;
}
//-------------------------------------------------
// Элементті жою
bool remove(Node **pbeg, Node **pend, int key){
   if(Node *pkey = find(*pbeg, key)){       // 1
      if (pkey == *pbeg)}                   // 2
         *pbeg = (*pbeg)->next;
         (*pbeg)->prev =0;}
      else if (pkey == *pend){              // 3
         *pend = (*pend)->prev;
         (*pend)->next =0;}
      else{                                 // 4
         (pkey->prev)->next = pkey->next;
         (pkey->next)->prev = pkey->prev;}
      delete pkey;
      return true;                          // 5
   }
   return false;                            // 6
}
//-------------------------------------------------
// Элементті кірістіру
Node * insert(Node * const pbeg, Node **pend, int key, int d){
   if(Node *pkey = find(pbeg, key)){
      Node *pv = new Node;
      pv->d = d;
```

```
        /* 1 – жаңа түйіннің келесі түйінмен  байланысын
           орнату: */
        pv->next = pkey->next;
        /* 2 - жаңа түйіннің алдыңғы түйінмен байланысын
           орнату: */
        pv->prev = pkey;
        /* 3 - алдыңғы түйіннің жаңа түйінмен байланысын
           орнату: */
        pkey->next = pv;
        /* 4 - келесі түйіннің жаңа түйінмен байланысын
           орнату: */
        if( pkey != *pend) (pv->next)->prev = pv;
        // Егер түйін тізім соңына қойылса,
        // тізім соңына нұсқауышты жаңарту.
        else *pend = pv;
        return pv;
    }
    return 0;
}
```

Программаның жұмысының нәтижесі:

```
1 2 200 3 4
```

Функция ішінде өзгертілмейтін барлық параметрлер **const** модификаторы арқылы берілуі тиіс. Өзгеруі мүмкін нұсқауыштар, (мысалы, тізімнен соңғы элементті өшірген кезде тізім соңына нұсқауышты өзгерту керек) адрес бойынша беріледі.

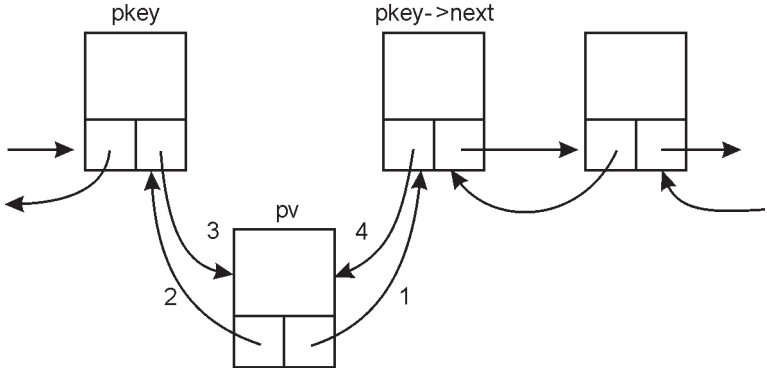
remove *тізімнен элементті өшіру функциясын* толығырақ қарастырайық. Оның параметрлері ретінде тізім басына және тізім соңына нұсқауыштар және өшірілуі тиіс элементтің кілті қабылданады. 1 жолында **pkey** жергілікті (локалды) нұсқауышына жады бөлінеді, оған **find** элементті кілті бойынша табу функциясының нәтижесі меншіктеледі. Егер іздестіру нәтижесі сәтті болса, бұл функция элементке нұсқауышты, ал мұндай кілті бар элемент табылмаған жағдайда 0 мәнін қайтарады. Егер **pkey** нөлдік емес мән қабылдаса, онда **if** операторындағы шарт ақиқат болады (ондай элемент бар), басқару 2 операторға беріледі, әйтпесе **false** мәнді функциядан қайту орындалады (6 оператор).

Элемент тізімнің басында, соңында немесе ортасында орналасқанына байланысты оны тізімнен өшіру әртүрлі болып орындалады. 2 операторға өшірілетін элементтің тізім басында орналасу шарты тексеріледі, жауап оң болса, онда **pbeg** атты тізім басына нұсқауыш алғашқы элементтің **next** өрісінде адресі орналасқан тізімнің келесі элементіне нұсқайтындай етіп өзгертілуі керек. Тізімнің жаңа бастапқы элементінің өзінен бұрын орналасқан элементке нұсқау өрісіне 0 мәні берілуі керек.

Егер өшірілетін элемент тізім соңында орналасса (3 оператор), онда **pend** тізім соңы нұсқауышын соңғы элементтен бұрын орналасқан элементке ауыстыру керек, оның адресі соңғы элементтің **prev** өрісінде жазылған. Жаңа соңғы элемент үшін келесі элементке нұсқауыш мәнін нөлге теңестіру қажет. Егер өшіру тізім ортасынан орындалса, онда осы элементтен бұрын және одан кейін орналасқан элементтер арасында қосбағытты байланыс орнату жеткілікті. Нұсқауыштарды түзеткеннен кейін элементке бөлінген жады босатылады және функция **true** мәнін қайтарады.

Тізімге элементті кірістіру функциясының жұмысы 3.2-суретте көрсетілген. Бағыттауыш сызықтардың жанындағы нөмірлер түсініктемелердегі операторлар нөмірлеріне сәйкес келеді.

Байланысқан тізімді сұрыптау элементтер арасындағы байланыстарды өзгертумен сипатталады. Алгоритмнің барысы: бастапқы тізім қаралып, оның әрбір элементі жаңа тізімде өз кілтінің мәні бойынша анықталатын орынға кірістіріледі.



3.2-сурет. Тізімге элементті кірістіру

Төменде *реттелген тізімді қалыптастыру функциясы* келтірілген (алғашқы элемент бар деп есептеледі):

```
void add_sort(Node **pbeg, Node **pend, int d){
   Node *pv = new Node;        // қосылатын элемент
   pv->d = d;
   Node * pt = *pbeg;
   while (pt){                 // тізімді қарап шығу
      if (d < pt->d){          // ағымдағы элемент (pt)
                               // алдына орналастыру
         pv->next = pt;
         if (pt == *pbeg){     // тізім басына
            pv->prev = 0;
            *pbeg = pv;}
```

```
            else{                    // тізім ортасына
               (pt->prev)->next = pv;
               pv->prev = pt->prev;}
            pt->prev = pv;
            return;
         }
         pt = pt->next;
      }
      pv->next = 0;                  //тізім соңына
      pv->prev = *pend;
      (*pend)->next = pv;
      *pend = pv;
   }
```

Стектер

Стек – бұл құрамына элемент қосу және оны таңдау стектің төбесі деп аталатын бір басынан ғана орындалатын бірбағытты тізімнің дербес түрі. Стекке қолданылатын басқа операциялар анықталмаған. Таңдау кезінде элемент стектен алынып тасталады. Стек LIFO (last in – first out, соңғы болып енеді – бірінші болып шығады) қызмет көрсету принципін жүзеге асырады. Стекті бір ұшы жабық, ішіне доп лақтырылатын тар ұзынша құбыр ретінде елестетуге болады. Ең бірінші салынған допты алу үшін, одан кейін салынған барлық доптарды шығарып алу керек. Жергілікті (локалды) айнымалыларға жады LIFO принципі бойынша бөлінетін болғандықтан, стек сегменті дәл осылай аталған. Стектер жүйелі программалық жабдықтамаларда, компиляторларда, әртүрлі рекурсивті алгоритмдерде кеңінен қолданылады.

Төмендегі программа 5 саннан тұратын (1, 2, 3, 4, 5) стек құрады және оны экранға шығарады. Дәстүр бойынша стекке орналастыру функциясын **push**, ал таңдау функциясын **pop** деп атайды. Стекпен жұмыс жасауға арналған нұсқауыш (**top**) әрқашан оның төбесіне сілтеме жасайды.

```
#include <iostream.h>
struct Node{
   int d;
   Node *p;
};
Node * first(int d);
void push(Node **top, int d);
int pop(Node **top);
//-----------------------------------------------
int main(){
   Node *top = first(1);
```

```
   for (int i = 2; i<6; i++) push(&top, i);
   while (top)
      cout << pop(&top) << ' ';
   return 0;
}
//-----------------------------------------------
// Стекті бастапқы қалыптастыру
Node * first (int d){
   Node *pv = new Node;
   pv->d = d;
   pv->p = 0;
   return pv;
}
//-----------------------------------------------
// Стекке орналастыру
void push(Node **top, int d){
   Node *pv = new Node;
   pv->d = d;
   pv->p = *top;
   *top = pv;
}
//-----------------------------------------------
// Стектен таңдау
int pop(Node **top){
   int temp = (*top)->d;
   Node *pv = *top;
   *top = (*top)->p;
   delete pv;
   return temp;
}
```

Программаның жұмыс істеу нәтижесі:

```
5 4 3 2 1
```

Кезектер

Кезек – бұл элемент қосу тізімнің бір басынан, ал таңдау екінші басынан орындалатын бірбағытты тізімнің дербес түрі. Кезектерге қолданылатын басқа опрерациялар анықталмаған. Таңдау кезінде элементті кезектен алып тастайды. Кезек FIFO (first in – first out, бірінші болып енеді – бірінші болып шығады) қызмет көрсету принципін жүзеге асырады. Мұның жұмыс істеу принципін кезекте бірнеше сағат тұру арқылы көзге жеңіл елестетуге болады. Программалауда кезектердің қолданылу мысалдары: модельдеу, операциялық жүйенің есептерін диспетчерлеу, буферленген енгізу/шығару әрекеттері.

Төмендегі программа 5 бүтін саннан тұратын кезек құрады және соларды экранға шығарады. Кезектің соңына орналастыру функциясы – **add**, ал таңдау функциясы – **del** деп аталады. Кезектің басына нұсқауыш – **pbeg**, ал оның соңына нұсқауыш – **pend** деп аталады.

```
#include <iostream.h>
struct Node{
   int d;
   Node *p;
};
Node * first (int d);
void add(Node **pend, int d);
int del(Node **pbeg);
//-------------------------------------------
int main(){
   Node *pbeg = first(1);
   Node *pend = pbeg;
   for (int i = 2; i<6; i++) add(&pend, i);
   while (pbeg)
      cout << del(&pbeg) << ' ';
   return 0;
}
//-------------------------------------------
// Кезекті бастапқы қалыптастыру
Node * first(int d){
   Node *pv = new Node;
   pv->d = d;
   pv->p = 0;
   return pv;
}
//-------------------------------------------
// Соңына элемент қосу
void add(Node **pend, int d){
   Node *pv = new Node;
   pv->d = d;
   pv->p = 0;
   (*pend)->p = pv;
   *pend = pv;
}
//-------------------------------------------
// Таңдау
int del(Node **pbeg){
   int temp = (*pbeg)->d;
   Node *pv = *pbeg;
   *pbeg = (*pbeg)->p;
```

```
    delete pv;
    return temp;
}
```

Программа жұмысының нәтижесі:

```
1 2 3 4 5
```

Бинарлы бұтақтар

Бинарлы бұтақ – бұл динамикалық мәліметтер құрылымы, ол түйіндерден тұрады, ал түйіндердің әрқайсысының құрамында мәліметтермен қатар, басқа бинарлы бұтақтарға екіден артпайтын сілтеме болады. Әрбір түйінге бір сілтемеден жасалады. Бастапқы түйін бұтақтың *түбірі* деп аталады.

3.3-суретте бинарлы бұтақтың мысалы көрсетілген (әдетте түбір жоғарыда суреттеледі, бірақ кітапты кері аударып қарасаңыз да болады)[1]. Ішкі бұтақтары болмайтын түйінді жапырақ деп атайды. Шығатын түйіндер ата-тегі, ал кіретін түйіндер ұрпақтары деп аталады. Бұтақтың биіктігі түйіндер орналасқан деңгейлер санымен анықталады.

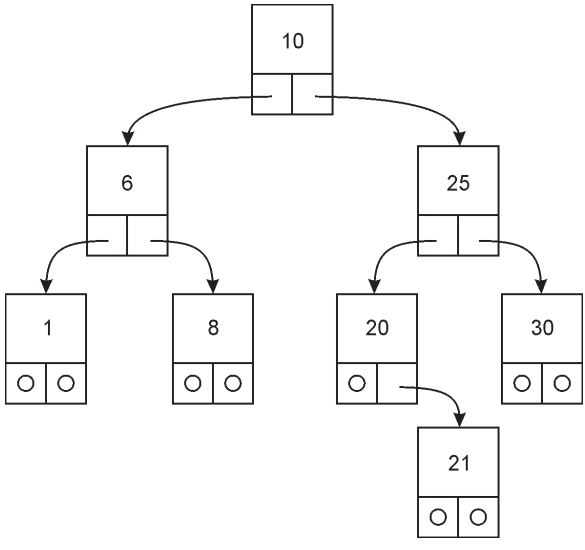
Егер бұтақта әрбір түйін үшін оның сол жақ ішкі бұтағының барлық кілттері осы түйін кілтінен кіші, ал оң жақ ішкі бұтағының кілттері одан үлкен болатындай етіп ұйымдастырылса, ондай бұтақты іздеу бұтағы деп атайды. Кілттердің бірдей болуы мүмкін емес. Іздеу бұтағында әрбір түбірдегі кілт мәніне байланысты түбірден оң жақ немесе сол жақ ішкі бұтағына қарай көше отырып, элементті оның кілті[2] арқылы табуға болады. Мұндай іздеу тізімнен іздегеннен гөрі тиімдірек, өйткені іздеу уақыты бұтақ биіктігі бойынша анықталады, ол түйіндер санының екілік логарифміне пропорционал.

Бұтақ рекурсивті мәліметтер құрылымы болып табылады, себебі оның әрбір ішкі бұтағы да бұтақ болып саналады. Мұндай құрылымдармен орындалатын әрекеттер рекурсивті алгоритмдердің көмегімен барынша бейнелі түрде сипатталады. Мысалы, бұтақтың барлық түйіндерін қарап шығу функциясын жалпы келесідей түрде сипаттауға болады:

```
function way_around ( бұтақ ){
    way_around ( сол жақ ішкі бұтақ )
    түбірге жету
    way_around ( оң жақ ішкі бұтақ )
}
```

[1]Жедел жадыда ұяшықтар адрестерінің өсу реті бойынша сызықтық түрде орналасқанын, ал бұтақ тек мәліметтерді логикалық ұйымдастыру әдісі екендігін ұмытпаған жөн.

[2]Сол жақ және оң жақ ішкі бұтақтардағы түйіндер санының айырмасы бірден артпайтын *теңдестірілген бұтақтың* биіктігі түйіндер санының екілік логарифміне тең. Сызықтық тізімді құрамындағы әрбір түйіннің бірден артық емес сілтемелері болатын ерекше бинарлы бұтақ ретінде қарастыру мүмкіндігі бар. Тізім үшін орташа іздеу уақыты тізімнің ұзындығының жартысына тең.



3.3-сурет. Бинарлы бұтақ

Бұтақты басқа реттілікпен де қарап шығуға болады, мысалы, алдымен түбір, содан кейін ішкі бұтақтарға көшу арқылы, мұндайда келтірілген функция нәтижесінде (шығысында) кілттердің реттелген тізбегін алуға мүмкіндік береді, өйткені алдымен сол жақ ішкі бұтақта орналасқан кілттері кіші төбелер қарастырылады. Төменде 3.3-суретте көрсетілген бұтақты қарап шығу нәтижесі берілген:

```
1, 6, 8, 10, 20, 21, 25, 30
```

Егер бұтақтарды қарап шығу функциясындағы оны алғашқы пайдалану оң жақ ішкі бұтаққа бағытталаған болса, онда оны қарап шығу нәтижесі басқаша болады:

```
30, 25, 21, 20, 10, 8, 6, 1
```

Осылайша, іздеу бұтағын мәндерді сұрыптау үшін де қолдануға болады. Бұтақтарды қарап шығуда түйіндер жойылмайды.

Бинарлы бұтақтар үшін анықталған операциялар:

- бұтаққа түйін қосу;
- бұтақ бойынша іздеу;
- бұтақты қарап шығу;
- түйінді жою.

Әрбір рекурсивті алгоритм үшін оның рекурсивті емес баламасын құруға болады. Төмендегі программада бұтақтан іздей отырып түйін қосуға арналған рекурсивті емес функция және бұтақтарды қарап шығуға арналған рекурсивті функция жүзеге асырылған. Бірінші функция берілген кілт бойынша іздеу

жүргізеді. Егер элемент табылса, функция осы элементке нұсқауышты қайтарады, кері жағдайда элементті бұтақтың тиісті жеріне орналастырады және оған нұсқауышты қайтарады. Элементті қосу үшін бұтақ бойымен бір қадам кейін өткен жолды есте сақтау керек және жаңа элементті қосу оның ата тегінің[1] оң немесе сол жақ ішкі бұтағына орындалатынын білу қажет.

Программа бүтін сандар жиымынан бұтақ қалыптастырады және оны экранға шығарады.

```
#include <iostream.h>
struct Node{
   int d;
   Node *left;
   Node *right;
};
Node * first(int d);
Node * search_insert(Node *root, int d);
void print_tree(Node *root, int l);
//-------------------------------------------------
int main(){
   int b[] = {10, 25, 20, 6, 21, 8, 1, 30};
   Node *root = first(b[0]);
   for (int i = 1; i<8; i++) search_insert(root, b[i]);
   print_tree(root, 0);
   return 0;
}
//-------------------------------------------------
// Бұтақтың алғашқы элементін қалыптастыру
Node * first(int d){
   Node *pv  = new Node;
   pv->d = d;
   pv->left = 0;
   pv->right = 0;
   return pv;
}
//-------------------------------------------------
// Қоса отырып іздеу
Node * search_insert(Node *root, int d){
   Node *pv = root, *prev;
   bool found = false;
   while (pv && !found){
      prev = pv;
      if      (d == pv->d) found = true;
      else if (d <  pv->d) pv    = pv->left;
      else                 pv    = pv->right;
```

[1]«Іздей отырып қосу алгоритміне сенімсіздік білдіру – заңды құбылыс» – классикалық «Алгоритмы + структуры данных = программы» кітабының [8] авторы Н.Вирт.

```
    }
    if (found) return pv;
    // Жаңа түйін құру:
    Node *pnew  = new Node;
    pnew->d     = d;
    pnew->left  = 0;
    pnew->right = 0;
    if (d < prev->d)
       // Тегінің сол жақ ішкі ағашына қосу:
       prev->left  = pnew;
    else
       // Тегінің оң жақ ішкі ағашына қосу:
       prev->right = pnew;
    return pnew;
}
//-----------------------------------------------
// Бұтақты қарап шығу
void print_tree(Node *p, int level){
    if (p){
       // сол жақ ішкі бұтақты шығару:
       print_tree(p->left, level + 1);
       for (int i = 0; i<level; i++) cout << "   ";
       // ішкі бұтақтың түбірін шығару:
       cout << p->d << endl;
       // оң жақ ішкі бұтақты шығару:
       print_tree(p->right, level + 1);
    }
}
```

Бұтақтан іздеуге арналған ағымдағы нұсқауыш **pv** арқылы, оның ата-тегіне нұсқауыш **prev** арқылы белгіленген, **pnew** айнымалысы бұтаққа қосылатын түйін үшін жады бөлуде қолданылады. Бір айнымалыны (**prev**) есте сақтап, жаңа түйінді қосу кезінде ол қай ішкі бұтаққа қосылатынын анықтайтын операторларды қайталау арқылы ол рекурсиясыз орындалды.

3.3-суретте бейнеленген бұтақ үшін программаның жұмыс істеу нәтижесі төмендегідей болады:

```
        1
    6
          8
10
        20
              21
  25
        30
```

Бұтақты қарап шығу функциясын толығырақ қарастырамыз. Оған екінші параметр ретінде түйіннің қай деңгейде жатқанын анықтайтын бүтін айнымалы беріледі. Түбір 0 деңгейінде орналасады. Бұтақ түбір сол жақта болатындай етіліп, көлденең бағытта басып шығарылады (программа жұмысының нәтижесіне басыңызды солға қарай еңкейтіп қараңыз және оны 3.3-суретпен салыстырыңыз). Бұтақтың құрылымын бейнелеу үшін түйін мәнінің алдында саны түйін деңгейіне пропорционал бос орындар шығарылады. Егер бос орындарды шығару циклін түсініктемеге алса, өсу ретімен сұрыпталған жиым бағана түрінде шығарылады. Бірнеше жолдан тұратын бұтақты қарап шығу функциясы кез келген өлшемдегі бұтақты басып шығара алады – тек стектің көлемі бұл ережеге бағынбайды.

Бұтақтан түйінді жою жеңіл орындалатын іс емес, өйткені жойылатын түйін түбірлік болуы немесе оның құрамында ішкі бұтақтарға екі, бір сілтеме болуы, тіпті сілтемелер болмауы да мүмкін. Құрамындағы сілтемелер саны екіден артпайтын түйіндерді жою қиындық тудырмайды. Екі сілтемесі бар түйінді жойған кезде бұтақтың реттелген күйін сақтап қалу үшін, бұл түйінді оған кілті барынша ең жақын түйінмен алмастырады. Бұл оның сол жақ ішкі бұтағының ең оң жақтағы түйіні немесе оң жақ ішкі бұтағының ең сол жағындағы түйіні болуы мүмкін (мысалы, 3.3-суреттегі бұтақтан кілті 25 болатын түйінді жою үшін оны 21 немесе 30 кілті бар түйіндерге алмастыру керек, ал 10 түйіні 20 немесе 8 түйіндеріне алмастырылады, т.с.с). Бұтақтан түйінді жою функциясының жүзеге асырылуы оқырманға өзіндік жұмыс ретінде қалдырылған.

Динамикалық құрылымдарды жиым көмегімен жүзеге асыру

Жадыны бөлу және босату операциялары – қымбатқа түсетін әрекеттер, сондықтан мәліметтің керекті үлкен көлемін оны қолданудан бұрын анықтау мүмкін болып, ол жұмыс барысында өзгермейтініні белгілі болса (мысалы, файл құрамын сұрыптауда), онда оған жадының үздіксіз аймағын бөлу әлдеқайда тиімді болар еді. Мұндайда элементтер арасындағы байланыстар нұсқауыштар арқылы емес, элементтер нөмірлері сақталатын қосалқы айнымалылар немесе жиымдар арқылы жүзеге асырылады.

Стекті осылайша жүзеге асыру жеңіл болады. Стектің мәліметтер типіне сәйкес келетін элементтер жиымына қосымша тек стектің төбесі болып табылатын жиым элементінің индексін сақтауға арналған бүтін типті бір айнымалы болғаны жеткілікті. Стекке орналастыру кезінде индекс бірге артып отырады да, ал таңдау кезінде ол бірге кемиді.

Кезекті жүзеге асыру үшін екі бүтін типті айнымалы – кезектің басы және соңы болатын жиым элементтерінің индекстерін сақтау үшін қажет.

Сызықтық тізімді құру үшін қосымша бүтін сандар жиымы және бір айнымалы керек, мысалы:

```
10  25  20  6  21  8  1  30 – мәліметтер жиымы
1   2   3   4  5   6  7  -1 – қосымша жиым
0 - тізімдегі алғашқы элементтің индексі
```

Қосымша жиымның **i** нөмірлі элементінде мәліметтер жиымының әрбір **i** нөмірлі элементі үшін оның келесі элементінің индексі сақталады. Теріс сан тізімнің соңының белгісі ретінде қолданылады. Бастапқы жиым сұрыптаудан кейін төмендегі түрде болады:

```
10  25  20  6  21  8  1  30 - мәліметтер жиымы
2   7   4   5  1   0  3  -1 - қосымша жиым
6 - тізімдегі алғашқы элементтің индексі
```

Бинарлы бұтақ құру үшін екі қосымша жиымды қолдануға болады (оның оң және сол жақ ішкі бұтақтары төбелерінің индекстері). Теріс сан бос сілтеменің белгісі ретінде қолданылады. Мысалы, 3.3-суретте көрсетілген бұтақты келесі түрде бейнелеуге болады:

```
10  25  20  6   21   8   1   30 – мәліметтер жиымы
3   2   -1  6   -1   -1  -1  -1 – сол жақ сілтеме
1   7   4   5   -1   -1  -1  -1 - оң жақ сілтеме
```

Егер жады көлемін тұрақты арқылы беруге болатын болса, онда мұндай құрылымдар үшін, компьютер жадын компиляция кезінде бөлуге болады немесе оны программаны орындау барысында да атқаруға болады, мысалы:

```
struct Node{
   Data d;   // Data мәліметтер типі
             // алдын ала анықталуы керек
   int i;
};
Node spisok1[1000];              // компиляция кезінде
Node *pspisok2 = new Node[m];     // орындалу кезінде
```

ЕСКЕРТУ

Осындай құрылымдармен жұмыс барысында индестердің жиым шекарасынан асып кетпеуін қадағалау керек.

Жоғарыда көрсетілген жүзеге асыру тәсілі динамикалық құрылымдардың (мысалға, жадыда физикалық орын ауыстырусыз, көлемді элементтерден тұратын құрылымдарды сұрыптау) артықшылықтарын қолдануға мүмкіндік бере отырып, әрбір мәліметтер элементі үшін жады бөлуге және босатуға артық уақыт жұмсамайды.

Динамикалық мәліметтер құрылымдарымен жұмыс істеу практикумның [11] тоғызыншы семинарында түбегейлі түрде қарастырылған.

I бөлімге арналған жаттығулар

Циклдік есептеу процестері

Осы тақырыпқа байланысты лабораториялық жұмыстарды іске асыратын программалар практикумның [11] 37 б. берілген.

1-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} ax^2+b, & x<0 \text{ және } b\neq 0 \text{ болғанда} \\ \dfrac{x-a}{x-c}, & x>0 \text{ және } b=0 \text{ болғанда} \\ \dfrac{x}{c}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер (Ац НЕМЕСЕ Вц) ЖӘНЕ (Ац НЕМЕСЕ Сц) өрнегі нөлге тең болмаса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЖӘНЕ мен НЕМЕСЕ операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

2-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} \dfrac{1}{ax}-b, & x+5<0 \text{ және } c=0 \text{ болғанда} \\ \dfrac{x-a}{x}, & x+5>0 \text{ және } c\neq 0 \text{ болғанда} \\ \dfrac{10x}{c-4}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер (Ац ЖӘНЕ Вц) НЕМЕСЕ (Вц ЖӘНЕ Сц) өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц

және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЖӘНЕ мен НЕМЕСЕ операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, a, b, c мәндерін пернетақтадан енгізіңдер.

3-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} ax^2+bx+c, & a<0 \text{ және } c\neq 0 \text{ болғанда} \\ \dfrac{-a}{x-c}, & a>0 \text{ және } c=0 \text{ болғанда} \\ a(x+c), & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер Ац ЖӘНЕ (Вц НЕМЕСЕ Сц) өрнегі нөлге тең емес болса, F функциясы нақты мәнді қабылдауы керек, кері жағдайда бүтін мәнді қабылдау керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЖӘНЕ мен НЕМЕСЕ операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX*, a, b, c* мәндерін пернетақтадан енгізіңдер.

4-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} -ax-c, & c<0 \text{ және } x\neq 0 \text{ болғанда} \\ \dfrac{x-a}{-c}, & c>0 \text{ және } x=0 \text{ болғанда} \\ \dfrac{bx}{c-a}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер Ац НЕМЕСЕ Вц НЕМЕСЕ Сц өрнегі нөлге тең емес болса, F функциясы нақты мәндерді қабылдауы керек, кері жағдайда бүтін мәнді қабылдау керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЖӘНЕ мен НЕМЕСЕ операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

5-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} a-\dfrac{x}{10+b}, & x<0 \text{ және } b\neq 0 \text{ болғанда} \\ \dfrac{x-a}{x-c}, & x>0 \text{ және } b=0 \text{ болғанда} \\ 3x+\dfrac{2}{c}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер (Ац НЕМЕСЕ Вц) ЖӘНЕ Сц өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЖӘНЕ мен НЕМЕСЕ операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

6-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} ax^2+b^2x, & c<0 \text{ және } b\neq 0 \text{ болғанда} \\ \dfrac{x+a}{x+c}, & c>0 \text{ және } b=0 \text{ болғанда} \\ \dfrac{x}{c}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер (Ац ЖӘНЕ Вц) НЕМЕСЕ (Ац ЖӘНЕ Сц) өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЖӘНЕ мен НЕМЕСЕ операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

7-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} -ax^2-b, & x<5 \text{ және } c\neq 0 \text{ болғанда} \\ \dfrac{x-a}{x}, & x>5 \text{ және } c=0 \text{ болғанда} \\ \dfrac{-x}{c}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер (Ац НЕМЕСЕ Вц) МОД2 (Ац НЕМЕСЕ Сц) өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЖӘНЕ, НЕМЕСЕ, МОД2 (2 модулі бойынша қосу) операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

8-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} -ax^2, & c<0 \text{ және } a\neq 0 \text{ болғанда} \\ \dfrac{a-x}{cx}, & c>0 \text{ және } a=0 \text{ болғанда} \\ \dfrac{x}{c}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер (Ац МОД2 Вц) ЖӘНЕ ЕМЕС (Ац НЕМЕСЕ Сц) өрнегі нөлге тең болмаса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЖӘНЕ, НЕМЕСЕ, МОД2 (2 модулі бойынша қосу) операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

9-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} ax^2+b^2x, & a<0 \text{ және } x\neq 0 \text{ болғанда} \\ x-\dfrac{a}{x-c}, & a>0 \text{ және } x=0 \text{ болғанда} \\ 1+\dfrac{x}{c}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер ЕМЕС(Ац НЕМЕСЕ Вц) ЖӘНЕ (Ац НЕМЕСЕ Сц) өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЖӘНЕ, НЕМЕСЕ, ЕМЕС операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

10-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} ax^2-bx+c, & x<3 \text{ және } b\neq 0 \text{ болғанда} \\ \dfrac{x-a}{x-c}, & x>3 \text{ және } b=0 \text{ болғанда} \\ \dfrac{x}{c}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер ЕМЕС(Ац НЕМЕСЕ Вц) ЖӘНЕ (Ац МОД2 Сц) өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЖӘНЕ, НЕМЕСЕ, ЕМЕС, МОД2 (2 модулі бойынша қосу) операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

11-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} ax^2+\dfrac{b}{c}, & x<1 \text{ және } c\neq 0 \text{ болғанда} \\ \dfrac{x-a}{(x-c)^2}, & x>1 \text{ және } c=0 \text{ болғанда} \\ \dfrac{x^2}{c^2}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер (Ац ЖӘНЕ Вц) МОД2 Сц өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЖӘНЕ, МОД2 (2 модулі бойынша қосу) операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

12-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} ax^3+b^2+c, & x<0{,}6 \text{ және } b+c\neq 0 \text{ болғанда} \\ \dfrac{x-a}{x-c}, & x>0{,}6 \text{ және } b+c=0 \text{ болғанда} \\ \dfrac{x}{c}+\dfrac{x}{a}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер (Ац НЕМЕСЕ Вц) ЖӘНЕ Сц өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЖӘНЕ, НЕМЕСЕ операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

13-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F = \begin{cases} ax^2 + b, & x\text{-}1<0 \text{ және } b\text{-}x\neq 0 \text{ болғанда} \\ \dfrac{x-a}{x}, & x\text{-}1>0 \text{ және } b+x=0 \text{ болғанда} \\ \dfrac{x}{c}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер (Ац НЕМЕСЕ Вц) МОД2 (Вц ЖӘНЕ Сц) өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЖӘНЕ, НЕМЕСЕ, МОД2 (2 модулі бойынша қосу) операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

14-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F = \begin{cases} -ax^3 - b, & x+c<0 \text{ және } a\neq 0 \text{ болғанда} \\ \dfrac{x-a}{x-c}, & x+c>0 \text{ және } a=0 \text{ болғанда} \\ \dfrac{x}{c}+\dfrac{c}{x}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер (Ац МОД2 Вц) НЕМЕСЕ (Ац МОД2 Сц) өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, НЕМЕСЕ, МОД2 (2 модулі бойынша қосу) операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

15-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} -ax^2+b, & x<0 \text{ және } b\neq 0 \text{ болғанда} \\ \dfrac{x}{x-c}+5.5, & x>0 \text{ және } b=0 \text{ болғанда} \\ \dfrac{x}{-c}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер ЕМЕС(Ац НЕМЕСЕ Вц НЕМЕСЕ Сц) өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, НЕМЕСЕ және ЕМЕС операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

16-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} a(x+c)^2-b, & x=0 \text{ және } b\neq 0 \text{ болғанда} \\ \dfrac{x-a}{-c}, & x=0 \text{ және } b=0 \text{ болғанда} \\ a+\dfrac{x}{c}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер (Ац МОД2 Вц) ЖӘНЕ ЕМЕС(Ац НЕМЕСЕ Сц) өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, НЕМЕСЕ, ЕМЕС, ЖӘНЕ, МОД2 (2 модулі бойынша қосу) операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

17-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F = \begin{cases} a\,x^2 - cx + b, & x+10<0 \text{ және } b\neq 0 \text{ болғанда} \\ \dfrac{x-a}{x-c}, & x+10>0 \text{ және } b=0 \text{ болғанда} \\ \dfrac{-x}{a-c}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер (Ац НЕМЕСЕ Вц) ЖӘНЕ ЕМЕС(Ац НЕМЕСЕ Сц) өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, НЕМЕСЕ, ЕМЕС, ЖӘНЕ операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

18-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F = \begin{cases} a\,x^3 + bx^2, & x<0 \text{ және } b\neq 0 \text{ болғанда} \\ \dfrac{x-a}{x-c}, & x>0 \text{ және } b=0 \text{ болғанда} \\ \dfrac{x+5}{c\,(x-10)}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер ЕМЕС(Ац ЖӘНЕ Вц ЖӘНЕ Сц) өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, ЕМЕС, ЖӘНЕ операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

19-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} a\,(x+7)^2-b, & x<5 \text{ және } b\neq 0 \text{ болғанда} \\ \dfrac{x-cd}{ax}, & x>5 \text{ және } b=0 \text{ болғанда} \\ \dfrac{x}{c} & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер (Ац МОД2 Вц) НЕМЕСЕ (Ац МОД2 Сц) өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, НЕМЕСЕ, МОД2 (2 модулі бойынша қосу) операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

20-нұсқа

Хбас. және Хсоң. аралығындағы F функциясының мәндерін dX қадамымен есептеп, кесте түрінде экранға шығару.

$$F=\begin{cases} -\dfrac{2x-c}{cx-a}, & x<0 \text{ және } b\neq 0 \text{ болғанда} \\ \dfrac{x-a}{x-c}, & x>0 \text{ және } b=0 \text{ болғанда} \\ -\dfrac{x}{c}+\dfrac{-c}{2x}, & \text{қалған жағдайларда} \end{cases}$$

мұндағы *a, b, c* – нақты сандар.

Егер ЕМЕС(Ац НЕМЕСЕ Вц) ЖӘНЕ ЕМЕС(Ац НЕМЕСЕ Сц) өрнегі нөлге тең емес болса, F функциясы нақты мән, кері жағдайда бүтін мән қабылдауы керек. Ац, Вц және Сц арқылы *a, b, c* мәндерінің бүтін бөліктері белгіленген, НЕМЕСЕ, ЕМЕС, ЖӘНЕ операциялары разрядтар бойынша орындалады. Хбас., Хсоң., dX, *a, b, c* мәндерін пернетақтадан енгізіңдер.

Бірөлшемді жиымдар

Бұл тақырып бойынша лабораториялық жұмыстар тапсырмаларын жүзеге асыратын программалар практикумның [11] 55 б. келтірілген.

1-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның теріс элементтерінің қосындысын;
2) жиымның ең үлкен (максимал) және ең кіші (минимал) элементтерінің аралығында орналасқан элементтерінің көбейтіндісін анықтаңыз;

Жиым элементтерін өсуі бойынша реттеңдер.

2-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның оң элементтерінің қосындысын;
2) жиымның модулі бойынша ең үлкен (максимал) және модулі бойынша ең кіші (минимал) элементтерінің аралығында орналасқан элементтерінің көбейтіндісін анықтаңыз;

Жиым элементтерін кемуі бойынша реттеңдер.

3-нұсқа

n бүтін элементтерден тұратын бірөлшемді жиымда:

1) жиымның жұп нөмірлі элементтерінің көбейтіндісін;
2) жиымның алғашқы және екінші нөлге тең екі элементінің аралығында орналасқан элементтерінің қосындысын анықтаңыз;

Жиымның барлық оң элементтері оның алдыңғы жағында, ал теріс элементтері соңғы жағында орналасатындай етіп түрлендіріңдер.

4-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның тақ нөмірлі элементтерінің қосындысын;
2) жиымның алғашқы және ең соңғы теріс екі элементінің аралығында орналасқан элементтерінің қосындысын анықтаңыз;

Жиымның модульдері 1-ден кіші элементтерін жойып, қысқартыңыз да, жиымның соңында бос қалған элементтерін нөлдермен толтырыңыз.

5-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның ең үлкен элементін;
2) жиымның алғашқы теріс элементіне дейін орналасқан элементтерінің қосындысын анықтаңыз;

Жиымның модульдері [a, b] аралығында орналасқан элементтерін жойып, қысқартыңыз да, жиымның соңында бос қалған элементтерін нөлдермен толтырыңыз.

6-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның ең кіші элементі мен оның нөмірін;
2) жиымның ең алғашқы және ең соңғы оң элементтері аралығында орналасқан элементтерінің қосындысын анықтаңыз;

Жиымның алдыңғы жағында тек нөлге тең элементтері, сонан кейін барып басқалары орналасатындай етіп түрлендіріңдер.

7-нұсқа

n бүтін элементтерден тұратын бірөлшемді жиымда:

1) жиымның ең үлкен элементі мен оның нөмірін;
2) жиымның ең алғашқы және ең соңғы нөлге тең элементтері аралығында орналасқан элементтерінің көбейтіндісін анықтаңыз;

Жиымның бірінші жартысында – оның тақ нөмірлі элементтері, ал екінші жартысында – жұп нөмірлі элементтері орналасатындай етіп түрлендіріңдер.

8-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның ең кіші элементінің нөмірін;
2) жиымның ең алғашқы және одан кейінгі теріс элементтері аралығында орналасқан элементтерінің қосындысын анықтаңыз;

Жиымның алғашқы жағында – оның модульдері 1-ден кіші элементтері, ал соңында – қалған элементтері орналасатындай етіп түрлендіріңдер.

9-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның модулі бойынша ең үлкен элементін;
2) жиымның ең алғашқы және одан кейінгі оң элементтері аралығында орналасқан элементтерінің қосындысын анықтаңыз;

Жиымның нөлге тең элементтері оның соңғы жағында орналасатындай етіп түрлендіріңдер.

10-нұсқа

n бүтін элементтерден тұратын бірөлшемді жиымда:

1) жиымның модулі бойынша ең кіші элементін;
2) жиымның нөлге тең алғашқы элементінен кейінгі элементтерінің модульдерінің қосындысын анықтаңыз;

Жиымның бірінші жартысында – оның жұп нөмірлі элементтері, ал екінші жартысында – тақ нөмірлі элементтері орналасатындай етіп түрлендіріңдер.

11-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның модулі бойынша ең кіші элементі нөмірін;

2) жиымның алғашқы теріс элементінен кейін орналасқан элементтерінің модульдері қосындысын анықтаңыз;

Жиымның мәндері [a, b] аралығында орналасқан элементтерін жойып, қысқартыңыз да, жиымның соңында бос қалған элементтерін нөлдермен толтырыңыз.

12-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның модулі бойынша ең үлкен элементі нөмірін;
2) жиымның алғашқы оң элементінен кейін орналасқан элементтерінің қосындысын анықтаңыз;

Жиымның мәндерінің бүтін бөлігі [a, b] аралығында орналасқан элементтерін алдыңғы жағына, қалғандарын – соңғы жағына орналасатындай етіп түрлендіріңдер.

13-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның мәндері A мен B аралығында жатқан элементтері санын;
2) жиымның ең үлкен элементінен кейін орналасқан элементтерінің қосындысын анықтаңыз;

Жиым элементтерін оның модульдерінің кемуі бойынша реттеңдер.

14-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның мәндері нөлге тең элементтері санын;
2) жиымның ең кіші элементінен кейін орналасқан элементтерінің қосындысын анықтаңыз;

Жиым элементтерін оның модульдерінің өсуі бойынша реттеңдер.

15-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның – мәндері берілген C санынан үлкен элементтерінің қанша екенін;
2) жиымның модулі бойынша ең үлкен элементінен кейін орналасқан элементтерінің көбейтіндісін анықтаңыз;

Жиымның барлық теріс элементтері оның алдыңғы жағында, ал оң элементтері (нөлге тең элементті оң деп санау керек) соңғы жағында орналасатындай етіп түрлендіріңдер.

16-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның мәндері теріс элементтерінің қанша екенін;
2) жиымның модулі бойынша ең кіші элементінен кейін орналасқан элементтері модульдерінің қосындысын анықтаңыз;

Жиымның барлық теріс элементтерін олардың квадраттарымен алмастырып, шыққан жиым элементтерін өсуі бойынша реттеңдер.

17-нұсқа

n бүтін элементтерден тұратын бірөлшемді жиымда:

1) жиымның оң элементтерінің қанша екенін;
2) жиымның соңғы нөлге тең элементінен кейін орналасқан элементтерінің қосындысын анықтаңыз;

Жиымның алғашқы жағында – оның мәндерінің бүтін бөлігі 1-ден кіші элементтері, ал соңында – қалған элементтері орналасатындай етіп түрлендіріңдер.

18-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның – мәндері берілген `t` санынан кіші элементтерінің қанша екенін;
2) жиымның соңғы теріс элементінен кейін орналасқан элементтері мәндерінің бүтін бөліктерінің қосындысын анықтаңыз;

Жиымның алғашқы жағында – оның ең үлкен мәнінен 20%-ға алшақ мәндері бар элементтері, ал соңында – қалған элементтері орналасатындай етіп түрлендіріңдер.

19-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның теріс элементтерінің көбейтіндісін;
2) жиымның ең үлкен элементіне дейін орналасқан оң элементтерінің қосындысын анықтаңыз;

Жиым элементтерін олар керісінше орналасатындай етіп түрлендіріңдер.

20-нұсқа

n нақты элементтерден тұратын бірөлшемді жиымда:

1) жиымның оң элементтерінің көбейтіндісін;
2) жиымның ең кіші элементіне дейін орналасқан элементтерінің қосындысын анықтаңыз;

Жиым элементтерінің жұп орында тұрғандарын және тақ орында тұрғандарын жеке-жеке өсуі бойынша реттеп шығыңыз.

Екіөлшемді жиымдар

Бұл тақырып бойынша лабораториялық жұмыстар тапсырмаларын жүзеге асыратын программалар практикумның [11] 71 б. келтірілген.

1-нұсқа

Берілген тікбұрышты бүтін сандық матрицада:

1) нөлдік элементі жоқ жолдар санын;

2) берілген матрицада бір реттен артық кездесетін сандардың ішіндегі ең үлкенін анықтаңыз.

2-нұсқа

Тікбұрышты бүтін сандық матрица берілген. Нөлдік элементі жоқ бағаналар санын анықтау керек.

Бүтін сандық матрица жолының сипаттамасы деп оның оң жұп элементтерінің қосындысын атайық. Берілген матрицаның жолдарының орнын ауыстыра отырып, оларды сипаттамаларының артуы бойынша орналастырыңыз.

3-нұсқа

Берілген тікбұрышты бүтін сандық матрицада:

1) кемінде бір нөлдік элементі бар бағаналар санын;
2) бірдей элементтердің ең ұзақ тізбегі орналасқан жолдың нөмірін анықтаңыз.

4-нұсқа

Берілген квадратты бүтін сандық матрицада:

1) теріс элементтері жоқ жолдардағы элементтер көбейтіндісін;
2) матрицаның бас диагоналына параллель орналасқан диагональдердің элементтерінің қосындыларының арасындағы ең үлкенін анықтаңыз.

5-нұсқа

Берілген квадратты бүтін сандық матрицада:

1) теріс элементтері жоқ бағандардағы элементтердің қосындысын;
2) матрицаның қосалқы диагоналына параллель орналасқан диагональдарының элементтері модульдерінің қосындыларының ішіндегі ең кішісін анықтаңыз.

6-нұсқа

Берілген бүтін сандық тікбұрышты матрицада:

1) кемінде бір теріс элементі бар жолдардағы элементтерінің қосындысын;
2) матрицаның барлық қайқы нүктелерінің жолдары мен бағандарының нөмірлерін анықтаңыз.

ЕСКЕРТУ

Егер матрицаның A_{ij} элементі *i*-ші жолдағы ең кіші және *j*-ші бағанадағы ең үлкен элементі болса, онда ол A матрицасының қайқы нүктесі болып табылады.

7-нұсқа

Берілген 8x8 өлшемді матрица үшін оның *k* нөмірлі жолы *k* нөмірлі бағанасымен бірдей болып келетін *k* мәндерін табыңыз.

Матрицаның кемінде бір теріс элементі бар жолдарындағы элементтерінің қосындысын табыңыз.

8-нұсқа

Бүтін сандық матрица бағанасының сипаттамасы деп теріс тақ элементтерінің модульдерінің қосындысын атайық. Берілген матрица бағаналарының реттілігін сипаттамаларының артуы бойынша орналастырыңыз.

Матрицаның кемінде бір теріс элементі бар бағаналарындағы элементтерінің қосындысын табыңыз.

9-нұсқа

Матрицадағы A_{ij} элементінің көршілері деп $\mathbf{i-1 \le k \le i+1,\ j-1 \le l \le j+1,\ (k,\ l) \ne (i,\ j)}$ теңсіздіктері орындалатын A_{kl} элементтерін атаймыз. Матрицаны тегістеу операциясы деп өлшемі бастапқы матрицамен бірдей, бірақ оның әрбір элементі бастапқы матрицаның сәйкес элементі көршілерінің арифметикалық ортасы ретінде анықталатын жаңа матрицаны айтады. Берілген 10x10 өлшемді нақты матрицаға тегістеу операциясын қолданып, жаңа матрица құру керек.

Тегістелген матрицадағы бас диагональдан төмен орналасқан элементтер модульдерінің қосындысын табыңыз.

10-нұсқа

Егер матрицаның элементі оның барлық көршілерінен міндетті түрде кіші болып келетін болса, онда ол жергілікті минимум деп аталады. Берілген 10x10 өлшемді матрицаның жергілікті минимумдарының санын табыңыз.

Матрицаның бас диагоналынан жоғары орналасқан элементтері модульдерінің қосындысын табыңыз.

11-нұсқа

Сызықтық теңдеулер жүйесінің коэффициенттері тікбұрышты матрица түрінде берілген. Орындауға болатын түрлендірулер көмегімен жүйені үшбұрышты түрге келтіріңдер.

Матрица элементтерінің арифметикалық ортасы берілген шамадан кіші болатын жолдар санын табыңыз.

12-нұсқа

Берілген матрицадағы нөлдерден тұратын жолдар мен бағаналарды алып тастау арқылы оны тығыздап ықшамдау керек.

Кемінде бір оң элементі бар алғашқы жолдың нөмірін табыңыз.

13-нұсқа

Тікбұрышты матрица элементтерін *n* элементке оңға немесе төмен қарай (көрсетілген бағытқа байланысты) циклдік түрде ығыстыру керек. *n* санының мәні жолдағы немесе бағанадағы элементтер санынан артық болуы мүмкін.

14-нұсқа

MxN өлшемді квадрат матрица элементтерін *k* элементке оңға қарай циклдік ығыстыруды келесідей түрде жүзеге асыру керек: 1-ші жол элементтері соңғы бағанаға жоғарыдан төмен қарай орналастырылады, сол бағана элементтері соңғы жолға оңнан солға қарай, сол жол элементтері бірінші бағанаға төменнен жоғарыға, сол бағанадағы элементтер бірінші жолға ығыстырылып орналастырылады; қалған элементтер үшін дәл осы үлгідегі әрекеттер орындалуы тиіс.

15-нұсқа

Бүтін сандардан тұратын тікбұрышты матрица берілген. Кемінде бір нөлдік элементі бар алғашқы бағананың нөмірін анықтаңыз.

Бүтін сандық матрица жолдарының сипаттамасы деп оның теріс жұп элементтерінің қосындысын атаймыз. Берілген матрица жолдарының орындарын ауыстыра отырып, оларды сипаттамаларының кемуі бойынша орналастырыңыз.

16-нұсқа

Бүтін сандық тікбұрышты матрицаның жолдарын әр жолдағы шамалары бірдей элементтер санының өсу реті бойынша орналастырыңыз.

Теріс элементтері жоқ алғашқы бағана нөмірін анықтаңыз.

17-нұсқа

Нақты мәнді квадрат матрица элементтерінің орнын ауыстыру арқылы ең үлкен элемент сол жақ жоғарғы бұрышта, үлкендігі бойынша келесі элемент (2,2) позициясында, үлкендігі бойынша одан кейінгі элемент (3,3) позициясында және т.с.с. орналасатындай етіп бас диагональді толтыру керек.

Матрицадағы оң элементтері жоқ алғашқы жол нөмірін анықтаңыз.

18-нұсқа

Берілген бүтін сандық тікбұрышты матрицада:

1) кемінде бір нөлдік элементі бар жолдардың санын;
2) қатар тұрған бірдей элементтердің ең ұзақ тізбегі орналасқан бағана нөмірін анықтаңыз.

19-нұсқа

Берілген квадратты бүтін сандық матрицада:

1) теріс элементтері жоқ жолдардағы элементтер қосындысын;
2) матрицаның бас диагоналына параллель орналасқан диагональдардағы элементтер қосындыларының ішіндегі ең кішісін анықтаңыз.

20-нұсқа

Берілген бүтін сандық тікбұрышты матрицада:

1) кемінде бір нөлдік элементі бар жолдардағы теріс элементтер санын;
2) матрицаның барлық қайқы нүктелері орналасқан жолдары мен бағаналарының нөмірлерін анықтаңыз.

ЕСКЕРТУ:

Егер матрицаның A_{ij} элементі *i*-ші жолдағы ең кіші және *j*-ші бағанадағы ең үлкен элементі болса, онда ол А матрицасының қайқы нүктесі болып табылады.

Бірөлшемді және екіөлшемді жиымдар

1-нұсқа

Жазықтықта екі дөңес көпбұрыш төбелерінің координаталары арқылы төбелерінің тізбектеле орналасқан (шекараны бойлау) реттілігімен берілген. Көпбұрыштардың аудандарын анықтап, олардың біреуі екіншісінің ішіне толығымен кіретіндігін анықтау қажет.

2-нұсқа

Жазықтықта берілген нүктелер жиынынан әртүрлі үш нүктені келесі шартты қанағаттандыратындай етіп таңдаңыз: осы үш нүкте арқылы өтетін шеңбермен шектелген дөңгелектің ауданы мен төбелері осы үш нүктеде орналасқан үшбұрыш ауданының айырмасы минималды болуы тиіс.

3-нұсқа

Жазықтықта екі нүктелер жиыны берілген. Бірінші жиыннан әртүрлі үш нүктені осы нүктелер арқылы өтетін шеңбермен шектелген дөңгелек екінші жиынның барлық нүктелерін қамтитындай және оның ауданы минималды болатындай етіп таңдау керек.

4-нұсқа

Жазықтықта екі нүктелер жиыны берілген. Бірінші жиыннан әртүрлі төрт нүктені, төбелері осы нүктелерде жататын шаршы екінші жиынның барлық нүктелерін қамтитындай және ауданы минималды болатындай етіп таңдау керек.

5-нұсқа

Жазықтықта екі нүктелер жиыны берілген. Бірінші жиыннан әртүрлі үш нүктені төбелері осы нүктелерде жататын үшбұрыш екінші жиынның барлық нүктелерін қамтитындай және ауданы минималды болатындай етіп таңдау керек.

6-нұсқа

Жазықтықта екі нүктелер жиыны берілген. Бірінші жиынның *n* (n>=3) нүктелерінен өтетін және өз құрамында бірінші мен екінші жиындардың сандары бірдей нүктелерін қамтитындай шеңбердің центрі мен радиусын табыңыз.

7-нұсқа

Жазықтықта екі нүктелер жиыны берілген. Бірінші жиыннан әртүрлі үш нүктені келесі шартты қанағаттандыратындай етіп таңдаңыз: төбелері осы

нүктелерде жататын үшбұрыш бірінші және екінші жиындарынан алынған бірдей нүктелер санынан тұру керек.

8-нұсқа

Жазықтықта М нүктелер жиыны мен дөңгелек берілген. М жиынынан екі нүктені келесі шартты қанағаттандыратындай етіп таңдаңыз: осы нүктелер арқылы өтетін түзуден екі жағында жатқан дөңгелектегі нүктелер сандарының айырмасы минималды болуы тиіс.

9-нұсқа

Жазықтықта *3n* нүкте берілген, олардың ішіндегі ешқандай үш нүкте бір түзудің бойында жатпайды. Төбелері осы нүктелерде жататын *n* үшбұрыштардың жиынын, ешқандай екі үшбұрыш қиылыспайтындай және бір-бірінің ішінде орналаспайтындай етіп құру керек.

10-нұсқа

Жазықтықта берілген нүктелер жиынынан әртүрлі үш нүктені келесі шарттар орындалатындай етіп таңдаңыз: төбелері осы нүктелерде жататын үшбұрыштың ішінде және одан тыс жататын нүктелер сандарының айырмасы минималды болуы керек.

11-нұсқа

Жазықтықта берілген нүктелер жиынынындағы кемінде әртүрлі үш нүкте арқылы өтетін және өз ішінде осы жиынның барынша көп нүктелерін қамтитын шеңбердің радиусы мен центрін анықтаңыз.

12-нұсқа

Жазықтықта А нүктелер жиыны мен онда жатпайтын *d* нүктесі берілген. *abcd* төртбұрышы параллелограмм болатындай А жиынындағы әртүрлі реттелмеген *a, b, c* үштіктерінің санын анықтау қажет.

13-нұсқа

Жазықтықта А нүктелер жиыны мен В шеңберлер жиыны берілген. А жиынынан әртүрлі екі нүктені келесі шарт орындалатындай етіп таңдаңыз: осы екі нүкте арқылы өтетін түзу В жиынындағы шеңберлердің максималды санымен қиылысуы керек.

14-нұсқа

Жазықтықта нүктелер жиыны берілген. Осы жиыннан шаршылардың төбелері болып табылатын барлық төрт нүктелерді табу керек. Ішінде жиынның ең көп нүктелер саны жататын шаршыны анықтаңыз.

15-нұсқа

Жазықтықта берілген нүктелер жиынындағы кемінде әртүрлі үш нүкте арқылы өтетін шеңберлер сызып, солардың ішіндегі ең кіші радиусы бар шеңбердің центрін табу қажет.

16-нұсқа

Төбелері жазықтықтағы берілген нүктелер жиынында жататын үш үшбұрышты келесі шарт орындалатындай етіп құрыңыз: екінші үшбұрыш бірінші үшбұрыштың, ал үшінші үшбұрыш екінші үшбұрыштың ішінде жатуы тиіс.

17-нұсқа

Жазықтықта нүктелер жиыны берілген. Төбелері осы нүктелер жиынында жататын барлық мүмкін болатын үшбұрыштарды құрып, солардың ішінен қабырғалары үшбұрыштардың максималды санымен қиылысатынын табу керек.

18-нұсқа

Жазықтықта нүктелер жиыны мен радиусы R және центрі координаталар басында жататын шеңбер берілген. Төбелері берілген жиын нүктелерінде жататын және үш қабырғасы да берілген шеңбермен қиылысатындай барлық үшбұрыштар жиынын құрып, солардың ішіндегі ең ауданы кіші үшбұрышты табу керек.

19-нұсқа

Қабырғаларының ұзындықтары әртүрлі және төбелері жазықтықта берілген жиын нүктелерінде жататын тең қабырғалы үшбұрыштар санын есептеп, солардың қиылысатынын немесе қиылыспайтынын анықтау керек.

20-нұсқа

Үшөлшемді кеңістікте нүктелер жиыны берілген. Әрқайсысы берілген әрбір үш нүктенің екеуін басып өтетіндей түрде екі жазықтық тұрғызып, осындай жазықтықтар ішінен бір-біріне параллель емес жазықтықтардың максималды ішкі жиынын табу керек.

Құрылымдар

Осы тақырыпқа байланысты лабораториялық жұмыстарды жүзеге асыратын программалар практикумның [11] 110 б. берілген.

1-нұсқа

1. Келесі өрістерден тұратын STUDENT атты құрылымды сипаттаңыз:
   - тегі мен инициалдары;
   - топтың нөмірі;
   - үлгерімі (бес элементтен тұратын жиым).
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - STUDENT типті он құрылымнан тұратын жиымке пернетақтадан мәліметтер енгізу; жазбалар топ нөмірінің өсуі бойынша реттеліп орналасуы керек;

- □ жиымға енген барлық студенттердің орташа балы 4,0-ден үлкен болса, олардың тегі мен топ нөмірлерін дисплейге шығару;
- □ егер мұндай студенттер жоқ болса, онда сол туралы хабарлама шығару.

2-нұсқа

1. Келесі өрістерден тұратын STUDENT атты құрылымды сипаттаңыз:
   - □ тегі мен инициалдары;
   - □ топтың нөмірі;
   - □ үлгерімі (бес элементтен тұратын жиым).
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - □ STUDENT типті он құрылымнан тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар орташа балдың өсуі бойынша реттеліп орналасуы керек;
   - □ 4 және 5 деген баға алған барлық студенттер үшін солардың тегі мен топ нөмірлерін дисплейге шығару;
   - □ егер мұндай студенттер жоқ болса, соған сәйкес хабарлама шығару.

3-нұсқа

1. Келесі өрістерден тұратын STUDENT атты құрылымды сипаттаңыз:
   - □ тегі мен инициалдары;
   - □ топтың нөмірі;
   - □ үлгерімі (бес элементтен тұратын жиым).
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - □ STUDENT типті он құрылымнан тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар алфавит бойынша реттеліп орналасуы керек;
   - □ кемінде бір екілік бағасы бар студенттер үшін олардың тегі мен топ нөмірлерін дисплейге шығару;
   - □ егер мұндай студенттер жоқ болса, соған сәйкес хабарлама шығару.

4-нұсқа

1. Келесі өрістерден тұратын AEROFLOT атты құрылымды сипаттаңыз: рейстің баратын пунктінің атауы;
   - □ рейс нөмірі;
   - □ ұшақ типі.
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - □ AEROFLOT типті жеті құрылымнан тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар рейс нөмірінің өсуі бойынша реттелуі керек;
   - □ атауы пернетақтадан енгізілген атаумен сәйкес келетін пунктке ұшатын рейс нөмірлері мен ұшақ типтерін экранға шығару;
   - □ егер мұндай рейстер болмаса, дисплейге сол туралы хабарлама шығару.

5-нұсқа

1. Келесі өрістерден тұратын AEROFLOT атты құрылымды сипаттаңыз:
   - □ рейстің баратын пунктінің атауы;

- □ рейс нөмірі;
- □ ұшақ типі.

2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - □ AEROFLOT типті жеті құрылымнан тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар белгіленген пунктердің аттарына қарай алфавит бойынша реттеліп орналасуы керек;
   - □ типі пернетақтадан енгізілген типпен сәйкес келген ұшақтың қызметін пайдаланатын рейстердің баратын пункттері мен нөмірлерін экранға шығару;
   - □ егер мұндай рейстер болмаса, онда дисплейге сол туралы хабарлама шығару.

6-нұсқа

1. Келесі өрістерден тұратын WORKER атты құрылымды сипаттаңыз:
   - □ жұмысшының тегі мен инициалдары;
   - □ атқаратын қызметінің атауы;
   - □ жұмысқа тұрған жылы.
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - □ WORKER типті он құрылымнан тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар алфавит бойынша реттелуі тиіс;
   - □ ұйымдағы еңбек өтілі пернетақтадан енгізілген мәннен артық болатын жұмысшылардың тегін дисплейге шығару;
   - □ егер мұндай жұмысшылар болмаса, онда дисплейге сол туралы хабарлама шығару.

7-нұсқа

1. Келесі өрістерден тұратын TRAIN атты құрылымды сипаттаңыз:
   - □ баратын пунктінің атауы;
   - □ пойыз нөмірі;
   - □ жолға шығу (жүру) уақыты.
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - □ TRAIN типті сегіз құрылымнан тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар баратын пункт аттарына байланысты алфавит бойынша реттеліп орналасуы тиіс;
   - □ пернетақтадан енгізілген уақыттан кейін жол жүретін пойыздар туралы ақпаратты экранға шығару;
   - □ егер мұндай пойыздар болмаса, онда дисплейге сол туралы хабарлама шығару.

8-нұсқа

1. Келесі өрістерден тұратын TRAIN атты құрылымды сипаттаңыз:
   - □ баратын пунктінің атауы;
   - □ пойыз нөмірі;

□ жолға шығу уақыты.

2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   □ TRAIN типті алты элементтен тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар пойыздардың жолға шығу уақыты бойынша реттеліп орналасуы керек;
   □ пернетақтадан енгізілген пунктке баратын пойыздар туралы ақпаратты экранға шығару;
   □ егер мұндай пойыздар болмаса, онда дисплейге сол туралы хабарлама шығару.

9-нұсқа

1. Келесі өрістерден тұратын TRAIN атты құрылымды сипаттаңыз:
   □ баратын пунктінің атауы;
   □ пойыз нөмірі;
   □ жолға шығу уақыты.
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   □ TRAIN типті сегіз элементтен тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар пойыздардың нөмірлері бойынша реттеліп орналасуы керек;
   □ нөмірі пернетақтадан енгізілген пойыз туралы ақпаратты экранға шығару;
   □ егер мұндай пойыздар болмаса, онда дисплейге сол туралы хабарлама шығару.

10-нұсқа

1. Келесі өрістерден тұратын MARSH атты құрылымды сипаттаңыз:
   □ жүру бағытының бастапқы пунктінің атауы;
   □ жүру бағытының соңғы пунктінің атауы;
   □ жүру бағытының нөмірі.
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   □ MARSH типті сегіз элементтен тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар жүру бағытының нөмірі бойынша реттелуі тиіс;
   □ нөмірі пернетақтадан енгізілген жүру бағыты туралы ақпаратты экранға шығару;
   □ егер мұндай бағыттар болмаса, онда дисплейге сол туралы хабарлама шығару.

11-нұсқа

1. Келесі өрістерден тұратын MARSH атты құрылымды сипаттаңыз:
   □ жүру бағытының бастапқы пунктінің атауы;
   □ жүру бағытының соңғы пунктінің атауы;
   □ жүру бағытының нөмірі.

2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - MARSH типті сегіз элементтен тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар жүру бағытының нөмірі бойынша реттелуі тиіс;
   - атауы пернетақтадан енгізілген пункттан басталатын немесе аяқталатын бағыттар туралы ақпаратты экранға шығару;
   - егер мұндай бағыттар болмаса, дисплейге сол туралы хабарлама шығару.

12-нұсқа

1. Келесі өрістерден тұратын NOTE атты құрылымды сипаттаңыз:
   - тегі, аты;
   - телефон нөмірі;
   - туған күні (үш саннан тұратын жиым).
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - NOTE типті сегіз элементтен тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар туған күн мерзімі бойынша реттелуі тиіс;
   - нөмірі пернетақтадан енгізілген адам туралы ақпаратты экранға шығару;
   - егер мұндай адам жоқ болса, онда дисплейге сол туралы хабарлама шығару.

13-нұсқа

1. Келесі өрістерден тұратын NOTE атты құрылымды сипаттаңыз:
   - тегі, аты;
   - телефон нөмірі;
   - туған күні (үш саннан тұратын жиым).
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - NOTE типті сегіз элементтен тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар алфавит бойынша реттеліп орналасуы керек;
   - туған күні пернетақтадан мәні енгізілген айға сәйкес келетін адамдар туралы ақпаратты экранға шығару;
   - егер мұндай адам жоқ болса, онда дисплейге сол туралы хабарлама шығару.

14-нұсқа

1. Келесі өрістерден тұратын NOTE атты құрылымды сипаттаңыз:
   - тегі, аты;
   - телефон нөмірі;
   - туған күні (үш саннан тұратын жиым).
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - NOTE типті сегіз элементтен тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар телефон нөмірінің алғашқы үш цифры бойынша реттелуі тиіс;

□ тегі пернетақтадан енгізілген адам туралы ақпаратты экранға шығару;
□ егер мұндай адам жоқ болса, дисплейге сол туралы хабарлама шығару.

15-нұсқа

1. Келесі өрістерден тұратын ZNAK атты құрылымды сипаттаңыз:
   □ тегі, аты;
   □ Зодиак (жұлдыз жорамал) белгісі;
   □ туған күні (үш саннан тұратын жиым).
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   □ ZNAK типті сегіз элементтен тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар туған күн мерзімі бойынша реттелуі тиіс;
   □ тегі пернетақтадан енгізілген адам туралы ақпаратты экранға шығару;
   □ егер мұндай адам болмаса, онда дисплейге сол туралы хабарлама шығару.

16-нұсқа

1. Келесі өрістерден тұратын ZNAK атты құрылымды сипаттаңыз:
   □ тегі, аты;
   □ Зодиак (жұлдыз жорамал) белгісі;
   □ туған күні (үш саннан тұратын жиым).
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   □ ZNAK типті сегіз элементтен тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар туған күн мерзімі бойынша реттелуі тиіс;
   □ атауы пернетақтадан енгізілген Зодиак белгісінде туған адамдар туралы ақпаратты экранға шығару;
   □ егер мұндай адамдар жоқ болса, онда дисплейге сол туралы хабарлама шығару.

17-нұсқа

1. Келесі өрістерден тұратын ZNAK атты құрылымды сипаттаңыз:
   □ тегі, аты;
   □ Зодиак белгісі;
   □ туған күні (үш саннан тұратын жиым).
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   □ ZNAK типті сегіз элементтен тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар Зодиак белгілері бойынша реттелуі тиіс;
   □ мәні пернетақтадан енгізілген айда туылған адамдар туралы ақпаратты экранға шығару;
   □ егер мұндай адамдар жоқ болса, онда дисплейге сол туралы хабарлама шығару.

18-нұсқа

1. Келесі өрістерден тұратын PRICE атты құрылымды сипаттаңыз:
   □ тауардың атауы;

- тауар сатылатын дүкеннің атауы;
- тауардың құны, теңгемен.

2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - PRICE типті сегіз элементтен тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар тауарлар атаулары бойынша алфавиттік ретпен орналасу керек;
   - атауы пернетақтадан енгізілген тауар туралы ақпаратты экранға шығару;
   - егер мұндай тауарлар жоқ болса, сәйкес хабарламаны дисплейге шығару;

19-нұсқа

1. Келесі өрістерден тұратын PRICE атты құрылымды сипаттаңыз:
   - тауардың атауы;
   - тауар сатылатын дүкеннің атауы;
   - тауардың құны, теңгемен.
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - PRICE типті сегіз элементтен тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар дүкендердің атаулары бойынша алфавиттік ретпен орналасуы керек;
   - атауы пернетақтадан енгізілген дүкенде сатылатын тауарлар туралы ақпаратты экранға шығару;
   - егер мұндай дүкен жоқ болса, сәйкес хабарламаны дисплейге шығару;

20-нұсқа

1. Келесі өрістерден тұратын ORDER атты құрылымды сипаттаңыз:
   - төлеушінің есеп айырысу шоты;
   - қабылдаушының есеп айырысу шоты;
   - аударылатын қаражат теңге бағамымен.
2. Келесі іс-әрекеттерді орындайтын программа жазыңыз:
   - ORDER типті сегіз элементтен тұратын жиымға пернетақтадан мәліметтер енгізу; жазбалар төлеушілердің есеп айырысу шоттары бойынша алфавиттік ретпен орналасуы керек;
   - пернетақтадан енгізілген төлеушінің есеп айырысу шотынан алынған қаражат туралы ақпаратты экранға шығару;
   - егер мұндай есеп айырысу шоты жоқ болса, сәйкес хабарламаны дисплейге шығару;

Нұсқауыштар

«Екіөлшемді жиымдар» және «Құрылымдар» бөлімдерінде берілген тапсырмаларды компьютер жадын динамикалық бөлуді пайдалану арқылы орындаңыз.

Қарапайым функциялар

Осы тақырыпқа байланысты лабораториялық жұмыстарды іске асыратын программалар практикумның [11] 132 б. берілген.

«Екіөлшемді жиымдар» бөлімінде берілген жаттығуларды тапсырманың әрбір пунктін функция түрінде ұйымдастыру арқылы орындаңыз. Функцияларға қажетті барлық мәліметтер оларға параметрлер ретінде берілуі тиіс. Функцияларда ауқымды айнымалыларды қолдануға тыйым салынады.

Функциялар және файлдар

1-нұсқа

Түрлі түсті планшетті сканерлер туралы мәліметтерді сақтау үшін келесідей құрылымды сипаттаңыз:

```
struct scan_info{
   char model[25]; // модельдің атауы
   int price;      // бағасы
   double x_size;  // сканерлеу аймағының көлденең өлшемі
   double y_size;  // сканерлеу аймағының тік өлшемі
   int optr;       // оптикалық шешу мүмкіндігі
   int grey;       // сұр түстің градациялар саны
};
```

Берілген құрылымнан сканер туралы мәліметтерді бинарлық файлға жазатын функция құру керек. Файлдың құрылымы: алғашқы екі байтта файлға енгізілген жазбалар санын анықтайтын **int** типті мән орналасады; одан кейін сканерлер туралы жазбалар бос орын қалдырмастан жазылады.

Осы файлдан сканер туралы мәліметтерді **scan_info** типті құрылымға көшіретін функция жазыңыз. Қажетті жазбаның нөмірі міндетті параметр болып табылады. Егер файлдан оқу сәтті орындалса, функция нөлдік мән, кері жағдайда -1 мәнін қайтаруы тиіс.

Сканерлер туралы 6-8 жазбадан тұратын мәліметтер (мәліметтер пернетақтадан енгізіледі) сақталған файл құратын және сұратылған жазба туралы мәліметтерді дисплейге шығаратын программа мысалын келтіріңдер.

Функциялар үшін барлық қажетті мәліметтер оларға параметрлер түрінде беріледі. Функцияларда ауқымды айнымалыларды пайдалануға болмайды.

2-нұсқа

Планшетті сканерлер туралы мәліметтерді сақтау үшін 1-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Берілген құрылымнан сканер туралы мәліметтерді бинарлық файлға жазатын функция құру керек. Файлдың құрылымы: алғашқы екі байтта файлға енгізілген жазбалар санын анықтайтын **int** типті мән орналасады; одан кейін сканерлер туралы жазбалар бос орын қалдырмастан жазылады.

Жоғарыдағы бинарлы файлдағы жазбаларды келесі сипаттамалардың бірі – бағасы немесе сұр түстің градациялар саны бойынша сұрыптайтын функция құрыңыз. Сұрыптау критериін бекітетін көрсеткіш міндетті параметр болып табылады.

Сканерлер туралы мәліметтер (мәліметтер пернетақтадан енгізіледі) сақталған сегізден кем болмайтын жазбадан тұратын файл құратын және оның сұрыпталуын жүзеге асыратын программаға мысал келтіріңдер.

Функциялар үшін барлық қажетті мәліметтер оларға параметрлер ретінде берілуі тиіс. Функцияларда ауқымды айнымалыларды пайдалануға тыйым салынады.

3-нұсқа

Планшетті сканерлер туралы мәліметтерді сақтау үшін 1-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Берілген құрылымнан сканер туралы мәліметтерді бинарлық файлға жазатын функция құрыңыз. Файлдың құрылымы: алғашқы төрт байтта файлға енгізілген жазбалар санын анықтайтын **long** типті мән орналасады; одан кейін сканерлер туралы жазбалар бос орын қалдырмастан жазылады.

Жоғарыда аталған бинарлық файлдағы жазбаларды сканер моделінің аттары бойынша сұрыптайтын функция құрыңыз.

Сканерлер туралы мәліметтер (мәліметтер пернетақтадан енгізіледі) сақталған сегізден кем болмайтын жазбалардан тұратын файл құратын және оның сұрыпталуын жүзеге асыратын программаға мысал келтіріңдер.

Функцияларға барлық қажетті мәліметтер параметрлер ретінде берілуі тиіс. Функцияларда ауқымды айнымалыларды пайдалануға тыйым салынады.

4-нұсқа

Планшетті сканерлер туралы мәліметтерді сақтау үшін 1-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Құрылымдар жиымына жадыны динамикалық түрде бөлетін (алты элементтен кем емес), оны сұхбаттасу режимінде мәліметтермен толтыратын және жиымды бинарлық файлға жазатын функция құрыңыз. Файлдың құрылымы: алғашқы екі байтта файлға енгізілген жазбалар санын анықтайтын **int** типті мән орналасады; одан кейін сканерлер туралы жазбалар бос орын қалдырмастан жазылады.

Аталған файлдан сканер туралы мәліметтерді **scan_info** типті құрылымға көшіретін функция құрыңыз. Қажетті жазбаның нөмірі міндетті параметр болып табылады. Егер файлдан оқу сәтті орындалса, функция нөлдік мән, кері жағдайда -1 мәнін қайтаруы тиіс.

Сканерлер туралы мәліметтер (мәліметтер пернетақтадан енгізіледі) сақталған сегізден кем болмайтын жазбалардан тұратын файлды құратын және сұратылған жазба туралы мәліметтерді дисплейге шығаратын программа мысалын келтіріңдер.

Функциялар үшін барлық қажетті мәліметтер оларға параметрлер ретінде берілуі тиіс. Функцияларда ауқымды айнымалыларды пайдалануға тыйым салынады.

5-нұсқа

Планшетті сканерлер туралы мәліметтерді сақтау үшін 1-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Берілген құрылымнан сканер туралы мәліметтерді бинарлық файлдағы қажетті позицияға жазатын функция құрыңыз. Файлдың құрылымы: алғашқы екі байтта файлға енгізілген жазбалар санын анықтайтын **`int`** типті мән орналасады; одан кейін сканерлер туралы жазбалар бос орын қалдырмастан жазылады. Жазба кез келген позицияға орындалуы мүмкін, егер енгізіліп жатқан және соңғы жазба (немесе файлдың басы) арасында бос орындар бар болса, олар нөлдермен толтырылады.

Жоғарыда сипатталған бинарлық файлды құрамындағы барлық нөлдерден тұратын жазбаларды жою арқылы оны «ықшамдап тығыздайтын» функция құрыңыз.

Сканерлер туралы мәліметтер (мәліметтер пернетақтадан енгізіледі) сақталған алтыдан кем емес жазбалардан тұратын файл құратын және оны ықшамдайтын программа мысалын келтіріңдер.

Функциялар үшін барлық қажетті мәліметтер оларға параметрлер ретінде берілуі тиіс. Функцияларда ауқымды айнымалыларды пайдалануға тыйым салынады.

6-нұсқа

Планшетті сканерлер туралы мәліметтерді сақтау үшін 1-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Құрылымдар жиымына жадыны динамикалық түрде бөлетін (алты элементтен кем емес), оны сұхбат режимінде мәліметтермен толтыратын және жиымды бинарлық файлға жазатын функция құрыңыз. Файлдың құрылымы: алғашқы екі байтта файлға енгізілген жазбалар санын анықтайтын **`int`** типті мән орналасады; одан кейін сканерлер туралы жазбалар бос орын қалдырмастан жазылады.

Сканер туралы мәліметтерді сұхбаттасу режимінде сұрайтын және бинарлық файлдағы жазбаларды берілген нөмірі бойынша алмастыратын функция құрыңыз. Қажетті жазбаның нөмірі міндетті параметр болып табылады. Егер файлдан оқу сәтті орындалса, функция нөлдік мән, кері жағдайда -1 мәнін қайтаруы тиіс.

Сканерлер туралы мәліметтер (мәліметтер пернетақтадан енгізіледі) сақталған сегізден аспайтын жазбалардан тұратын файлды құратын және сканер туралы жаңа жазбалар кірістіруді жүзеге асыратын программа мысалын келтіріңдер.

Функциялар үшін барлық қажетті мәліметтер параметрлер ретінде берілуі тиіс. Функцияларда ауқымды айнымалыларды пайдалануға тыйым салынады.

7-нұсқа

Планшетті сканерлер туралы мәліметтерді сақтау үшін 1-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Берілген құрылымнан сканер туралы мәліметтерді бинарлық файлға жазатын функция құрыңыз. Файлдың құрылымы: алғашқы екі байтта файлға енгізілген жазбалар санын анықтайтын **int** типті мән орналасады; одан кейін сканерлер туралы жазбалар бос орын қалдырмастан жазылады.

Аталған файлдан сканер туралы мәліметтерді **scan_info** типті құрылымға пернетақтадан енгізетін, қажетті сканер туралы ақпарат файлда табылмаған жағдайда құрылым құрамындағы ақпаратты файл соңына орналастыратын, кері жағдайда сәйкес хабарлама шығаратын функция құрыңыз.

Сканерлер туралы мәліметтер (мәліметтер пернетақтадан енгізіледі) сақталған файл – 6-8 жазба құратын және файлды 2-3 сканер туралы пернетақтадан енгізілетін жазбалармен толықтыратын программаға мысал келтіріңдер.

Функциялар үшін барлық қажетті мәліметтер оларға параметрлер ретінде берілуі тиіс. Функцияларда ауқымды айнымалыларды пайдалануға тыйым салынады.

8-нұсқа

Планшетті сканерлер туралы мәліметтерді сақтау үшін 1-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Берілген құрылымнан сканер туралы мәліметтерді бинарлық файлға жазатын функция жазыңыз. Файлдың құрылымы: алғашқы екі байтта файлға енгізілген жазбалар санын анықтайтын **int** типті мән орналасады; одан кейін сканерлер туралы жазбалар бос орын қалдырмастан жазылады.

Аталған файлдан сканер туралы мәліметтерді **scan_info** типті құрылымға пернетақтадан енгізетін және оның құрамын файлдағы алғашқы жазбаның орнына жазатын функция құрыңыз. Мұндай файл алдын ала құрылған болуы тиіс. Бұл кезде бұрынғы уақытта бірінші позицияда орналасқан жазба екінші позицияға, екінші жазба үшіншісіне, т.с.с. ығысып орналастырылады.

Сканерлер туралы мәліметтер (мәліметтер пернетақтадан енгізіледі) сақталған файл – 6-8 жазба құратын және файлды пернетақтадан енгізілетін 1-2 жаңа жазбамен толықтыратын программаға мысал келтіріңдер.

Функциялар үшін барлық қажетті мәліметтер оларға параметрлер ретінде берілуі тиіс. Функцияларда ауқымды айнымалыларды пайдалануға тыйым салынады.

9-нұсқа

Планшетті сканерлер туралы мәліметтерді сақтау үшін 1-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Енгізілген ақпаратқа сәйкес сканерлер санын анықтайтын, сәйкес өлшемдегі құрылымдар жиымына жадыны динамикалық түрде бөлетін және

оны сұхбаттасу режимінде толтыратын (пернетақтадан) функция құрыңыз. Мұнда сканер атауында бос орындар болуы мүмкін.

Берілген жиымды жаңадан құрылатын бинарлық файлға жазатын функция құрыңыз. Егер сканер бағасы 200-ден кем болса, бұл сканер туралы мәліметтер файлға жазылмайды. Басқа сканерлер туралы ақпарат бинарлық файлға орналастырылады, мұнда алдымен атауы бас әріптен басталатын, содан кейін кіші әріптен басталып жазылатын барлық сканерлер туралы ақпараттар жазылады.

Файлдың құрылымы: алғашқы төрт байтта файлға енгізілген жазбалар санын анықтайтын **long** типті мән орналасады; одан кейін сканерлер туралы жазбалар бос орын қалдырмастан жазылады.

Сканерлер туралы мәліметтер (мәліметтер пернетақтадан енгізіледі) сақталған файл құратын және қажетті жазба туралы мәліметтерді (барлығын немесе нөмірі бойынша) дисплейге шығаруды жүзеге асыратын программаға мысал келтіріңдер.

Функциялар үшін барлық қажетті мәліметтер оларға параметрлер ретінде берілуі тиіс. Функцияларда ауқымды айнымалыларды пайдалануға тыйым салынады.

10-нұсқа

Ноутбуктар туралы мәліметтерді сақтау үшін (қажет болса, толықтыра отырып) келесідей құрылымды сипаттаңыз:

```
struct NOTEBOOK{
   char model[21];      // атауы
   struct size{         // габариттік өлшемдері
      float x;
      float y;
      float z;
   };
   float w;              // салмағы
   int price;            // бағасы
}
```

Ноутбуктар туралы мәліметтерді note.txt файлынан берілген түрдегі құрылымға оқитын (170 б. қараңыз) функция құрыңыз. Құрылым құрамындағы мәліметтерді бинарлық файлдың соңына жазатын функция құру қажет. Бинарлық файлдың құрылымы: алғашқы екі байт (бүтін) – файлдағы жазбалар саны; одан кейін NOTEBOOK құрылымының форматындағы жазбалар.

Құрылған функциялар негізінде процессор жиілігі 120 МГц-тен артық ноутбуктар үшін ғана мәліметтерді оқып, оларды бинарлық файлға бағасының кемуі бойынша жазуды жүзеге асыратын программа құрыңыз.

11-нұсқа

Ноутбуктар туралы мәліметтерді сақтау үшін 10-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Ноутбуктар туралы мәліметтерді note.txt файлынан берілген түрдегі құрылымға оқитын (170 б. қараңыз) функция жазыңыз. Құрылым құрамындағы мәліметтерді бинарлық файлдың соңына жазатын функция құру қажет. Бинарлық файлдың құрылымы: алғашқы екі байт (бүтін) – файлдағы жазбалар саны; одан кейін NOTEBOOK құрылымының форматындағы жазбалар.

Құрылған функциялар негізінде HDD көлемі 1 Гбайт-тан аспайтын ноутбуктар үшін ғана мәліметтерді оқып, оларды бинарлық файлға атауы бойынша алфавиттік ретпен жазуды жүзеге асыратын программа құрыңыз.

12-нұсқа

Ноутбуктар туралы мәліметтерді сақтау үшін 10-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Ноутбуктар туралы мәліметтерді note.txt файлынан берілген түрдегі құрылымға оқитын (170 б. қараңыз) функция жазыңыз. Құрылым құрамындағы мәліметтерді бинарлық файлдың соңына жазатын функция құру қажет. Бинарлық файлдың құрылымы: алғашқы екі байт (бүтін) – файлдағы жазбалар саны; одан кейін NOTEBOOK құрылымының форматындағы жазбалар.

Құрылған функциялар негізінде бір шаршы метрдегі саны 285 данадан аспайтын ноутбуктар үшін ғана мәліметтерді оқып, оларды екілік файлға атауы бойынша алфавиттік ретпен жазуды жүзеге асыратын программа құрыңыз.

13-нұсқа

Ноутбуктар туралы мәліметтерді сақтау үшін 10-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Ноутбуктар туралы мәліметтерді note.txt файлынан берілген түрдегі құрылымға оқитын (170 б. қараңыз) функция жазыңыз. Құрылым құрамындағы мәліметтерді бинарлық файлдың соңына жазатын функция құру қажет. Бинарлық файлдың құрылымы: алғашқы екі байт (бүтін) – файлдағы жазбалар саны; одан кейін NOTEBOOK құрылымының форматындағы жазбалар.

Құрылған функциялар негізінде жедел есте сақтау құрылғысына –ЖЕСҚ ең үлкен (максималды) көлемі 40 Мбайт-тан аспайтын ноутбуктар үшін ғана мәліметтерді оқып, оларды екілік файлға көлемі бойынша сұрыпталған мәліметтерді жазуды жүзеге асыратын программа құрыңыз.

14-нұсқа

Ноутбуктар туралы мәліметтерді сақтау үшін 10-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Ноутбуктар туралы мәліметтерді note.txt файлынан берілген түрдегі құрылымға оқитын (170 б. қараңыз) функция жазыңыз. Құрылым құрамындағы мәліметтерді бинарлық файлдың соңына жазатын функция құру қажет. Бинарлық файлдың құрылымы: алғашқы екі байт (бүтін) – файлдағы жазбалар саны; одан кейін NOTEBOOK құрылымының форматындағы жазбалар.

Құрылған функциялар негізінде дисплей диагоналі он бір дюймнен асатын ноутбуктар үшін ғана мәліметтерді оқып, оларды екілік файлға жазуды жүзеге асыратын программа құрыңыз.

15-нұсқа

Ноутбуктар туралы мәліметтерді сақтау үшін (қажет болса, толықтыра отырып) келесідей құрылымды сипаттаңыз:

```
struct NOTEBOOK{
   struct disp_res{     // дисплейдің шешуш мүмкіндігі
      int x;             // көлденеңнен
      int y;             // тігінен
   };
   int f;                // регенерация жиілігі
   float d;              // дисплей диагоналінің ұзындығы
   int price;            // бағасы
   char model[21];      // атауы
}
```

Ноутбуктар туралы мәліметтерді note.txt файлынан берілген түрдегі құрылымға оқитын (170 б. қараңыз) функция жазыңыз. Құрылым құрамындағы мәліметтерді бинарлық файлдың соңына жазатын функция құру қажет. Бинарлық файлдың құрылымы: алғашқы екі байт (бүтін) – файлдағы жазбалар саны; одан кейін NOTEBOOK құрылымының форматындағы жазбалар.

Құрылған функциялар негізінде салмағы 7 кг-нан аспайтын ноутбуктар үшін ғана мәліметтерді оқып, оларды екілік файлға бағасының өсуі бойынша реттеп жазуды жүзеге асыратын программа құрыңыз.

16-нұсқа

Ноутбуктар туралы мәліметтерді сақтау үшін 15-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Ноутбуктар туралы мәліметтерді note.txt файлынан берілген түрдегі құрылымға оқитын (170 б. қараңыз) функция жазыңыз. Құрылым құрамындағы мәліметтерді бинарлық файлдың соңына жазатын функция құру қажет. Бинарлық файлдың құрылымы: алғашқы екі байт (бүтін) – файлдағы жазбалар саны; одан кейін NOTEBOOK құрылымының форматындағы жазбалар.

Құрылған функциялар негізінде бейнежады көлемі 2 Мбайт болатын ноутбуктар үшін ғана мәліметтерді оқып, оларды екілік файлға процессордың тактілік жиілігінің кему ретімен жазуды жүзеге асыратын программа құрыңыз.

17-нұсқа

Ноутбуктар туралы мәліметтерді сақтау үшін 15-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Ноутбуктар туралы мәліметтерді note.txt файлынан берілген түрдегі құрылымға оқитын (170 б. қараңыз) функция жазыңыз. Құрылым құрамындағы

мәліметтерді бинарлық файлдың соңына жазатын функция құру қажет. Бинарлық файлдың құрылымы: алғашқы екі байт (бүтін) – файлдағы жазбалар саны; одан кейін NOTEBOOK құрылымының форматындағы жазбалар.

Құрылған функциялар негізінде HDD көлемі 1 Гбайт-тан асатын ноутбуктар үшін ғана мәліметтерді оқып, оларды екілік файлға дисплей диагоналі ұзындығының өсу реті бойынша жазуды жүзеге асыратын программа құрыңыз.

18-нұсқа

Ноутбуктар туралы мәліметтерді сақтау үшін 15-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Ноутбуктар туралы мәліметтерді note.txt файлынан берілген түрдегі құрылымға оқитын (170 б. қараңыз) функция жазыңыз. Құрылым құрамындағы мәліметтерді бинарлық файлдың соңына жазатын функция құру қажет. Бинарлық файлдың құрылымы: алғашқы екі байт (бүтін) – файлдағы жазбалар саны; одан кейін NOTEBOOK құрылымының форматындағы жазбалар.

Құрылған функциялар негізінде процессорының тактілік жиілігі 120 МГц-тен асатын ноутбуктар үшін ғана мәліметтерді оқып, оларды екілік файлға салмақтарының кему ретімен жазуды жүзеге асыратын программа құрыңыз.

19-нұсқа

Ноутбуктар туралы мәліметтерді сақтау үшін (қажет болса, толықтыра отырып) келесідей құрылымды сипаттаңыз:

```
struct NOTEBOOK{
   struct disp_res{     // дисплейдің шешу мүмкіндігі
      int x;            // көлденеңнен
      int y;            // тігінен
   };
   int f;               // регенерация жиілігі
   float d;             // дисплей диагоналінің ұзындығы
   int price;           // бағасы
   char model[21];      // атауы
}
```

Ноутбуктар туралы мәліметтерді note.txt файлынан берілген түрдегі құрылымға оқитын функция жазыңыз. Құрылым құрамындағы мәліметтерді бинарлық файлдың соңына жазатын функция құру қажет. Бинарлық файлдың құрылымы: алғашқы екі байт (бүтін) – файлдағы жазбалар саны; одан кейін NOTEBOOK құрылымының форматындағы жазбалар.

Құрылған функциялар негізінде процессорының тактілік жиілігі 120 МГц-тен асатын ноутбуктар үшін ғана мәліметтерді оқып, оларды екілік файлға бағаларының өсу ретімен жазуды жүзеге асыратын программа құрыңыз.

20-нұсқа

Ноутбуктар туралы мәліметтерді сақтау үшін 19-нұсқада көрсетілгендей құрылымды сипаттаңыз.

Ноутбуктар туралы мәліметтерді note.txt файлынан берілген түрдегі құрылымға оқитын функция жазыңыз. Құрылым құрамындағы мәліметтерді бинарлық файлдың соңына жазатын функция құру қажет. Бинарлық файлдың құрылымы: алғашқы екі байт (бүтін) – файлдағы жазбалар саны; одан кейін NOTEBOOK құрылымының форматындағы жазбалар.

Құрылған функциялар негізінде бағасы $3500-дан асатын ноутбуктар үшін ғана мәліметтерді оқып, оларды екілік файлға процессордың тактілік жиілігінің өсу ретімен жазуды жүзеге асыратын программа құрыңыз.

note.txt файлының мысалы:

```
Acer Note Light    2699 5.6 02.0x11.8x08.3 100 40 10.4 1 1024x0768 60 0.774
ASW ND5123T        3489 7.2 02.3x11.8x10.1 133 32 12.1 2 1024x0768 70 1.300
ARMNote TS80CD     3699 7.2 02.0x11.5x08.8 133 64 11.3 1 1024x0768 75 1.300
AST Ascentia P50   4499 7.5 02.3x11.3x09.0 133 40 11.3 1 0800x0600 70 0.774
BSI NP8657D        2605 8.0 02.3x11.8x09.3 133 40 11.3 1 1024x0768 60 0.810
BSI NP5265A        3765 8.2 02.5x12.0x09.0 150 32 12.1 2 1024x0768 70 1.300
Dell Xpi P100SD    3459 6.0 02.3x11.0x08.8 100 40 10.3 1 1024x0768 60 0.773
Digital HiNote     4799 4.0 01.3x11.0x08.8 120 40 10.4 1 0800x0600 56 1.000
Gateway Solo S5    4499 5.6 02.0x11.9x08.8 133 40 11.3 2 1024x0768 60 0.686
Hertz Z-Optima NB  3995 8.0 02.3x11.9x09.0 150 40 11.2 2 1024x0768 75 1.000
HP OmniBook 5500   6120 7.1 02.0x11.5x09.0 133 64 11.4 1 1024x0768 75 1.300
IBM ThinkPad 560   3749 4.1 01.3x11.8x08.8 120 40 12.1 2 1024x0768 85 0.774
NEC Versa 4080H    4780 6.6 02.3x11.8x09.5 120 48 10.4 1 0800x0600 70 0.776
Polywell Poly 500  3300 7.9 02.3x11.9x09.0 120 40 10.4 1 1024x0768 72 1.000
Samsung SENS 810   3667 8.7 02.3x11.5x09.5 100 32 11.4 2 1024x0768 75 0.773
Twinhead Slimnote  2965 7.4 02.0x11.5x08.0 075 64 10.4 1 1024x0768 70 0.772
```

Note.txt файлында ноутбуктар туралы мәтіндік ақпарат сақталған. Әрбір жолда бір модель туралы мәліметтер жазылған. Жолдағы мәліметтер келесі өрістерде орналасады:

1 : 20 – модельдің атауы;

21 : 24 – бағасы, доллар есебімен (бүтін сан);

26 : 28 – ноутбуктың салмағы, кг есебімен (төрт таңбадан тұратын ондық нүктелі сан);

30 : 43 – ноутбуктің габариттік өлшемдері дюйм бірлігімен (БИІКТІК×ҰЗЫНДЫҚ×ЕН – ондық нүктелі үш сан ('×' арқылы бөлінген нүктені қосқанда – 4 символ);

44 : 47 – процессор жиілігі, МГц (үш символдан тұратын бүтін сан);

49 : 50 – мегабайтпен алынған жедел есте сақтау құрылғысының максималды көлемі (екі таңбадан тұратын бүтін сан);

52 : 55 – дюймммен алынған дисплей диагоналінің ұзындығы (нүктені қоса есептегенде төрт таңбадан тұратын ондық нүктелі сан);

57 – мегабайтпен алынған бейнежады көлемі – бір таңбадан тұратын бүтін сан;

59 : 67 – пикселмен алынған дисплейдің шешу қабілеті ('x' арқылы бөлінген екі бүтін сан);

69 : 70 – дисплейдің регенерация жиілігі, Гц (екі таңбалы бүтін сан);

72 : 76 – HDD көлемі, Гбайт (бес таңбадан тұратын ондық нүктелі сан).

Барлық сипатталмаған позициялар бос орын белгілерімен толтырылған.

Тіркестермен және символдармен жұмыс істеуге арналған кітапхана функциялары

Осы тақырыпқа байланысты лабораториялық жұмыстарды іске асыратын программалар практикумның [11] 89 б. берілген.

1-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- кездейсоқ пернені басқан кезде мәтіннің әрбір сөйлемін кезегімен біртіндеп ерекшелеу;
- мәтіндегі сөйлемдер санын анықтау.

2-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- кездейсоқ пернені басқан кезде мәтіннің әрбір сөзін кезегімен біртіндеп ерекшелеу;
- мәтіндегі сөздер санын анықтау.

3-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- кездейсоқ пернені басқан кезде мәтіннің дауысты әріпке аяқталатын әрбір сөзін кезегімен біртіндеп ерекшелеу;
- мәтіндегі дауысты әріпке аяқталатын сөздер санын анықтау.

4-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек. Мәтін үш сөйлемнен тұруы тиіс (бір жол мәтіннің ұзындығы 70 символдан артық болмауы қажет). Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- кездейсоқ пернені басқан кезде мәтіннің әрбір сөйлемін 2, 1, 3 реттілігімен кезек бойынша біртіндеп ерекшелеу.

5-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- кездейсоқ пернені басқан кезде мәтіндегі алғашқы және соңғы символдары сәйкес келетін әрбір сөзді кезегімен біртіндеп ерекшелеу;
- мәтіндегі алғашқы және соңғы символдары бірдей болып келетін сөздер санын анықтау.

6-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- кездейсоқ пернені басқан кезде мәтіннің дауысты әріптен басталатын әрбір сөзін кезегімен біртіндеп ерекшелеу;
- мәтіндегі дауысты әріптен басталатын сөздер санын анықтау.

7-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- ең ұзақ сөздегі символдар санын анықтау;
- кездейсоқ пернені басқан кезде мәтіндегі ең көп символдар санынан тұратын әрбір сөзді кезегімен біртіндеп ерекшелеу.

8-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- ең қысқа сөздегі символдар санын анықтау;
- кездейсоқ пернені басқан кезде мәтіндегі ең аз символдар санынан тұратын әрбір сөзді кезегімен біртіндеп ерекшелеу.

9-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- мәтіннің әрбір сөйлеміндегі әріптерден және бос орындардан басқа символдар санын анықтау;
- кездейсоқ пернені басқан кезде мәтіннің әрбір сөйлемін кезегімен біртіндеп ерекшелеу, ал ерекшеленген сөйлемде әріптер мен бос орындардан басқа барлық символдарды кезегімен біртіндеп ерекшелеу.

10-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- мәтіндегі сөйлемдер санын және әрбір сөйлемдегі сөздер санын анықтау;
- кездейсоқ пернені басқан кезде мәтіннің әрбір сөйлемін, ал ерекшеленген сөйлемде барлық сөздерді кезегімен біртіндеп ерекшелеу.

11-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;

- мәтіннің соңғы сөзіндегі ‘а’ әріптерінің санын анықтау;
- кездейсоқ пернені басқан кезде мәтіннің соңғы сөзін, ал ерекшеленген сөзде кездесетін барлық ‘а’ әріптерін кезегімен біртіндеп ерекшелеу.

12-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- мәтіндегі ең ұзақ цифрлар тізбегін анықтау (екі цифр арасындағы бос орындардың кез келген саны цифрлар тізбегін үзбейді деп есептеледі);
- кездейсоқ пернені басқан кезде ең көп символдар санынан тұратын әрбір цифрлар тізбегін кезегімен біртіндеп ерекшелеу.

13-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- мәтіннің әрбір сөйлеміндегі берілген сөздің реттік нөмірін анықтау (берілген сөз пернетақтадан енгізіледі);
- кездейсоқ пернені басқан кезде мәтіннің әрбір сөйлемін, ал ерекшеленген сөйлемде берілген сөзді кезегімен біртіндеп ерекшелеу.

14-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 700 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- кездейсоқ пернені басқан кезде мәтіндегі берілген сөзді ерекшелеу (берілген сөз пернетақтадан енгізіледі);
- мәтіндегі берілген сөзді және басы артық бос орындарды жоя отырып, мәтінді дисплей экранына тағы бір рет шығару.

15-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 700 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- кездейсоқ пернені басқан кезде орындары алмастырылуы тиіс берілген екі сөзді кезегімен біртіндеп ерекшелеу (берілген сөздер пернетақтадан енгізіледі);
- мәтіндегі берілген сөздердің орындарын өзара алмастырып және артық бос орындарды жоя отырып, мәтінді дисплей экранына тағы бір рет шығару.

16-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 700 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- бастапқы мәтінді дисплей экранына шығару;
- кездейсоқ пернені басқан кезде мәтіндегі берілген сөзді кезек бойынша біртіндеп ерекшелеу (берілген сөз пернетақтадан енгізіледі);
- мәтіндегі берілген сөзді тырнақшаға ала отырып, дисплей экранына тағы бір рет шығару және берілген сөзді тырнақшалармен бірге кезегімен біртіндеп ерекшелеу.

17-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 700 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- бастапқы мәтінді дисплей экранына шығару;
- пернетақтадан бастапқы мәліметтер ретінде енгізілген сөзді әрбір сөйлемге соңғы сөз ретінде кірістіре отырып, дисплей экранына мәтінді тағы бір рет шығару;
- кездейсоқ пернені басқан кезде мәтіндегі кірістірілген сөзді кезегімен біртіндеп ерекшелеу.

18-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 500 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;

- кездейсоқ пернені басқан кезде мәтіндегі сөздер арасындағы артық бос орындарды ерекшелеу;
- мәтіннің сөздері арасындағы артық бос орындарды өшіріп және әрбір сөйлемді жаңа жолдан бастап жаза отырып, мәтінді дисплей экранына тағы бір рет шығару.

19-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 700 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- кездейсоқ пернені басқан кезде мәтіндегі берілген сөзді кезек бойынша біртіндеп ерекшелеу (берілген сөз пернетақтадан енгізіледі);
- берілген сөздегі кіші әріптерді бас әріптермен алмастыра отырып, мәтінді дисплей экранына тағы бір рет шығару.

20-нұсқа

Мәтіндік редактордың көмегімен ұзындығы 1000 символдан аспайтын мәтін жазылған файл құру керек (бір жол мәтіннің ұзындығы 70 символдан артық болмауы тиіс).

Файл атауының кеңейтілуі DAT болуы тиіс.

Келесі әрекеттерді орындайтын программа жазыңыз:

- мәтінді дисплей экранына шығару;
- мәтіндегі қатар орналасқан бос орындардың максималды санын анықтау;
- кездейсоқ пернені басқан кезде максималды ұзындықты бос орындар тізбегінің әрқайсысын кезекпен бірыңдеп ерекшелеу.

Функциялар шаблондары

«Бірөлшемді жиымдар» бөлімінде берілген жаттығуларды тапсырманың әрбір пунктін функция шаблоны түрінде ұйымдастыру арқылы орындаңыз. Функцияларға қажетті барлық мәліметтер оларға параметрлер ретінде берілуі тиіс. Функцияларда ауқымды айнымалыларды қолдануға тыйым салынады. Осы шаблондарды **`int`**, **`float`** және **`double`** типтері үшін қолданатын программаға мысалдар келтіріңдер.

Модульдік программалау

Осы тақырыпқа байланысты лабораториялық жұмыстарды жүзеге асыратын программалар практикумның [11] 132 б. берілген.

«Функциялар және файлдар» бөлімінде берілген жаттығуларды келесі шарттарды қанағаттандырылатындай етіп орындаңыз: құрылымның спаттамасын тақырыптық файлға, ал функциялардың анықтамалары мен программаның негізгі функцияларын екі жеке файлға орналастыру керек.

Мәліметтердің динамикалық құрылымдары

Осы тақырыпқа байланысты лабораториялық жұмыстарды жүзеге асыратын программалар практикумның [11] 169 б. берілген.

1-нұсқа

Автобус паркінде автобустардың болуы туралы динамикалық ақпараттан тұратын программа құрыңыз.

Әрбір автобус туралы төмендегідей ақпараттар бар:

- автобус нөмірі;
- жүргізушінің тегі мен инициалдары;
- жүру бағытының нөмірі.

Программа келесі іс-әрекеттердің орындалуын қамтамасыз етуі тиіс:

- парктегі барлық автобустар туралы мәліметтерді бастапқы тізім түрінде қалыптастыру;
- әрбір автобус парктан шыққан кезде автобус нөмірі енгізіледі және программа осы автобус туралы мәліметтерді паркте орналасқан автобустар тізімінен өшіріп, бұл мәліметтерді жолда жүрген автобустар тізіміне жазады;
- әрбір автобус паркке кірген кезде автобус нөмірі енгізіледі және программа осы автобус туралы мәліметтерді жолда жүрген автобустар тізімінен өшіріп, бұл мәліметтерді паркте орналасқан автобустар тізіміне жазады;
- сұраныс бойынша паркте орналасқан немесе бағыт бойынша жолда жүрген автобустар туралы мәліметтер беріледі.

2-нұсқа

Кітапханадағы кітаптар туралы ағымдағы ақпаратты қамтитын программа құрыңыз.

Әрбір кітап туралы төмендегідей ақпараттар бар:

- ӘОЖ – әмбебап ондық жіктелу (УДК) нөмірі;
- автордың тегі мен инициалдары;
- атауы;
- басылып шығарылған жылы;
- кітапханадағы осы кітаптар саны.

Программа келесі іс-әрекеттердің орындалуын қамтамасыз етуі тиіс:

- кітапханадағы барлық кітаптар туралы мәліметтерді екілік бұтақ түрінде бастапқы қалыптастыру;
- кітапханаға жаңадан түсетін кітаптар туралы мәліметтер қосу;

□ есептен шығарылатын кітаптар туралы мәліметтерді өшіру;

□ сұраныс бойынша кітапханадағы басылып шығарылған жылы бойынша реттелген кітаптар туралы мәліметтер беріледі.

3-нұсқа

Әуе билеттеріне жасалған тапсырыстар туралы ағымдағы ақпаратты қамтитын программа құрыңыз.

Әрбір тапсырыс туралы төмендегідей ақпараттар бар:

□ баратын пункт атауы;

□ рейс нөмірі;

□ жолаушының тегі мен инициалдары;

□ қалайтын ұшу күні.

Программа келесі іс-әрекеттердің орындалуын қамтамасыз етуі тиіс:

□ барлық тапсырыстарды тізім түрінде сақтау;

□ тізімге тапсырыстарды қосу;

□ тапсырыстарды өшіру;

□ тапсырыстарды берілген рейс нөмірі және ұшу күні бойынша басылып шығару;

□ барлық тапсырыстарды шығару.

4-нұсқа

Әуе билеттеріне жасалған тапсырыстар туралы ағымдағы ақпаратты қамтитын программа құрыңыз.

Әрбір тапсырыс туралы төмендегідей ақпараттар бар:

□ баратын пункт атауы;

□ рейс нөмірі;

□ жолаушының тегі мен инициалдары;

□ қалайтын ұшу күні.

Программа келесі іс-әрекеттердің орындалуын қамтамасыз етуі тиіс:

□ барлық тапсырыстарды екілік бұтақ түрінде сақтау;

□ тапсырыстарды қосу және өшіру;

□ берілген рейс нөмірі және ұшу күні бойынша тапсырыстарды басылып шығарылу және кейіннен өшіру;

□ барлық тапсырыстарды шығару.

5-нұсқа

Кітапханадағы кітаптар туралы ағымдағы ақпаратты қамтитын программа құрыңыз.

Әрбір кітап туралы төмендегідей ақпараттар бар:

□ ӘОЖ– әмбебап ондық жіктелу (УДК) нөмірі;

□ автордың тегі мен инициалдары;

□ атауы;

□ басылып шығарылу жылы;

□ кітапханадағы осы кітаптар саны.

Программа келесі іс-әрекеттердің орындалуын қамтамасыз етуі тиіс:

□ кітапханадағы барлық кітаптар туралы мәліметтерді бастапқы тізім түрінде қалыптастыру;

□ әрбір кітапты алу кезінде ӘОЖ нөмірі енгізіледі және программа кітаптар санын бірге кемітеді немесе қажетті кітап кітапханада жоқ немесе қолда екендігі туралы хабарлама береді;

□ әрбір кітап қайтарылған кезде ӘОЖ нөмірі енгізіледі және программа кітаптар санының мәнін бірге арттырады;

□ сұраныс бойынша кітапханада кітаптардың бар-жоқтығы туралы ақпарат беріледі.

6-нұсқа

Автобустар паркінде бар автобустар туралы динамикалық ақпараттан тұратын программа құрыңыз.

Әрбір автобус туралы төмендегідей ақпараттар бар:

□ автобус нөмірі;

□ жүргізушінің тегі мен инициалдары;

□ жүру бағытының нөмірі;

□ автобустың жолда немесе паркте болуының белгісі.

Программа келесі іс-әрекеттердің орындалуын қамтамасыз етуі тиіс:

□ барлық автобустар туралы мәліметтерді бастапқы тізім түрінде қалыптастыру;

□ әрбір автобус парктан шыққан кезде автобус нөмірі енгізіледі және программа «автобус жолда» белгісін орнатады;

□ әрбір автобус паркке кірген кезде автобус нөмірі енгізіледі және программа «автобус паркте» белгісін орнатады;

□ сұраныс бойынша паркте орналасқан немесе бағыт бойынша жолда жүрген автобустар туралы мәліметтер беріледі.

7-нұсқа

Лабиринттан дұрыс шығатын жолды табатын программа құрыңыз. Лабиринт шаршылардан тұратын матрица түрінде бейнеленеді. Әрбір шаршы ашық немесе жабық болады. Жабық шаршыға кіруге тыйым салынған. Егер шаршы ашық болса, оған қабырға жағынан кіруге болады, бірақ бұрыш жақтан кіруге болмайды. Әрбір шаршы өзінің матрицадағы координаталарымен анықталады.

Программа лабиринттан шығатын жолды берілген кіріс нүктесінен бастап табады. Дұрыс жолды тапқаннан кейін программа оны шаршының координаталары түрінде шығарады. Жолды есте сақтау үшін стекті қолданыңыз.

8-нұсқа

Машиналар қойылатын гараждық тұрақ бір қатар жолақтан ғана тұрады және олардың кіру және шығу нүктелері жолақтың бір басында орналасқан.

Егер көлік иесі өз машинасын алып кетуге келіп, бірақ көлігі шығатын жолға ең жақын орналаспаған болса, жолды бөгеп тұрған барлық көліктер жойылады, берілген адамның көлігі тұрақтан шығарылады да, жойылған көліктер тұраққа бастапқы ретпен қайта орналастырылады.

Көліктердің кіру және шығу әрекеттерін модельдейтін программа құрыңыз. Көліктің кіруі немесе шығуы командалық жол арқылы беріледі, мұнда көліктің келуінің немесе кетуінің белгісі және көлік нөмірі көрсетіледі. Программа кез келген көліктің келген және кеткен уақыты жайлы хабарлама шығаруы керек. Көлік тұрақтан шыққан кездегі хабарламада басқа көліктердің шығуын қамтамасыз ету үшін осы көліктің тұрақтан неше рет жойылғандығын көрсету керек.

9-нұсқа

Иілгіш магниттік дискінің мәліметпен толуын модельдейтін программа құрыңыз.

Дискідегі жадының жалпы көлемі 360 Кбайт. Файлдардың ұзындықтары 18 байттан 32 Кбайтқа дейінгі аралықтағы кез келген мәнді қабылдай алады. Жұмыс барысында файлдар дискіге жазылады немесе дискіден өшіріледі.

Жұмыс басында файлдар бірінен соң бірі кезекпен жазылады. Файлды өшіргеннен кейін дискіде бос жады аймағы пайда болады және жаңадан жазылатын файл осы бос аймаққа немесе файл бұл аймаққа сыймаған жағдайда соңғы жазылған файлдан кейін орналастырылады.

Егер файлдың көлемі дискідегі бос аймақ көлемінен асып кетсе, қате туралы хабарлама шығарылады. Файлдарды жазу немесе өшіру туралы сұраныс команда жолында беріледі, мұнда файл аты, оның байтпен берілген көлемі, жазылу немесе жойылу белгісі көрсетіледі. Программа сұраныс бойынша дискідегі бос немесе пайдаланылатын аймақтары туралы мәліметтерді шығарып беруі керек.

Нұсқау: дискідегі бос және пайдаланылатын аймақтар тізімін жеке-жеке құрған жөн.

10-нұсқа

Файлдық жүйеде файлдар каталогы сызықтық тізім түрінде ұйымдастырылған.

Әрбір файл үшін каталогта келесідей мәліметтер сақталады:

- □ файл аты;
- □ құрылған күні;
- □ файлды пайдалану саны.

Программа келесі іс-әрекеттердің орындалуын қамтамасыз етуі тиіс:

- □ файлдар каталогын бастапқы қалыптастыру;
- □ файлдар каталогын шығару;
- □ құрылған уақыты көрсетілген мерзімнен бұрын ашылған файлдарды өшіру;
- □ пайдаланылу саны ең көп болатын файлды таңдау.

Программа меню көмегімен сұхбаттасуды және енгізу кезінде кеткен қателерді тексеруді қамтамасыз етуі тиіс.

11-нұсқа

Пәндік көрсеткіш сызықтық тізім түрінде ұйымдастырылған.

Көрсеткіштң әрбір компонентінде сөз және осы сөз кездесетін беттердің нөмірлері жазылады. Бір сөзге қатысты беттер нөмірлерінің саны бірден онға дейін болуы мүмкін.

Келесі іс-әрекеттердің орындалуын қамтамасыз ететін программа құрыңыз:

□ пәндік көрсеткішті бастапқы қалыптастыру;

□ пәндік көрсеткішті шығару;

□ берілген сөз үшін беттер нөмірлерін шығару.

Программа меню көмегімен сұхбаттасуды және енгізу кезінде кеткен қателерді тексеруді қамтамасыз етуі тиіс.

12-нұсқа

Белгілі бір программаға арналған көмекші мәтін сызықтық тізім түрінде ұйымдастырылған.

Мәтіннің әрбір компонентінде термин (сөз) және осы терминге түсініктеме беретін мәтін жазылады. Бір терминге қатысты мәтін жолдарының саны бірден беске дейін болуы мүмкін.

Келесі іс-әрекеттердің орындалуын қамтамасыз ететін программа құрыңыз:

□ көмекші мәтінді бастапқы қалыптастыру;

□ көмекші мәтінді шығару;

□ берілген термин үшін көмекші мәтінді экранға шығару.

Программа меню көмегімен сұхбаттасуды және енгізу кезінде кеткен қателерді тексеруді қамтамасыз етуі тиіс.

13-нұсқа

Пәтер алмастыру бюросының картотекасы сызықтық тізім түрінде ұйымдастырылған.

Әрбір пәтер туралы төмендегі мәліметтер сақталады:

□ бөлмелер саны;

□ қабаты;

□ ауданы;

□ адресі (мекен-жайы).

Келесі іс-әрекеттердің орындалуын қамтамасыз ететін программа құрыңыз:

□ картотеканы бастапқы қалыптастыру;

□ пәтер алмастыруға тапсырыс енгізу;

□ картотекадан жарамды нұсқаны іздеу: бөлмелер саны мен қабаты бірдей, ал аудандарының айырмашылығы 10% шегінде болған жағдайда сәйкес картішке шығарылады және тізімнен өшіріледі, кері жағдайда түскен тапсырыс тізімге қосылады;

□ тізімді толық шығару.

Программа меню көмегімен сұхбаттасуды және енгізу кезінде кеткен қателерді тексеруді қамтамасыз етуі тиіс.

14-нұсқа

Ағылшын-қазақ сөздігі екілік бұтақ түрінде ұйымдастырылған.

Әрбір компонентте ағылшын тіліндегі сөз, оған сәйкес келетін қазақ тіліндегі сөз және осы компонентті қолдану санауышы сақталады.

Бастапқы кезде бұтақ ағылшын алфавитіне сәйкес құрылады. Сөздікті пайдалану барысында компонентті әрбір қолдану кезінде санауышқа бір қосылады.

Келесі іс-әрекеттердің орындалуын қамтамасыз ететін программа құрыңыз:

□ пайдалану санауышының нақты мәндері бар сөздікті бастапқы енгізуді қамтамасыз ету;

□ сөздіктің жаңа бейнесі келесі алгоритм бойынша екілік бұтақ түрінде қалыптастырылады: а) ескі сөздіктен пайдалану санауышының мәні ең үлкен болатын компонент ізделеді; ә) табылған компонент жаңа сөздікке енгізіліп, ескі сөздіктен өшіріледі; б) бастапқы сөздік аяқталғанға дейін а) пунктіне көшу;

□ жаңа және бастапқы сөздіктерді экранға шығару.

Программа меню көмегімен сұхбаттасуды және енгізу кезінде кеткен қателерді тексеруді қамтамасыз етуі тиіс.

15-нұсқа

Көпшілікке арналған сауалнаманың анкетасында сұрақтардың екі тобы бар.

Бірінші топта респондент туралы келесідей ақпарат болады:

□ жасы;

□ жынысы;

□ білімі (бастапқы, орташа, жоғары).

Екінші топта анкета сұрағының өзі жазылған, оған ИӘ немесе ЖОҚ деген жауап беріледі.

Келесі іс-әрекеттердің орындалуын қамтамасыз ететін программа құрыңыз:

□ анкеталарды бастапқы енгізуді қамтамасыз ету және олардан сызықтық тізім қалыптастыру;

□ анкеталарды талдау негізінде келесі сұрақтарға жауап беріледі: а) 40 жастан асқан, жоғары білімді қанша ер адам анкета сұрағына ИӘ деп жауап берді; ә) 30 жасқа толмаған, орташа білімді қанша әйел адам анкета сұрағына ЖОҚ деп жауап берді; б) 25 жасқа толмаған, бастапқы білімді қанша ер адам анкета сұрағына ИӘ деген жауап қайтарды;

□ барлық анкеталар мен сұрақтарға жауаптарды экранға шығару.

Программа меню көмегімен сұхбаттасуды және енгізу кезінде кеткен қателерді тексеруді қамтамасыз етуі тиіс.

16-нұсқа

Кітапханадағы кітаптар туралы ағымдағы ақпаратты қамтитын программа құрыңыз.

Әрбір кітап туралы төмендегідей ақпараттар бар:

□ ӘОЖ – әмбебап ондық жіктелу (УДК) нөмірі;

□ автордың тегі мен инициалдары;

□ кітап аты;

□ басылып шығарылған жылы;

□ кітапханадағы осы кітап саны.

Программа келесі іс-әрекеттердің орындалуын қамтамасыз етуі тиіс:

□ кітапханадағы барлық кітаптар туралы мәліметтерді бастапқы тізім түрінде қалыптастыру;

□ кітапханаға жаңадан түсетін кітаптар туралы мәліметтер қосу;

□ есептен шығарылатын кітаптар туралы мәліметтерді өшіру;

□ сұраныс бойынша кітапханадағы басылып шығарылған жылы бойынша реттелген кітаптардың бар-жоқтығы туралы мәліметтер беріледі.

17-нұсқа

Қаралық телефон станциясындағы телефондар мен олардың иелері туралы ақпарат жазылған абоненттер картотекасы сызықты тізім түрінде ұйымдастырылған.

Келесі іс-әрекеттердің орындалуын қамтамасыз ететін программа құрыңыз:

□ картотеканы сызықтық тізім түрінде бастапқы қалыптастыруды қамтамасыз етеді;

□ картотеканы толығымен экранға шығарады;

□ телефон нөмірі мен сөйлесу уақытын енгізеді;

□ телефон арқылы болған байланысты төлеуге хабарлама шығарады.

Программа меню көмегімен сұхбаттасуды және енгізу кезінде кеткен қателерді тексеруді қамтамасыз етуі тиіс.

18-нұсқа

Қаралық телефон станциясындағы телефондар мен олардың иелері туралы ақпарат жазылған абоненттер картотекасы екілік бұтақ түрінде ұйымдастырылған.

Келесі іс-әрекеттердің орындалуын қамтамасыз ететін программа құрыңыз:

□ картотеканы екілік бұтақ түрінде бастапқы қалыптастыруды қамтамасыз ету;

□ картотеканы толығымен экранға шығару;

□ телефон нөмірі мен сөйлесу уақытын енгізеді;

□ телефон байланысын төлеуге хабарлама шығарады.

Программа меню көмегімен сұхбаттасуды және енгізу кезінде кеткен қателерді тексеруді қамтамасыз етуі тиіс.

19-нұсқа

Темір жол бекетіндегі автоматтандырылған ақпараттық жүйеде алысқа жүретін пойыздардың жүру ережесі сақталған. Әр пойыз үшін төмендегі мәліметтер көрсетіледі:

□ пойыз нөмірі;

□ баратын станциясы;

□ жүру уақыты.

Ақпараттық жүйедегі мәліметтер сызықтық тізім түрінде ұйымдастырылған.

Келесі іс-әрекеттердің орындалуын қамтамасыз ететін программа құрыңыз:

□ ақпараттық жүйеге мәліметтерді бастапқы енгізуді және сызықтық тізімді құруды қамтамасыз етеді;

□ тізімді толығымен шығаруды орындайды;

□ пойыз нөмірін енгізіп, ол туралы барлық ақпаратты шығарады;

□ баратын станциясының атын енгізіп, сол станцияға дейін жүретін барлық пойыздар туралы мәліметтерді экранға шығарады.

Программа меню көмегімен сұхбаттасуды және енгізу кезінде кеткен қателерді тексеруді қамтамасыз етуі тиіс.

20-нұсқа

Темір жол бекетіндегі автоматтандырылған ақпараттық жүйеде алысқа жүретін пойыздардың жүру ережесі сақталған. Әр пойыз үшін төмендегі мәліметтер көрсетіледі:

□ пойыз нөмірі;

□ баратын станциясы;

□ жүру уақыты.

Ақпараттық жүйеде мәліметтер екілік бұтақ түрінде ұйымдастырылған.

Келесі іс-әрекеттердің орындалуын қамтамасыз ететін программа құрыңыз:

□ ақпараттық жүйеге бастапқы мәліметтерді енгізуді және екілік бұтақ құруды қамтамасыз етеді;

□ бұтақты толығымен экранға шығаруды орындайды;

□ пойыз нөмірін енгізіп, ол туралы барлық ақпаратты шығарады;

□ баратын станциясының атын енгізіп, сол станцияға дейін жүретін барлық пойыздар туралы мәліметтерді экранға шығарады.

Программа меню көмегімен сұхбаттасуды және енгізу кезінде кеткен қателерді тексеруді қамтамасыз етуі тиіс.

II БӨЛІМ

Объектіге бағытталған программалау

Кез келген программа ең соңғы үлгісінде процессорға арналған нұсқаулар жиынтығы болып табылады. Кез келген программалау тіліндегі жазылған командалар – осы нұсқаулар жиынтығының әрі ыңғайлы, әрі қарапайым жазбасы, ол программаны жазуды, түзетіп жөндеуді және уақыт өте келе өзгерістер енгізуді жеңілдетеді. Программалау тілінің деңгейі неғұрлым жоғары болған сайын, іс-әрекеттердің жазылу формасы соғұрлым қарапайым болып келеді. Мысалы, ассемблер тілінде циклді жүзеге асыру үшін регистрлерге айнымалылардың орналасуын қамтамасыз еткеннен кейін ғана нұсқаулар тізбегін жазуға болады, ал С немесе Паскаль тілдерінде бұл үшін бір-ақ операторды жазу жеткілікті.

Программа көлемінің ұлғаюына байланысты компьютер жадында барлық мәліметтерді тиянақты түрде сақтау мүмкін болмайды да, мәліметтерді құрылымдау керек болады, яғни негізгі мәліметтерді белгілеп алып, қалған маңызды еместерін ескермеу қажеттігі туындайды. Бұл процесс программаның абстракциялық деңгейін арттыру деп аталады.

Абстракцияны арттырудың алғашқы қадамы *функцияларды қолдану* болып табылады, мұнда функцияны жазып, түзетіп жөндегеннен кейін оның жүзеге асырылу ерекшеліктерін ескерудің қажеті жоқ, өйткені функцияны шақыру үшін тек оның интерфейсін білу жеткілікті. Егер ауқымды айнымалылар қолданылмаса, интерфейс толығымен функцияның тақырыбы арқылы анықталады.

Келесі қадам – ақпаратты құрылымдау және топтау арқылы оны табиғи түрге барынша жақын етіп бейнелеуді қамтамасыз ететін *өзіндік мәліметтер типтерін сипаттау* болып саналады. Мысалы, бір ғана құрылымның көмегімен қоймадағы тауардың бір түріне қатысты әртекті ақпаратты толығынан сипаттауға болады.

Өзіндік мәліметтер типтерімен жұмыс істеу үшін арнайы функциялар қажет. Оларды осы мәліметтер типтерінің сипаттамаларымен бірге программаның бір жеріне топтастырып, мүмкіндігінше программаның басқа бөліктерінен жекелеген жөн. Осындай типтер мен функцияларды қолдану үшін олардың қалай жазылғанын толық білудің қажеті жоқ, тек қана солардың интерфейстерінің сипаттамаларын білу жеткілікті. Олармен жұмыс істеуге арналған мәліметтер типтері мен функциялардың сипаттамаларын *модульдерге біріктіре отырып,*

модуль қолданушысынан онша маңызды емес ұсақ-түйектерді жасыру программаны құрылымдаудың одан ары дамуы болып табылады.

Жоғарыда айтылған абстракцияны арттырудың үш тәсілінің де мақсаты программа құрылымын жеңілдету, яғни оны *көлемдері үлкен болғанмен, жалпы саны аз болып келетін блоктар түрінде көрсету және олардың арасындағы байланыстарды барынша азайту* болып табылады. Бұлар ақпараттың аса үлкен көлемдерін басқарып, аса күрделі программаларды жеңіл түзетуге мүмкіндік береді.

Класс түсінігін енгізу модульділік идеясының заңды жалғасы болып саналады. Класта мәліметтер құрылымдары мен оларды өңдеу функциялары біріктіріледі. Класс тек өз интерфейсі арқылы ғана қолданылады, оның орындалу ерекшеліктері класты пайдаланушы үшін маңызды емес. Кластар идеясы нақты әлем объектілері құрылымын бейнелейді – кез келген заттың немесе процестің сипаттамалары немесе ерекше белгілері болады, басқаша айтқанда, қасиеттері мен тәртіптері болады емес пе. Программалар, көбінесе нақты әлемдегі заттарды, құбылыстарды және процестерді модельдеуге арналады[1], сондықтан программалау тілінде модельдерді бейнелеуге арналған құрал болғаны жөн.

Класс қолданушы анықтайтын мәліметтер типі болып табылады. Класта қандай да бір заттың немесе процестің қасиеттері мен тәртіптері мәліметтер өрістері (құрылымдағы сияқты) және солармен жұмыс істеуге арналған функциялар түрінде беріледі. Құрылатын мәліметтер типінің қасиеттері стандартты типтердің қасиеттерімен бірдей деуге болады. Тип компьютердің жадындағы мәліметтердің ішкі бейнесін, осы типтегі шамалар қабылдай алатын мәндер жиынын, сонымен қатар осы шамаларға қолданылатын операциялар мен функцияларды анықтайтынын айта кетейік. Осылардың барлығын класс құрамында да беруге болады.

Кластың маңызды қасиеті – оның жұмыс істеу ерекшеліктерінің қолданушыдан интерфейс арқылы жасырылуында (мысалы, нақты өмірде де автокөлікті оның қозғалтқышының қалай жұмыс істейтінін білместен-ақ жүргізуге болады, телефонды да «сигнал қалай жүретінін, байланыс принциптерін және кабельді кім тартқанын білмей-ақ қолдана аламыз»[2]). Класс интерфейстері болып оның әдістерінің тақырыптары есептеледі. Сонымен, класты нақты әлем объектісінің моделі ретінде сыртқы ортаға қатысты тұйықталған қара жәшік ретінде қарастыруға болады.

Кластар идеясы объектіге бағытталған программалаудың (ОБП) негізі болып табылады. ОБП негізгі принциптері Simula-67[3] және Smalltalk тілдерінде де құрылған болатын, алайда оны түсіну қиыншылықтары мен жүзеге асы-

[1]Мысалы, Б. Страуструп C++ тілін телефон коммутациялық жүйелерін модельдеу үшін құрған болатын.

[2]©БГ

[3]Шыққан жылдарына назар аударыңыз: программалауда жаңа ойлар айтарлықтай жиі пайда болмайды.

рылу тиімділігі төмен болғандықтан, кең қолданыс таппады. C++ тіліне бұл концепциялар тиімді, көркем, әрі қайшылықсыз жүзеге асырылды, осының арқасында бұл тіл кең таралып, оның көптеген мүмкіндіктері басқа да программалау тілдеріне енгізіле бастады.

ОБП – бұл тілге қосылған жай ғана жаңа құралдар жиынтығы емес (C++ тілінде ОБП қолданбай-ақ жақсы программалар жазуға және керісінше, объектілерді сүйемелдеудің арнайы құралдары жоқ тілде де объектілік программа жазуға болады). ОБП-ны жиі программалаудың жаңа парадигмасы деп атайды.

«Парадигма» термині білімді ұйымдастыру тәсілін құратын теориялар, стандарттар және әдістер жиынының бірігуін, басқаша айтқанда, әлемге деген жаңа көзқарасты білдіреді. Программалауда бұл термин есептеу моделін анықтау үшін, яғни ақпаратты құрылымдауда, есептеулер мен мәліметтерді ұйымдастыруда кең қолданылады. Объектіге бағытталған программа объектілер мен солардың өзара байланыстарын көрсететін терминдер арқылы құрылады.

Абстракция дәрежесін таңдау шешілетін есеп типімен анықталады. Қарапайым есептерді шығару үшін күрделі технологияларды қолданбаған жөн, ал күрделі мәселелерді қарапайым құралдармен шешуге тырысу сәтсіздікке ұшыратады. Басқа жағынан қарасақ, күрделі технологияларды меңгеру уақыттың мол жұмсалуын талап етеді.

Мысалы, егер хат мәтінін теру керек болса, онда бұл үшін 10 минутта өз әжеңді де үйретуге болатын, мүмкіндіктері аз, қарапайым мәтіндік редактор жетіп жатыр; формулалары бар мақала дайындау үшін одан күрделілеу Microsoft Word сияқты мәтіндік процессорды меңгеру қажет, ал көрнекі суреттері бар жарнамалық құжат дайындау үшін үйренуге бірнеше апта жұмсалатын шағын баспа құралдарының бірін игеру керек. Программалауда да дәл осылай: ОБП идеяларын түсініп, оны тәжірибе жүзінде қолдану оңай іс емес (оларды дұрыс қолданбаудың пайдасынан гөрі зияны көп болып келеді), ал қазіргі қолданылып жүрген стандартты кітапханаларды меңгеру уақытты да және бастапқы дайындықтың да жоғарғы деңгейін талап етеді.

Мәліметтер типінің «класс» ұғымының нақты шамалары *класс даналары (экземплярлары) немесе объектілер* деп аталады. Объектілер бір-бірімен хабарламаларды жіберу және қабылдау арқылы өзара әрекеттеседі. Хабарлама – бұл керекті параметрлер жиынын қамтитын, белгілі бір іс-әрекетті орындауға арналған сұраныс. Хабарламалар механизмі соларға сәйкес функцияларды шақыру арқылы жүзеге асырылады. Сонымен, объектіге бағытталған программалау көмегімен «оқиғалық-басқарылатын модель» жеңіл жүзеге асырылады, мұнда мәліметтер екпінді түрде болады және программалық кодтың кез келген бір бөлігін шақыруды басқарады[1].

[1]Оқиғалық-басқарылатын модель ОБП-ның бір бөлігі емес және ол объектілерді қолданбай-ақ жүзеге асырылуы мүмкін (мысалы, API функцияларын қолданып Windows ортасында C тілінде программалау).

ЕСКЕРТУ

Оқиғалық-басқарылатын модельдің жүзеге асырылу мысалы ретінде меню арқылы басқарылатын кез келген программаны алуға болады. Орындалуға жіберілгеннен кейін мұндай программа қолданушының іс-әрекеттерін күтіп бақылап отырады және олардың кез келгеніне дұрыс жауап қайтаруы тиіс. Оқиғалық модель дәстүрлі (директивалық) үлгіге қарама-қарсы болып табылады, дәстүрлі үлгіде код мәліметтерді басқарады: программа іске қосылғаннан кейін қолданушыға бекітілген алгоритмнің орындалуын қатаң қадағалайтын әрекеттер тізбегін орындауға ұсыныс береді (мәліметтер енгізу, режимді таңдау).

ОБП-ның негізгі қасиеттері инкапсуляция, мұралау және полиморфизм болып табылады. Төменде олардың мағыналары қысқаша түсіндіріледі, ал олар туралы толық мағлұматты[1] кітаптың осы бөлімін оқу арқылы алуға болады.

Мәліметтердің оларды өңдеу функцияларымен бірігуі кезінде осы мәліметтерді қолдануға қажетсіз ақпаратты жасырудың жүзеге асуы *инкапсуляция* (encapsulation) деп аталады. Бұл идея жаңалық емес, ол құрылымдық және модульдік программалауда да қолданылған, ал ОБП-да ол өзінің логикалық жалғасын тапты. Инкапсуляция программаның абстракция деңгейін жоғарылатады: класс мәліметтері және оның функцияларының жүзеге асырылуы абстракция деңгейінен төмен орналасқан және программа жазу үшін олар туралы ақпараттың қажеті жоқ. Сонымен қатар, интерфейс өзгермеген жағдайда инкапсуляция кластың жүзеге асырылуын программаның негізгі бөлігін өзгертпестен түрлендіруге мүмкіндік береді (мысалы, мәліметтердің сақталу тәсілін жиымнан стекке өзгерту керек болғанда). Жоғарыда бірнеше рет қайталап айтылғандай, программаны өзгерту қарапайымдылығы оның сапасының өте маңызды алғышарттарының бірі болып табылады.

Инкапсуляция класты басқа ортада қолданған кезде ол өзіне тиісті емес жады аймақтарын бүлдірмейді және көптеген программаларда қолдануға арналған кластар кітапханаларын құруға мүмкіндік береді.

Мұралау – бұл ұрпақтары өздерінің ата-тектерінің барлық қасиеттерін мұралайтындай, оларды өзгерту және жаңа қасиеттер қосу мүмкіндіктеріне ие болатындай кластар иерархиясын құру мүмкіндігі. Мұралау кезінде қасиеттер қайталап сипатталмайды, сол себепті ол программа көлемін қысқартады. Әртүрлі кластардың ортақ белгілерін бір класс-тегіне тағайындау абстракцияның күшті механизмі болып табылады – кез келген ғылым да қарастырылатын пәндік аймақтың күрделілігін жеңуге мүмкіндік беретін абстракциялаудан және жіктеуден басталатыны белгігі ғой.

Кластар иерархиясы бұтақ тәріздес құрылым түрінде бейнеленеді, мұнда жалпыға ортақ кластар түбірге жақын, ал арнайы кластар ішкі бұтақтар мен жапырақтарда орналасады. C++ тілінде әрбір кластың ұрпақтары мен ата-тегінің саны шектелмеген. Кейде ата-тегі сыртқы класс немесе суперкласс, ал ұрпақтары субкласс немесе ішкі класс деп аталады.

ОБП-ның үшінші негізі *полиморфизм* болып табылады. *Полиморфизм – иерархияның әртүрлі кластарында мағыналары ұқсас әрекеттер үшін бір*

атауды қолдану және қажетті әрекетті программаның атқарылуы барысында еркін түрде таңдау мүмкіндігі.

С++ тілінде полиморфизм түсінігі кең қолданылады. Полиморфизмнің қарапайым мысалы ретінде кітаптың бірінші бөлімінде қарастырылған функцияларды асыра жүктеуді алуға болады, бұл кезде прототипінің берілетін параметрлерге сәйкестігі бойынша бірнеше нұсқалардың ішінен барынша жарамды функция таңдалады. Басқа мысал ретінде функциялар шаблондарын қолдануды атауға болады (алдағы уақытта біз кластар шаблондарын да қарастырамыз), бұл кезде бір ғана код параметр ретінде берілген типке сәйкес өзгертіледі. Полиморфизм түсінігін виртуалды тәсілдер механизмімен (ол 222 б. қарастырылады) жиі байланыстырады.

Программаның объектілер тәртіптері терминдерімен бейнеленуінің арқасында программалау кезінде пәндік аймаққа жақынырақ түсініктер қолданылады, демек, бұл программаны оқу мен түсінуді оңайлатады. Осы қасиет ОБП-дың үлкен артықшылығы болып табылады. Алайда объектіге бағытталған программалауды жобалау күрделі мәселе болып келеді, өйткені процеске тағы бір маңызды кезең – *кластар иерархиясын құру* қосылады.

Нашар жобаланған иерархия күрделі және түсініксіз программалардың құрылуына алып келеді. ОБП-ды қолданудың басқа бір кедергісі меңгерілуі қажет ақпарат көлемінің үлкендігі және оны меңгерудің қиындығы болып саналады.

Жобалауды бастаудан бұрын объектіге бағытталған тәсілді қолдану қажеттілігін анықтап алған дұрыс. Егер кластардың иерархиясы қажет болмаса, әдетте модульдік технологиямен шектелу жеткілікті, мұндайда стандартты кітапхананың кластарын және өзіндік кластарды тиімді пайдалануға болады. Стандартты кластарды қолдану үшін алдымен қажетті синтаксис пен механизмдерді, содан кейін барып, осы кластардың нақты қасиеттерін[1] меңгерген жөн болатыны сөзсіз.

[1]Осы кітапта көрсетілген программаларда cin және cout стандартты кітапхана кластары қолданылған. Басқа кластарды қолдану оңай сияқты көрінбеуі мүмкін, бірақ нәтижесі соған жұмсалған күшті ақтайды

4-ТАРАУ

Кластар

Бұл тарауда объектіге бағытталған программаны жазудың іргетасын қалайтын объектілерді сипаттаудың синтаксистік ережелері қарастырылады. Кітаптың осы тарауының соңында ОБП-дың принциптері мен объектілі программаларды жобалау әдістерін талқылауға қайта ораламыз, өйткені «тілдің қасиеттерін қалыптастыратын идеяларды жақсы түсіну арқылы ғана шеберлікке қол жеткізуге болады» (Б. Страуструп).

Класты сипаттау

Класс қолданушы анықтайтын мәліметтердің абстрактылы типі болып табылады, ол мәліметтер және олармен жұмыс істеуге арналған функциялар түрінде берілген шынайы объектінің үлгісін бейнелейді.

Класс мәліметтері *өрістер* (құрылым өрістеріне ұқсас), ал класс функциялары *әдістер*[1] деп аталады. Өрістер мен әдістерді *класс элементтері* деп атайды. Алғашқы қарастырыстырудағы класс сипаттамасы төмендегідей түрде болады:

```
class <атауы>{
   [ private: ]
      <жасырын элементтерді сипаттау>
   public:
      <қолжетімді элементтерді сипаттау>
};  // Сипаттау нүктелі үтірмен аяқталады
```

Мысалдағы **private** және **public** спецификаторлары класс элементтерінің көрінуін басқарады. **private** түйінді сөзінен кейін сипатталған элементтер класс ішінде ғана көрінеді. Келісім бойынша, класта қатынас құрудың осы түрі қабылданған. Класс интерфейсі **public** спецификаторынан кейін сипатталады. Кез келген спецификатордың әрекеті келесі спецификаторға дейін немесе класс соңына дейін сақталады. Бірнеше **private** және **public** секцияларын беруге болады, олардың жазылу реттілігі маңызды емес.

Класс өрістері:

- осы класс типінен басқа кез келген типті қабылдай алады (алайда, осы класқа нұсқауыш немесе сілтемелер болуы да мүмкін);

[1]Әдебиетте «мәлімет-мүшелер» және «функция-мүшелер», сонымен қатар «компоненттік мәліметтер» және «компоненттік функциялар» терминдері жиі кездеседі, бірақ түсіндіруге келмейтін себептермен мен «өрістер» және «әдістер» терминдерін пайдаланамын. Контекст бойынша әдеттегі функция емес, кластың функция-мүшелері туралы сөз болғаны түсінікті жағдайларда әдістерді жай ғана «функциялар» деп атайды.

- **const** модификаторы көмегімен сипатталуы мүмкін, мұндайда олар тек бір рет қана инициалданады (конструктор арқылы) және өзгеруі мүмкін емес;
- **static** модификаторының көмегімен сипатталуы мүмкін (бұл туралы «Статикалық өрістер» бөлімінде, 201 б. жазылады), бірақ **auto**, **extern** және **register** модификаторлары арқылы жазылмайды.

Сипаттау кезінде өрістерді инициалдауға тыйым салынған.

Кластар ауқымды (кез келген блоктан тыс жарияланған) және жергілікті (блок ішінде, мысалы, функцияның немесе басқа кластың ішінде жарияланған) болуы мүмкін.

Төменде жергілікті (локалды) кластың кейбір ерекшеліктері көрсетілген:

- жергілікті кластың ішінде өзі сипатталған аймағындағы типтерді, статикалық (**static**) және сыртқы (**extern**) айнымалыларды, сыртқы функциялар мен тізбелер элементтерін қолдануға болады; осы аймақтағы автоматты айнымалыларды қолдануға тыйым салынған;
- жергілікті кластың статикалық элементтері болмайды;
- осы кластың әдістерін тек кластың ішінде сипаттауға болады;
- егер бір класс басқа класс ішіне кірістірілсе, олардың бір-бірінің элементтеріне қол жеткізе алатын ерекше мүмкіндіктері болмайды, оларға тек ортақ ережелерге сәйкес түрде пайдаланыла алады.

Мысал ретінде компьютерлік ойын кейіпкерін модельдейтін класс құрайық. Бұл үшін оның қасиеттері (мысалы, қармалауыштар санын, гранататқыштардың болуын және оның күшін) мен тәртіптерін беруіміз керек. Әрине, мұндай мысал схематикалық түрде болады, өйткені ол тек синтаксисті көрсету үшін келтірілген.

```
class monstr{
      int health, ammo;
   public:
      monstr(int he = 100, int am = 10){ health = he;
                                         ammo = am;}
      void draw(int x, int y, int scale, int position);
      int get_health() {return health;}
      int get_ammo() {return ammo;}
};
```

Бұл кластың құрамында екі жасырын өріс бар, олар – **health** және **ammo**, олардың мәндерін сырттан **get_health()** және **get_ammo()** әдістерінің көмегімен алуға болады. Мұнда өрістерді әдістер арқылы пайдалану қолдан жасалған күрделілік болып көрінеді, алайда шынайы кластардың өрістері ретінде күрделі динамикалық құрылымдар алынуы мүмкін және олардың элементтерінің мәндеріне қол жеткізу оңай емес екенін ескеру керек. Сонымен қатар, класс интерфейсіне тиместен, осы құрылымдарға өзгерістер енгізу мүмкіндігі өте маңызды болып табылады.

Кластың барлық әдістері жасырын өрістерді тікелей пайдалана алады, басқаша айтқанда, класс функцияларының тұлғалары класс элементтерінің **private** көріну аймағына кіреді.

Берілген класта әдістердің үш анықтамасы және бір жариялануы (**draw** әдісі) берілген. Егер әдіс тұлғасы (денесі) класс ішінде анықталған болса, онда ол құрамдас (**inline**) болып табылады. Көбінесе қысқа әдістерді құрамдас әдістер ретінде жасайды. Егер класс ішінде тек қана әдістің жариялануы (тақырыбы) берілген болса, әдістің өзі программының басқа жерінде көріну аймағына қол жетімділік операциясы (**::**) арқылы анықталуы керек:

```
void monstr::draw(int x, int y, int scale, int position){
   /* әдіс тұлғасы */
}
```

Әдісті **inline** (қарапайым функциялардағы сияқты ол ұсыныс сипатында болады) директивасының көмегімен кластан тыс та құрамдас әдіс ретінде анықтауға болады.

```
inline int monstr::get_ammo(){
   return ammo;
}
```

Әрбір класс құрамында атауы класс атауымен бірдей болып келетін кем дегенде бір әдіс болады. Ол конструктор деп аталады және класс объектісін құру кезінде автоматты түрде шақырылады. Конструктор объектіні инициалдау үшін қажет. Конструкторды автоматты түрде шақыру инициалданбаған айнымалыларды пайдалануға байланысты туындайтын қателердің алдын алуға мүмкіндік береді. Конструкторлар туралы толығырақ мәлімет «Конструкторлар» бөлімінде, 196 б. қарастырылады.

struct және **union** сияқты мәліметтер типтері кластың түрлері болып табылады, олардың арасындағы айырмашылық кейінірек, 226-227 б. түсіндіріледі. Класты сипаттаудың басқа бір мысалы, сонымен қатар жергілікті класс мысалы «Кластардың шаблондарын құру» бөлімінде, 228 б. қарастырылады.

Объектілерді сипаттау

«Класс» типті нақты көрсетілген айнымалылар класс даналары (экземплярлары) немесе объектілер деп аталады. Объектілердің өмірлік кезеңдері мен көріну мүмкіндігі олардың түрлеріне және сипатталу орнына байланысты болады және C++ тілінің жалпы ережелеріне бағынады:

```
monstr Vasia;          // Келісім бойынша параметрлер
                       //берілген monstr класының объектісі
monstr Super(200, 300); // Тікелей инициалданған объект
monstr stado[100];     // Келісім бойынша параметрлер
                       //берілген жиым объектілері
```

```
monstr *beavis = new monstr (10); // Динамикалық объект
        //(екінші параметр келісім бойынша беріледі)
monstr &butthead = Vasia;          // Объектіге сілтеме
```

Әрбір объектіні құру кезінде оның барлық өрістерін сақтауға жеткілікті жады көлемі бөлінеді және оларды инициалдайтын конструктор автоматты түрде шақырылады. Кластың әдістері қайталанып жаппай таралмайды. Объект әрекет ету аймағынан шыққан кезде ол жойылады да, бірден автоматты түрде деструктор шақырылады (деструкторлар 204 б. сипатталған).

Объект элементтеріне қол жеткізу құрылым өрістеріне қол жеткізуге ұқсас орындалады. Бұл үшін элементті объект атауы арқылы шақырғанда **.** (нүкте) операциясы және нұсқауыш арқылы шақырғанда **->** операциясы қолданылады, мысалы:

```
int n = Vasia.get_ammo();
stado[5].draw;
cout << beavis->get_health();
```

Мұндай жолмен тек **public** спецификаторы бар элементтерге қол жеткізуге болады. **private** спецификаторы бар элементтердің мәндерін алу немесе өзгерту үшін соған сәйкес әдістерге жүгіну керек.

Өрістерін өзгертуге тыйым салынатын тұрақтылық объект құруға болады. Оған тек тұрақтылық әдістер қолданылуы тиіс:

```
class monstr{
...
int get_health() const {return health;}
};
const monstr Dead(0,0);   //Тұрақтылық объект
cout << Dead.get_health();
```

Тұрақтылық әдіс:

- параметрлер тізімінен кейін **const** түйінді сөзі арқылы жарияланады;
- класс өрістерінің мәндерін өзгерте алмайды;
- тек тұрақтылық әдістерді шақыра алады;
- кез келген объектілер үшін (тек тұрақтылық объектілер үшін ғана емес) шақыртылуы мүмкін.

Өріс мәндерін алуға арналған әдістерді тұрақтылық әдістер ретінде сипаттаған жөн.

this нұсқауышы

Әрбір объектіде класс өрістерінің өзіндік экземпляры, яғни данасы болады. Компьютер жадында класс әдістері тек бір экземплярда (данада) болады және барлық объектілер оны ортақ түрде қолданады, сондықтан әдістері шақырылған объект өрістерімен тек осы әдістердің жұмыс істеуін қамтамасыз ету керек. Мұндай әрекет функцияға **this** жасырын параметрін беру арқылы

орындалады, бұл параметрде функцияны шақырған объектіге тұрақтылық нұсқауыш сақталады. **this** нұсқауышы объект элементтеріне сілтеме жасау үшін әдіс ішінде айқындалмаған түрде қолданылады. Бұл нұсқауыш айқын түрде, негізінен, әдістен нұсқауышты **(return this;)** немесе шақырған объектіге сілтемені **(return *this;)** қайтару үшін қолданылады.

this нұсқауышының қолданылуын көрсету үшін жоғарыдағы **monstr** класына жаңа әдіс қосамыз, екі мүмкіндіктің біріне сілтеме қайтаратын бұл әдіс құбыжықтардың ішіндегі денсаулығы мықтысын (**health** өрісі) таңдайды, олардың алғашқысы – әдісті шақырып, екіншісі – оған параметр ретінде берілетін (әдісті класты сипаттайтын **public** секциясына орналастыру керек) сілтеме қайтарады:

```
monstr & the_best(monstr &M){
   if( health > M.health) return *this;
   return M;
}
... monstr Vasia(50), Super(200);
// Жаңа  Best объектісі Super объектісі өрістерінің
// мәндерімен инициалданады:
monstr Best = Vasia.the_best(Super);
```

Сонымен қатар, **this** нұсқауышын класс өрісінің атауы әдістің формалды параметрінің атауымен сәйкес келген жағдайда, класс өрісін идентификациялау үшін қолдануға болады. Өрісті идентификациялаудың басқа бір тәсілі – көріну аймағына қол жеткізу операциясын пайдаланады:

```
void cure(int health, int ammo){
   this -> health += health; // this сөзін қолдану
   monstr:: ammo += ammo;    // :: операциясын қолдану
}
```

Конструкторлар

Конструктор объектіні инициалдау үшін қажет және ол құрылған кезде автоматты түрде шақырылады. Төменде конструкторлардың негізгі қасиеттері көрсетілген.

□ Конструктор *мән қайтармайды,* тіпті **void** типті мәнді де. Конструкторға нұсқауыш алу мүмкін емес.

□ Кластың құрамында әртүрлі инициалдау түрлеріне арналған әртүрлі параметрлі *бірнеше конструкторлар* болуы мүмкін (мұндайда асыра жүктеу механизмі қолданылады).

□ Параметрсіз шақырылатын конструктор *келісім бойынша құрылған конструктор* деп аталады.

□ *Конструктор параметрлері* осы кластан басқа кез келген типте болуы мүмкін. Параметрлер мәндерін келісім бойынша тағайындауға болады. Олар конструкторлардың тек біреуінде берілуі мүмкін.

□ Егер программалаушы бірде-бір конструкторды көрсетпеген болса, онда компилятор оны *автоматты түрде* құрады. Мұндай конструктор класс өрістеріне арналған келісім бойынша конструкторларды және базалық кластардың келісім бойынша конструкторларын шақырады («Қарапайым мұралау» бөлімін қараңыз, 217 б.). Егер класс құрамында тұрақтылар немесе сілтемелер бар болатын болса, онда класс объектісін құруға талпыныс жасау кезінде қателік туындайды, өйткені оларды нақты мәндермен инициалдау қажет, ал келісім бойынша конструктор мұны жасай алмайды.

□ *Конструкторлар мұраланбайды*.

□ Конструкторларды **const, virtual** және **static** модификаторлары арқылы сипаттауға болмайды.

□ Ауқымды объектілердің конструкторлары **main** функциясынан бұрын шақырылады. Жергілікті объектілер олардың әрекет ету аймағы екпінді болған сәтте бірден құрылады. Конструктор уақытша объектіні құрған кезде де іске қосылады (мысалы, функциядан объектіні беру кезінде).

□ Егер программада төмендегі синтаксистік конструкциялардың бірі кездесетін болса, онда конструктор шақырылады:

```
класс_атауы объект_атауы [(параметрлер тізімі)];
// Параметрлер тізімі бос болмауы керек
класс_атауы (параметрлер тізімі);
// Атауы жоқ объект құрылады (тізім бос болуы мүмкін)
класс_атауы объект_атауы = өрнек;
//  Атауы жоқ объект құрылады және көшіріледі
```

Мысалдар:

```
monstr Super(200, 300), Vasia(50), Z;
monstr X = monstr(1000);
monstr Y = 500;
```

Бірінші операторда үш объект құрылады. Көрсетілмеген параметрлердің мәндері келісім бойынша тағайындалады.

Екінші операторда **health = 1000** параметрі бар (екінші парметрдің мәні келісім бойынша бекітіледі) атауы жоқ объект құрылады. Атауы жоқ объект көшірілетін **X** объектісіне компьютер жады бөлінеді.

Соңғы операторда **health = 500** параметрі бар (екінші парметрдің мәні келісім бойынша бекітіледі) атауы жоқ объект құрылады. Атауы жоқ объект көшірілетін **Y** объектісіне жады бөлінеді. Объектіні құрудың мұндай формасын объектіні инициалдау үшін бір ғана параметрді көрсету жеткілікті болғанда ғана қолдануға болады.

Бірнеше конструкторы бар класс мысалы ретінде жоғарыда сипатталған **monstr** класына түсін (**skin**) және атын (**name**) беретін өрістерді қосу арқылы жетілдіреміз:

```
enum color {red, green, blue}; // Түстің мүмкін мәндері
class monstr{
      int health, ammo;
      color skin;
      char *name;
   public:
      monstr(int he = 100, int am = 10);
      monstr(color sk);
      monstr(char * nam);
      int get_health() {return health;}
      int get_ammo() {return ammo;}
   ...
};
//-----------------------------------
monstr::monstr(int he, int am){
   health = he; ammo = am; skin = red; name = 0;
}
//-----------------------------------
monstr::monstr(color sk){
   switch (sk){
      case red:   health = 100; ammo = 10; skin = red;
                  name = 0; break;
      case green: health = 100; ammo = 20; skin = green;
                  name = 0; break;
      case blue:  health = 100; ammo = 40; skin = blue;
                  name = 0; break;
   }
}
//-----------------------------------
monstr::monstr(char * nam){
   name = new char [strlen(nam) + 1];
   // Нөл-символды сақтау үшін тіркес ұзындығына 1 қосылады
    strcpy(name, nam);
    health = 100;
    ammo = 10; skin = red;
}
...
//-----------------------------------
monstr * m = new monstr ("Ork");
monstr Green (green);
```

Жоғарыда көрсетілген конструкторлардың біріншісі келісім бойынша құрылған конструктор болып табылады, себебі оны параметрлерсіз шақыруға болады. Енді **monstr** класының объектілерін әртүрлі тәсілдермен инициалдауға болады, қажетті конструктор жақшалар ішіндегі мәндер тізіміне байланысты шақырылады. Бірнеше конструкторды бергенде асыра жүктелген функцияларды жазу кезінде қолданылатын ережелер сақталуы тиіс – компилятордың қажетті нұсқаны тану мүмкіндігі болуы керек.

ЕСКЕРТУ

Конструкторларды ғана емес, сонымен қатар кластың басқа да әдістерін асыра жүктеуге болады.

Конструктордағы өрістерді инициалдаудың тағы да бір тәсілі бар (жоғарыдағы программада қолданылған өрістерге формалды параметрлер мәндерін меншіктеуден басқа) – тақырып пен конструктор тұлғасының аралығындағы қос нүктеден кейін орналасқан инициализаторлар тізімінің көмегімен:

```
monstr::monstr(int he, int am);
   health (he), ammo (am), skin (red), name (0){}
```

Өрістер үтір арқылы тізіліп жазылады. Әрбір өріс үшін жақшалар ішінде оны инициалдаушы мән көрсетіледі, ол өрнек болуы мүмкін. *Өріс-тұрақтыларды, өріс-сілтемелерді* және *өріс-объектілерді* инициалдауды бұл тәсілді қолданбай орындау мүмкін емес. Соңғы аталған жағдайда жақша ішінде көрсетілген параметрлерге сәйкес конструктор шақырылады.

ЕСКЕРТУ

Конструктор инициалдау кезінде туындайтын қателік туралы хабарлау үшін мән қайтара алмайды. Бұл үшін ерекше жағдайларды (аластамаларды) өңдеу механизмін қолдануға болады («Конструкторлар мен деструкторлардағы аластамалар» бөлімін қараңыз, 247 б.).

Көшіру конструкторы

Көшіру (көшірме) конструкторы – бұл жалғыз параметр ретінде дәл осы кластың объектісіне сілтемені қабылдайтын конструктордың арнайы түрі:

```
T::T(const T&) { ... /* Конструктор тұлғасы */ }
```

мұндағы T – кластың атауы.

Жаңа объект бұрыннан бар объектіні көшіру арқылы құрылған жағдайда осы конструктор:

- жаңа объектіні басқа объектіні инициализациялау арқылы сипаттаған кезде;
- объектіні функцияға мәні арқылы беру кезінде;
- объектіні функциядан қайтару кезінде[1] шақырылады.

Егер программалаушы бірде-бір көшіру конструкторын көрсетпеген болса, онда компилятор оны автоматты түрде құрады. Мұндай конструктор өрістерді бір-бір элементтен көшіруді орындайды. Егер класс құрамында нұсқауыштар немесе сілтемелер бар болса, бұл дұрыс емес болар, өйткені көшірме де, түпнұсқа да жадының бір ғана аймағына сілтеме жасайды.

monstr класы үшін көшіру конструкторын жазайық. Оның құрамында символдар тіркесіне нұсқайтын **name** атты өріс бар болғандықтан, көшіру конструкторы жаңа тіркеске жады бөліп, оған бастапқы тіркесті көшіруі тиіс:

```
monstr::monstr(const monstr &M){
   if (M.name){
      name = new char [strlen(M.name) + 1];
      strcpy(name, M.name);}
   else name = 0;
   health = M.health; ammo = M.ammo; skin = M.skin;
}
...
monstr Vasia (blue);
monstr Super = Vasia;      // Көшіру конструкторы
                           // жұмыс істейді
monstr *m = new monstr ("Ork");
monstr Green = *m;         // Көшіру конструкторы
                           // жұмыс істейді
```

ЕСКЕРТУ

Кез келген басқа типтегі бір параметрді қабылдайтын класс конструкторы *түрлендіру конструкторы* деп аталады, өйткені ол параметр типінен осы класс типіне түрлендіруді жүзеге асырады.

Иерархияға кіретін кластардың конструкторларын жазу ережелері «Қарапайым мұралау» бөлімінде, 217 б. сипатталған.

Кластың статикалық элементтері

static модификаторының көмегімен кластың статикалық өрістерін және әдістерін сипаттауға болады. Оларды класс аймағының шеңберінде ғана қолжетімді ауқымды айнымалылар немесе функциялар ретінде қарастыруға болады.

[1]Сонымен қатар, ерекше жағдайларды өңдеу кезінде.

Статикалық өрістер

Статикалық өрістер кластың барлық объектілеріне ортақ мәліметтерді, мысалы, объектілер санын немесе барлық объектілер бөле отырып пайдаланатын ресурсқа сілтемені сақтау үшін қолданылады. Бұл өрістер кластың барлық объектілері үшін жалғыз нұсқада болады, яғни олар қайталанып жазылмайды.

Төменде статикалық өрістердің ерекшеліктері көрсетілген.

□ Құрылған объектілер санына тәуелсіз түрде (олар жоқ болған жағдайда да) статикалық өріске жады инициалдау кезінде бір рет бөлінеді және таңдау операциясы арқылы емес, әрекет ету аймағына қатынасу операциясының көмегімен инициалданады (анықтамасы функциядан тыс жазылу керек):

```
class A{
   public:
      static int count; // Класс құрамында жариялау
};
...
int A::count;           // Ауқымды аймақта анықтау
// Келісім бойынша нөлмен инициалданады
/* int A::count = 10; Кездейсоқ мәнмен инициалдану мысалы */
```

□ Статикалық өрістеріне класс атауы арқылы да, объект атауы арқылы да қол жеткізуге болады:

```
A *a, b;
...
cout << A::count << a->count << b.count;
// бірдей нәтиже шығарылады
```

□ *Статикалық өрістерге қол жеткізу спецификаторларының әрекеті таралады, сондықтан* **private** *түрінде сипатталған статикалық өрістерді* жоғарыда көрсетілген әрекет ету аймағына қол жеткізу операциясының көмегімен өзгертуге болмайды. Мұны тек қана статикалық әдістердің көмегімен орындауға болады (ары қарай қараңыз).

□ *Статикалық өріс пайдаланатын жады объект көлемін* **sizeof** *операциясы көмегімен анықтау кезінде ескерілмейді.*

Статикалық әдістер

Статикалық әдістер кластың статикалық өрістерін пайдалану үшін қажет. Олар тек тікелей статикалық өрістерді пайдалана алады және кластың тек басқа статикалық әдістерін шақыра алады, өйткені оларға жасырын **this** нұсқауышы берілмейді. Статикалық әдістерді пайдалану дәл статикалық өрістерді пайдалану сияқты – класс атауы арқылы немесе кластың кем дегенде бір объектісі құрылған жағдайда объект атауы арқылы орындалады.

```
class A{
      static int count;  // count өрісі жасырын
   public:
      static void inc_count() { count++; }
      ...
};
...
A::int count;      // ауқымды аймақта анықтау
void f(){
   A a;
   // a.count++ түрінде жазуға болмайды, count өрісі
   // жасырын
   // Өрісті статикалық әдіс көмегімен өзгерту:
   a.inc_count();   // немесе A::inc_count();
}
```

Статикалық әдістер тұрақтылық (**const**) немесе виртуалды **(virtual**) бола алмайды.

Достас функциялар және кластар

Кейде кластың жасырын өрістеріне тікелей қол жеткізу мүмкіндігі болғаны, яғни класс интерфейсін кеңейткен жөн. Бұл үшін достас функциялар және достас кластар қызмет етеді.

Достас функциялар

Достас функциялар кластың жасырын өрістеріне қол жеткізу үшін қолданылады және олар әдістердің балама түрі (альтернативасы) болып саналады. Әдіс, көбінесе объектінің қасиеттерін жүзеге асыру үшін қолданылады, ал достас функциялар түрінде класс қасиеттерін көрсетпейтін, бірақ оның интерфейсіне концептуалды түрде енетін және оның жасырын өрістеріне қол жеткізуді қажет ететін әрекеттер қалыптастырылады, мысалы объектілерді экранға шығарудың қайта анықталған операциялары (304 б. қараңыз).

Төменде достас функцияларды сипаттау ережелері мен олардың ерекшеліктері көрсетілген.

- Достас функция объектілеріне қол жеткізу қажет болатын класс ішінде **friend** түйінді сөзі арқылы жарияланады. Параметр ретінде оған объект немесе класс объектісіне сілтеме берілуі керек, өйткені оған **this** нұсқауышы берілмейді.
- Достас функция әдеттегі функция немесе бұрынырақ анықталған басқа кластың әдісі болуы мүмкін. Оған қол жеткізу спецификаторла-

ры әсер етпейді, оның кластағы жариялану орны да маңызды болып саналмайды.

- Бір функция бірден бірнеше кластарға достас болуы мүмкін.

Мысал ретінде төменде **monstr** класына достас екі функцияның сипаттамалары берілген. **kill** функциясы **hero** класының әдісі болып табылады, ал **steal_ammo** функциясы ешбір класқа жатпайды. Екі функцияға да параметр ретінде **monstr** класының объектісіне сілтеме беріледі.

```
class monstr;      // Кластың алдын ала анықталуы
class hero{
   public:
      void kill(monstr &);
   ...
};
class monstr{
   ...
   friend int steal_ammo(monstr &);
   friend void hero::kill(monstr &);
   // hero класы ертерек анықталуы керек
};
int steal_ammo(monstr &M) {return --M.ammo;}
void hero::kill(monstr &M) {
    M.health = 0; M.ammo = 0;}
```

Достас функцияларды мүмкіндігінше аз қолдануға тырысу керек, өйткені олар инкапсуляция қағидасына қайшы келеді (бұзады), осылайша программаны түзетіп жөндеуді және өзгертуді қиындатады.

Достас класс

Егер кез келген бір кластың барлық әдістері басқа кластың жасырын өрістерін пайдалануы қажет болса, онда ол класс түгелдей **friend** түйінді сөзінің көмегімен достас класс ретінде жарияланады. Төменде келтірілген мысалда **mistress** класы **hero** класына достас болып жарияланған:

```
class hero{
   ...
   friend class mistress;
}
class mistress{
   ...
   void f1();
   void f2();
}
```

Мысалдағы **f1** және **f2** функциялары **hero** класына достас болып табылады (**friend** түйінді сөзі жазылмағанына қарамастан) және оның барлық өрістеріне қол жеткізе алады.

Мұндағы **friend** түрінде жариялану қол жеткізу спецификаторы болып табылмайды және мұраланбайды.

ЕСКЕРТУ

Қандай функциялар мен кластардың достас немесе достас емес екенін класс өзі анықтайтынына назар аударыңыз.

Деструкторлар

Деструктор – бұл объектіге жұмсалып тұрған жады аймағын босату үшін қолданылатын әдістің ерекше түрі. Объект көріну аймағынан шыққан кезде деструктор автоматты түрде шақырылады:

- жергілікті объектілер үшін – олар жарияланған блоктан шығу кезінде;
- ауқымды объектілер үшін – **main** функциясынан шығу процедурасының бір бөлігі ретінде;
- нұсқауыштар арқылы берілген объектілер үшін деструктор **delete** операциясын колдану кезінде тікелей емес түрде шақырылады.

НАЗАР АУДАРЫҢЫЗ

Объектіге нұсқауыштың әрекет ету аймағынан шығу кезінде деструкторды автоматты түрде шақыру жүзеге асырылмайды.

Деструктордың атауы тильда белгісінен (~) басталады, одан кейін класс аты жазылады. Деструктордың қасиеттері:

- аргументтері және қайтарылатын мәні болмайды;
- **const** немесе **static** ретінде жариялануы мүмкін емес;
- мұраланбайды;
- виртуалды болуы мүмкін («Виртуалды әдістер» бөлімін қараңыз 221 б.).

Егер деструктор айқын түрде анықталмаған болса, компилятор автоматты түрде бос деструктор құрады.

Объект құрамында динамикалық түрде бөлінетін жадыға нұсқауыштар бар болса, класта деструкторды тікелей айқын түрде сипаттау қажет, әйтпесе объектіні жою кезінде оның өріс-нұсқауыштары сілтеме жасаған жады бос аймақ ретінде белгіленбейді. Деструкторға нұсқауыш анықтау мүмкін емес.

Бұрыннан қарастырылып келе жатқан мысал үшін деструктор (204 б.) төмендегідей түрде болуы тиіс:

```
monstr::~monstr() {delete [] name;}
```

Деструктордың толық нақтыланған атын көрсету арқылы оны тікелей түрде шақыруға болады, мысалы:

```
monstr *m; ...
m -> ~monstr();
```

Бұл асыра жүктелген **new** операциясының көмегімен нақты көрсетілген жады адресі бөлінген объектілер үшін керек болуы мүмкін. Объект деструкторын ол қажет болмаған жағдайларда айқын түрде тікелей шақыру керек емес.

Операциялардың асыра жүктелуі

C++ тілі көптеген операциялардың әрекеттерін нақты көрсетілген кластың объектілерімен бірге қолдану кезінде, олар бекітілген функцияларын орындайтындай етіп қайта анықтауға мүмкіндік береді. Бұл өзіндік мәліметтер типтерін дәл стандартты мәліметтер типтері сияқты қолдануға жол ашады. Өзіндік операциялардың белгілерін енгізуге болмайды. C++ тілінде төменде көрсетілген операциялардан басқа осы тілде бар барлық операцияларды асыра жүктеуге болады:

```
.     .*     ?:     ::     #     ##     sizeof
```

Операцияларды асыра жүктеу арнайы түрдегі әдістер көмегімен (функция-операциялар) жүзеге асырылады және келесі ережелерге бағынады:

- операцияларды асыра жүктеу кезінде аргументтер саны, операция басымдықтары және стандартты мәлімет типтерінде қолданылатын ассоциация ережелері (оңнан солға немесе солдан оңға қарай) сақталады;
- стандартты мәлімет типтері үшін операцияларды қайта анықтауға болмайды;
- функция-операциялардың келісім бойынша берілген аргументтері болуы мүмкін емес;
- функция-операциялар мұраланады (= операциясынан басқасы);
- функция-операциялардың **static** ретінде анықталуы мүмкін емес.

Функция-операцияны үш тәсілмен анықтауға болады: ол класс әдісі, немесе кластың достас функциясы, немесе қарапайым функция болуы керек. Соңғы екі жағдайда функция класс типті, класқа нұсқауыш немесе сілтеме типті кем дегенде бір аргументті қабылдауы керек[1].

Функция-операция құрамында **operator** түйінді сөзі, одан кейін қайта анықталатын операция белгісі орналасады:

```
тип operator операция ( параметрлер тізімі ) {
   функция тұлғасы
}
```

[1]Ерекше жағдай: алғашқы параметрі стандартты типте болатын функция-операция класс әдісі ретінде анықтала алмайды.

Унарлы операциялардың асыра жүктелуі

Класс ішінде анықталатын унарлы функция-операция статикалық емес әдістің көмегімен параметрсіз түрде берілуі керек, мұнда операнд ретінде оны шақырған объект қабылданады, мысалы:

```
class monstr{
   ...
   monstr & operator ++() {++health; return *this;}
}
monstr Vasia;
cout << (++Vasia).get_health();
```

Егер функция *кластан тыс* анықталатын болса, оның құрамында класс типіндегі бір параметр болуы тиіс:

```
class monstr{
   ...
   friend monstr & operator ++( monstr &M);
};
monstr& operator ++(monstr &M) {++M.health; return M;}
```

Егер класс ішіндегі функцияны достас ретінде сипаттамайтын болсақ, өзгертілетін өрістердің қол жетімділігін ескеру керек. Берілген жағдайда **health өрісіне** сырттан қол жеткізуге болмайды, өйткені ол **private** специ-фикаторы арқылы сипатталған, сондықтан оны өзгерту үшін соған сәйкес әдіс қолдану қажет болады. monstr класының сипаттамасына **change_health** әдісін енгіземіз, ол **health** өрісінің мәнін өзгертуге мүмкіндік береді:

```
void change_health(int he){ health = he;}
```

Осылай болса, инкремент операциясын кластан тыс сипатталған қарапайым функция көмегімен асыра жүктеуге болады:

```
monstr& operator ++(monstr &M){
   int h = M.get_health(); h++; M.change_health(h);
   return M; }
```

Постфикстік инкремент және декремент операцияларының алғашқы параметрінің типі **int** болуы тиіс. Ол бұларды префикстік формадан ажыратып алу үшін ғана қолданылады:

```
class monstr{
   ...
   monstr operator ++(int){
      monstr M(*this); health++;
      return M; }
};
monstr Vasia;
cout << (Vasia++).get_health();
```

Бинарлы операциялардың асыра жүктелуі

Класс ішінде анықталатын бинарлы функция-операция параметрлі статикалық емес әдістің көмегімен сипатталуы тиіс, мұнда оны шақырған объект алғашқы операнд болып саналады:

```
class monstr{
   ...
   bool operator >(const monstr &M){
      if( health > M.health) return true;
      return false;
   }
};
```

Егер функция *кластан тыс* анықталса, оның құрамында класс типіндегі екі параметр болу керек:

```
bool operator >(const monstr &M1, const monstr &M2){
   if( M1.get_health() > M2.get_health()) return true;
   return false;
}
```

Меншіктеу операциясының асыра жүктелуі

Меншіктеу операциясы кез келген класта келісім бойынша жекелеген элементтер бойынша көшіру ретінде анықталған. Бұрыннын бар бір объектіге басқа объектінің мәнін меншіктеу кезінде әрқашанда осы операция шақырылады. Егер класта компьютер жады динамикалық түрде бөлінетін өрістер бар болса, онда өзіндік меншіктеу операциясын анықтау керек. Меншіктеу семантикасын сақтау үшін объектіге арнайы шақырылған операция-функция осы объектіге сілтеме қайтаруы керек және параметр ретінде жалғыз аргументті – меншіктелетін объектіге сілтемені қабылдауы тиіс.

```
const monstr& operator = (const monstr &M){
   // Проверка на самоприсваивание:
   if (&M == this) return *this;
   if (name) delete [] name;
   if (M.name){
      name = new char [strlen(M.name) + 1];
      strcpy(name, M.name);}
   else name = 0;
   health = M.health; ammo = M.ammo; skin = M.skin;
   return *this;
}
```

Объектіге нұсқауыш функциясынан қайта оралу келесі меншіктеу операциялары тізбегін жазуға мүмкін етеді:

```
monstr A(10), B, C;
C = B = A;
```

Меншіктеу операциясын тек класс әдісі ретінде ғана анықтауға болады. Ол мұраланбайды.

new және delete операцияларының асыра жүктелуі

Жадыны басқарудың альтернативті нұсқаларын қамтамасыз ету мақсатында объект пен объетілер жиымына сәйкесінше динамикалық жады бөлу үшін **new** және **new[]**, сонымен қатар оны босату үшін **delete** және **delete []** операцияларының өзіндік (меншікті) нұсқаларын анықтауға болады.

Мұндай функция-операциялар келесідей ережелерге сәйкес келуі керек:

□ оларға класс типіндегі параметрлерді берудің қажеті жоқ;

□ **new** және **new[]** функцияларына алғашқы параметр ретінде **size_t** типті объект өлшемі берілуі керек (**sizeof** операциясы қайтаратын бұл тип **<stddef.h>** тақырыптық файлында анықталады); шақыру кезінде ол функцияға айқын емес түрде беріледі;

□ **return** командасы басқа типтерге (көбінесе кластарға) нұсқауыштар қайтаратын болса да, бұл функциялар қайтарылатын мәннің **void*** типімен анықталуы тиіс;

□ **delete** операциясының қайтарылу типі **void** және алғашқы аргументінің типі **void*** болуы тиіс;

□ компьютер жадын бөлу және босату операциялары кластың статикалық элементтері болып табылады.

Асыра жүктелген операциялардың тәртіптері олардың келісім бойынша орындайтын әрекеттеріне сәйкес келуі тиіс**.** **new** операциясы үшін бұл оның дұрыс мән қайтаруы, нөлдік өлшемдегі жадыны бөлуге сұранысты дұрыс өңдеуі және мұндай сұраныстың орындалуы мүмкін болмаған жағдайда ерекше жағдайлар (аластамалар туралы «Аластамалы жағдайларды өңдеу» бөлімінде, 240 б. айтылады) туындауы керек екендігін білдіреді. **delete** операциясы үшін нөлдік нұсқауышты жою қауіпсіз болуы керек деген шартты орындаған жөн, сол себепті операция ішінде нұсқауыштың нөлге тең екендігін тексеру керек және тең болған жағдайда ешбір әрекеттің орындалмауы қадағалануы тиіс.

Кластың әрекет ету аймағында жадыны бөлу және босатудың стандартты операцияларын асыра жүктелген операциялармен қатар қолдануға болады (көріну аймағына қол жеткізу **::** операциясының көмегімен осы класс объектілері үшін және тікелей кез келген басқа объектілер үшін).

Жадыны бөлу операциясының асыра жүктелуі жадыны үнемдеу үшін, программаның орындалу жылдамдығын жоғарылату немесе мәліметтерді қандай да бір нақты көрсетілген аймақта орналастыру үшін қолданылады. Мысалы, қандай да бір объектіге нұсқауышы бар төмендегідей класс сипатталатын болсын:

```
class Obj { ... };
class pObj{
      ...
   private:
      Obj *p;
};
```

Стандартты **new** операциясының көмегімен **pObj** типті объектіге жады бөлу кезінде

```
pObj *p = new pObj;
```

байттардың нақты саны **sizeof(pObj)** шамасынан артық болады, өйткені **new** операциясы, әдетте, бөлінген аймақтың бас жағына оның өлшемін жазады (**delete** операциясы дұрыс жұмыс істеуі үшін).

Шағын объектілер үшін мұндай қосымша шығындардың көлемі айтарлықтай үлкен болуы мүмкін. Жадыны үнемдеу үшін **pObj** класы үшін өзіндік **new** операциясын жазуға болады, ол жадының үлкен блогын бөліп, содан кейін онда **Obj** объектісіне нұсқауыштарды орналастырады. Бұл үшін **pObj** объектісіне **headOfFree** статикалық өрісі енгізіледі, бұл өрісте кезекті объектіні орналастыру үшін блоктың алғашқы бос объектісіне нұсқауыш сақталады.

Қолданылмайтын ұяшықтар тізімге байланыстырылады. Байланыс өрісіне жады жұмсамау үшін біріктірме (**union**) қолданылады, оның көмегімен бір ғана ұяшық объектіге нұсқауышты орналастыру үшін де немесе келесі бос ұяшықпен байланыс орнату үшін де қолданылады:

```
class pObj{
   public:
      static void * operator new(size_t size);
      ...
   private:
      union{
         Obj *p;         // Объектіге нұсқауыш
         pObj *next;     // Келесі бос ұяшыққа нұсқауыш
      };
      static const int BLOCK_SIZE;// Блоктың өлшемі
      // Бос ұяшықтар тізімінің тақырыбы:
      static pQbj *headOfFree;
};
void * pObj::operator new(size_t size){
   // new стандартты операциясының дұрыс көрсетілмеген
   // жады көлеміне сұранысын қайта бағыттау:
   if (size != sizeof(pObj)) return ::operator
                                              new(size);
   pObj *p = headOfFree; //Алғашқы бос ұяшыққа нұсқауыш
   // Бос ұяшықтар тізімінің нұсқауышын жылжыту:
   if (p) headOfFree = p -> next;
```

```
    // Егер жадыда бос орын болмаса, кезекті блокты бөлеміз:
    else {
       pObj *newblock = static_cast<pObj*>[1]
          (::operator new(BLOCK_SIZE * sizeof(pObj)));
       // Біріншіден басқа барлық ұяшықтар бос (ол
       // қолданылуда), оларды байланыстырамыз:
       for (int i = 1; i< BLOCK_SIZE - 1; ++i)
          newblock[i].next = &newblock[i + 1];
       newblock[BLOCK_SIZE - 1].next = 0;
       // Бос ұяшықтар тізімінің басын белгілейміз:
       headOfFree = &newblock[1];
       p = newblock;
    }
 return p; // Белгіленген жадыға нұсқауышты қайтарамыз
 }
```

Асыра жүктелген **new** операциясы мұраланады, сондықтан ол туындайтын объектілер үшін шақырылады. Егер олардың көлемдері негізгі объект көлеміне сәйкес келмесе (осылай болуы әбден ықтимал), онда қиындықтар туындауы мүмкін. Олардың алдын алу үшін операцияның басында көлемдердің сәйкес келуі тексеріледі. Егер объектінің көлемі **new** операциясы асыра жүктелген көлемге тура келмесе, жадыны бөлуге сұраныс стандартты **new** операциясына беріледі.

pObj класын қолданатын программада оның статикалық өрістерін инициалдау әрекеті орындалуы керек (статикалық өрістер 201 б. қарастырылды):

```
pObj *pObj::headOfFree; // Келісім бойынша 0 көрсетіледі
const int pObj::BLOCK_SIZE = 1024;
```

Бұл мысалдан көрініп тұрғандай, жадыны үнемдеумен қатар, жоғары жылдамдыққа қол жеткізіледі, өйткені көбінесе жадыны бөлу үшін бірнеше қарапайым операторлар қолданылады.

Егер **new** операциясы асыра жүктелген болса, **delete** операциясы үшін де осы әрекет орындалуы керек (мысалы, біздің жағдайымызда стандартты **delete** операциясы объектінің басында оның көлемдері туралы дұрыс ақпарат таба алмайды, бұл өз кезегінде, программа іс-әрекеттерінің анықталмауына алып келеді).

Қарастырылған мысалда **delete** операциясы босатылған жады ұяшығын бос ұяшықтар тізіміне қосуы керек:

```
void pObj::operator delete(void * ObjToDie, size_t size){
   if (ObjToDie == 0) return;
   if (size != sizeof(pObj)){
      ::operator delete(ObjToDie); return;
```

[1]Бұл жерде **static_cast** операциясының көмегімен типті тікелей түрде түрлендіру қолданылған. Ол туралы «**static_cast** операциясы» бөлімінде, 255 б. айтылады.

```
    }
    pObj *p = static_cast<pObj*>(ObjToDie);
    p->next = headOfFree;
    headOfFree = p;
}
```

delete операциясында **new** операциясындағы сияқты объектілер көлемдерінің сәйкестігін тексеру орындалған.

Типті келтіру операциясының асыра жүктелуі

Класс объектісін басқа типке түрлендіруді орындайтын функция-операцияларды анықтауға болады. Форматы:

```
operator жаңа_типтің_атауы ();
```

Қайтарылатын мән типін және параметрді көрсетудің қажеті жоқ. Типтерді түрлендірудің виртуалды функцияларын анықтауға болады.

Мысал:

```
monstr::operator int(){return health;}
...
monstr Vasia; cout << int(Vasia);
```

Функцияны шақыру операциясының асыра жүктелуі

Функцияны шақыру операциясы анықталған класс *функционалды* деп аталады. Мұндай кластан басқа өрістер мен әдістердің болуы талап етілмейді:

```
class if_greater{
   public:
      int operator () (int a, int b) const {
         return a > b;
      }
};
```

Мұндай класты қолданудың синтаксисі ерекше болып келеді. Мысал қарастырайық:

```
if_greater x; cout << x(1, 5) << endl;  // Нәтижесі - 0
cout << if_greater()(5, 1) << endl;     // Нәтижесі – 1
```

Мысалдағы **if_greater** класында екі параметрі бар функцияны шақыру операциясы анықталғандықтан, **x(1,5)** өрнегін жазуға болады (дәл осыны **x.operator () (1,5)** түрінде де жазуға болады). Мысалдан көріп отырғанымыздай, функционалды кластың объектісі функция сияқты қолданылған.

Шығару операторының екіншісінде **if_greater()** өрнегі **if_greater** класының келісім бойынша берілген конструкторын шақыру үшін қолданылады. Бұл өрнектің орындалу нәтижесі **if_greater** класының

объектісі болып табылады. Одан ары, осының алдындағы жағдайдағы сияқты, осы объект үшін дөңгелек жақша ішінде жазылған екі аргументі бар функция шақырылады.

Келесі **()** операциясын тек класс әдісі ретінде анықтауға болады. Аргументтерінің саны әртүрлі функцияларды шақырудың асыра жүктелген операцияларын анықтау мүмкіндігі бар. Функционалды объектілер C++ стандартты кітапханасында кеңінен қолданылады.

Индекстеу операциясының асыра жүктелуі

Әдетте, класс типі индекстеудің мағынысы болатын мәндер жиынын көрсететін кезде индекстеу операциясы **[]** асыра жүктеледі. Индекстеу операциясы жиынға тиісті элементке сілтеме қайтаруы керек. Мұны бүтін сандар жиымын сақтауға және онымен сенімді жұмыс істеуге арналған **Vect** класының мысалы арқылы көрсетейік:

```
#include <iostream.h>
#include <stdlib.h>
class Vect{
public:
   explicit Vect(int n = 10);
   Vect(const int a[], int n); //Жиыммен инициалдау
   ~Vect() { delete [] p; }
   int& operator [] (int i);
   void Print();
...
private:
   int* p;
   int size;
};
Vect::Vect(int n) : size(n){
   p = new int[size];
}
Vect::Vect(const int a[], int n) : size(n){
   p = new int[size];
   for (int i = 0; i < size; i++) p[i] = a[i];
}
// Индекстеу операциясының асыра жүктелуі:
int& Vect::operator [] (int i){
   if(i < 0 || i >= size){
      cout << "Қате индекс: (i = " << i << ")" << endl;
      cout << "Программаны аяқтау" << endl; exit(0);
   }
   return p[i];
}
void Vect::Print(){
```

```
    for (int i = 0; i < size; i++)
        cout << p[i] << " ";
        cout << endl;
}
int main(){
    int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    Vect a(arr, 10);
    a.Print();
    cout << a[5] << endl;
    cout << a[12] << endl;
    return 0;
}
```

Программа жұмысының нәтижесі:

```
1 2 3 4 5 6 7 8 9 10
6
Қате индекс: (i = 12)
Программаны аяқтау
```

Асыра жүктелген индекстеу операциясы бүтін аргументті қабылдайды да, оның мәні жиым диапазонының аралығында жататынын немесе жатпайтынын тексереді. Егер сол диапазонда жататын болса, онда стандартты индекстеу операция семантикасына сәйкес элемент адресі қайтарылады.

Берілген мысалда келісім бойынша параметрді қабылдаған конструктор тікелей емес түрде шақырылатын типті түрлендіру конструкторы (200 б. қараңыз) болмауы үшін **explicit** ретінде жарияланған. Мұндағы **explicit** түйінді сөзі бұл конструктордың тек тікелей айқын түрде шақырылатындығын көрсетеді.

[] операциясын тек кластың әдісі ретінде анықтауға болады.

Класс элементтеріне нұсқауыштар

Класс элементтерін нұсқауыштар көмегімен де пайдалануға болады. Мұны жүзеге асыру үшін **.*** және **->*** операциялары анықталған. Кластың өрістері мен әдістеріне арналған нұсқауыштар әртүрлі болып анықталады.

*Класс әдісіне нұсқауышты*ң форматы:

```
қайт_тип(класс_атауы::*нұсқауыш_атауы)(параметрлер);
```

Мысалы, **monstr** класының

```
int get_health() {return health;}
int get_ammo() {return ammo;}
```

әдістеріне (сонымен қатар, кластың сигнатурасы дәл осындай болып келетін басқа да әдістеріне) нұсқауыштың сипаттамасы келесідей түрде болады:

```
int (monstr:: *pget)();
```

Мұндай нұсқауышты функцияның параметрі ретінде беруге болады. Бұл функцияға әдістің атауын беру мүмкіндігін жүзеге асырады:

```
void fun(int (monstr:: *pget)()){
   (*this.*pget)();    // Функцияны .* операциясы
                       // арқылы шақыру
   (this->*pget)();    // Функцияны ->* операциясы
                       // арқылы шақыру
}
```

Нұсқауышты нақты көрсетілген әдіске адресті алу операциясының көмегімен баптауға болады:

```
// Нұсқауышқа мән меншіктеу:
pget = & monstr::get_health;
monstr Vasia, *p;
p = new monstr;
// Функцияны .* операциясы арқылы шақыру:
int Vasin_health = (Vasia.*pget)();
// Функцияны ->* операциясы арқылы шақыру:
int p_health = (p->*pget)();
```

Төменде нұсқауыштарды класс әдістеріне қолдану ережелері көрсетілген:

- Әдіс нұсқауышына тек сәйкес тақырыбы бар әдістердің адрестерін меншіктеуге болады.
- Кластың статикалық әдісіне нұсқауышты анықтауға болмайды.
- Әдіске нұсқауышты класс элементі болып табылмайтын қарапайым функция нұсқауышына түрлендіруге болмайды.

Қарапайым функцияларға нұсқауыштар сияқты әдістерге нұсқауыштар атауы белгісіз әдістерді шақыру керек болған кезде қолданылады. Бірақ оның айнымалыға немесе қарапайым функцияға нұсқауыштан айырмашылығы – әдістерге нұсқауыштар жадының белгілі бір адресіне сілтеме жасамайды. Ол жиымдағы индекске көбірек ұқсайды, өйткені ығыстыру шамасын тағайындайды. Компьютер жадындағы нақты көрсетілген адрес әдіске нұсқауышты нақты көрсетілген объектіге нұсқауышпен үйлестіру арқылы алынады.

ЕСКЕРТУ

Нұсқауыштар арқылы шақырылатын әдістер виртуалды болуы мүмкін. Мұндайда нұсқауыш қолданылған объект типіне сәйкес келетін әдіс шақырылады.

Класс өрісіне нұсқауыштың форматы:

```
мәліметтер_типі(класс_атауы::*нұсқауыш_атауы);
```

Нұсқауыш анықтамасына оның инициалдануын мынадай түрде енгізуге болады:

```
&класс_атауы::*өріс_атауы; // Өріс  public болуы тиіс
```

Егер **health** өрісі **public** ретінде жарияланған болса, онда оған нұсқауыштың анықтамасы келесідей түрде болатын еді:

```
int (monstr::*phealth) = &monstr::health;
cout << Vasia.*phealth;   // .* операциясын қолдану
cout << p->*phealth;      // ->* операциясын қолдану
```

Класс өрістеріне нұсқауыштар қарапайым нұсқауыштар болмайтынына назар аударыңыз – оларға мәндерді меншіктеген кезде олар нақты жады адресіне сілтеме жасамайды, өйткені жады кластар үшін емес, класс объектілері үшін бөлінеді.

Класс құрамы бойынша ұсыныстар

Кластар құру, олармен жұмыс істеу мысалдарына практикумның [12] бірінші семинары арналған.

Көбінесе, қолданушы анықтаған тип ретінде класс құрамында жасырын (**private**) өрістер және келесі функциялар қамтылуы керек:

- класс объектілерінің калай инициалданатынын анықтайтын *конструкторлар*;
- *кластың қасиеттерін жүзеге асыратын әдістер* жиынтығы (мұндайда кластың жасырын өрістерінің мәндерін қайтаратын әдістер **const** модификаторы арқылы сипатталады, бұл модификатор өрістер мәндерінің өзгертілмейтіндігін көрсетеді);
- объектілерді көшіруге, меншіктеуге, салыстыруға және олармен кластың қасиеттеріне байланысты басқа да әрекеттерді орындауға мүмкіндік беретін *операциялар* жиынтығы;
- ерекше жағдайларды генерациялау арқылы қателер туралы хабарламалар үшін пайдаланылатын *аластамалар класы* (ол туралы 240 б. айтылатын болады).

Егер класпен немесе бірнеше класпен интерфейс арқылы жұмыс істейтін (яғни жасырын өрістерге енудің қажеті жоқ) функциялар бар болса, онда интерфейстерді асыра жүктемеу үшін осы функцияларды кластан тыс сипаттауға болады, ал логикалық байланысты қамтамасыз ету үшін оларды осы кластармен ортақ атаулар кеңістігіне орналастыру керек, мысалы:

```
namespace Staff{
   class monstr { /* ...*/ };
   class hero { /*...*/ };
   void interact(hero, monstr);
   ...
}
```

5-ТАРАУ

Мұралау

Кластарды мұралау механизмі туынды кластар негіздік немесе базалық кластардың элементтерін қолданып, оларды толықтыру немесе олардың қасиеттерін өзгерту мүмкіндігіне ие болатындай иерархияларды құруға мүмкіндік береді. Бір-бірімен ешбір байланысы жоқ кластардың саны көп болған жағдайда оларды басқару мүмкін болмайды. Мұралау бұл қиындықты кластарды реттеу немесе деңгейлеу жолымен, яғни бірнеше класс үшін ортақ қасиеттерді бір класта біріктіру және осы класты базалық класс ретінде қолдану арқылы шешуге мүмкіндік береді.

Иерархия басына жақын орналасқан кластар өзінен төмен орналасқан барлық кластарға барынша ортақ сипаттамаларды өздеріне жинақтайды. Иерархия бойымен төмендеген сайын кластардың нақты сипаттамалары арта түседі. Көптік мұралау бір кластың екі немесе одан да артық басқа базалық класс қасиеттеріне ие болуына мүмкіндік береді.

Қол жеткізу кілттері

Класты сипаттау кезінде оның тақырыбында оған базалық болып келетін барлық кластар тізіліп көрсетіледі. Осы кластардың элементтерін пайдалану мүмкіндігі **private**, **protected** және **public** сияқты қол жеткізу кілттерінің көмегімен реттеліп отырады:

```
class атауы : [private | protected | public] базалық_класс
   { класс тұлғасы };
```

Егер базалық класс бірнешеу болса, олар үтір арқылы бөлініп тізбектеле жазылады. Қол жеткізу кілттері әрбір класс алдында тұруы мүмкін, мысалы:

```
class    A { ... };
class    B { ... };
class    C { ... };
class    D: A, protected B, public C { ... };
```

Келісім бойынша кластар үшін **private** қол жеткізу кілті, ал құрылымдар үшін **public** кілті қолданылады.

Осыған дейін біз тек класс элементтеріне қолданылатын **private** және **public** *қол жеткізу спецификаторларын* қарастырыдық. Кластың кез келген элементіне **protected** спецификаторы да қолданылуы мүмкін, ол иерархияға кірмейтін жеке кластар үшін **private** спецификаторымен бірдей болып табылады. Олардың арасында айырмашылық мұралау кезінде көрінеді, оны келесі кестеден көруге болады:

Қол жеткізу кілті	Базалық кластағы спецификатор	Туынды кластан қол жеткізу
private	**private**	жоқ
	protected	**private**
	public	**private**
protected	**private**	жоқ
	protected	**protected**
	public	**protected**
public	**private**	жоқ
	protected	**protected**
	public	**public**

Кестеден көрініп тұрғандай, кілттен тәуелсіз түрде базалық кластың **private** элементтеріне туынды кластан қол жеткізуге болмайды. Оларды тек базалық класс әдістері арқылы ғана пайдалануға болады.

protected элементтері **private** кілті арқылы мұралау кезінде туынды класта **private** түрінде болады, ал басқа жағдайларда оларға қол жеткізу құқықтары өзгермейді.

Мұралау кезінде **public** элементтеріне қол жеткізу кілтке сәйкес орындалады.

Егер базалық класс **private** кілті арқылы мұраланса, онда туынды кластың **public** секциясында көріну аймағына қол жеткізу операциясының көмегімен оның кейбір элементтерін таңдамалы түрде туынды кластан қол жеткізугу болатындай етуге болады:

```
class Base{
   ...
   public: void f();
};
class Derived : private Base{
   ...
   public: Base::void f();
};
```

Қарапайым мұралау

Туынды кластың бір ғана ата-тегі (базасы) болса, мұндай мұралау түрі қарапайым мұралау деп аталады. Кластың әртүрлі әдістері үшін әртүрлі мұралау ережелері бар – мысалы, туынды класта конструкторлар және меншіктеу операциясы мұраланбайды, ал деструкторлар мұраланады. Кластардың мұралануын және мұндайда туындайтын қиындықтарды бір мысал арқылы қарастырайық.

monstr класынан туындайтын, кейбір кездерде пайдалы болып табылатын ойлау қабілеті қосылған **daemon** класын құрайық. Базалық класс осыдан бұрынғы бөлімдерде енгізілген кейбір әдістермен бірге сипатталған:

```
enum color {red, green, blue};
// ----------- monstr класы -----------
class monstr{
      // ------ Кластың жасырын өрістері:
      int health, ammo;
      color skin;
      char *name;
   public:
      // ------ Конструкторлар:
      monstr(int he = 100, int am = 10);
      monstr(color sk);
      monstr(char * nam);
      monstr(monstr &M);
      // ------      Деструктор:
      ~monstr() {delete [] name;}
      // ------ Операциялар:
      monstr& operator ++(){
         ++health; return *this;
      }
      monstr operator ++(int){
         monstr M(*this); health++; return M;
      }
      operator int(){
         return health;
      }
      bool operator >(monstr &M){
         if( health > M.get_health()) return true;
         return false;
      }
      const monstr& operator = (monstr &M){
         if (&M == this) return *this;
         if (name) delete [] name;
         if (M.name){
            name = new char [strlen(M.name) + 1];
            strcpy(name, M.name);}
         else name = 0;
         health = M.health; ammo = M.ammo; skin = M.skin;
         return *this;
      }
      // ------ Өрістерге қол жеткізу әдістері:
      int get_health() const {return health;}
      int get_ammo() const {return ammo;}
```

```
      // ------ Өрістердің мәндерін өзгертетін әдістер:
      void change_health(int he){ health = he;}
      // ------ Басқа әдістер:
      void draw(int x, int y, int scale, int position);
};
// ------- monstr класының жүзеге асырылуы -------
monstr::monstr(int he, int am):
   health (he), ammo (am), skin (red), name (0){}
monstr::monstr(monstr &M){
   if (M.name){
      name = new char [strlen(M.name) + 1];
      strcpy(name, M.name);}
   else name = 0;
   health = M.health; ammo = M.ammo; skin = M.skin;
}
monstr::monstr(color sk){
   switch (sk){
      case red:   health = 100; ammo = 10; skin = red;
                  name = 0; break;
      case green: health = 100; ammo = 20; skin =green;
                  name = 0; break;
      case blue:  health = 100; ammo = 40; skin = blue;
                  name = 0; break;
   }
}
monstr::monstr(char * nam){
   name = new char [strlen(nam)+1];
   strcpy(name, nam);
   health = 100; ammo = 10; skin = red;
}
void monstr::draw(int x, int y, int scale, int position) {
/* ... monstr суреттелуі */ }
// ------------     daemon класы ------------
class daemon : public monstr{
      int brain;
   public:
      // ------     Конструкторлар:
      daemon(int br = 10){brain = br;};
      daemon(color sk) : monstr (sk) {brain = 10;}
      daemon(char * nam) : monstr (nam) {brain = 10;}
      daemon(daemon &M) : monstr (M) {brain = M.brain;}
      // ------     Операциялар:
      const daemon& operator = (daemon &M){
         if (&M == this) return *this;
         brain = M.brain;
         monstr::operator = (M);
```

```
        return *this;
    }
    // ------ Өрістердің мәндерін өзгертетін әдістер:
    void think();
    // ------     Басқа әдістер:
    void draw(int x, int y, int scale, int position);
};
// -------- daemon класының жүзеге асырылуы: -------
void daemon::think(){ /* ... */ }
void daemon::draw(int x, int y, int scale, int position) {
   /* ... daemon суреттелуі */ }
```

daemon класына **brain** өрісі және **think** әдісі енгізілген, өзіндік конструкторлар мен меншіктеу операциясы анықталған, сонымен қатар **draw** сурет салу әдісі қайта анықталға. **monstr** класының барлық өрістері, операциялары (меншіктеуден басқа) және **get_health**, **get_ammo** және **change_health** әдістері **daemon** класына мұраланады, ал деструктор келісім бойынша қалыптастырылады.

Әртүрлі әдістердің мұралану ережелерін қарастырайық.

Конструкторлар мұраланбайды, сондықтан туынды кластың өзіндік конструкторлары болуы керек. Конструкторларды шақыру реттілігі төменде көрсетілген ережелермен анықталады.

□ Егер туынды класс конструкторында базалық класс конструкторының тікелей шақырылуы кездеспесе, онда автоматты түрде базалық кластың келісім бойынша берілген конструкторы (яғни, параметрлерсіз шақыруға болатын конструктор) шақырылады. Осы ереже **daemon** класы конструкторларының біріншісінде қолданылған.

□ Бірнеше деңгейлерден тұратын иерархия үшін базалық класс конструкторлары ең жоғарғы деңгейден бастап шақырылады. Осыдан кейін объектілер болып табылатын класс элементтерінің конструкторлары класта жариялану тәртібіне сәйкес орындалады да, содан соң барып класс конструкторы орындалады.

□ Бірнеше базалық класс болған жағдайда, олардың конструкторлары жариялану тәртібіне сәйкес шақырылады.

НАЗАР АУДАРЫҢЫЗ

Егер базалық класс конструкторында параметрлерді көрсету талап етілсе, ол туынды класс конструкторындағы инициализация тізімінде нақты түрде шақырылуы керек (бұл соңғы үш конструкторда көрсетілген).

Меншіктеу операциясы да мұраланбайды, сондықтан оны да **daemon** класында нақты түрде анықтап алу керек. Функция-операцияның жазылуына назар аударыңыз: оның тұлғасында базалық кластан меншіктеу функция-операциясын нақты түрде шақыру қолданылған. Шақыру синтаксисін жақсы

елестету үшін **operator** түйінді сөзін операция белгісімен бірге функция-операцияның атауы ретінде түсінуге болады.

Базалық класс функциясын шақыру әрекеті базалық класс функцияларынан код үзінділерін туынды класс функцияларына көшіруге қарағанда басымырақ болып саналады. Код көлемін азайтумен қатар, осы арқылы программаны өзгерту жеңілдігіне қол жеткізуге болады: өзгерістерді программаның бір ғана нүктесіне енгізу қажет, бұл мүмкін болатын кателіктер санын азайтады.

Төменде *деструкторлардың* мұралану ережелері көрсетілген:

□ *Деструкторлар мұраланбайды* және программалаушы туынды класта деструкторды сипаттамаған болса, онда ол келісім бойынша құрылады және барлық базалық кластардың деструкторларын шақырады.

□ Конструкторларға қарағанда, туынды класс деструкторын жазу кезінде оның құрамында *базалық класс деструкторларын нақты түрде шақыру талап етілмейді,* өйткені бұл автоматты түрде орындалады.

□ Бірнеше деңгейден тұратын кластар иерархиясы үшін деструкторлар *конструкторлардың шақырылу реттілігіне қатаң түрде қарама-қарсы тәртіппен шақырылады:* алдымен класс деструкторы, содан кейін класс элементтерінің деструкторлары, олардың соңынан базалық класс деструкторы шақырылады.

monstr класынан мұраланған өрістерді туынды класс функциялары пайдалана алмайды, өйткені олар базалық класта **private** түрінде анықталған. Егер **daemon** класында анықталған функциялардың осы өрістермен жұмыс істеуі қажет болса, онда оларды базалық класта **protected** түрінде анықтауға немесе оларды **monstr** класындағы функциялар көмегімен пайдалануға болады, немесе оларды осының алдындағы бөлімде көрсетілгендей, нақты түрде **daemon** класында қайта анықтау керек (218 б.).

Әдістердің мұралануын қарастыра отырып, **daemon** класында сипатталған **draw** әдісі **monstr** класында дәл осы атауға ие болған әдісті қайта анықтайтынына назар аударыңыз (себебі әртүрлі кейіпкерлерді бейнелеу әртүрлі орындалады). Осылайша, туынды класс базалық кластың іс-әрекеттерін тек толықтырып қана қоймай, сонымен қатар түзете де алады[1]. Базалық кластың қайта анықталған әдісіне туынды кластан қол жеткізу көрініу аймағына қол жеткізу операциясының (`::`) көмегімен нақтыланған оның атауы арқылы орындалады.

Виртуалды әдістер

Объектілермен жұмыс көбінесе нұсқауыштар арқылы орындалады. Базалық класс нұсқауышына кез келген туынды кластың объектісі адресінің мәнін меншіктеуге болады (ашық мұралау кезінде):

[1]Туынды класта тек виртуалды әдістерді қайта анықтаған жөн (келесі бөлімді қараңыз).

```
// Базалық класқа нұсқауыш сипатталады:
monstr *p;
// Нұсқауыш туынды класс объектісіне сілтеме жасайды:
p = new daemon;
```

Объектінің әдістерін шақыру нұсқауыш сілтеме жасайтын объектінің нақты типіне емес, нұсқауыштың типіне сәйкес орындалады, сондықтан оператор орындалғанда, мысалы,

```
p->draw(1, 1, 1, 1);
```

операторының орындалуы кезінде **daemon** класының емес, **monstr** класының әдісі шақырылады, өйткені әдістерге сілтемелер программаны жинақтау барысында рұқсат етіледі. Бұл үрдіс *ерте байланыстыру* деп аталады. **daemon** класының әдісін шақыру үшін нұсқауыш типін тікелей түрде түрлендіруді қолдануға болады:

```
((daemon * p))->draw(1, 1, 1, 1);
```

Бұл әрқашан мүмкін бола бермейді, өйткені әртүрлі уақыт мезетінде нұсқауыш иерархияның әртүрлі кластарының объектілеріне сілтеме жасауы мүмкін және программаның компиляциясы кезінде нақты класс белгісіз болуы мүмкін. Мысал ретінде параметрі базалық класс объектісіне нұсқауыш болып табылатын функцияны келтіруге болады. Программаның орындалуы кезінде оның орнына кез келген туынды класқа нұсқауыш берілуі мүмкін. Басқа мысал – иерархияның әртүрлі объектілеріне нұсқауыштардың байланысқан тізімі, бұл тізіммен бір сипатта жұмыс істеу керек.

Ерте байланыстырумен қатар, C++ тілінде *кейінгі байланыстыру* механизмі жүзеге асырылған, мұнда әдіске сілтемелер жасауға рұқсат беру әдісті шақырған объектінің нақты типіне байланысты программаның атқырылу кезеңінде орындалады. Бұл механизм виртуалды әдістер көмегімен жүзеге асырылған, ол келесі бөлімде қарастырылады.

Виртуалды әдісті анықтау үшін **virtual** спецификаторы қолданылады, мысалы:

```
virtual void drawdnt x, int y, int scale, int position);
```

Виртуалды әдістерді сипаттау және пайдалану ережелерін қарастырайық:

□ Егер базалық класта әдіс виртуалды ретінде анықталған болса, туынды класта *дәл осындай атаумен және параметрлер жиынтығымен* анықталған әдіс автоматты түрде виртуалды болады, ал *параметрлер жиынтығы әртүрлі* болса, әдіс қарапайым болады.

□ Виртуалды әдістер *мұраланады,* яғни оларды туынды класта қайта анықтау тек айырықша әрекеттерді тағайындау қажет болғанда орындалады. Қайта анықтау кезінде оларға қол жеткізу құқықтарын өзгертуге болмайды.

□ Егер виртуалды әдіс туынды класта қайта анықталған болса, осы класс объектілері көріну аймағына қол жеткізу операциясының көмегімен базалық класс әдісін пайдалану мүмкіндігіне ие бола алады.

□ Виртуалды әдісті **static** модификаторы арқылы жариялауға болмайды, бірақ ол достас әдіс ретінде жариялануы мүмкін.

□ Егер класта виртуалды әдістің сипаттамасы енгізілетін болса, ол ең болмағанда, таза виртуалды әдіс ретінде анықталуы керек.

Таза виртуалды әдісте оның тұлғасының орнына **= 0** белгісі жазылады, мысалы:

```
virtual void f(int) = 0;
```

Таза виртуалды әдіс туынды класта қайта анықталуы тиіс (қайтадан таза виртуалды әдіс ретінде анықталуы мүмкін).

Егер **monstr** класындағы **draw** әдісін виртуалды әдіс ретінде анықтасақ, қай кластың әдісін шақыру қажеттігі туралы шешім нұсқауыш сілтеме жасайтын объектінің типіне байланысты қабылданатын болады:

```
monstr *r. *p;
r = new monstr; // monstr класының объектісі құрылады
p = new daemon; // daemon класының объектісі құрылады
r->draw(1, 1, 1, 1); // monstr::draw әдісі шақырылады
p->draw(1, 1, 1, 1); // daemon::draw әдісі шақырылады
p-> monstr::draw(1, 1, 1, 1);  // Виртуалды әдістер
                               // механизмін қолданбау
```

Егер **daemon** класының объектісі **draw** әдісін тікелей емес, жанама түрде (яғни, **monstr** класында анықталған басқа әдістен) шақыратын болса, онда **daemon** класының **draw** әдісі шақырылады.

Сонымен, программаның орындалу кезеңінде өзіне сілтеме жасауға рұқсат етілетін әдіс виртуалды деп аталады (ағылшынның әдемі **virtual** сөзінің бұл жердегі аудармасының мағынасы – «нақтылы түрде», яғни шақыру нақты орындалған кезде сілтеме рұқсат етіледі).

Кейінгі байланыстыру механизмі

Ең кем дегенде бір виртуалды әдісі бар әрбір класс (объект емес!) үшін компилятор *виртуалды әдістер кестесін* (**vtbl**) құрады, мұнда әрбір виртуалды әдіс үшін оның компьютер жадындағы адресі жазылады. Кестедегі әдістердің адрестері олардың кластардағы сипатталу реттілігіне сәйкес орналасады. Кез келген виртуалды әдістің адресі иерархия құрамындағы әрбір класс үшін **vtbl** кестесінде бірдей ығысуға ие болады.

Vtbl кестесіне әрбір объектінің *сілтеме жасайтын* жасырын қосымша **vptr** деп аталатын өрісі болады. Ол объект құру кезінде конструктор көмегімен толтырылады (бұл үшін компилятор конструктор тұлғасының бастапқы жағына соған сәйкес нұсқауларды қосады).

Компиляция кезеңінде виртуалды әдістерге жасалған сілтемелер **vptr** объектісі арқылы **vtbl** кестесін пайдалану әрекеттеріне алмастырылады, ал программаның орындалуы кезеңінде әдісті пайдалану кезінде оның адресі

кестеден таңдалып алынады. Осылайша, қарапайым әдістер мен функциялармен салыстырғанда, виртуалды әдісті шақыру кестеден әдіс адресін алудың қосымша кезеңі арқылы жүзеге асырылады. Бұл программаның орындалуын бірқатар баяулатады.

Динамикалық объектіге бөлінген жадының дұрыс босатылуын қамтамасыз ету үшін, *деструкторларды виртуалды етіп құру ұсынылыды,* өйткені мұндайда кез келген уақытта объектінің нақты типіне сәйкес келетін деструктор таңдалып алынады. Деструктор **`delete`** операциясына **`size_t`** типіндегі объектінің өлшемін жібереді. Егер өшірілетін объект туынды болса және оның құрамында виртуалды деструктор анықталмаса, онда жіберілген объект өлшемі дұрыс болмауы мүмкін.

Әдісті виртуалды жасаудың нақты ережесі жоқ. Туынды класта қайта анықталу ықтималдығы бар әдістерді виртуалды ретінде жариялау туралы ұсыныс қана беруге болады. Барлық иерархия деңгейлерінде өзгеріссіз қалатын немесе туынды кластар қолданбайтын әдістерді виртуалды етудің қажеті жоқ. Басқа жағынан алғанда, иерархияны жобалау кезінде базалық кластардың қандай жолмен кеңейтілетінін алдын ала болжау әрқашанда мүмкін бола бермейді (әсіресе кластар кітапханаларын жобалағанда), ал әдісті виртуалды ретінде жариялау икемділік пен кеңейту мүмкіндігін қамтамасыз етеді.

Соңғы жағдайды түсіндіру үшін **`draw`** әдісін шақыру объектінің орнын ауыстыру әдісінен орындалады деп есептейік. Егер орын ауыстыру әдісінің мәтіні орны ауыстырылатын объект типіне байланысты болмаса (барлық объектілердің орын ауыстыру принциптері бірдей болғандықтан, ал суреттеу үшін нақты әдіс шақырылатындықтан), туынды кластарда бұл әдісті қайта анықтаудың қажеті жоқ және оны виртуалды әдіс ретінде сипаттамауға болады. Егер **`draw`** әдісі виртуалды болса, орын ауыстыру әдісі қайта компиляциялаусыз кез келген туынды класс объектілерімен (тіпті оны жазу кезінде белгісіз болған объектілермен де) жұмыс істей алады.

Виртуалды механизм тек объектілерге нұсқауыштарды немесе сілтемелерді қолдану кезінде ғана жұмыс істейді. Нұсқауыш немесе сілтеме арқылы анықталып, виртуалды әдістерді қамтитын объект полиморфты деп аталады. Мұндайдағы полиморфизмнің мәні – әдіске бір ғана хабарласу арқылы әрбір уақыт мезетінде нұсқауыш сілтеме жасайтын типке байланысты әртүрлі әрекеттер орындалады.

Абстрактылы кластар

Кем дегенде бір таза виртуалды әдісі бар класс абстрактылы деп аталады. Абстрактылы кластар туынды кластарда нақтылауды талап ететін жалпы ұғымдарды тағайындау үшін қажет. Абстрактылы класс басқа кластар үшін тек *базалық класс ретінде* қолданыла алады – абстрактылы класс объектілерін құруға болмайды, өйткені таза виртуалды әдісті тікелей немесе жанамалы түрде шақыру олардың орындалуы кезіндегі қателіктерге алып келеді.

Абстрактылы класты анықтау кезінде келесі жағдайларды ескеру керек:

□ абстрактылы класты нақты түрде типтерді түрлендіру кезінде параметр типін және функцияның қайтаратын мәнінің типін сипаттау үшін қолдануға болмайды;

□ егер инициалдау кезінде уақытша объект құрудың қажеті жоқ болса, абстрактылы класқа сілтемелерді және нұсқауыштарды жариялауға рұқсат берілген;

□ егер абстрактылы кластан туындаған жаңа класс барлық таза виртуалды функцияларды анықтамайтын болса, ол да абстрактылы класс болып табылады.

Сонымен, параметрі абстрактылы класқа нұсқауыш болып табылатын функция құруға болады. Программаның орындалу кезінде бұл параметрдің орнына кез келген туынды кластың объектісіне нұсқауышты беруге болады. Бұл бір иерархия көлемінде кез келген типтегі объектімен жұмыс істейтін *полиморфты функцияларды* құруға мүмкіндік береді.

Көптік мұралау

Көптік мұралау кластың бірнеше базалық класы бар екендігін білдіреді. Егер базалық кластарда атаулары бірдей элементтер бар болса, онда идентификаторлар қайшылығы туындауы мүмкін, оны көріну аймағына қол жеткізу операциясының көмегімен жоюға болады:

```
class monstr{
   public: int get_health();
   ...
};
class hero{
   public: int get_health();
   ...
};
class ostrich: public monstr, public hero{
   ...
};
int main(){
   ostrich A;
   cout << A.monstr::get_health();
   cout << A.hero::get_health();
}
```

Мысалдан көрініп тұрғандай, **get_health** әдісін шақыру үшін ол сипатталған класты нақты түрде көрсету керек. Класс әдісін шақырудың қарапайым **A.get_health()** конструкциясын қолдану қателікке алып келеді, өйткені компилятор базалық кластардың қайсы әдісін пайдалану керек екенін түсіне алмайды.

Егер базалық кластардың ортақ ата-тегі бар болса, онда осы базалық кластардан туындайтын класс ата-тегі өрістерінің екі данасын (экземплярын) мұралайды, бұлай болғаны жақсы емес. Мұндай жағдайдың алдын алу үшін ортақ ата-тегінен мұралау кезінде оны виртуалды класс ретінде анықтау керек:

```
class monstr{
   ...
};
class daemon: virtual public monstr{
   ...
};
class lady: virtual public monstr{
   ...
};
class baby: public daemon, public lady{
   ...
};
```

baby класында **monstr** класы өрістерінің бір ғана данасы (экземпляры) болады. Егер базалық класс виртуалды түрде де, қарапайым түрде де мұраланатын болса, онда туынды класта әрбір виртуалды емес нұсқа үшін жеке даналар және виртуалды нұсқа үшін тағы бір дана кездеседі.

Көптік мұралау туынды класты екі немесе одан көп базалық кластардың қасиеттерімен қамтамасыз ету үшін қолданылады. Көбінесе осы кластардың біреуі негізгі болып табылады, ал қалғандары кейбір қосымша қасиеттермен қамтамасыз етеді, сондықтан олар *араластыру кластары* деп аталады. Араластыру кластары мүмкіндігінше виртуалды болуы тиіс және параметрсіз конструкторлар көмегімен құрылуы керек, сонда ромб түріндегі мұралау (базалық кластардың ортақ ата-тегі болатын жағдай) кезінде туындайтын көптеген қиындықтарды жеңуге мүмкіндік береді.

Құрылымдар мен біріктірмелердің кластардан айырмашылығы

Құрылымдар (**struct**) мен біріктірмелер (**union**) кластардың жеке бір түрлері болып табылады.

Құрылымдардың кластардан айырмашылығы – құрылымдарда элементтерге қол жеткізу және мұралау кезінде базалық класс келісім бойынша **public** болып есептеледі. Құрылымдарды барлық элементтеріне қол жеткізуге болатын кластар үшін қолданған жөн.

Біріктірмелердің кластардан айырмашылықтары төменде көрсетілген:

□ біріктірмелерде оларға қол жеткізу тәсілі келісім бойынша **public** болып табылады, сонымен қатар мұнда қол жеткізу спецификаторларын тікелей түрде мүлде қолдануға болмайды;

□ біріктірме кластар иерархиясына қатыса алмайды;

□ біріктірме элементтері ретінде конструкторлары мен деструкторлары бар объектілерді алуға болмайды;

□ біріктірме құрамында конструкторлар және басқа да әдістер болуы мүмкін, бірақ онда статикалық әдістер болмайды;

□ жасырын, яғни анонимдік біріктірмеде әдістерді сипаттауға болмайды.

Мұралануға және де кластардың басқа да қатынастарына – ассоциацияға, агрегацияға және тәуелділікке практикумның [12] екінші семинары арналған.

6-ТАРАУ

Класс шаблондары

Осы кітаптың бірінші бөлімінде функциялардың шаблондары (88 б.) қарастырылды, олардың көмегімен типті параметр ретінде беру арқылы алгоритмді ол жұмыс істейтін нақты мәлімет типтерінен бөліп алуға болады. Кластардың шаблондары осыған ұқсас мүмкіндіктерді қамтамасыз ете отырып, параметрленген кластарды құруға мүмкіндік береді.

Параметрленген класс туыстас кластар жиынын құрады, оларды параметр ретінде берілетін кез келген мәліметтер типіне қолдануға болады. Шаблондар *контейнерлік кластарды* құруда кеңінен қолданылады. Контейнерлік деп қандай да бір жолмен ұйымдастырылған мәліметтерді сақтауға және солармен жұмыс жасауға арналған класс аталады. C++ тілінің стандартты кітапханасында әртүрлі мәліметтер құрылымдарын ұйымдастыруға арналған контейнерлік кластар жиыны бар (олар «Контейнерлік кластар» бөлімінде, 318 б. сипатталған).

Шаблондарды қолданудың артықшылығы – мәліметтермен жұмыс істеу алгоритмі анықталғаннан және жөндеп түзетілгеннен кейін, оны кез келген мәліметтер типтері үшін кодты қайта жазбай қолдануға болады.

Кластардың шаблондарын құру

Кластың шаблонын құруды мысал арқылы қарастырайық. «Сызықтық тізімдер» (124 б.) бөлімінде екібайланысты тізім сипатталып, онымен жұмыс істеу алгоритмдері көрсетілген болатын. Тізімдер мәліметтерді ұйымдастыру үшін жиі қолданылатын болғандықтан, тізімді класс түрінде сипаттаған ыңғайлы, ал әртүрлі типтегі мәліметтерді сақтау керек болғандықтан, бұл класс параметрленген болуы тиіс.

Алдымен «тізім» класының параметрленбеген нұсқасын қарастырайық.

Тізім нұсқауыштар көмегімен өзара байланысқан түйіндерден тұрады. Әрбір түйін тізім кілті болып табылатын бүтін санды сақтайды. Тізімнің бір түйінін бейнелейтін қосымша класты сипаттайық:

```
class Node{
   public:
      int d;          // Мәліметтер
      Node *next;     // Келесі түйінге нұсқауыш
      Node *prev;     // Алдыңғы түйінге нұсқауыш
      Node(int dat = 0){   // Конструктор
         d = dat; next = 0; prev =0;
      }
};
```

Бұл класс тізімді бейнелейтін кластың ішінде сипатталатын болғандықтан, ішкі кластан қол жеткізуді жеңілдету мақсатында өрістер қолжетімді (**public**) етіп құрылған. Бұл қол жеткізу функциясының көмегіне жүгінбеуге және өрістерді өзгертпеуге мүмкіндік береді. Тізім класын **List** деп атайық:

```
class List{
   class Node{
      ...
   };
   Node *pbeg, *pend; // Тізімнің басына және соңына
                      // нұсқауыштар
   public:
      List(){ pbeg = 0; pend = 0;} // Конструктор
      ~List();                     // Деструктор
      void add(int d);    // Түйінді тізім соңына қосу
      Node * find(int i);  // Түйінді кілті бойынша іздеу
      // d түйінін key кілті бар түйіннен кейін
      // кірістіру:
      Node * insert(int key, int d);
      bool remove(int key);   // Түйінді жою
      void print();            // Тізімді тура бағытта
                               // басып шығару
      void print_back();       // Тізімді кері бағытта
                               // басып шығару
};
```

Класс әдістерінің жүзеге асырылуын қарастырайық. **add** әдісі жаңа **Node** типті объектіге жады бөледі және тізім басына және соңына нұсқауыштарды жаңарта отырып, объектіні тізімге қосады:

```
void List::add(int d){
   Node *pv = new Node(d);      // Жаңа түйінге жады бөлу
   if (pbeg == 0) pbeg = pend = pv; // Тізімнің алғашқы
                                    // түйіні
   else{
      // Жаңа түйінді алдыңғысымен байланыстыру:
      pv->prev = pend;
      pend->next = pv;
      pend = pv;}      // Тізім соңына нұсқауышты жаңарту
}
```

Сұрыпталған тізімді алу үшін бұл әдісті «Сызықтық тізімдер» бөлімінде, 124 б. келтірілген **add_sort** сұрыпталған тізімді қалыптастыру функциясына ұқсас әдіске алмастыруға болады.

Жоғарыдағы мысалда көрсетілген **find** әдісі берілген кілті бойынша түйінді іздеуді орындайды да, іздеу сәтті өткен жағдайда нұсқауышты түйінге қайтарады, ал тізімде мұндай түйін болмаған жағдайда, 0 мәнін қайтарады:

```
Node * List::find( int d ){
   Node *pv = pbeg;
   while (pv){
      if(pv->d == d)break;
      pv = pv->next;
   }
   return pv;
}
```

insert әдісі тізім ішіне жаңа түйінді **key** кілті бар түйіннен кейін орналастырады және кірістірілген түйінге нұсқауышты қайтарады. Егер мұндай түйін тізімде жоқ болса, кірістіру орындалмайды да, 0 мәні қайтарылады:

```
Node * List::insert(int key, int d){
   if(Node *pkey = find(key)){  // key кілті бар
                                // түйінді іздеу
      // Жаңа түйінге жады бөлу және оны инициалдау
      Node *pv = new Node(d);
      // Жаңа түйіннің келесі түйінмен байланысын орнату
      pv->next = pkey->next;
      // Жаңа түйіннің өзінен бұрынғы түйінмен
      // байланысын орнату:
      pv->prev = pkey;
      // Алдыңғы түйіннің жаңа түйінмен
      // байланысын орнату:
      pkey->next = pv;
      // Келесі түйіннің жаңа түйінмен
      // байланысын орнату:
      if( pkey != pend) (pv->next)->prev = pv;
      // Егер түйін тізім соңына кірістірілсе,
      // тізім соңына нұсқауышты жаңарту:
      else pend = pv;      '
      return pv;
   }
   return 0;
}
```

remove әдісі тізімнен берілген кілті бар түйінді өшіреді және өшіру сәтті орындалған жағдайда t**rue** мәнін, мұндай кілті бар түйін тізімде табылмаса, **false** мәнін қайтарады:

```
bool List::remove(int key){
   if(Node *pkey = find(key)){
      if (pkey == pbeg){        // Тізім басынан өшіру
         pbeg = pbeg->next;
         pbeg->prev = 0; }
```

```
      else if (pkey == pend){   // Тізім соңынан өшіру
         pend = pend->prev;
         pend->next = 0;}
      else {                    // Тізім ортасынан өшіру
         (pkey->prev)->next = pkey->next;
         (pkey->next)->prev = pkey->prev;}
      delete pkey;
      return true;}
   return false;
}
```

Тізімді тура және кері бағыттарды басып шығару әдістері соларға сәйкес сілтемелер арқылы көше отырып, тізімді жеке элементтер бойынша қарап шығады:

```
void List::print(){
   Node *pv = pbeg;
   cout << endl << "list: ";
   while (pv){
      cout << pv->d << ' ';
      pv = pv->next;}
   cout << endl;
}
void List::print_back(){
   Node *pv = pend;
   cout << endl << " list back: ";
   while (pv){
      cout << pv->d << ' ';
      pv = pv->prev;}
   cout << endl;
}
```

Тізім деструкторы оның барлық элементтерінен жадыны босатады:

```
List::~List(){
   if (pbeg != 0){
      Node *pv = pbeg;
      while (pv){
         pv = pv->next;
         delete pbeg;
         pbeg = pv;}
   }
}
```

Төменде **List** класын қолданатын программа мысалы көрсетілген. Программа 125 б. келтірілген программаға ұқсас: ол 5 саннан тұратын тізімді қалыптастырады, оны экранға шығарады, санды тізімге қосады, тізімнен санды өшіреді және оны қайтадан экранға шығарады:

```
int main(){
   List L;
   for (int i = 1; i<6; i++) L.add(i);
   L.print();
   L.print_back();
   L.insert(2, 200);
   if (!L.remove(5)) cout << «not found»;
   L.print();
   L.print_back();
}
```

List класы бүтін сандарды сақтауға арналған. Онда кез келген типтегі мәліметтерді сақтау үшін бұл класты шаблон ретінде сипаттап, типті параметр ретінде беру керек.

Шаблонды сипаттау синтаксисі:

```
template <шаблон_параметрлерін_сипаттау> класс_анықтамасы;
```

Шаблонның параметрлері үтір арқылы бөліне отырып, тізбектеле жазылады. Параметрлер ретінде типтер, шаблондар және айнымалылар қолданылуы мүмкін.

Типтер стандартты немесе қолданушы анықтаған тип болуы мүмкін. Оларды сипаттау үшін **class** түйінді сөзі қолданылады. Шаблонның ішінде тип спецификациясын қолдануға болатын кез келген жерде тип параметрін пайдалануға болады, мысалы:

```
template <class Data> class List{
   class Node{
      public:
         Data d;
         Node *next;
         Node *prev;
         Node(Data dat = 0){d = dat; next = 0; prev = 0;}
   };
...
}
```

Data класын формалды параметр ретінде қарастыруға болады, компиляция кезінде оның орнына нақты көрсетілген мәліметтер типі қойылады.

Шаблонның кез келген параметрлері үшін келісім бойынша мәндер тағайындауға болады, мысалы:

```
template<class T> class myarray { /* ... */ };
...
template<class K, class V, template<class T> class C = myarray>
class Map{
   C<K> key;
```

```
   C<V> value;
...
};
```

Шаблон параметрінің әрекет ету аймағы оның сипатталу нүктесінен бастап шаблонның соңына дейінгі кеңістік болады, сондықтан параметрді одан кейін орналасқан параметрлерді сипаттау кезінде де қолдануға болады, мысалы:

```
template<class T, T* p, class U = T> class X { /*...*/};
```

Класс шаблонының әдістері автоматты түрде функция шаблондарына айналады. Егер әдіс шаблоннан тыс сипатталса, онда оның тақырыбында келесі элементтер болуы тиіс:

```
template <шаблон_параметрлерінің_сипаттамасы>
   қайт_тип класс_атауы <шаблон_параметрлері >::
   функция_атауы (функцияның параметрлер_тізімі)
```

Функцияның тақырыбындағы шаблон параметрлерінің сипаттамасы класс шаблонына сәйкес болуы керек, мұнда параметрлердің атаулары бірдей болмауы мүмкін. Шаблон әдістерінің сипатталу синтаксисін мысал арқылы қарастырған жеңіл болады:

```
template <class Data> void List<Data>::print()
   { /* функция тұлғасы */ }
```

Мұндағы **<class Data>** – шаблон параметрінің сипаттамасы, **void** – функцияның қайтаратын мәнінің типі, **List** – класс атауы, **<Data>** – шаблон параметрі, **print** – параметрлерсіз функцияның атауы.

Бірнеше параметр болған жағдайда **параметрлер_сипаттамасындағы** және **шаблон_параметрлеріндегі** олардың орналасу реттілігі бірдей болуы керек, мысалы:

```
template<class T1, class T2> struct A{
   void f1();
};
template<class T2, class T1> void A<T2, T1>::f1(){...}
```

Төменде шаблондарды сипаттау ережелері көрсетілген.

- Жергілікті кластар шаблондарды өздерінің элементтері ретінде кабылдай алмайды.
- Әдістердің шаблондары виртуалды бола алмайды.
- Класс шаблондарында статикалық элементтер, достас функциялар және кластар болуы мүмкін.
- Шаблондар басқа шаблондардан немесе қарапайым кластардан туынды болып, сонымен қатар шаблондар мен қарапайым кластар үшін базалық болуы да мүмкін.
- Шаблон ішінде **friend**-шаблондарды анықтауға болмайды.

Шаблонның мысалы ретінде **List** екібайланысты тізімінің параметрленген класының толық сипаттамасын қарастырайық.

```
template <class Data> class List{
   class Node{
      public:
         Data d;
         Node *next, *prev;
         Node(Data dat = 0){d = dat; next = 0; prev = 0;}
   };
   Node *pbeg, *pend;
   public:
      List(){pbeg = 0; pend = 0;}
      ~List();
      void add(Data d);
      Node * find(Data i);
      Node * insert(Data key, Data d);
      bool remove(Data key);
      void print();
      void print_back();
};
// ------------------------------
template <class Data>
   List <Data>::~List(){
      if (pbeg !=0){
         Node *pv = pbeg;
         while (pv){
            pv = pv->next; delete pbeg;
            pbeg = pv;}
      }
}
// ------------------------------
template <class Data>
   void List <Data>::print(){
      Node *pv = pbeg;
      cout << endl << "list: ";
      while (pv){
         cout << pv->d << ' ';
         pv = pv->next;}
      cout << endl;
}
// ------------------------------
template <class Data>
   void List <Data>::print_back(){
```

```
      Node *pv = pend;
      cout << endl << " list back: ";
      while (pv){
         cout << pv->d << ' ';
         pv = pv->prev;}
      cout << endl;
}
// ------------------------------
template <class Data>
   void List <Data>::add(Data d){
      Node *pv = new Node(d);
      if (pbeg == 0)pbeg = pend = pv;
      else{
         pv->prev = pend;
         pend->next = pv;
         pend = pv;}
}
// ------------------------------
template <class Data>
   Node * List <Data>::find(Data d){
      Node *pv = pbeg;
      while (pv){
         if(pv->d == d)break;
         pv = pv->next;
      }
      return pv;
}
// ------------------------------
template <class Data>
   Node * List <Data>::insert(Data key, Data d){
      if(Node *pkey = find(key)){
         Node *pv = new Node(d);
         pv->next = pkey->next;
         pv->prev = pkey;
         pkey->next = pv;
         if( pkey != pend) (pv->next)->prev = pv;
         else pend = pv;
         return pv;}
      return 0;
}
// ------------------------------
template <class Data>
   bool List <Data>::remove(Data key){
      if(Node *pkey = find(key)){
```

```
            if (pkey == pbeg){
               pbeg = pbeg->next; pbeg->prev = 0;}
            else if (pkey == pend){
               pend = pend->prev; pend->next = 0;}
            else {
               (pkey->prev)->next = pkey->next;
               (pkey->next)->prev = pkey->prev;}
            delete pkey; return true;}
         return false;
}
```

Егер **List** шаблонын құрамдас типтегі емес, қолданушы анықтаған типтегі мәліметтерді сақтау үшін қолдану керек болса, онда осы типтің сипаттамасында ағымға шығару (бұл туралы «Қолданушы анықтаған ағымдар мен типтер» бөлімінде, 305 б. жазылған) және теңдікке тексеру операцияларының асыра жүктелуін қосу керек, ал егер оның өрістері үшін жадыны динамикалық бөлу қолданылса, онда меншіктеу операциясын да қосу керек болады.

Шаблон синтаксисін анықтау кезінде оған типтер мен үлгілермен қоса, *айнымалыларды* да беруге болатынын айтылған. Олар бүтін немесе тізбелік типті, сонымен қатар объектіге немесе функцияға нұсқауыштар немесе сілтемелер болуы мүмкін. Олар шаблон тұлғасының тұрақтылық өрнекті пайдалануға болатын кез келген жерінде қолданылуы мүмкін. Мысал ретінде ұзындығы мен типі белгілі жады блогын қамтитын класс шаблонын құрайық:

```
template <class Type, int kol>
   class Block{
      public:
         Block(){p = new Type [kol];}
         ~Block(){delete [] p;}
         operator Type *();
      protected:
         Type * p;
};
template <class Type, int kol>
   Block <Type, kol>:: operator Type *(){
      return p;
}
```

Класс шаблондарын құру мен жөндеп түзетуден кейін тақырыптық файлдарға орналастырған ыңғайлы болады.

Класс шаблондарын қолдану

Шаблондар көмегімен нақты бір кластың нақты объектісін құру үшін (бұл үрдіс инстанциялау деп аталады) объектіні сипаттау кезінде шаблон атауынан кейін бұрыштық жақша ішінде оның аргументтері тізіліп жазылады:

```
шаблон_атауы <аргументтер>
объект_атауы [(конструктор_параметрлері)];
```

Аргументтер шаблонның параметрлерімен сәйкес келуі тиіс. Шаблонның аргументтерімен бірге жазылған атауын кластың нақтыланған атауы ретінде қабылдауға болады. Объектілерді өткен бөлімде сипатталған шаблондар бойынша құру мысалдары:

```
List <int> List_int;
List <double> List_double;
List <monstr> List_monstr;
Block <char, 128> buf;
Block <monstr, 100> stado;
```

Шаблонның келісім бойынша параметрлерін қолдану кезінде аргументтер тізімі бос болып шығуы мүмкін, мұндайда бұрыштық жақшаларды тастап кетуге болмайды:

```
template<class T = char> class String;
String<>* p;
```

Егер шаблонның параметрі специализациясы бар шаблон болса, ол инстанциялау кезінде ескеріледі (шаблондардың специализациясы туралы келесі бөлімде жазылады):

```
template<class T> class A{        // Бастапқы шаблон
   int x;
};
template<class T> class A<T*> {   // Шаблонның
                                  // специализациясы
   long x;
};
template<template<class U> class V> class C{
   V<int> y;
   V<int*> z;
};
...
C<A> c;
```

Бұл мысалда **C<A>** ішінде **V<int>** бастапқы шаблонды қолданады, сол себепті **c.y.x** типі **int** болады, ал **V<int*>** шаблон специализациясын қолданады, сондықтан **c.z.x** типі **long** болады.

Бүтін типті айнымалылар болып табылатын формалды параметрлердің орындарында тұрақтылық өрнектер болу керек.

Шаблонның көмегімен объектілерді құрғаннан кейін олармен қарапайым кластардың объектілерімен бірдей жұмыс істеуге болады, мысалы:

```
for (int i = 1; i<10; i++) List_double.add(i * 0.08);
List_double.print();
```

```
// ---------------------------------
for (int i = 1; i<10; i++) List_monstr.add(i);
List_monstr.print();
// ---------------------------------
strcpy(buf, "Өте маңызды хабарлама");
cout << buf << endl;
```

Кластардың шаблондарын қолдануды жеңілдету үшін **typedef** көмегімен типтерге қайта атау беруді жүзеге асыруға болады:

```
typedef List <double> Ldbl;
Ldbl List_double;
```

Класс шаблондарының специализациясы

Шаблон бойынша құрылатын кластың немесе функцияның әрбір нұсқасының базалық кодтары бірдей болады; тек шаблон параметрлерімен байланысы бар элементтер өзгереді. Мұндайда әртүрлі мәліметтер типтері үшін құрылатын нұсқалардың жұмыс істеу тиімділігі де әркелкі болуы мүмкін.

Егер қандай да бір мәліметтер типі үшін бұрынғысынан тиімдірек код бар болса, онда бұл тип үшін жеке әдістердің арнайы жүзеге асырылуын қарастыруға болады немесе класс шаблонын толығымен қайта анықтау (специализациялау) керек болады.

Әдісті специализациялау үшін оның тақырыбында нақты мәліметтер типін көрсету арқылы код нұсқасын анықта алу керек. Мысалы, егер **List** шаблонының **print** жалпылама әдісінің тақырыбы мынадай түрде болса:

```
template <class Data> void List <Data>:: print();
```

онда символдардың тізімін шығаруға арналған әдіс келесідей болады:

```
void List <char>::print(){
   ... // print әдісінің арнайы нұсқасының тұлғасы
}
```

Егер программада **char** типті **List** шаблонының данасын (экземплярын) құратын болсақ, әдістің сәйкес нұсқасы автоматты түрде шақырылады.

Бүтін класты специализациялау кезінде кластың жалпылама нұсқасын сипаттағаннан кейін специализацияланған кластың толық сипаттамасы орналастырылады, бұл кезде оның барлық әдістерін қайта анықтау қажет. 100 бүтін сан түріндегі шаманы сақтау үшін алдынғы бөлімде сипатталған **Block** шаблоныны специализациялау қажет болсын делік[1]:

```
class Block<int, 100>{
   public:
      Block(){p = new int [100];}
```

[1]Мұның терең мағынасын іздеудің қажеті жоқ – мысал синтаксисті көрсету үшін берілген.

```
      ~Block(){delete [] p;}
      operator int *();
   protected:
      int * p;
};
Block<int, 100>::operator int *(){
   return p;
}
```

Block шаблонының даналарын (экземплярларын) **int** және 100 параметрлері арқылы анықтау кезінде специализацияланған нұсқа іске қосылады.

Шаблондардың артықшылықтары мен кемшіліктері

Шаблондар әртүрлі мәліметтер типтерімен жұмыс істеудің қуатты әрі тиімді құралы болып табылады, оны басқаша параметрлі полиморфизм деп те атауға болады. Сонымен қатар, шаблондар препроцессор макростарына қарағанда, типтерді қауіпсіз қолдануды қамтамасыз етеді. Шаблондарды қолданатын программада әрбір туынды тип үшін толық код болатынын ескеру керек, бұл орындалатын файлдың көлемін арттыруы мүмкін. Сонымен қатар, кейбір мәліметтер типтерімен шаблондар басқа типтерге қарағанда айтарлықтай тиімді жұмыс жасамауы мүмкін. Мұндайда шаблонның специализациясын қолданған жөн.

C++ стандартты кітапханасы мәліметтерді сақтау мен өңдеуді ұйымдастырудың әртүрлі тәсілдеріне арналған шаблондардың үлкен жиынтығы болып келеді («Контейнерлік кластар» бөлімін, 318 б. қараңыз).

Класс шаблондары практикумның [12] үшінші семинарында түпкілікті қарастырылған.

7-ТАРАУ

Аластамалы жағдайларды өңдеу

Ерекше жағдай немесе *аластама* – аппаратураны дұрыс қолданбаудан туындайтын көзделмеген және төтенше жағдайлардың пайда болуы. Мысалы, нөлге бөлу немесе жадының жоқ адресіне хабарласу. Әдетте осындай оқиғалар программаның қателік туралы жүйелік хабарламасымен аяқталуына алып келеді. C++ тілі программалаушыға программаны қалпына келтіруге және оның жұмысын жалғастыруға мүмкіндік береді.

C++ тілінің аластамалары құрал-жабдықтардың қателіктері немесе программа үзілістерін өңдеу, мысалы, Ctrl+C батырмаларын басуды өңдеу сияқты асинхронды оқиғаларды өңдеуді қолдамайды. Аластамалар механизмі программаның жұмысы нәтижесінде пайда болатын және тікелей түрде көрсетілетін оқиғаларға ғана арналған. Программаның қандай да бір бөлігі өзінен талап етілген әрекетті орындай алмаған кезде аластамалар туындайды. Олар жұмысқа кіріскенде, программаның басқа бір бөлігі басқа іс-әрекет орындауға талпынуы мүмкін.

Аластамалар есептеу процесін логикалық түрде екі бөлікке бөлуге мүмкіндік береді, олар – төтенше (апаттық) жағдайды анықтау және оны өңдеу. Бұл тек қана программаны жақсы құрылымдау үшін ғана маңызды емес. Мұның басты себебі – қателікті тапқан функция оны түзету үшін не істеу керектігін білмей, ал осы функцияны қолданатын код қателікті қалай түзетуге болатынын біліп, тек оның пайда болу орнын анықтай алмауы мүмкін. Бұл, әсіресе, кітапханалық функциялар мен көптеген модульдерден құралатын программаларды қолдану кезінде өте өзекті жұмыс болып табылады.

Аластамалардың басқа бір артықшылығы – шақырушы функцияға қате туралы ақпаратты беру үшін қайтарылатын мәнді, параметрлерді немесе ауқымды айнымалыларды қолданудың қажеті жоқ, сондықтан функцияның интерфейсі кеңейтілмейді. Мысалы, синтаксисі бойынша мән қайтара алмайтын конструкторлар үшін бұл өте маңызды болып саналады.

ЕСКЕРТУ

Жалпы алғанда, аластамалар ретінде тек қателіктерді емес, мәліметтерді өңдеу кезінде туындайтын қалыпты жағдайларды да қарастыруға болады, алайда бұл әрекеттің басқа шешімдерден басымдылығы жоқ және ол программаның құрылымы мен оқылу жеңілдігін жақсартпайды.

Аластамаларды өңдеудің жалпы механизмі

Қателік туындауы мүмкін орын бақылаудағы блоктың құрамына кіруі тиіс, бақылаудағы блок – бұл алдынан **`try`** түйінді сөзі жазылатын құрамдас оператор.

Ерекше жағдайларды өңдеу қалайша жүзеге асырылатынын қарастырайық.

□ Аластаманы өңдеу қате туындаған кезден басталады. Қателік пайда болған функция аластаманы генерациялайды. Бұл үшін аластаманың түрін анықтайтын параметрі бар **throw** түйінді сөзі қолданылады. Параметр тұрақты, айнымалы немесе объект болуы мүмкін және ол аластама туралы ақпаратты аластама өңдеуішіне жөнелту үшін қолданылады.

□ Аластамаға сәйкес келетін өңдеуіш ізделіп, оған басқару беріледі.

□ Егер аластаманы өңдеуіш табылмаса, **terminate** стандартты функциясы шақырылады, ол, өз кезегінде, ағымдағы процесті апаттық жағдайда аяқтайтын **abort** функциясын шақырады. Процесті аяқтайтын өзіндік функцияны да тағайындауға болады.

Кітаптың бірінші бөлімінде («Функциялар» бөлімін, 74 б. қараңыз) әрбір функцияны шақыру кезінде стекте жергілікті (локалды) айнымалыларды және шақырушы функцияға қайту адресін сақтауға арналған жады аймағы құрылатыны туралы сөз болған. *Шақырулар стегі* термині шақырылған, бірақ әлі жұмысы аяқталмаған функциялар тізбегін білдіреді. Стекті тарқату деп жергілікті айнымалыларға бөлінген жадыны босатып, басқаруды шақырушы функцияға қайтару процесін айтады. Функция жұмысын аяқтаған кезде, стектің табиғи түрде тарқатылуы орындалады. Дәл осы механизм аластамаларды өңдеуде де қолданылады. Сондықтан аластама бар екені белгілі болғаннан кейін, программаның орындалуын аластаманы генерациялау нүктесінен ары қарай жалғастыру мүмкін емес. Бұл механизм келесі бөлімде толығырақ қарастырылады.

Аластамалардың синтаксисі

try түйінді сөзі бақылаудағы блокты белгілеу үшін қолданылады, мұндағы бақылаудағы блок – аластама генерациялануы мүмкін болатын программа коды. Блок жүйелі жақшаларға алынып жазылады:

```
try{
    ....
}
```

try-блоктан тікелей немесе жанамалы түрде шақырылатын барлық функциялар оған тиісті болып саналады.

Аластаманың генерациясы (туындауы) **throw** түйінді сөзі арқылы орындалады, ол параметрмен бірге де немесе параметрсіз де қолданыла береді:

```
throw [ өрнек ];
```

throw сөзінен кейін орналасатын өрнектің типі туындайтын аластаманың типін анықтайды. Аластама генерацияланған кезде ағымдағы блоктың орын-

далуы тоқтатылады да, соған сәйкес келетін өңдеуіш ізделіп, оған басқару беріледі. Көбінесе аластама тікелей **try**-блокта емес, оған тура немесе жанама түрде қабаттастырылып енгізілген функцияларда генерацияланады.

Ішкі блокта пайда болған аластама әрқашан бірден дұрыс өңделе бермейді. Мұндайда қабаттасқан бақылаудағы блоктар қолданылады да, аластама параметрсіз **throw** түйінді сөзінің көмегімен жоғары орналасқан деңгейге беріледі.

Аластамаларды өңдеу **catch** түйінді сөзінен басталады да, оның артынан өңделетін аластама типі жазылады. Олар тікелей **try**-блоктан кейін орналасуы тиіс. Өңделетін аластамалар типтеріне сәйкес бір немесе бірнеше өңдеуіштерді жазуға болады. Өңдеуіштердің синтаксисі аластама типіндегі бір параметрі бар функцияның анықтамасына ұқсас болады. Өңдеуіштер синтаксисін жазудың үш формасы бар:

```
catch(тип атауы){ ... /* өңдеуіш тұлғасы  */ }
catch(тип){ ... /* өңдеуіш тұлғасы*/ }
catch(...){ ... /* өңдеуіш тұлғасы */ }
```

Бұлардың бірінші формасы – өңдеуіш тұлғасында кез келген бір әрекеттерді орындау үшін параметр аты пайдаланылғанда, мысалы, аластама туралы ақпаратты шығару үшін қолданылады. Екінші формада аластама туралы ақпараттың қолданылуы қарастырылмаған, мұнда тек оның типі ғана маңызды рөл атқарады. Параметрдің орнындағы көп нүкте барлық ерекше жағдайларды өңдеуіштің өзі жүзеге асыратынын (тауып өңдейтінін) білдіреді. Өңдеуіштер жазылу реттілігімен қарастырылатындықтан, үшінші типтегі өңдеуішті оның басқа формаларынан кейін орналастырған жөн. Мысалы:

```
catch(int i){
   ... // int типіндегі аластамаларын өңдеу
}
catch(const char *){
   ... // const char* типіндегі аластамаларын өңдеу
}
catch(Overflow){
   ... // Overflow класы аластамаларын өңдеу
}
catch(j){
   ...   // барлық қарастырылмаған аластамаларды өңдеу
}
```

Аластаманы өңдеуден кейін басқару бірден тікелей аластаманы өңдеуіштен соң орналасқан алғашқы операторға беріледі. Егер аластама **try**-блокта генерацияланбаса, онда басқару барлық өңдеуіштердің кодын орындамай, бірден осы аталған операторға беріледі.

Аластамаларды айқындап атқару

throw түйінді сөзінің көмегімен аластама генерацияланған кезде, C++ атқарушы кітапханасының функциялары келесідей әрекеттерді орындайды:

1) статикалық объект түрдегі **throw** параметрінің көшірмесін жасайды, ол аластама өңделіп болғанша жойылмайды;

2) сәйкес келетін өңдеуішті табу мақсатында әрекет ету аймағынан тыс орналасқан жергілікті объектілердің деструкторларын шақыру арқылы стекті тарқату ісі атқарылады;

3) обект және әрекеттерді басқару осы объектімен үйлесімді типтегі параметрі бар өңдеуішке беріледі.

Стекті тарқату кезінде барлық өңдеуіштер өздеріне сәйкес келетіні табылғанша ішкі блоктан сыртқысына қарай әрбір деңгейде тізбектеле қарастырылады.

Егер **throw** түйінді сөзінен кейін көрсетілген объектінің типі төмендегілердің біріне сәйкес келсе:

- **catch** параметрінде көрсетілген типпен бірдей болғанда (параметр **T**, **const T**, **T&** немесе **const T&** түрінде жазылуы мүмкін, мұндағы **T** – аластаманың типі);
- **catch** параметрінде көрсетілген типтің туынды типі болған жағдайда (егер мұралау **public** қол жеткізу кілтімен жүзеге асырылса);
- нұсқауыштарды түрлендірудің стандартты ережелері бойынша **catch** параметріндегі нұсқауыш типіне түрлендірілуі мүмкін нұсқауыш болып табылған кездерде өңдеуіш табылған болып саналады.

Аталған ережелерге сәйкес, туынды кластардың өңдеуіштерін базалық класс өңдеуіштерінен бұрын орналастыру керек, өйткені бұған кері жағдайда басқару оларға еш уақытта берілмейді. **void** типті нұсқауыштың өңдеуіші кез келген басқа типтің нұсқауышын автоматты түрде жасырады, сондықтан оны да нақты тип нұсқауыштарының өңдеуіштерінен кейін орналастыру қажет.

Мысал қарастырайық.

```
#include <fstream.h>
class Hello{
// Өзінің құрылғаны және жойылғаны туралы
// хабарлайтын класс
public:
   Hello(){cout << "Hello!" << endl;}
   ~Hello(){cout << "Bye!" << endl;}
};
void f1(){
   // Файл ашамыз
   ifstream ifs("\\INVALID\\FILE\\NAME");
   if (!ifs){
```

```
      cout << "Аластама генерациялаймыз" << endl;
      throw "Файл ашуда қате кетті";}
}
void f2(){
   Hello H; // Жергілікті объект құрамыз
   f1();    // Аластама генерациялайтын
            // функцияны шақырамыз
}
int main(){
   try{
      cout << "try-блокқа кіреміз" << endl;
      f2();
      cout << "try-блоктан шығамыз" << endl;
   }
   catch(int i){
      cout << "int өңдеуіші шақырылды, аластама - "
           << i << endl;
      return -1;
   }
   catch(const char * p){
      cout << "const char* өңдеуіші шақырылды,
              аластама - " << p << endl;
      return -1;
   }
   catch(...){
      cout << "Барлық аластамалардың өңдеуіші
              шақырылды" << endl;
      return -1;
   }
   return 0; // Барлығы сәтті орындалды
}
```

Программаның орындалу нәтижесі:

```
try-блокқа кіреміз
Hello!
Аластама генерациялаймыз
Bye!
const char * өңдеуіші шақырылды, аластама – Файл ашуда
қате кетті
```

Аластаманың туындауынан кейін **f1** функциясынан басқару **main** функциясында орналасқан өңдеуішке берілгендігіне қарамастан, жергілікті объектінің деструкторы шақырылғанына назар аударыңыздар. Мұнда «**try**-блоктан шығамыз» хабарламасы шығарылған жоқ. Программада файлмен жұмыс істеу үшін ағымдар қолданылды (олар туралы «Ағымдық кластар» бөлімінде, 283 б. жазылады).

Сонымен, аластамалар механизмі қателіктер туындаған кезде объектілерді дұрыс жоюға мүмкіндік береді. Сондықтан *ресурстарды бөлу және босатуды кластар түрінде ұйымдастырған тиімді* болып саналады, өйткені олардың конструкторы ресурсты бөледі, ал деструкторы – босатады. Мысал ретінде файлмен жұмыс істеуге арналған класты келтіруге болады. Класс конструкторы файлды ашады, ал деструкторы – жабады. Мұндайда қателік туындаған кезде файл дұрыс жабылып, ақпарат жойылмайтынына кепілдік берілетінін айтуға болады.

Жоғарыда көрсетілгендей, аластама стандартты немесе қолданушы анықтаған типте болуы мүмкін. Бұл типті ауқымды түрде анықтаудың қажеті жоқ, оның аластама туындаған нүктеде және аластаманы өңдеу нүктесінде белгілі болуы жеткілікті. Аластаманы бейнелейтін класты, онымен жұмыс істегенде осы аластама туындауы мүмкін басқа кластың ішінде сипаттауға болады. Осы кластың көшірме конструкторы **public** болып жариялануы тиіс, өйткені оған кері жағдайда аластама туындағанда объектінің көшірмесін құру мүмкін болмай қалады (келісім бойынша құрылатын көшірме конструкторының спецификаторы **public** болады).

Функцияның аластамалар тізімі

Функцияның тақырыбында ол тікелей немесе жанама түрде туындатуы мүмкін аластамалар тізімін беруге болады. Тақырып бөлігі функцияның интерфейсі болғандықтан, мұнда аластамалар тізімін беру арқылы қолданушылар функцияны пайдалану туралы ақпарат алады және де көзделмеген аластама пайда болса, бұл жағдай анықталатыны туралы кепілдік беріледі.

Аластаманы өңдеу алгоритмі 7.1-суретте көрсетілген.

Аластамалардың типтері функцияның параметрлер тізімінен кейін орналасқан **throw** түйінді сөзінің соңынан жақша ішінде үтір арқылы бөліне отырып, тізбектеле жазылады, мысалы:

```
void f1() throw (int,const char*){/* Функция тұлғасы */ }
void f2() throw (Oops*){ /* Функция тұлғасы */ }
```

Мысалдағы **f1** функциясы тек **int** және **const char*** типтерінің аластамаларын туындатуы керек.

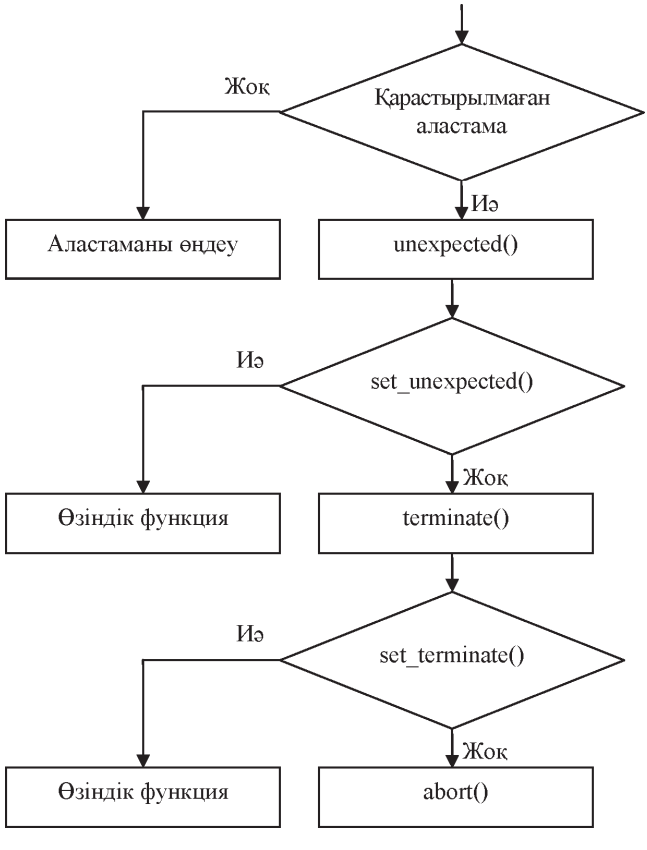
Келесі **f2** функциясы тек **Oops** класына немесе одан туынды кластарға нұсқауыш типінің ерекше жағдайларын генерациялауы тиіс.

Егер **throw** түйінді сөзі көрсетілмесе, функция кез келген аластаманы генерациялауы мүмкін. Бос тізім функция аластамаларды тудырмайтынын білдіреді:

```
void f() throw (){
   // аластамаларды тудырмайтын функцияның тұлғасы
}
```

Аластамалар функцияның прототипіне енгізілмейді. Туынды класта виртуалды функцияны қайта анықтау кезінде базалық кластың соған сәйкес функциясындағы сияқты немесе оған қарағанда шектелген аластамалар тізімін беруге болады.

Аластамалар тізімін көрсетудің ешбір салдары жоқ – функция қолданбауға уәде берген аластаманы тікелей немесе жанамалы түрде туындатуы мүмкін. Мұндай жағдай программаны орындау кезінде көрінеді де, **unexpected** стандартты функциясының шақырылуына әкеліп соғады, ол, өз кезегінде, келісім бойынша **terminate** функциясын шақырады. **set_unexpected** функциясының көмегімен **terminate** функцияның орнына шақырылатын және көзделмеген аластама туындаған кездегі программаның әрекетін анықтайтын өзіндік функцияны тағайындауға болады.



7.1-сурет. Аластаманы өңдеу алгоритмі

terminate функциясы келісім бойынша программаның орындалуын аяқтайтын **abort** функциясын шақырады. **set_terminate** функциясының көмегімен **abort** функциясының орнына шақырылып, программаны аяқтау тәсілін анықтайтын өзіндік (меншікті) функцияны тағайындауға болады. **set_unexpected** және **set_terminate** функциялары **<exception>** тақырыптық файлында сипатталған.

Конструкторлар мен деструкторлардағы аластамалар

C++ тілі конструктор мен деструктордан мән қайтаруға жол бермейді. Аластамалар механизмі объектінің конструкторында немесе деструкторында туындаған қателік туралы хабарлауға мүмкіндік береді. Осыны көрнекі түрде көрсету үшін қажетті жады көлемі шектелетін **Vector** класын құрайық.

```
class Vector{
   public:
      class Size{};          // Аластама класы
      enum {max = 32000};    // Вектордың максималды
                             // ұзындығы
      Vector(int n)          // Конструктор
         { if (n<0 || n>max) throw Size(); ... }
      ...
};
```

Vector класын қолдану кезінде **Size** типті аластамаларды айқындап атқаруды алдын ала қарастыруға болады.

```
try{
   Vector *p = new Vector(i);
   ...
}
catch(Vector::Size){
    ... // Вектор өлшемінің қателігін өңдеу
}
```

Өңдеуіште қателік және қалпына келтіру туралы хабарламалар берудің негізгі тәсілдерінің стандартты жиынтығы қолданылуы мүмкін. Аластаманы анықтайтын кластың ішінде аластама туралы өңдеуішке берілетін ақпарат сақталуы мүмкін. Бұл техниканың мәні – қателік туралы ақпаратты табылған нүктесінен қатені өңдеуге қажетті мүмкіндіктері бар жерге дейін жеткізуді қамтамасыз ету болып табылады.

Егер объектінің конструкторында аластама генерацияланатын болса, онда автоматты түрде осы блокта ағымдағы сәтке дейін толығымен құрылған объектілер үшін, сонымен қатар ағымдағы объектінің объектілер болып табылатын мәліметтер өрістері үшін және оның базалық кластары үшін деструктор-

лар шақырылады. Мысалы, егер аластама объектілердің жиымын құру кезінде пайда болса, деструкторлар тек сәтті құрылған элементтер үшін шақырылады.

Егер объект динамикалық жадыда **new** операциясының көмегімен құрылып, конструкторда аластама туындаған болса, онда объектіге бөлінген жады дұрыс босатылады.

Аластамалардың иерархиялары

Өзіндік аластамалар кластарын қолдану стандартты мәліметтер типтерін қолдануға қарағанда басымдыққа ие болады. Кластардың көмегімен аластама туралы ақпаратты беруді айтарлықтай икемді ұйымдастыруға, аластамалардың өңделуін оңай ажыратуға болады, сонымен қатар кластардың иерархияларын қолдануға мүмкіндік туады.

Аластамаларды басқару механизмі базалық класс үшін өңдеуішті құруға мүмкіндік беретін болғандықтан, көбінесе туыстас аластамаларды иерархия түрінде көрсетуге болады. Ортақ базалық кластан аластамаларды туындата отырып, полиморфизмді қолдану арқылы өңдеуіште базалық класқа сілтемені немесе нұсқауышты айқындап атқару (ұстап қалу) мүмкіндігі бар. Мысалы, математикалық кітапханада кластарды келесідей түрде ұйымдастыруға болады:

```
class Matherr{};
class Overflow: public Matherr{};     // Асыра толу
class Underflow: public Matherr{};    // Дәреженің жоғалуы
class ZeroDivide: public Matherr{}; // Нөлге бөлу
```

Енгізу/шығару қателерін көрсету үшін келесі кластар қолданылуы мүмкін:

```
class IOerr;
class Readerr: public IOerr{};      // Оқу қателігі
class Writerr: public IOerr{};      // Жазу қателігі
class Seekerr: public IOerr{};      // Іздеу қателігі
```

Пайда болатын жағдайларға байланысты туынды аластамаларды да айқындап атқаратын (ұстап қалып отыратын) базалық кластың аластама өңдеуішін немес туынды кластардың өзіндік өңдеуіштерін қолдануға болады.

C++ операциялары немесе функциялары генерациялайтын бірқатар *стандартты аластамалар* бар («Стандартты кітапхананың басқа құралдары» деп аталатын 16-бөлімді, 411 б. қараңыз). Олардың барлығы **<stdexcept>** тақырыптық файлында сипатталған **exception** кітапханалық класынан туынды болып табылады. Мысалы, **new** операциясы жадыны сәтсіз бөлу жағдайында **bad_alloc** типті аластаманы генерациялайды.

Программалаушы стандартты аластамалардан туынды болатын өзіндік аластамаларын анықтай алады.

Аластамаларды өңдеу практикумның [12] үшінші семинарында қарастырылады.

8-ТАРАУ

Типтерді түрлендіру

Программаны орындау кезінде шамаларды бір типтен екінші бір типке тікелей немесе жанама түрде түрлендірулер жасалады. Жанама түрлендірулер 32 б. және 3-қосымшада көрсетілген ережелерге сәйкес орындалады. C++ тілінде типтің нақты түрде тікелей түрлендірілуін жүзеге асыруға арналған бірсыпыра операциялар тобы бар, олар: **const_cast, dynamic_cast, reinterpret_cast** және **static_cast** және де осы кітапта бұдан бұрын да қолданылып келе жатқан (54 б. қараңыз), C тілінен мұраланған типті келтіру операциясы бар. Алдымен осы операцияны толығырақ қарастырып шығайық.

C стиліндегі типтерді келтіру операциясы

Операция екі түрлі формада жазылуы мүмкін:

```
тип (өрнек)
(тип) өрнек
```

Операцияның нәтижесі берілген типтегі мән болып табылады, мысалы:

```
int a = 2;
float b = 6.8;
printf("%lf %d", double (a), (int) b);
```

Бұл мысалдағы **a** шамасы **double** типіне, ал **b** айнымалысы бөлшек бөлігі алып тасталған **int** типіне түрлендіріледі, осы екі жағдайда да түрлендіру операциясының нәтижесін бейнелеудің ішкі формасы оның бастапқы мәнінің формасынан өзгеше болып келеді.

Типті түрлендіру қажеттілігі, мысалы, нақты көрсетілген типті айнымалымен қандай да бір әрекеттерді орындау үшін, функция оған меншіктелуі тиіс **void** типіне нұсқауышты қайтаратын кезде туындайды:

```
float *q = (float *) malloc(100 * sizeof(float));
```

Типті тікелей түрде түрлендіру кейбір қателіктердің туындауына себепші болып табылады, өйткені оның нәтижесі үшін жауапкершілік толығымен тек программалаушыға жүктеледі. Сондықтан C++ тілінде орындалатын түрлендірулерге ішінара аздаған бақылау жасауға немесе программалаушының ниетін түсінуді жеңілдетуге мүмкіндік беретін операциялар енгізілген. Жоғарыда қарастырылған C стиліндегі түрлендіру C++ ортасында тек тілдер үйлесімділігін сақтау үшін қалдырылған, оны *кеңінен қолдану ұсынылмайды.* Мұнда қажет етілетін түрлендіру түріне байланысты соған сәйкес келетін лайықты типті келтіру операциясы қолданылуы тиіс.

ЕСКЕРТУ

C++ компиляторларының алғашқы нұсқалары **const_cast**, **reinterpret_cast**, **dynamic_cast** және **static_cast** сияқты типті келтіру операцияларын сүйемелдемейді.

const_cast операциясы

Бұл операция **const** модификаторын жою үшін қажет. Көбінесе ол функциядағы **const** модификаторы жоқ формалды параметрдің орнына тұрақтылық нұсқауышты беру кезінде қолданылады. Операция форматы:

const_cast <тип> (өрнек)

Мұндағы тип өрнек типімен бірдей болуы керек, бірақ ол **const** модификаторы бола алмайды. Әдетте бұл нұсқауыш болып табылады. Операция көрсетілген типтегі нәтиже алуды қалыптастырады.

Бұл операцияны енгізу қажеттілігі – функцияны жүзеге асыратын программалаушы ондағы өзгермейтін формалды параметрлерді **const** түрінде сипаттауға міндетті болмағанымен, бірақ көбінесе осылай істеу ұсынылады. C++ тілінің ережелері қарапайым нұсқауыштың орнына тұрақтылық нұсқауышты беруге тыйым салады. **const_cast** операциясы бұл шектеуді айналып өту үшін енгізілген. Әрине, функция берілетін нұсқауыш сілтеме жасайтын мәнді өзгертуге талпынбауы тиіс, әйтпесе программаның орындалу нәтижесі анықталмаған болып саналады.

Мысал:

```
void print(int *p){ // Функция *p мәнін өзгертпейді
   cout << *p;
}
const int *p;
...
/* print(p); Қате, өйткені p тұрақтыға нұсқауыш ретінде
             жарияланған */
```

Программалаушы функцияның тұлғасында нұсқауыш сілтеме жасайтын мән өзгермейтініне сенімді болған жағдайда ғана, **const_cast** операциясы қолданылады. Әрине, формалды параметрдің сипаттамасына **const** модификаторын қосу мүмкіндігі болса, бұл әрекет функцияны шақыру кезінде типті түрлендіруді қолдануға қарағанда дұрысырақ болып саналады.

dynamic_cast операциясы

Бұл операция иерархияның туыстас кластарының нұсқауыштарын, негізінен, базалық класс нұсқауышын туынды класс нұсқауышына түрлендіру үшін қолданылады, мұнда программаның орындалу барысында түрлендірудің жасалу мүмкіндігі тексеріледі.

Операция форматы:

```
dynamic_cast <тип *> (өрнек)
```

өрнек класқа нұсқауыш немесе сілтеме болып, ал **тип** осы класс үшін базалық немесе туынды класс болуы тиіс. Түрлендірудің орындалу мүмкіндігі тексерілгеннен кейін, егер ол сәтті орындалса, операция берілген типтегі нәтиже қалыптастырады, кері жағдайда нұсқауыш үшін нәтиже нөлге тең[1] болады, ал сілтеме үшін – **bad_cast** аластамасы туындайды. Егер берілген тип пен өрнектің типі бір иерархияға жатпаса, түрлендіруді жүзеге асыру мүмкін болмайды.

Базалық кластан туынды класқа түрлендіру *төмендететін түрлендіру* (downcast) деп аталады, өйткені графикалық түрдегі мұралау иерархиясында туынды кластар базалық кластардан төменірек бейнеленеді. Туынды кластан базалық класқа келтіру *жоғарылататын түрлендіру* (upcast) деп, ал бір базалық класс ішіндегі туынды кластар арасындағы немесе керісінше, бір туынды кластың базалық кластары арасындағы келтіру операциялары *қиылысқан түрлендіру* (crosscast) деп аталады.

Жоғарылататын түрлендіру

Жоғарылататын түрлендіруді **dynamic_cast** операциясы арқылы орындау қарапайым меншіктеумен (ашық мұралау кезінде) бірдей болып саналады:

```
class B{ /* ... */ };
class C: public B{ /* ... */ };
C* c = new C;
B* b = dynamic_cast<B*>(c);  // B* b = c эквивалентті;
```

Төмендететін түрлендіру

dynamic_cast операциясы көбінесе төмендететін түрлендіруде – компилятордың түрлендіру дұрыстығын тексеру мүмкіндігі болмаған кезде қолданылады.

Туынды кластар құрамында базалық кластарда жоқ функциялар болуы мүмкін. Оларды базалық кластың нұсқауышы арқылы шақыру үшін осы нұсқауыштың расында да туынды кластың объектісіне сілтеме жасайтынына сенімді болу керек. Мұндай тексеру **RTTI** (run-time type information) – «программаның атқарылуы кезіндегі тип туралы ақпарат» арқылы типті келтіруді жүзеге асыру сәтінде орындалады. Түрлендіруді тексеру мүмкіндігі орындалуы үшін **dynamic_cast** операциясының аргументі полиморфты типте болуы тиіс, яғни оның кем дегенде бір виртуалды әдісі (222 б. қараңыз) болуы керек.

[1]Егер өрнек нөлге тең болса, оның нәтижесі де нөлге тең болады.

ЕСКЕРТУ

RTTI ақпаратын қолдану үшін программаға **<typeinfo>** тақырыптық файлын қосу керек. Сонымен қатар, компилятордың соған сәйкес режимі де орнатылуы тиіс.

Полиморфты объект үшін **dynamic_cast** операциясының жүзеге асырылуы айтарлықтай тиімді болады, өйткені объектінің типі туралы ақпаратқа сілтеме виртуалды әдістер кестесіне енгізіледі де, оған қол жеткізу жеңіл іске асырылады.

Логикалық тұрғыдан алғанда да, объектінің полиморфты болу талабы дұрыс шешім болып табылады: егер кластың виртуалды әдістері болмаса, нұсқауыштың нақты түрдегі типін білместен, оны қауіпсіз түрде қолдану мүмкін емес. Ал егер тип белгілі болса, онда **dynamic_cast** операциясын қолданудың қажеті жоқ.

Нұсқауышқа **dynamic_cast** операциясын қолданудың нәтижесін әрқашанда нақты түрде тексеру керек. Төмендегі мысалда **B** полиморфты базалық класы және одан туынды болып келетін, құрамында **f2** функциясы анықталған **C** класы сипатталған. **demo** функциясынан туынды класс объектісіне нұсқауыш берілген жағдайда ғана **f2** функциясын **demo** функциясынан шақыру үшін түрлендіру нәтижесінің тексерілуі орындалатын **dynamic_cast** операциясы қолданылады:

```
#include <iostream.h>
#include <typeinfo.h>
class B{
   public: virtual void f1(){};
};
class C: public B{
   public: void f2(){cout << "f2";};
};
void demo(B* p){
   C* c = dynamic_cast<C*>(p);
   if (c) c->f2();
   else cout << "Берілген C класы емес";
}
int main(){
   B* b = new B;
      demo(b); // "Берілген C класы емес"
               // хабарламасы шығарылады
      C* c = new C;
      demo(c); // "f2" (дұрыс) хабарламасы шығарылады
      return 0;
}
```

Бұл мысалдағы **dynamic_cast** орнына C стиліндегі типтерді келтіруді қолданатын болсақ, мысалы:

```
C* c = (C*) p;
```

операцияның орындалу мүмкіндігін қадағалау мүмкін емес және **p** нұсқауышы **C** класының объектісіне сілтеме жасамайтын болса, бұл қателіктің пайда болуына алып келеді.

С стилінде түрлендірудің тағы бір кемшілігі – виртуалды базалық класты туынды класқа түрлендіру мүмкін емес, бұған синтаксистік түрде тыйым салынған. Мұндай түрлендіру класс полиморфты және түрлендіру екімағыналы болмаған жағдайда, **dynamic_cast** операциясының көмегімен орындалуы мүмкін. *Виртуалды базалық кластың төмендететін түрлендірілуі* орындалатын мысалды қарастырайық:

```
#include <iostream.h>
#include <typeinfo.h>
class A{                              //   A
   public: virtual ~A(){};};          //  /  \
class B: public virtual A{};          // B   C
class C: public virtual A{};          //  \  /
class D: public B, public C{};        //    D
void demo(A *a){
   D* d = dynamic_cast<D*>(a);
   if (d) { ... }
}
int main(){
   D *d = new D; demo(d);
   return 0;
}
```

Сілтемелерді түрлендіру

Аргумент-сілтеме үшін түрлендіру операциясының мағынасы нұсқауыштарқа қарағанда өзгешелеу болып келеді. Сілтеме әрқашанда нақты көрсетілетін объектіге нұсқайтын болғандықтан, **dynamic_cast** операциясы дәл осы объектінің типіне түрлендіруді орындауы керек. Келтіру операциясының дұрыстығы автоматты түрде тексеріледі, сәйкессіздік жағдайында **bad_cast** аластамасы туындайды:

```
#include <iostream.h>
#include <typeinfo.h>
class B{
   public: virtual void f1(){};
};
class C: public B{
   public: void f2(){ ... };
};
void demo(B& p){
```

```
   try{
      C& c = dynamic_cast<C&>(p); c.f2();
   }
   catch(Bad_cast){
      ...
   }
}
int main(){
   B* b = new B; demo(*b); // Аластама туындайды
   C* c = new C; demo(*c); // Дұрыс
   return 0;
}
```

Қиылысқан түрлендіру

dynamic_cast операциясы бір базалық кластың туынды кластарының арасындағы түрлендіруді қауіпсіз түрде орындауды қамтамасыз етеді, мысалы:

```
#include <iostream.h>
#include <typeinfo.h>
class B{
   public: virtual void f1(){};
};
class C: public B{
   public: void f2(){ ... };
};
class D: public B{ ... };
void demo(D* p){
   C* c = dynamic_cast<C*>(p);
   if(c)c->f2();
   else cout <<" орындалмады ";
}
int main(){
   B* b = new C; demo((D*)b);
   return 0;
}
```

C және **D** кластары **B** класынан туынды болып табылады. **demo** функциясына **D** класының нұсқауышы беріледі, ол шын мәнінде, оған «ағайынды» болып табылатын **C** класына нұсқауыш болып табылады, сондықтан **D** класынан **C** класына типті динамикалық түрде түрлендіру **demo** функциясында сәтті аяқталады.

ЕСКЕРТУ

Функцияға берілетін нұсқауыш жасанды түрде тек типті динамикалық түрлендірудің мүмкіндігін көрсету үшін қалыптастырылған.

Қажет болса, бір туынды кластың базалық кластары арасындағы түрлендіруді де жүзеге асыруға болады, мысалы:

```
#include <iostream.h>
#include <typeinfo.h>
class B{
   public: virtual void f1(){ ... };   // B  C
};                                     // \ /
class C{                               //  D
   public: virtual void f2(){ ... };
};
class D: public B, public C{};
   void demo(B* b){
   C* c = dynamic_cast<C*>(b);
   if(c)c->f2();
}
int main(){
   D* d = new D; demo(d);
   return 0;
}
```

D класы **B** және **C** кластарының ұрпағы болып табылады, сондықтан оның құрамында осы екі кластың да әдістері сақталады. Егер **demo** функциясына **B** класына емес, **D** класына деген нұсқауыш берілетін болса, онда оны екінші **C** базалық класына түрлендіруге болады.

static_cast операциясы

static_cast операциясы компиляция кезеңінде типтерді түрлендіруді келесі типтер арасында жүзеге асыру үшін қолданылады:

- бүтін типтер;
- бүтін және нақты типтер;
- бүтін және тізбелік типтер;
- түрлендіру бірмәнді түрде болып, ол виртуалды базалық класты төмендететін түрлендірумен байланысты болмаған жағдайда, бір иерархия объектілеріне нұсқауыштар мен сілтемелер арасында атқарылады.

Операцияның форматы:

```
static_cast <типі> (мәні)
```

Операция нәтижесі көрсетілген типте болады, ол сілтеме, нұсқауыш, арифметикалық немесе тізбелік тип болуы мүмкін.

Операцияны орындау кезінде мәліметтердің ішкі бейнесі өзгертілуі мүмкін, бірақ сандық мәні өзгермейді. Мысалы:

```
float f = 100;
int i = static_cast <int> (f);
// Бүтін және нақты типтердің ішкі бейнесі әртүрлі
```

Мұндай түрлендірулер мәліметтерді жоғалту мүмкіндігі туралы компилятордың хабарламаларын шығармай басып тастау үшін қажет, бірақ мұндайда дәл осы әрекет орындалатынына сенімді болу керек. Түрлендіру нәтижесі программалаушының жауапкершілігінде қалады.

static_cast операциясы туынды кластан базалық класқа және керісінше түрлендіруді шектеусіз түрде жүзеге асыруға мүмкіндік береді.

```
class B{};
class C: public B{};
C c;
B *bp = static_cast<B*>(c); // Туынды -> базалық
B b;
C &cp = static_cast<C&>(b); // Базалық -> туынды
```

Түрлендіру компиляция барысында жүзеге асырылады, бұл кезде объектілер полиморфты болмауы да мүмкін. Программалаушы түрлендірілген шамалармен одан әрі жұмыс істеу мүмкіндігін өзі қадағалауы тиіс.

Жалпы жағдайда иерархияның туыстас кластарының нұсқауыштарын түрлендіру үшін **dynamic_cast** операциясын қолданған жөн. Мұндайда, егер түрлендіруді компиляция кезеңінде жүргізу мүмкін болса, **static_cast** үшін генерацияланған кодпен бірдей код туындайды. Сонымен қатар, **dynamic_cast** қиылысқан түрлендіру мен виртуалды базалық кластың төмендететін түрлендірілуін жүзеге асыра алады және түрлендірудің орындалу мүмкіндігін оның атқарылу кезінде тексереді.

reinterpret_cast операциясы

reinterpret_cast операциясы өзара байланыспаған типтерді, мысалы, нұсқауыштарды бүтін типке және керісінше, **void*** типті нұсқауыштарды нақты бір көрсетілген типке түрлендіру үшін қолданылады. Мұндайда мәліметтердің ішкі бейнесі өзгеріссіз қалады да, тек компилятордың мәліметке көзқарасы өзгереді.

Операция форматы:

```
reinterpret_cast <тип> (өрнек)
```

Операцияның нәтижесі көрсетілген типке ие болады, ол сілтеме, нұсқауыш, бүтін немесе нақты тип болуы мүмкін.

Мысал:

```
char *p = reinterpret_cast <char*>(malloc(100));
long l = reinterpret_cast <long>(p);
```

static_cast және **reinterpret_cast** операцияларының арасындағы айырмашылық **static_cast** қолданылған жағдайда компилятордың минималды түрде тексеру жүргізуіне, ал программалаушының **reinterpret_cast** көмегімен қауіпті түрлендірулерді белгілеуіне мүмкіндік береді. Түрлендірудің нәтижесі программалаушы жауапкершілігіне жүктеледі.

Типті динамикалық түрде анықтау

Программаның орындалу барысында (RTTI) типті анықтау механизмі ағымдағы уақыт мезетінде нұсқауыш қандай типке сілтеме жасап тұрғанын анықтауға және объектілердің типтерін салыстыруға мүмкіндік береді. RTTI механизмін пайдалану үшін тіл стандартына **typeid** операциясы және **type_info**[1] класы енгізілген.

typeid операциясының форматы:

```
typeid (тип)
typeid (өрнек)
```

Операция параметр ретінде типтің атауын немесе өрнекті қабылдап, тип туралы ақпаратты сақтайтын **type_info** класының объектісіне сілтеме қайтарады. Егер операция операндтың типін анықтай алмаса, **bad_typeid** аластамасы туындайды.

Операнд полиморфты типке нұсқауыш немесе сілтеме болған кезде, нәтиже тип туралы динамикалық ақпарат түрінде болады (яғни, **type_info** объектісінде дәл осы уақытта нұсқауыш сілтеме жасап тұрған объектінің типі туралы ақпарат бар).

Егер операнд полиморфты емес типке сілтеме түріндегі өрнек болса, онда **type_info** объектісінде өрнек сілтеме жасайтын объектінің типі туралы емес, өрнектің типі туралы мәлімет сақталады.

typeid операциясын мәліметтердің негізгі типтерімен де, туынды типтерімен де пайдалануға болады.

type_info класы **<typeinfo>** тақырыптық файлында төмендегідей түрде сипатталған:

```
class type_info{
   public:
      virtual ~type_info();
      bool operator==(const type_info& rhs) const;
      bool operator!=(const type_info& rhs) const;
      bool before(const type_info& rhs) const;
      const char* name() const;
   private:
```

[1]Компиляторлардың ескі нұсқаларында бұл класс **Type_info** немесе **typeinfo** деп аталуы мүмкін.

```
        type_info(const type_info& rhs);
        type_info& operator=(const type_info& rhs);
};
```

Мысалдағы **name** тәсілі **type_info** типті объектімен сипатталатын типтің атауын көрсететін жолға нұсқауышты қайтарады.

Виртуалды деструктор **type_info** класын полиморфты етеді.

Көшіру конструкторы және меншіктеу операциясы кездейсоқ көшіру және класс объектілерін меншіктеу мүмкіндігін жою үшін **private** түрінде жарияланған.

== және **!=** операциялары екі объектіні теңдікке және теңсіздікке, ал **before** функциясы екі типтің атауларын әріп бойынша салыстыруға мүмкіндік береді. Салыстыру үшін келесі түрдегі конструкция қолданылады:

```
typeid(T1).before(typeid(T2))
```

Егер **T1** типтің атауы **T2** атауынан лексикографикалық түрде ертерек орналасқан болса, нәтиже ақиқат болады.

Объектінің типі туралы нақты мәлімет программаның орындалу барысында, мысалы, программалаушы қандай да бір кітапханалық базалық кластың функционалдылығын туынды класс көмегімен кеңейткен кезде базалық класқа виртуалды функцияларды енгізу мәнсіз немесе мүмкін емес болғанда қажет болуы мүмкін. Мысалы:

```
#include <typeinfo.h>
class B{
   public: virtual ~B(){};
};
class C: public B{
   public: virtual void some_method(){ ... };
};
void demo(B* p){
if (typeid(*p) == typeid(C))
   dynamic_cast<C*>(p)->some_method();
}
int main(){
   C* c = new C; demo(c);
   return 0;
}
```

Тип туралы ақпаратты диагностикалық мақсаттарда да қолданған пайдалы:

```
void print_type(some_obj *p){
   cout << typeid(*p).name();
}
```

typeid операциясы виртуалды функциялардың орнына және объектінің типін компиляция кезеңінде анықтауға болатын жағдайларда қолданылмауы тиіс.

9-ТАРАУ

Программалау бойынша ұсыныстар

Кітаптың осы бөлімінде C++ тілінің объектіге бағытталған мүмкіндіктері қарастырылған болатын, енді программаны жазу барысында туындайтын мәселелерге тоқталатын уақыт жетті, олардың қатарында: әртүрлі жағдайларда тілдің қандай құралдарын таңдаған дұрыс, программа тиімді әрі сүйемелдеуге жеңіл болу үшін қандай мәселелерге ерекше көңіл бөлу қажет және неден аулақ жүру керек.

Әрқашанда бірмәнді ұсыныстар беру мүмкін емес – тек C++ тілінің жұмыс істеу механизмін түсіну арқылы оны сауатты түрде қолдануға болады.

Кез келген программалық өнімді құру жобалау үрдісінен басталады, бұл оның барысында туындайтын мәселелердің бірі – программаның объектіге бағытталған болуының міндетті немесе міндетті емес екендігін анықтау. Объектіге бағытталған программалаудың қажеттілігі жоқ есептер үшін оны қолдану тек программаның көлемін арттырады және жазу күрделілігін жоғарылатады. Егер мәселенің қойылымын талдау барысында кластар иерархиясын қолданудың қажеттілігі жоқ екендігі анықталса, көбінесе құрылымдық тәсілді пайдаланып программалау жеткілікті болып табылады. Б. Страуструптың тұжырымы бойынша, «анықталған типтегі бір объектіден артық қажет болмайтын барлық жерлерде, модульдердің көмегімен мәліметтер жасырылатын программалау стилін қолдану жеткілікті».

Екі тәсілді бір жобада арластырып қолданбаған жөн, себебі дайын өнімде әрі құрылымдық, әрі объектіге бағытталған программа құру принциптерінің кемшіліктері табылуы мүмкін.

Программаларды жобалау технологиясына арналып көптеген әдебиеттер жазылған және осы мәселені қарастыру бұл кітаптың міндеттеріне кірмейді. Түбегейлі программалық жобаларды қолға алғысы келетіндер үшін алдымен классикаға айналған Ф. Брукстың [6], Г. Бучтың [7] және Б. Страуструптың [17] кітаптарымен танысу керек. Осыдан кейін ғана программаларды құрудың келесі кезеңінде – программалық кодты жазу үрдісінде көңіл аударылуы тиіс мәселелер қарастырылады.

Класты, яғни жаңа мәліметтер типін құру барысында оның интерфейсін – класты қолданатын программалаушылар қол жеткізе алатын жұмыс істеу құралдарын – *жақсылап ойластыру керек.* Жақсы жобаланған кластың интерфейсі бір қарағаннан-ақ интуитивтік түрде түсінікті, қарама-қайшылықсыз және көрнекі болуы тиіс. Көбінесе оның құрамында тек әдістер қарастырылып, мәліметтер өрістері болмауы тиіс. Мәліметтер өрістері жасырын (**`private`**) түрде берілуі керек. Осы арқылы кейіннен кластың жүзеге асырылуын оның интерфейсіне өзгеріс енгізбестен түрлендіруге, сонымен

қатар класс өрістерінің қолжетімділігін қолданушыға ұсынылатын әдістер жиынтығының көмегімен реттеуге мүмкіндік туады.

Кластың барлық жасырын өрістері үшін **get/set** *типті әдістерді анықтамаған жөн* – бұл айтарлықтай күрделі жолмен оларды пайдалануға жол ашып берумен пара-пар. Класс өрістері оның интерфейсінде әдістер көмегімен бейнеленген класс қасиеттерін жүзеге асыру үшін ғана енгізілетінін естен шығармаған дұрыс. Әрине, егер кластың қасиеті осылайша жүзеге асырылатын болса, онда қандай да бір өрістің мәнін әдістің көмегімен алуға немесе орнатуға болады (осылайша, код өзгеріссіз қалады да, ал бірақ оның мағынасы мүлде басқаша болады.

Класс интерфейсін қажеттіліксіз, «керек болып қалар жағдай» үшін кеңейту керек емес, өйткені әдістер санының артуы қолданушының класты түсінуінде қиындықтар тудырады[1]. Мінсіз жағдайда интерфейс толық болуы тиіс, яғни класқа қатысты кез келген саналы әрекеттерді орындауға мүмкіндік беруі тиіс және *интерфейс минималды болуы керек,* яғни оның құрамында қайталаулар болмай, әдістер мүмкіндіктерінің қиылысуы да кездеспеуі тиіс.

Әдістер ретінде тек кластың қасиеттерін жүзеге асыратын әрекеттерді анықтаған жөн. Егер қандай да бір әрекетті кластың жасырын өрістерін пайдаланбай-ақ жүзеге асыруға болатын болса, оны әдіс ретінде сипаттаудың қажеттілігі жоқ; оны класпен ортақ атаулар кеңістігіне орналастырып, қарапайым функция ретінде сипаттаған дұрыс. Егер функция класс қасиеті болып табылмайтын әрекетті орындап, бірақ оның жасырын өрістерін пайдалануы қажет болса, онда оны достас функция ретінде жариялау керек. Дегенмен, жалпы жағдайда *достас функцияларды және кластарды қолданбаған жөн,* өйткені ОБП-дың негізгі идеясы инкапсуляцияланған кластар арасында байланыстарды ең төменгі деңгейде ұстау (минималдау) болып табылады.

Программаның өнімділігін арттыру үшін барынша жиі шақырылатын әдістерді кіріктірілген (**inline – құрамдас**) деп қарастыруға болады. Негізінен, бұлар қысқа әдістерге, яғни тұлғасы осы әдістерді шақыру үшін генерацияланатын код көлемінен кіші болатын әдістерге қатысты болып саналады. Осындай жолмен шақыруларды болдырмау арқылы программа жылдамдығының артуымен қатар, компилятордың толық тиімділік орнатуына мүмкіндік туады. Алайда, **inline** директивасы компилятор үшін ұсыну сипатында болатынын ұмытпау керек, яғни компилятор оны ескермей кететін кезі де болып жатады, мысалы, егер әдіс құрамында күрделі циклдер болса немесе ол виртуалды ретінде жарияланған жағдайда оны орындамауы да мүмкін. Сонымен қатар, егер программа немесе компилятор әдіске немесе функцияға нұсқауыш қолданса, олардың кіріктірілмейтін көшірмелері қосымша генерацияланады. Назар аударатын тағы да бір мәселе – көптеген жөндеп түзету істерін орындау кездерінде кіріктірілген функциялармен жұмыс істеу қиындықтары туындайды.

[1]Қолданушы ретінде класты пайдаланатын программалаушыны түсіну керек.

Конструкторлар мен деструкторларды кіріктірілген түрде құру ұсынылмайды, өйткені олардың құрамында компилятор кірістіретін қосымша кодтар болады және олардың көлемі өте үлкен де болуы мүмкін (мысалы, туынды класс конструкторында барлық базалық және кіріктірілген кластардың конструкторлары шақырылуы тиіс).

Кластың асыра жүктелген операцияларының жалпыға ортақ интуитивті түсінікті мағынасы болу керек (мысалы, + операциясына қосудан немесе мәнді арттырудан басқа амалдарды орындатпауға тырысу керек). Негізінен, операцияларды асыра жүктеу программалаушы құратын арифметикалық типтер үшін қолданылады, ал басқа жағдайларда класс объектілерімен қандай да бір әрекеттерді жүзеге асыру үшін әдістерді қолданған жөн, өйткені оларға мағыналы атаулар беруге болады.

Егер қандай да бір операция асыра жүктелген болса, мүмкіндігінше, лдан басқа осыған ұқсас операцияларды да, мысалы **+**, **+=** және **++** операцияларын да асыра жүктеген жөн (компилятор оны автоматты түрде орындамайды). Мұндайда операциялар семантикасы олардың өзіне ұқсас стандартты операция семантикаларымен бірдей болуы тиіс.

Кез келген кластың негізі *оның конструкторлары мен деструкторлары* болып табылады. Нақты берілген конструкторлар, көшірме конструкторы және меншіктеу операциясы болмаған кездерде оларды компилятор *автоматты* түрде құрады.

Келісім бойынша құрылатын көшірме конструкторы да, меншіктеу операциясы да бастапқы аймақтан қабылдаушы аймаққа элементтер бойынша көшіруді орындайды. Егер объект құрамында нұсқауыштар бар болса, онда көшіруден кейін әртүрлі екі объектінің сәйкес нұсқауыштары бір ғана жады аймағына сілтеме жасайды. Объектілердің біріншісін жойған жағдайда бұл жады аймағы бостылады, ал екінші объектіні жою кезінде сол жады аймағын қайталап босатуға талпыну программаның анықталмаған әрекеттер орындауына алып келеді Сондықтан, өріс-нұсқауыштары бар кластар үшін объектінің динамикалық өрістері үшін жадыны бөлетін көшірме конструкторын және меншіктеу операциясын әрқашанда нақты түрде тікелей анықтап алған дұрыс.

Сонымен қатар, *объект конструкторында бөлінген динамикалық жады оның деструкторында босатылуы тиіс.* Бұл талаптың орындалмауы жадының көбірек жұмсалуына әкеліп соғады. Нөлдік нұсқауышты жою қауіпсіз болып саналады (мұндайда ешқандай әрекет орындалмайды), сондықтан егер конструкторлар, көшіру конструкторлары және меншіктеу операциялары дұрыс жазылған болса, онда кез келген нұсқауыш бөлінген жады аймағына сілтеме жасайды немесе ол нөлге тең болады да, оған тексерусіз **delete** операциясын қолдануға болады.

Көшірме конструторы және меншіктеу операциясы арасындағы айырмашылық – соңғысы қабылдаушы объект бар болған жағдайда да жұмыс істейді, сондықтан мұнда динамикалық жадыны бөлудің алдында бұрын

қолданысқа берілген аймақты босату керек. Осының салдарынан, өрісі-нұсқауыштары бар кластар үшін меншіктеу операциясын жүзеге асыру кезінде өзіндік меншіктеуге тексеру жүргізу керек және мұндайда объектіні өзгеріссіз қалдыру қажет. Сонымен қатар, меншіктеу операциясы тұрақтыға сілтеме қайтару керектігін есте сақтаған жөн. Сонымен, меншіктеу операциясының анықтамасы келесі түрде болуы тиіс:

```
class X{
   const X & operator=(const X & r);
   ...
};
const X & X::operator=(const X & r){
   if(this != &r){            // Көшіру
   }
   return *this;
}
```

Конструкторларда өрістерге бастапқы мәндер беру үшін меншіктеу операциясын емес, инициалдауды қолдану ұсынылады:

```
class X{
      string s;
   public:
      X(const char * str);
};
// Меншіктеуі бар конструктор нұсқасы:
X::X(const char * str){ s = str; }
// Инизиалдау қолданылған конструктор нұсқасы:
X::X(const char * str): s(str){}
```

Инициалдау басқаларына қарағанда, әмбебап әрекет болып табылады, өйткені оның меншіктеуді пайдалануға болмайтын жағдайларда қолданылуы мүмкін (мысалы, тұрақтылық өрістерге немесе сілтемелерге мәндер беру кезінде). Сонымен қатар, ол тиімдірек жұмыс атқарады, себебі C++ тілінде объектіні құру оның өрістерін келісім бойынша конструктордың инициалдауынан басталады, содан кейін барып шақырылатын конструктор орындалады. Осылайша, қарастырылған мысалдың бірінші нұсқасы үшін келісім бойынша **string** класының конструкторы шақырылады да, тек содан кейін барып меншіктеу орындалады, ал екінші нұсқа керекті конструкторды бірден шақыруға мүмкіндік береді. Иерархия құрайтын күрделі кластар үшін келтірілген нұсқалар арасындағы айырмашылық айтарлықтай үлкен болады.

Өрістердің инициалдануы олардың тізімдегі кездесу реттілігі бойынша емес, жариялану реттілігі бойынша жүзеге асырылатынын да ескеру керек. Сондықтан, мүмкін болатын қателіктер санын азайту үшін конструктордың инициалдау тізіміндегі өрістерді жазу реті олардың класта жариялану реттілігіне сәйкес келуі тиіс.

Статикалық өрістер конструкторда инициалданбауы тиіс, себебі оларға бастапқы мәнді әрбір класс үшін тек бір рет меншіктеу керек, ал конструктор кластың әрбір объектісі үшін орындалады. Статикалық өрістер ауқымды анықталу аймағында (кез келген функциядан тыс) инициалданады.

Көшірме конструкторлары да өрістердің инициалдану тізімін қолдануы керек, өйткені базалық кластар және қабаттастырылған объектілер үшін келісім бойынша анықталған конструкторлар шақырылады:

```
class X{
   public:
      X(void);
      // Көшірме конструкторы:
      X(const X & r);
};
class Y: public X{
      string s;         // Қабаттастырылған объект
   public:
      // Көшіру конструкторы:
      Y(const Y & r): X(r), s(r.s){}
};
```

Меншіктеу операциясы мұраланбайды, сондықтан ол туынды кластарда анықталуы тиіс. Мұндайда оның ішінен нақты түрде базалық кластың сәйкес операциясын шақырған жөн (220 б. қараңыз).

Кластарды мұралау программалаушыға мол мүмкіндіктерді – виртуалды және виртуалды емес базалық кластар, ашық, қорғалған және жабық мұралау, виртуалды әдістер мен басқа да мүмкіндіктерді орындауды қамтамасыз етеді. Нақты жобаның мақсаттарына барынша сәйкес келетін мүмкіндіктерді таңдау олардың жұмыс істеу және өзара әрекеттесу механизмдерін білуге негізделген.

X класынан Y класын ашық түрде мұралау Y класы X класының бір түрі екендігін, яғни нақтыланған дербес концепциясы *болып табылатынын білдіреді*. Y класына қарағанда X базалық класы айтарлықтай жалпы ұғым болып табылады[1]. X класын қолдануға болатын барлық жерлерде Y класын да қолдануға болады, бірақ бұл кері бағытта орындалмайды (базалық класқа сілтемелердің орнына туынды кластардың кез келгеніне сілтеме беруге болатынын еске түсіріңдер). Программаның орындалуы кезінде кластар иерархиясы және туынды кластардан базалық кластар объектілеріне хабарламалар жіберу болмайтындығын есте сақтаған жөн, яғни тек компиляция сатысында иерархия негізінде өрістері қалыптасатын кластардың нақты объектілері бар болады.

Құрамында барлық туынды кластары бар, бірақ базалық класс деңгейінде жүзеге асырыла алмайтын барлық әдістер виртуалды болып табылады. Мысалы, иерархияның барлық объектілері өздері туралы ақпаратты шығара

[1]Мысалы, әрбір программалаушы – ол адам, бірақ әрбір адам программалаушы болмайды.

алатындай дәрежеде болуы тиіс. Ол туынды кластардың әртүрлі өрістерінде сақталатын болғандықтан, бұл функцияны базалық класта жүзеге асыруға болмайды. Оны барлық кластарда бірдей атпен атау және виртуалды функция ретінде сипаттау керек, осы арқылы базалық кластың басқа әдістері оны өздері жұмыс істейтін объектінің нақты типіне байланысты шақыра алады. Осы себептерге байланысты *деструкторлар виртуалды болып жарияланады.*

Виртуалды әдістерді қайта анықтау кезінде аргументтің келісім бойынша мұраланған мәнін өзгертуге болмайды, себебі C++ ережелері бойынша ол әдісті шақырған объектінің нақты типі арқылы емес, нұсқауыштың типі бойынша анықталады:

```
#include <iostream.h>
class X{
   public:
      virtual void fun(int a = 0){cout << a;}
};
class Y: public X{
   public:
      virtual void fun(int a = 1) {cout << a;}
};
int main(){
   X *px = new X;
   px->fun();        // 0 шығарылады
   X *py = new Y;
   py->fun();        // 0 шығарылады
}
```

Виртуалды емес әдістерді туынды кластарда қайталап анықтау қажет емес, себебі туынды кластар базалық кластардың қасиеттерін мұралап алуы керек. Басқаша айтқанда, виртуалды емес әдіс специализацияға қатысты инвариантты болып саналады, яғни туынды кластың қалай нақтыланатынына тәуелсіз базалық кластан мұралануы тиіс қасиеттерді бойында сақтайды. Туынды кластың специализациялануы жаңа әдістерді қосу жолымен және бұрыннан бар виртуалды әдістерді қайта анықтау арқылы орындалады.

Мұралаудың негізгі артықшылығы – базалық класс деңгейінде әмбебап код жазып, соның көмегімен виртуалды әдістер арқылы туынды кластардың объектілерімен жұмыс жасауға болады.

Мұралауға альтернативті түрде бір кластың екінші класты пайдалануы үшін енгізілген әдіс *қабаттастыру* болып табылады, мұнда бір класс басқа бір кластың өрісі болып саналады:

```
class X{ ... };
class Y{
   X x;
   ...
};
```

Қабаттастыру кластардың «Y құрамында X» немесе «Y X арқылы жүзеге асырылады» деген сияқты қатынастарын сипаттайды. Мұралаудың орнына қабаттастыруды қолданудың қажеттілігін «Y құрамында X класының бірнеше объектісі болуы мүмкін бе» деген сұрақты қоя отырып анықтауға болады. Мысалы, егер ұшақты модельдеуге арналған класты сипаттау қажет болса, оның құрамында «қозғалтқыш» типті өрісті сипаттаған жөн болады: ұшақ құрамында қозғалтқыш бар, бірақ ол ұшақтың бір түрі болып саналмайды.

Y класы X класының бір түрі болады деп айта алмайтын, бірақ Y класы X класы функционалдылығының бір бөлігін қолданатын кезде «... арқылы жүзеге асырылады» қатынасы мұралаудың орнына пайдаланылады. Мұралау орнына қабаттастыруды қолдану дұрысырақ болып табылады.

Кластар арасында логикалық өзара байланыс болмай, тек бір класс кодының бөлігін басқа бір класта қолдану керек болған жағдайда жабық мұралауды (**`private`** қол жеткізу кілті бар) қолданған тиімді болуы мүмкін. Бұл тәсіл туынды класта базалық кластың қорғалған элементтеріне қол жеткізу және оның виртуалды әдістердің орнын басу керек болған кезде қолданылады.

Класс шаблондары іс-әрекеттері объектілер типіне байланысты болмайтын класс топтарын құру үшін қолданылады. Шаблондарды мұқият болып, әрбір тип үшін өзіндік шаблон көшірмесі құрылатындықтан, код көлемі артуы мүмкін екендігін есте сақтай отырып қолдану керек. Кодтың көлемін азайту үшін объект типі туралы ақпаратты пайдаланбайтын барлық әдістерді базалық класқа шығару және одан шаблондық әдісті барлық басқа әдістермен бірге мұралау ұсынылады. Осы арқылы мәтіндері шаблон параметрлеріне тәуелді болмайтын әдістердің қайталануынан сақтанамыз.

Көбінесе *ерекше жағдайлар* қателік туралы хабар берудің басқа тәсілі жоқ болған кезде (мысалы, қателік конструкторда немесе асыра жүктелген операцияда туындаса) пайдаланылады, сонымен қатар олар түзетілмейтін немесе өте сирек кездесетін күтілмеген қателіктер туындағанда қолданылады. Ерекше жағдайларды өңдеу программаның өнімділігін айтарлықтай төмендетеді, автордың көзқарасы бойынша, ол программаның оқылуын да нашарлатады, сондықтан ерекше жағдайларды қажет болмаған кезде қолданбаған жөн.

Осы бөлімде қарастырылған сұрақтарды толығырақ меңгеру үшін А. Голубтың [9] және С. Мейерстің [13] кітаптарын қолдану ұсынылады.

II бөлімге арналған жаттығулар

Кластар

Бұл тақырып бойынша лабораториялық жұмыстар тапсырмаларын жүзеге асыратын программалар практикумның [12] бірінші семинарында келтірілген.

1-нұсқа

Стекті жүзеге асыратын класты сипаттау керек. Осы класты темір жолдағы Т-түріндегі сұрыптаушы түйінді модельдеу үшін қолданатын программа жазыңыз. Программа вагондардың екі типінен тұратын пойыз құрамын екі бағытқа бөлуі тиіс (әрбір бағытқа бір типті вагондардан тұратын пойыз құрамы қалыптастырылады). Пойыз құрамының файлдан және пернетақта арқылы қалыптастырылу мүмкіндігін қарастырыңыз.

2-нұсқа

Жаңа элементтерді қосу, бар элементтерді жою, кілті бойынша элементті іздеу, сонымен қатар барлық элементтерге тізбекті түрде қол жеткізу мүмкіндіктері бар бинарлы бұтақты жүзеге асыратын класты сипаттау керек.

Осы класты ағылшын-орыс сөздігін бейнелеу үшін қолданатын программа жазыңыз. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс. Сөздіктің файл және пернетақта арқылы қалыптастырылу мүмкіндігін қарастырыңыз.

3-нұсқа

Жазықтықтағы геометриялық фигураларды: шеңберді, шаршыны, тік төртбұрышты сипаттауға арналған кластар жүйесін құру керек. Объектілерді құруға, жазықтықта олардың орнын ауыстыруға, өлшемдерін өзгертуге және берілген бұрышқа айналдыруға арналған әдістерді қарастырыңыз.

Осы кластармен жұмыс істеуді сипаттайтын программа жазу қажет. Программада кластардың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

4-нұсқа

Ұйымның пошталық адресі туралы ақпаратты қамтитын кластың сипаттамасын құру қажет. Адрестің құрамдас бөліктерін жеке-жеке өзгерту, осы класс объектілерін құру және жою мүмкіндіктерін қарастырыңыз.

Осы класпен жұмыс істеуді сипаттайтын программа жазу қажет. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

5-нұсқа

Кешендік (комплекстік) сандарды бейнелеуге арналған кластың сипаттамасын құру қажет. Комплексті сандарды қосу, азайту және көбейту операцияларының орындалуын қамтамасыз етіңдер.

Осы класпен жұмыс істеуді сипаттайтын программа жазу қажет. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

6-нұсқа

Үшөлшемді кеңістікте ұшының координаталары арқылы берілетін объект-вектолар үшін кластың сипаттамасын құру қажет. Векторларды қосу және азайту операцияларының көмегімен жаңа вектор (қосындысының немесе айырымының векторын) алуды, екі вектордың скаляр көбейтіндісін, вектор ұзындығын, векторлар арасындағы бұрыштың косинусын есептеу операцияларын қамтамасыз етіңдер.

Осы класпен жұмыс істеуді сипаттайтын программа жазу қажет. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

7-нұсқа

Қабырғалары координата өстеріне параллель болатын тіктөртбұрыштар класының сипаттамасын құру қажет. Тіктөртбұрыштардың жазықтықтағы орындарын ауыстыру, өлшемдерін өзгерту, берілген екі тіктөртбұрышты қамтитын ең кіші тіктөртбұрышты және екі тіктөртбұрыштың ортақ бөліктері (қиылысуы) болып табылатын тіктөртбұрышты құру мүмкіндігін қарастырыңыз.

Осы класпен жұмыс істеуді сипаттайтын программа жазу қажет. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

8-нұсқа

Бүтін сандардың (векторлардың) бірөлшемді жиымдарын анықтауға арналған кластың сипаттамасын құру қажет. Жиымның шекараларынан шығуды бақылай отырып, оның жеке элементтерін пайдалану, объектіні құру кезінде индекстердің кездейсоқ шекараларын тағайындау, индекстерінің шекаралары бірдей болатын жиымдарды элемент бойынша қосу және азайту операцияларын орындау, жиымның барлық элементтерін скаляр шамаға көбейту және бөлу, берілген индексі бойынша жиымның элементін немесе жиымды толығымен экранға шығару мүмкіндіктерін қарастырыңыз.

Осы класпен жұмыс істеуді сипаттайтын программа жазу қажет. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

9-нұсқа

Бекітілген ұзындығы бар тіркестердің бірөлшемді жиымдарын анықтауға арналған кластың сипаттамасын құру қажет. Жиым шекараларына шығуды қадағалау, жиымның жеке тіркестерін индекстері арқылы пайдалану, екі жиымды элемент бойынша тізбектей отырып біріктіру арқылы жаңа жиымды қалыптастыру, қайталанатын элементтерді жоя отырып, екі жиымды біріктіру, берілген индексі бойынша жиымның элементін немесе жиымды толығымен экранға шығару мүмкіндігін қарастырыңыз.

Осы класпен жұмыс істеуді сипаттайтын программа жазу қажет. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

10-нұсқа

Көпмүшеліктің дәрежесі және коэффициенттері жиым арқылы берілетін бір айнымалыға тәуелді көпмүшелер класының сипаттамасын құру қажет. Берілген аргумент үшін көпмүшеліктің мәнін есептеу, көпмүшелерді қосу, азайту және көбейту операцияларының көмегімен жаңа объект-көпмүшені алу, көпмүшенің сипаттамасын экранға шығару әдістерін қарастырыңыз.

Осы класпен жұмыс істеуді сипаттайтын программа жазу қажет. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

11-нұсқа

Тіркестердің бірөлшемді жиымдары класының сипаттамасын құру керек, әрбір тіркес ұзындығы және өзіне бөлінген жадыға нұсқауыш арқылы беріледі. Жиымдар шекараларынан шығуды қадағалау, жиымның жеке тіркестерін индекстері арқылы пайдалану, екі жиымды элемент бойынша тізбектей біріктіру арқылы жаңа жиымды қалыптастыру, қайталанатын элементтерді жоя отырып, екі жиымды біріктіру, берілген индексі бойынша жиымның элементін немесе жиымды толығымен экранға шығару мүмкіндігін қарастырыңыз.

Осы класпен жұмыс істеуді сипаттайтын программа жазу қажет. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

12-нұсқа

Кездейсоқ өлшемдегі матрицаны бейнелейтін және оның жолдары мен бағаналарының санын өзгерту, кез келген өлшемдегі ішкі матрицаны немесе матрицаны толығымен экранға шығару мүмкіндіктерін қамтамасыз ететін кластың сипаттамасын құру қажет.

Осы класпен жұмыс істеуді сипаттайтын программаны жазыңыз. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

13-нұсқа

Тіркестермен тиімді жұмыс істеуге арналған, тіркестерді форматтауға және салыстыруға, тіркестерде сандық мәндерді сақтауға және оларды шығарып алуға мүмкіндік беретін класты жазыңыз. Ол үшін мыналарды жүзеге асыру қажет:

- □ асыра жүктелген меншіктеу операторлары және конкатенация;
- □ типтерді келтіру және салыстыру операциялары;
- □ кез келген типті санға түрлендіру;
- □ тіркесті форматты түрде шығару.

Осы класпен жұмыс істеуді сипаттайтын программаны жазыңыз. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

14-нұсқа

«Үй кітапханасы» класын сипаттау керек. Кітаптардың кездейсоқ санымен жұмыс істеу, белгілі бір сипаты бойынша кітаптарды іздеу (мысалы, авторы немесе шығарылған жылы бойынша), кітпаханаға жаңа кітаптарды

енгізу, одан кітаптарды өшіру, кітаптарды әртүрлі өрістер бойынша сұрыптау мүмкіндіктерін қарастырыңыз.

Осы класпен жұмыс істеуді сипаттайтын программаны жазыңыз. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

15-нұсқа

«Жазба кітапшасы» класын сипаттау керек. Кездейсоқ жазбалар санымен жұмыс істеу, белгілі бір сипаты бойынша жазбаны іздеу (мысалы, тегі, туған күні немесе телефон нөмірі бойынша), жазбаларды қосу және өшіру, оларды әртүрлі өрістері бойынша сұрыптау мүмкіндіктерін қарастырыңыз.

Осы класпен жұмыс істеуді сипаттайтын программаны жазыңыз. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

16-нұсқа

«Студенттік топ» класын сипаттау керек. Студенттердің айнымалы санымен жұмыс істеу, белгілі бір сипаты бойынша студентті іздеу (мысалы, тегі, туған күні немесе телефон нөмірі бойынша), жазбаларды қосу және өшіру, оларды әртүрлі өрістері бойынша сұрыптау мүмкіндіктерін қарастырыңыз.

Осы класпен жұмыс істеуді сипаттайтын программаны жазыңыз. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

17-нұсқа

«Нақты матрица» мәліметтер типін пайдаланатын және олармен жұмыс істеуді іске асыратын класты сипаттау керек. Класс матрицалармен келесі операциялардың орындалуын жүзеге асыруы тиіс:

- қосу, азайту, көбейту және бөлу (+, -, *, /) (басқа бір матрицаға көбейту мен бөлу операцияларымен қатар, сандарға да қатысты операциялар орындалуы мүмкін);
- аралас меншіктеу операциялары (+=, -=, *=, /=);
- теңдікке/теңсіздікке салыстыру операциялары;
- кері және транспонирленген матрицаны есептеу операциялары және дәрежелеу операциясы;
- детерминантты және норманы есептеу әдістері;
- матрицаның типін (квадраттық, диоганалдық, нөлдік, бірлік, симметриялы, жоғарғы үшбұрышты, төменгі үшбұрышты) тексеруді жүзеге асыратын әдістер;
- стандартты ағымдарға енгізу/шығару операциялары (305 б.)

Осы класпен жұмыс істеуді сипаттайтын программаны жазыңыз. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

18-нұсқа

Негізгі операцияларды – элементті қосу және өшіруді, жиындардың қиылысуын, біріктірілуін және айырмасын орындауға мүмкіндік беретін «жиын» класын сипаттау керек.

Осы класпен жұмыс істеуді сипаттайтын программа жазыңыз. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

19-нұсқа

Стекті жүзеге асыратын класты сипаттау керек. Осы класты лабиринттен шығатын жолды табу үшін қолданатын программа жазыңыз.

Лабиринт шаршылардан тұратын матрица түрінде беріледі. Әрбір шаршы ашық немесе жабық болады. Жабық шаршыға кіруге тыйым салынған. Егер шаршы ашық болса, онда оған қабырғасынан кіруге болады, бірақ бұрышынан кіру мүмкін емес. Әрбір шаршы өзінің матрицадағы координаталары арқылы анықталады. Программа шығатын жолды тапқаннан кейін осы жолды шаршылардың координаталары түрінде экранға шығарады.

20-нұсқа

«Пәндік нұсқауыш» класын сипаттау керек. Нұсқауыштың әрбір компоненті сөзді және осы сөз кездесетін беттердің нөмірлерін сақтайды. Бір сөзге қатысты беттер нөмірлерінің саны бір мен онның аралығында болады. Нұсқауышты пернетақта арқылы және файлдан енгізіп қалыптастыру, нұсқауышты шығару, берілген сөз үшін бет нөмірлерін шығару және нұсқауыштан элементті жою мүмкіндіктерін қарастырыңыз.

Осы класпен жұмыс істеуді сипаттайтын программаны жазыңыз. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

Мұралау

Бұл тақырып бойынша лабораториялық жұмыстар тапсырмаларын жүзеге асыратын программалар практикумның [12] екінші семинарында келтірілген.

1-нұсқа

Құрамында **`Open`**, **`Close`**, **`Seek`**, **`Read`**, **`Write`**, **`GetPosition`** және **`GetLength`** сияқты файлдармен жұмыс істеу функцияларын инкапсуляциялайтын **`CFile`** класын құру қажет. Осы кластың негізінде **`CmyDataFile`** туынды класын құрыңыз, бұл класс құрамында қандай да бір анықталған **`MyData`** типіндегі мәліметтер, сонымен қатар осы файлға қатынасуды жеңілдететін тақырыбы бар файл болып табылады.

Осы класпен жұмыс істеуді сипаттайтын программа жазу керек. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

2-нұсқа

`CPoint` атты нүкте класын құру керек. Осының негізінде **`CcoloredPoint`** және **`CLine`** кластарын құрыңыз. **`CLine`** класының негізінде **`CColoredLine`** және **`CPolyLine`** – көпбұрыш кластарын құру қажет. Барлық кластарда барлық координаталар мәндерін тағайындауға және қабылдауға, сонымен қатар түсті өзгертуге және ағымдағы түсті қабылдауға арналған әдістер болуы тиіс.

Динамикалық жадыда осы кластар объектілерінің тізімін қолданатын программа жазу керек.

3-нұсқа

`CVehicle` абстрактылы класын құру қажет. Оның негізінде **`CPlane`**, **`CCar`** және **`CShip`** кластарын жүзеге асырыңыз. Кластардың қатынас көліктерінің (қозғалыс құралдарының) координаталарын, параметрлерін (бағасы, жылдамдығы, шығарылған жылы) тағайындау және қабылдау мүмкіндіктері болуы тиіс. Ұшақ үшін биіктік, ал ұшақ пен кеме үшін жолаушылар саны анықталуы тиіс. Кеме үшін оны тізімге алу портын тағайындау керек.

Динамикалық жадыда осы кластар объектілерінің тізімін қолданатын программа жазу керек. Программада кластың барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

4-нұсқа

1. «Элемент» базалық класын сипаттау керек.

Оның өрістері:

- элемент атауы (символдар тіркесіне нұсқауыш);
- элементтің кіріс нүктелерінің саны;
- элемент шығыс нүктелерінің саны.

Әдістері:

- класс конструкторы;
- класс деструкторы;
- элемент атын беретін әдіс.

2. «Элемент» класының негізінде «комбинациялық (аралас) элемент» туынды класын сипаттау қажет, туынды класс бірнеше кірісі және бір шығысы болуы мүмкін комбинациялық (аралас) элементті (екілік тетікті) бейнелейді.

Өрістері:

- шығыстардың мәндері сақталған өрістерді динамикалық түрде орналастыру үшін қолданылатын нұсқауыш.

Әдістері:

- конструктор;
- көшірме конструкторы;
- деструктор;
- класс данасының (экземплярының) кірістеріндегі мәндерді беретін әдіс;
- класс данасының (экземплярының) жеке кірісінің қалып-күйін сұрастыруға мүмкіндік беретін әдіс;
- шығыстың мәнін есептейтін (тапсырма нұсқасы бойынша) әдіс;

3. «Элемент» класының негізінде триггерді бейнелейтін «Жады» туынды класын сипаттау керек. Триггерлердің олардың типіне сәйкес келетін кірістері (төмендегі тапсырма нұсқасын қараңыз), тағайындау және түсіру (мәнді

өшіру) кірістері болады. Барлық триггерлер синхронды болып саналады, ал синхрокірістің өзі триггер құрамына қосылмайды.

Өрістері:

□ класс объектісінің кірістер мәндерінің жиымы (статикалық түрде беріледі), жиымда барлық кірістер ескеріледі (басқарушы және ақпараттық);

□ триггердің тікелей шығысындағы оның қалып-күйі;

□ триггердің инверсті шығысындағы қалып-күйі;

Әдістері:

□ конструктор (келісім бойынша класс данасын (экземплярын) нөлдік қалыпқа түсіріп тастайды);

□ көшірме конструкторы;

□ деструктор;

□ класс данасының (экземплярының) кірістеріндегі мәндерді орнататын әдіс;

□ класс данасының (экземплярының) жеке кірісінің қалып-күйін сұрастыруға мүмкіндік беретін әдістер;

□ ағымдағы жағдайға (күйге) және кірістердегі мәндерге байланысты класс данасының (экземплярының) қалып-күйін анықтайтын әдіс (тапсырманың нұсқасы бойынша);

□ класс даналары (экземплярлары) үшін == операциясын қайта анықтайтын әдіс.

4. «Жады» класын енгізілетін класс ретінде қолдана отырып, «Регистр» класын құру керек.

Өрістері:

□ «Түсіру» («Мәнді өшіру») кірісінің қалып-күйі – класс данасы үшін біреу;

□ «Орнату» кірісінің қалып-күйі – класс данасы үшін біреу;

□ нұсқада берілген өлшемдегі «Жады» типті статикалық жиымы;

□ сәйкес кірістерінде «Жады» типті жиым элементтерінің мәндері болатын статикалық жиым(дар).

Әдістер:

□ класс данасының (экземплярының) кірістеріндегі мәнді орнататын әдіс (әдіске параметрлер ретінде мәндер жиымдарына нұсқауыштарды берген жөн);

□ класс данасының жеке шығысы үшін қалып-күйді сұрастыруға мүмкідік беретін әдіс;

□ класс данасының жаңа қалып-күйінің мәнін есептейтін әдіс.

«Элемент», «Комбинациялық» («Аралас») және «Жады» кластарының барлық өрістері **`private`** немесе **`protected`** түйінді сөздері арқылы сипатталуы тиіс.

Тапсырмада кластың міндетті әдістері мен мүшелері берілген. Қажет болған жағдайда қосымша мүшелер мен әдістерді беруге болады, бірақ олар міндеттілерге өзгертпеуі тиіс және берілген кластармен жұмыс істеуде қосымша қолайлылықты қамтамасыз етуі тиіс, мысалы, кіріс/қалып-күйді есептеу функцияларын виртуалды түрде сипаттау керек.

5. Құрылған кластардың функционалдылығын тексеру үшін осы кластарды қолданатын программа жазу керек. Программада құрылған кластардың барлық қасиеттері көрсетілуі тиіс.

Комбинациялық (аралас) элементтің нақты типі, триггердің типі және регистрдің разрядтылығы тапсырманың нұсқасына сәйкес таңдалады.

Нұсқа	Комбинациялық (аралас) элемент	Кірістер саны	Триггер	Регистр разрядтылығы
1	ЖӘНЕ-ЕМЕС	4	RS	8
2	НЕМЕСЕ	5	RST	10
3	МОД2-ЕМЕС	6	D	12
4	ЖӘНЕ	8	T	8
5	НЕМЕСЕ-ЕМЕС	8	V	9
6	ЖӘНЕ	4	RS	10
7	НЕМЕСЕ-ЕМЕС	5	JK	11
8	МОД2	5	D	8
9	ЖӘНЕ	4	T	10
10	НЕМЕСЕ	3	JK	8
11	ЖӘНЕ-ЕМЕС	3	RS	12
12	НЕМЕСЕ-ЕМЕС	4	RST	4
13	МОД2	5	D	10
14	МОД2-ЕМЕС	6	T	10
15	НЕМЕСЕ-ЕМЕС	8	V	10
16	ЖӘНЕ	8	JK	6
17	ЖӘНЕ-ЕМЕС	8	RS	10
18	НЕМЕСЕ	8	T	10
19	МОД2	6	JK	8
20	МОД2-ЕМЕС	5	V	10

5-нұсқа

ТІРКЕС базалық класын сипаттау керек.

Кластың міндетті өрістері:

□ **`char`** типіне нұсқауыш – тіркес символдарын орналастыру үшін динамикалық түрде бөлінген жадының адресін сақтайды.

□ **`int`** типті мән – тіркестің ұзындығын байт есебімен сақтайды.

Міндетті әдістер келесі әрекеттерді орындау керек:

□ параметрлерсіз конструктор;

□ параметр ретінде С-тіркесті қабылдайтын конструктор (нөлдік байтпен аяқталады);

□ параметр ретінде символды қабылдайтын конструктор;

□ көшірме конструкторы;

□ тіркес ұзындығын алу;

□ тіркесті тазарту (бос тіркес алу);

□ деструктор.

ТІРКЕС класынан туынды ТІРКЕС_ИДЕНТИФИКАТОР класын сипаттаңыз.

Бұл кластың тіркестері С тіліндегі идентификаторларды жазу ережелері бойынша құрастырылады және оларда тек С-идентификаторлардың құрамына

кіре алатын символдар болуы мүмкін. Егер бастапқы мәндер идентификаторды жазу ережелеріне қайшы келсе, онда бос ТІРКЕС_ИДЕНТИФИКАТОР құрылады.

Міндетті түрде болатын әдістері:

□ параметрлері жоқ конструктор;

□ параметр ретінде С-тіркесті қабылдайтын конструктор (нөлдік байтпен аяқталады);

□ параметр ретінде символды қабылдайтын конструктор;

□ көшірме конструкторы;

□ тіркестің барлық символдарын жоғарғы регистрге ауыстыру;

□ тіркестің барлық символдарын төменгі регистрге ауыстыру;

□ символдың тіркеске алғаш енуін іздеу;

□ деструктор.

Келесі операцияларды қайта анықтау қажет:

□ меншіктеу (=);

□ қосу (+) – тіркестер конкатенациясы операциясы;

□ азайту (-) – тіркестен (бірінші операнд) – екінші операнд тіркесіне кіретін барлық символдар жойылады, бұл кезде бос тіркес пайда болуы мүмкін;

□ (>) операциясы – үлкен болуын тексеру. Егер бірінші тіркестің і-ші (і 0 мен n-1 аралығында өзгереді, мұндағы n – екі тіркес ішіндегі қысқа тіркестің ұзындығы) позициясындағы символдың коды екінші тіркестің дәл осы позициясындағы символ кодынан үлкен болса, бірінші тіркес екіншісінен үлкен болып саналады, тіркестердің ұзындықтары бір-бірімен бірдей болмауы мүмкін;

□ (<) операциясы – кіші болуын тексеру. Егер бірінші тіркестің і-ші (і 0 мен n-1 аралығында өзгереді, мұндағы n – екі тіркес ішіндегі қысқа тіркестің ұзындығы) позициясындағы символдың коды екінші тіркестің дәл осы позициясындағы символ кодынан кіші болса, бірінші тіркес екіншісінен кіші болып саналады, тіркестердің ұзындықтары бір-бірімен бірдей болмауы мүмкін.

Программа құрушы қосымша өрістер мен әдістердің кез келген санын (қажеттілігін негіздей отырып) енгізуге құқылы болады.

Келесі әрекеттерді орындайтын тестілік программа жазыңыз:

□ динамикалық түрде базалық класқа нұқауыштар жиымын бөледі (4-6);

□ сұхбат режимінде осы жиымды туынды кластарға нұсқауыштармен толтырады, бұл кезде туынды кластардың даналары (экземплярлары) бастапқы мәндерін беру арқылы динамикалық түрде құрылады;

□ құрылған туынды кластардың даналары үшін қалыптастырылған барлық әдістерді тексере отырып, бастапқы мәліметтер мен нәтижелерді дисплейге шығарады.

Әрбір кластың көшірме конструкторлары үшін дисплейдің белгілі бір жеріне оның шақырылу санын диагностикалық түрде басып шығаруды қарастырыңыз (кластың статикалық мүшелерін қолданған жөн).

Сұхбат режимі иерархиялық меню көмегімен қамтамасыз етіледі.

6-нұсқа

ТІРКЕС базалық класын сипаттау керек.

Кластың міндетті өрістері:

□ **char** типіне нұсқауыш – тіркес символдарын орналастыру үшін динамикалық түрде бөлінген жадының адресін сақтайды.

□ **int** типті мән – тіркестің ұзындығын байт есебімен сақтайды.

Міндетті әдістер келесі әрекеттерді орындау керек:

□ параметрлерсіз конструктор;

□ параметр ретінде С-тіркесті қабылдайтын конструктор (нөлдік байтпен аяқталады);

□ параметр ретінде символды қабылдайтын конструктор;

□ көшірме конструкторы;

□ тіркес ұзындығын анықтау;

□ тіркесті тазарту (тіркесті бос ету);

□ деструктор.

ТІРКЕС класынан туынды БИТТІК_ТІРКЕС класын сипаттаңыз.

Бұл кластың тіркестерінде тек '0' немесе '1' символдары болуы мүмкін. Егер инициалдаушы тіркес құрамында мүмкін болатын символдардан өзгешелері кездесетін болса, БИТТІК_ТІРКЕС нөлдік мән қабылдайды. Бұл тіркестердің құрамы екілік сан ретінде қарастырылады.

Теріс сандар қосымша кодта сақталады.

Міндетті түрде болатын әдістері:

□ параметрлері жоқ конструктор;

□ параметр ретінде С-тіркесті қабылдайтын конструктор (нөлдік байтпен аяқталады);

□ көшірме конструкторы;

□ деструктор;

□ таңбаны қарама-қарсысына ауыстыру (санды қосымша кодқа түрлендіру).

Келесі операцияларды қайта анықтау қажет (нәтижелік тіркес ұзындығы екі тіркес ішіндегі ұзын тіркестің ұзындығына тең; қажет болған жағдайда қысқалау биттік тіркес солға қарай таңбалық разряд арқылы кеңейтіледі):

□ меншіктеу (=);

□ қосу (+) – тіркестердің арифметикалық қосындысы;

□ (==) операциясы – теңдікке тексеру.

Программа құрушы қосымша өрістер мен әдістердің кез келген санын (қажеттілігін негіздей отырып) енгізуге құқылы болады.

Келесі әрекеттерді орындайтын тестілік программа жазыңыз:

□ динамикалық түрде базалық класқа нұқауыштар жиымын бөледі (4-6);

□ сұхбат режимінде осы жиымды туынды кластарға нұсқауыштармен толтырады, бұл кезде туынды кластардың даналары (экземплярлары) бастапқы мәндерін беру арқылы динамикалық түрде құрылады;

□ құрылған туынды кластардың даналары үшін қалыптастырылған бар-

лық әдістерді тексере отырып, бастапқы мәліметтер мен нәтижелерді дисплейге шығарады.

Әрбір кластың көшірме конструкторлары үшін дисплейдің белгілі бір жеріне оның шақырылу санын диагностикалық түрде басып шығаруды қарастырыңыз (кластың статикалық мүшелерін қолданған жөн).

Сұхбат режимі иерархиялық меню көмегімен қамтамасыз етіледі.

7- нұсқа

ТІРКЕС базалық класын сипаттау керек.

Кластың міндетті өрістері:

□ **char** типіне нұсқауыш – тіркес символдарын орналастыру үшін динамикалық түрде бөлінген жадының адресін сақтайды.

□ **int** типті мән – тіркестің ұзындығын байт есебімен сақтайды.

Міндетті түрде болатын әдістер келесі әрекеттерді орындауы тиіс:

□ параметрлерсіз конструктор;

□ параметр ретінде С-тіркесті қабылдайтын конструктор (нөлдік байтпен аяқталады);

□ параметр ретінде символды қабылдайтын конструктор;

□ көшірме конструкторы;

□ тіркес ұзындығын анықтау;

□ тіркесті тазарту (тіркесті бос ету);

□ деструктор.

ТІРКЕС класынан туынды ОНДЫҚ_ТІРКЕС класын сипаттаңыз.

Бұл кластың тіркестерінде тек ондық цифр символдары және санның таңбасын көрсететін – және + символдары болуы мүмкін. Бұл – немесе + символдары санның тек алғашқы позициясында орналасуы мүмкін, мұнда + символы болмауы мүмкін, онда сан оң болып есептеледі. Егер инициалдаушы тіркестің құрамында мүмкін болатын символдардан басқалары кездесетін болса, ОНДЫҚ_ТІРКЕС нөлдік мән қабылдайды. Берілген тіркестердің құрамы ондық сан ретінде қарастырылады.

Міндетті түрде болатын әдістер:

□ параметрлері жоқ конструктор;

□ параметр ретінде С-тіркесті қабылдайтын конструктор (нөлдік байтпен аяқталады);

□ көшірме конструкторы;

□ деструктор;

□ берілген санды **int** форматында бейнелеу мүмкіндігін анықтайтын әдіс.

Келесі операцияларды қайта анықтау қажет:

□ меншіктеу (=);

□ азайту (-) – тіркестердің арифметикалық айырмасы;

□ (>) операциясы – үлкен болуын тексеру (мәні бойынша);

□ (<) операциясы – кіші болуын тексеру (мәні бойынша).

Программа құрушы қосымша өрістер мен әдістердің кез келген санын (қажеттілігін негіздей отырып) енгізуге құқылы болады.

Келесі әрекеттерді орындайтын тестілік программа жазыңыз:

□ динамикалық түрде базалық класқа нұқауыштар жиымын бөледі (4-6);

□ сұхбат режимінде осы жиымды туынды кластарға нұсқауыштармен толтырады, бұл кезде туынды кластардың даналары (экземплярлары) бастапқы мәндерін беру арқылы динамикалық түрде құрылады;

□ құрылған туынды кластардың даналары үшін қалыптастырылған барлық әдістерді тексере отырып, бастапқы мәліметтер мен нәтижелерді дисплейге шығарады.

Әрбір кластың көшірме конструкторлары үшін дисплейдің белгілі бір жеріне оның шақырылу санын диагностикалық түрде басып шығаруды қарастырыңыз (кластың статикалық мүшелерін қолданған жөн).

Сұхбат режимі иерархиялық меню көмегімен қамтамасыз етіледі.

8-нұсқа

ТІРКЕС базалық класын сипаттау керек.

Кластың міндетті өрістері:

□ **char** типіне нұсқауыш – тіркес символдарын орналастыру үшін динамикалық түрде бөлінген жадының адресін сақтайды.

□ **int** типті мән – тіркестің ұзындығын байт есебімен сақтайды.

Міндетті түрде болатын әдістер келесі әрекеттерді орындауы керек:

□ параметрлерсіз конструктор;

□ параметр ретінде С-тіркесті қабылдайтын конструктор (нөлдік байтпен аяқталады);

□ параметр ретінде символды қабылдайтын конструктор;

□ көшірме конструкторы;

□ тіркес ұзындығын анықтау;

□ тіркесті тазарту (тіркесті бос ету);

□ деструктор.

ТІРКЕС класынан туынды КОМПЛЕКС_САН класын сипаттаңыз.

Бұл кластың тіркестері і символымен ажыратылған екі өрістен тұрады.

Бірінші өріс санның нақты бөлігінің, ал екінші өріс жорамал бөлігінің мәнін береді. Өрістердің әрқайсысында тек ондық цифр символдары және саннық таңбасын көрсететін – және + символдары болуы мүмкін. Осы – немесе + символдары санның тек алғашқы позициясында орналасуы мүмкін, мұнда + символы болмауы да мүмкін, ондайда сан оң болып есептеледі. Егер инициалдаушы тіркестің құрамында мүмкін болатын символдардан басқалары кездесетін болса, КОМПЛЕКС_САН нөлдік мән қабылдайды. Тіркестердің мысалдары: 33i12, -7i100, +5i-21.

Міндетті түрде болатын әдістер:

□ параметрлері жоқ конструктор;

□ параметр ретінде C-тіркесті қабылдайтын конструктор (нөлдік байтпен аяқталады);

□ көшірме конструкторы;

□ деструктор.

Келесі операцияларды қайта анықтау қажет:

□ меншіктеу (=);

□ (==) операциясы – теңдікке тексеру;

□ көбейту (*) – сандарды көбейту.

Программа құрушы қосымша өрістер мен әдістердің кез келген санын (қажеттілігін негіздей отырып) енгізуге құқылы болады.

Келесі әрекеттерді орындайтын тестілік программа жазыңыз:

□ динамикалық түрде базалық класқа нұқауыштар жиымын бөледі (4-6);

□ сұхбат режимінде осы жиымды туынды кластарға нұсқауыштармен толтырады, бұл кезде туынды кластардың даналары (экземплярлары) бастапқы мәндерін беру арқылы динамикалық түрде құрылады;

□ құрылған туынды кластардың даналары үшін қалыптастырылған барлық әдістерді тексере отырып, бастапқы мәліметтер мен нәтижелерді дисплейге шығарады.

Әрбір кластың көшірме конструкторлары үшін дисплейдің белгілі бір жеріне оның шақырылу санын диагностикалық түрде басып шығаруды қарастырыңыз (кластың статикалық мүшелерін қолданған жөн).

Сұхбат режимі иерархиялық меню көмегімен қамтамасыз етіледі.

Класс шаблондары

Бұл тақырып бойынша лабораториялық жұмыстар тапсырмаларын жүзеге асыратын программалар практикумның [12] үшінші семинарында келтірілген.

1-нұсқа

«Стек» класының шаблонын құру керек. Оны «Кластар» бөліміндегі 1-тапсырманы орындаған кезде пайдаланыңыз.

2-нұсқа

«Стек» класының шаблонын құру керек. Оны «Мәліметтердің динамикалық құрылымдары» бөліміндегі 7-тапсырманы орындаған кезде пайдаланыңыз (бірінші бөлімге арналған жаттығуларды қараңыз).

3-нұсқа

«Стек» класының шаблонын құру керек. Оны «Мәліметтердің динамикалық құрылымдары» бөліміндегі 8-тапсырманы орындаған кезде пайдаланыңыз (бірінші бөлімге арналған жаттығуларды қараңыз).

4-нұсқа

«Бірбағытты сызықтық тізім» класының шаблонын құру керек. Оны «Мәліметтердің динамикалық құрылымдары» бөліміндегі 1-тапсырманы

орындаған кезде пайдаланыңыз (бірінші бөлімге арналған жаттығуларды қараңыз).

5-нұсқа

«Бірбағытты сызықтық тізім» класының шаблонын құру керек. Оны «Мәліметтердің динамикалық құрылымдары» бөліміндегі 3-тапсырманы орындаған кезде пайдаланыңыз (бірінші бөлімге арналған жаттығуларды қараңыз).

6-нұсқа

«Бинарлы бұтақ» класының шаблонын құру керек. Оны «Мәліметтердің динамикалық құрылымдары» бөліміндегі 2-тапсырманы орындаған кезде пайдаланыңыз (бірінші бөлімге арналған жаттығуларды қараңыз).

7-нұсқа

«Бинарлы бұтақ» класының шаблонын құру керек. Оны «Мәліметтердің динамикалық құрылымдары» бөліміндегі 4-тапсырманы орындаған кезде пайдаланыңыз (бірінші бөлімге арналған жаттығуларды қараңыз).

8-нұсқа

«Бинарлы бұтақ» класының шаблонын құру керек. Оны пернетақтадан немесе файлдан берілетін бүтін сандар мен тіркестерді сұрыптау үшін пайдаланыңыз.

9-нұсқа

«Кезек» класының шаблонын құру керек. Шаблон параметрлерінің әртүрлі типтері үшін осы шаблонмен жұмыс істеуді сипаттайтын программаны жазыңыз. Программада шаблонның барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

10-нұсқа

«Басымдылықты кезек» класының шаблонын құру керек. Осындай кезекке элементті енгізу кезінде оның нөмірі басымдылығы бойынша анықталады. Шаблон параметрлерінің әртүрлі типтері үшін осы шаблонмен жұмыс істеуді сипаттайтын программаны жазыңыз. Программада шаблонның барлық әдістерін тексеруге мүмкіндік беретін меню болуы тиіс.

11-нұсқа

Комплекс сандармен жұмыс істеуге арналған кластың шаблонын құру керек («Кластар» бөлімінің 5-нұсқасын қараңыз). Осы шаблонды қолдана отырып, санның нақты және жорамал бөліктерін **`double`** типті сандармен де, бүтін сандармен де беру арқылы программа жазу қажет.

12-нұсқа

Сандардың бірөлшемді жиымдарының класын параметрлеу керек («Кластар» бөлімінің 8-нұсқасын қараңыз). Осы шаблонды әртүрлі типтегі жиымдарды құру үшін қолданатын программа жазу қажет.

13-нұсқа

«Кластар» бөлімінің 12-нұсқасында сипатталған класты параметрлеу керек. Осы үлгіні әртүрлі типтегі матрицаларды құру үшін қолданатын программа жазу қажет.

14-нұсқа

Негізгі операцияларды – элементті қосуды және жоюды, жиындардың қиылысун, бірігуін және айырымын орындауға мүмкіндік беретін «жиын» класының шаблонын сипаттау керек.

Осы шаблонның әртүрлі мәліметтер типтері үшін жұмыс жасауын сипаттайтын программа жазыңыз.

15-нұсқа

«Бірбағытты сақиналық тізім» класының шаблонын құру керек. Оны «Мәліметтердің динамикалық құрылымдары» бөліміндегі 6-тапсырманы орындаған кезде пайдаланыңыз (бірінші бөлімге арналған жаттығуларды қараңыз).

16-нұсқа

«Екібағытты сақиналық тізім» класының шаблонын құру керек. Оны «Мәліметтердің динамикалық құрылымдары» бөліміндегі 5-тапсырманы орындаған кезде пайдаланыңыз (бірінші бөлімге арналған жаттығуларды қараңыз).

17-нұсқа

«Бірбағытты сызықтық нұсқауыштар тізімі» класының шаблонын құру керек. Оны «Мәліметтердің динамикалық құрылымдары» бөліміндегі 11-тапсырманы орындаған кезде пайдаланыңыз (бірінші бөлімге арналған жаттығуларды қараңыз).

18-нұсқа

«Бинарлы бұтақ» класының шаблонын құру керек. Оны «Кластар» бөліміндегі 2-тапсырманы орындаған кезде пайдаланыңыз.

19-нұсқа

«Бинарлы бұтақ» класының шаблонын құру керек. Оны «Мәліметтердің динамикалық құрылымдары» бөліміндегі 18-тапсырманы орындаған кезде пайдаланыңыз (бірінші бөлімге арналған жаттығуларды қараңыз).

20-нұсқа

Мәліметтер элементтеріне нұсқауыштары бар «бинарлы бұтақ» класының шаблонын құру керек. Оны «Мәліметтердің динамикалық құрылымдары» бөліміндегі 20-тапсырманы орындаған кезде пайдаланыңыз (бірінші бөлімге арналған жаттығуларды қараңыз).

III БӨЛІМ

Стандартты кітапхана

Кез келген программада тілдер операторымен қатар, программалау ортасына кіретін кітапхана құралдары қолданылады. Әртүрлі орталар программалаушының құзырына программаларды құруды жеңілдететін әртүрлі құралдар жиынтығын ұсынады, мысалы, Microsoft Visual C++ және Borland C++ компиляторларында Windows қосымшаларын жазуға арналған кластардың кітапханалары бар. Кітапханалардың бір бөлігі стандартталған, яғни олар қолданушыға C++ тілінің кез келген компиляторымен бірге берілуі керек.

C++ стандартты кітапханасын шартты түрде екі бөлікке бөлуге болады. Біріншісіне C кітапханасынан мұраланған функциялар, макростар, типтер және тұрақтылар, ал екіншісіне кластар және C++ тілінің басқа да құралдары жатқызылады. Кітапхананың бірінші бөлігі «Стандартты кітапхана функциялары» бөлімінде (92 б.) қарастырылған. Тұрақтылардың, макростардың және типтердің тізімі 5-қосымшада, ал функцялар тізімі 6қосымшада көрсетілген. Екінші бөлігінде кластар, шаблондар және басқа да стандартты типтегі ғана емес, қолданушылар анықтаған типтегі мәліметтерді де енгізуге, шығаруға, сақтауға және өңдеуге арналған құралдар қамтылған.

Стандартты кітапхананың кластарын олардың атқаратын қызметіне байланысты мынадай топтарға бөлуге болады:

- *Ағымдық кластар* жедел жады мен сыртқы құрылғылардың (мысалы, дискілер мен консоль) арасындағы, сондай-ақ жедел жады аймағында орналасқан мәліметтер ағымдарын басқаруға арналады.
- *Тіркестік класс* символдық тіркестермен ыңғайлы әрі қателерден қорғалған түрде жұмыс істеуге арналған.
- *Контейнерлік кластар* мәліметтерді сақтауға арналған барынша кең таралған құрылымдарды, мысалы, тізімдерді, векторларды және жиындарды жүзеге асырады. Кітапханаға, сонымен қатар, осы контейнерлерді қолданатын алгоритмдер де кіреді.
- *Итераторлар* контейнерлік және басқа да кластардың элементтеріне унификацияланған түрде қол жеткізуге арналған.
- *Математикалық кластар* жылжымалы нүктелі жиымдарды тиімді өңдеуді және комплекс сандармен тиімді жұмысты істеуді қолдайды.

□ *Диагностикалық кластар* типтерді динамикалық идентификациялауды және қателерді объектіге бағытталған түрде өңдеуді қамтамасыз етеді.

□ *Қалған кластар* жадыны динамикалық түрде бөліп үлестіруді, жергілікті (локалды) ерекшеліктерге бейімделуді, функционалды объектілерді өңдеуді және т.б. қамтамасыз етеді.

Контейнерлік кластар, алгоритмдер және итераторлар кіретін кітапхана бөлігін *шаблондардың стандартты кітапханасы* (STL — **Standard Template Library**) деп атайды.

Стандартты кітапхана құралдарын қолдану үшін программаға **#include** директивасының көмегімен қолданылатын кодтарға сәйкес тақырыптық файлдарды енгізу қажет. Мысалы, ағымдар **<iostream>**, ал тізімдер **<list>** файлында сипатталған. Тақырыптық файлдардың **.h** кеңейтілуі жоқ элементтері **std** атаулар кеңістігінде, ал атаулары дәл осындай **.h** кеңейтілуі бар элементтер ауқымды (глобалды) атаулар кеңістігінде анықталған. Стандартты кітапхана ортасындағы тақырыптық файлдар тізімі 4-қосымшада келтірілген.

C++ тілінің **std** атаулар кеңістігінде анықталған C кітапханасының функцияларына арналған тақырыптық файлдарының атаулары **c** әрпінен басталады, мысалы, **<cstdio>**, **<cstring>**, **<ctime>**. Әрбір **<cX>** түріндегі тақырыптық файл үшін дәл осы атауларды ауқымды атаулар кеңістігінде анықтайтын **<X.h>** файлы бар (107 б. қараңыз)[1].

[1]Ең үлкен жаңылыс тіркестер тақырыптарының атауларын пайдалану кезінде туындайды: **string** – C++ тілінің тіркестік кластарымен жұмыс істеуге арналған тақырып, **cstring** – C функциялары тақырыбының **std** атаулар кеңістігіндегі нұсқасы, **string.h** – ауқымды атаулар кеңістігіндегі дәл осындай тақырып, **cstring.h** – қоршаулардың біріндегі ауқымды атаулар кеңістігінде C++ тілінің тіркестік кластарымен жұмыс істеуге арналған ескі тақырып.

10-ТАРАУ

Ағымдық кластар

Ағым – бұл мәліметтердің шыққан жерінен қабылдаушысына кез келген түрде тасымалдануына байланысты айтылатын абстрактылы ұғым. C++ ағымдары, C стиліндегі енгізу/шығару функцияларымен салыстырғанда, тек стандартты ғана емес, қолданушылар анықтаған мәліметтер типтерімен де сенімді жұмыс істеуді қамтамасыз етеді, сонымен қатар оның синтаксисі де біртекті және түсінікті болып табылады.

Ағымнан мәліметтерді оқу оны ағымнан *шығарып алу* деп, ал ағымға мәлімет жазу оны *орналастыру* немесе *кірістіру* деп аталады. Ағым байттар тізбегі ретінде анықталады және ол алмасу жүргізілетін нақты құрылғыдан (жедел жады, дискідегі файл, пернетақта немесе принтер) тәуелсіз түрде орындалады. Көбінесе ақпараттардың тасымалдау жылдамдығын арттыру үшін ағыммен мәлімет алмасу жедел жадының арнайы аймағы болып табылатын *буфер* арқылы жүзеге асырылады. Мәліметтерді нақты түрде тасымалдау кезінде, ақпараттармен буферді толтырғаннан кейін – шығару әрекеті, ал буфердегі мәлімет біртіндеп таусылғанша – енгізу әрекеті жүзеге асырылады.

Ақпарат алмасу бағыты бойынша ағымды *кіріс* (мәліметтер компьютер жадына енгізіледі), *шығыс* (мәліметтер компьютер жадынан шығарылады) және *екібағыттағы* (мәліметтерді енгізуге де, шығаруға да болады) *ағымдар* деп бөлуге болады.

Ағым жұмыс істейтін құрылғылар түрлері бойынша оларды *стандартты, файлдық* және *тіркестік ағымдар* деп бөлуге болады.

Стандартты ағымдар мәліметтерді пернетақтадан дисплей экранына жіберуге, ал *файлдық ағымдар* сыртқы мәлімет жинақтауыштардағы (мысалы, магниттік дискідегі) файлдармен ақпарат алмасуға, ал *тіркестік ағымдар* компьютер жедел жадында символдар жиымымен жұмыс жасауға арналған.

Ағымдарды сүйемелдеу үшін C++ кітапханасының құрамында **ios** және **streambuf** сияқты екі базалық кластар негізінде құрылған кластар иерархиясы бар. Мұндағы **ios** класы мәліметтерді енгізу мен шығаруға арналған ортақ өрістер мен кластардан тұрады, ал **streambuf** класы ағымдарды буферлеуді және олардың физикалық құрылғылармен байланысуын қамтамасыз етеді. Бұл кластардан кіріс ағымдары үшін **istream** класы және шығыс ағымдары үшін **ostream** класы мұраланады. Соңғы аталған екі класс екібағытты ағымдарды жүзеге асыратын **iostream** класы үшін базалық класс болып табылады. Кластар иерархиясында осыдан төмен файлдық және тіркестік а ғындар орналасады. Төменде жиі қолданылатын ағымдар кластары көрсетілген.

ios — ағымдардың базалық класы;
istream — кіріс ағымдары класы;
ostream — шығыс ағымдары класы;
iostream — екібағытты ағымдар класы;
istringstream — тіркестік ағымдардың кіріс класы;
ostringrstream — тіркестік ағымдардың шығыс класы;
stringstream — екібағытты тіркестік ағымдар класы;
ifstream — кіріс файлдық ағымдар класы;
ofstream — шығыс файлдық ағымдар класы;
fstream — екібағытты файлдық ағымдар класы.

Класс сипаттамалары төмендегі тақырыптық файлдарда орналасады:

<ios> — енгізу/шығару ағымдарының базалық класы;
<iosfwd> — енгізу/шығару құралдарын алдын ала жариялау;
<istream> — енгізу ағымының шаблоны;
<ostream> — шығару ағымының шаблоны;
<iostream> — стандартты объектілер және енгізу/шығару ағымдарымен орындалатын операциялар;
<fstream> — файлдардан енгізу/шығару ағымдары;
<sstream> — тіркестерден енгізу/шығару ағымдары;
<streambuf> — енгізу/шығару ағымдарының буферленуі;
<iomanip> — манипуляторлар (олар туралы алдағы тарауларда айтылады, 290 б. қараңыз).

Программаға **<fstream>** және **<sstream>** файлдарын қосу арқылы **<iostream>** файлы да автоматты түрде қосылады, өйткені ол файлдар үшін базалық класс болып табылады.

Ағымдардың C кітапханасынан мұраланған енгізу/шығару функцияларынан негізгі артықшылығы типтерді басқару және де олардың кеңейтілу мүмкіндігі саналады, яғни қолданушы анықтаған типтермен жұмыс істеу мүмкіндігі болып табылады. Ол үшін ағымдардың операцияларын қайта анықтау қажет болады (бұл туралы «Қолданушы анықтаған ағымдар мен типтер» бөлімінде айтылады, 305 б.).

Сонымен қатар, ағымдар **wchar_t** сияқты кеңейтілген символдар жиынтығымен жұмыс істей алады. Ол үшін **wistream**, **wostream**, **wofstream** және т.б. кластар қолданылады[1]. Ағымдардың кемшіліктеріне программаның жұмыс істеу жылдамдығының төмендеуін жатқызуға болады, компилятордың

[1]Шын мәнінде ағымдар кітапханада үлгілер (шаблондардың) көмегімен жүзеге асырылған, олардың параметрі ретінде символ типі қолданылады. Аталған **char** және **wchar_t** параметрлері үшін класс атаулары **typedef** көмегімен анықталған.

жүзеге асырылуына байланысты жылдамдықтың төмендеу мәні айтарлықтай болуы мүмкін.

Стандартты ағымдар

<iostream> тақырыптық файлында енгізу/шығаруға арналған кластар сипаттамасымен қатар, алдын ала анықталған төрт объект бар.

Объект	Класс	Сипаттамасы
cin	**istream**	Пернетақтамен байланысады (стандартты буферленген енгізу әрекетімен)
cout	**ostream**	Экранмен байланысады (стандартты буферленген шығару әрекетімен)
cerr	**ostream**	Экранмен байланысады (стандартты буферленбеген шығару әрекетімен), мұнда қателік туралы хабарламалар шығарылады
clog	**ostream**	Экранмен байланысады (стандартты буферленген шығару әрекетімен), мұнда қателік туралы хабарламалар шығарылады

Бұл объектілер программмаға **<iostream>** тақырыптық файлын қосылғанда құрылады, мұнда олармен байланысты енгізу/шығару құралдары қолжетімді түрде болады. Бұл объектілердің атауларын басқа файлдарға немесе сиволдық буферлерге қайталап тағайындауға болады.

Жоғарыда сипатталған **istream** және **ostream** кластарында мәліметтерді ағымнан шығарып алу **>>** және ағымға орналастыру **<<** операциялары ығыстыру операцияларын асыра жүктеу жолымен анықталған. Мысалы:

```
#include <iostream.h>
int main(){
   int i;
   cin >> i;
   cout << "Сіз енгіздіңіз: " << i;
   return 0;
}
```

Шығарып алу және оқу операциялары өздерінің орындалу нәтижесі ретінде шығарып алу үшін **istream** типті объектіге және оқу үшін **ostream** типті объектіге сілтеме қалыптастырады. Бұл операциялар тізбектерін қалыптастыруға мүмкіндік береді, ол жоғарыдағы мысалдың соңғы операторы арқылы көрнекі түрде бейнеленген, онда мәлімет шығару солдан оңға қарай орындалады.

Басқа да асыра жүктелген операциялар сияқты, кірістіру мен шығарып алу үшін басымдылықтарды өзгерту мүмкін емес, сондықтан қажет болған жағдайларда жақшалар қолданылады:

```
// Жақша қажет емес - қосу басымдығы << операциясынан
// жоғары:
cout << i + j;
// Жақшалар қажет - қатынас операциясының басымдығы
// << операциясынан төмен:
cout << (i < j);
cout << (i << j);    // Оң жақтағы << операциясы
                     // ығыстыруды білдіреді
```

Шамаларды енгізу кезінде олар босорын символдарымен (босорындармен, табуляция және келесі жолға көшіру белгілерімен) ажыратылып жазылуы тиіс. Егер кезекті символ рұқсат етілмеген таңба болса, мәліметті шығарып алу тоқтатылады.

Егер ағымға орналастыру операциясында қандай да бір айнымалыны өзгертетін өрнек кездесетін болса, онда ол операциялар тізбегінде бір реттен артық орналаспауы тиіс, өйткені мұндайда нәтиже компилятордың жүзеге асырылуына байланысты (тәуелді) болуы мүмкін.

<< және **>>** операциялары барлық құрамдас мәліметтер типтері үшін асыра жүктелген, осы арқылы шамалар типіне сәйкес автоматты түрде енгізу мен шығаруды орындауға мүмкіндік туады. Бұл енгізу кезінде символдар тізбегі шығарып алу белгісінің оң жағында орналасқан шаманың ішкі көрінісіне түрлендірілетінін, ал шығару кезінде кері түрлендіру орындалатынын білдіреді, мысалы:

```
#include <iostream.h>
int main(){
   int i = 0xD;
   double d;
   // Енгізу ағымындағы символдар double типіне
   // түрлендіріледі:
   cin >> d;
   // int және double символдар тіркесіне
   // түрлендіріледі:
   cout << i << ' ' << d;
   return 0;
}
```

Осы операциялардың көмегімен әртүрлі типті мәліметтер қалай өңделетінін қарастырайық.

Сандық мәндерді ондық немесе он алтылық санау жүйесінде (0x префиксімен) таңбамен немесе таңбасыз енгізуге болады. Нақты сандар бектілген нүктелі немесе дәрежелі формада берілуі мүмкін. Мысалы, егер алдыңғы мысал үшін пернетақтадан **1.53e-2** символдар тізбегі енгізілсе, ол дәрежелі нақты сан ретінде қабылданады да, *double* типіне сәйкес келетін ішкі көрініске түрлендіріледі. Шығару кезінде кері түрлендіру орындалады және экранға төмендегідей символдар шығарылады:

```
13 0.0153
```

Енгізу буферлендірілген болғандықтан, енгізу буферіне мәліметтерді орналастыру келесі жолға көшу пернесін басқаннан кейін орындалады, осыдан кейін барып буферде ағымнан шығарып алу операциясы жүзеге асырылады. Бұл енгізілген символдарды **Enter** пернесі басылғанға дейін түзету мүмкіндігін береді.

Тіркестерді енгізу кезінде мәліметтерді шығару ең жақын орналасқан босорынға дейін орындалады (оның орнына тіркеске нөл-символ енгізіледі):

```
char str1[100], str2[100];
cin >> str1 >> str2;
```

Егер пернетақтадан «**бір екі үш төрт бес**» тіркесі енгізілсе, **str1** және **str2** айнымалылары сәйкесінше «**бір**» және «**екі**» мәндерін қабылдайды да, тіркестің қалған бөлігі ескерілмейді. Кіріс ағымынан тіркесті толығымен енгізу қажеттілігі туындағанда ('**\n**' символына дейін) **get** немесе **getline** әдістері қолданылады (292-293 б. қараңыз).

Нұсқауыштардың мәндері он алтылық санау жүйесінде шығарылады. Мәліметтерді шығару кезінде кез келген шамаға оны көрсету үшін қажетті позициялар саны бөлінеді. Бір шаманы екіншісінен бөліп алу үшін бос орындар қолданылады:

```
cout << i << ' ' << d << "    " << j;
```

Егер келісім бойынша қолданылатын шығару форматы программалаушының көңілінен шықпаса, оны енгізу/шығару кластары әдістерінің, форматтау жалаушалары және манипуляторлар көмегімен түзетуге болады. Бұл туралы төменде айтылады.

Мәліметтерді форматтау

Ағымдық кластарда форматтау үш түрлі тәсілмен – жалаушалар, манипуляторлар және форматтаушылар арқылы орындалады.

Жалаушалар және форматтаушы әдістер

Жалаушалар **ios** класының **long** типті **x_flags** өрісінде біріктірілген жеке биттер болып табылады. Жалаушалар 10.1-кестеде көрсетілген.

10.1- кесте. Форматтау жалаушалары

Жалауша	Күйі	Үнсіз келісім	Бит орнатылғандағы әрекеттің сипаттамасы
skipws	0x0001	+	Шығарып алу кезінде босорын символдары ескерілмейді
left	0x0002		Өрістің сол жақ шетіне қарай туралау
right	0x0004	+	Өрістің оң жақ шетіне қарай туралау

Жалауша	Күйі	Үнсіз келісім	Бит орнатылғандағы әрекеттің сипаттамасы
internal	0x0008		Санның таңбасы сол жақ шетте, ал сан оң жақ шетте шығарылады. Олардың арасы **x_fill** символдарымен (төменнен қараңыз), келісім бойынша бос орындармен толтырылады
dec	0x0010	+	Ондық санау жүйесі
oct	0x0020		Сегіздік санау жүйесі
hex	0x0040		Он алтылық санау жүйесі
showbase	0x0080		Санау жүйесінің негізі шығарылады (он алтылық сандар үшін 0x және сегіздік сандар үшін 0)
showpoint	0x0100		Нақты сандарды шығару кезінде ондық нүктені және санның бөлшек бөлігін басып шығару
uppercase	0x0200		Шығару кезінде жоғары регистр символдарын қолдану
showpos	0x0400		Оң сандарды шығару кезінде сан таңбасын басып шығару
scientific	0x0800		Нақты сандарды дәрежесі бар мантисса түрінде басып шығару
fixed	0x1000		Нақты сандарды бекітілген нүктелі түрде басып шығару (дәлдік **x_precision** өрісімен анықталады, төменнен қараңыз)
unitbuf	0x2000		Мәліметтерді әрбір шығарудан кейін барлық ағымдардың буферлерін шығару (босату)
stdio	0x4000		Мәліметтерді әрбір шығарудан кейін **stdout** және **stderr** ағымдарының буферлерін шығару (босату)

ЕСКЕРТУ

left, **right** және **internal**, **dec**, **oct** және **hex**, сонымен қатар **scientific** және **fixed** жалаушалары бір-біріне қарама-қарсы болып табылады, яғни кез келген уақыт сәтінде мұндай топтардағы тек бір ғана жалауша орнатылуы мүмкін.

ios класында жалаушаларды басқаратын **flags**, **setf** және **unsetf** әдістері бар:

long ios::flags(long); — ағымның сол сәттегі (ағымдағы) жалаушаларын қайтарады;

long ios::flags(long); — жалаушаларға параметр мәнін меншіктейді;

long ios::setf(long,long); — биттері бірінші параметрге орнатылған жалаушаларға екінші параметрдің сәйкес биттерінің мәнін меншіктейді;

long ios::setf(long); — параметрде биттері орнатылған жалаушаларды орнатады;

long ios::unsetf(long); — параметрде биттері орнатылған жалаушаларды алып тастайды.

Барлық функциялар ағымның бұрынғы жалаушаларын қайтарады.

Форматтау үшін, жалаушалардан басқа, **ios** класының келесі *өрістері* қолданылады:

int x_width — шығару өрісінің ең кіші (минималды) ені;

int x_precision — бекітілген нүктелі нақты сандарды шығару кезіндегі санның бөлшек бөлігіндегі цифрлар саны немесе санды мантисса және дәреже түрінде шығару кезіндегі мән сақтайтын цифрлардың жалпы саны;

int x_fill — мәліметтерді шығару өрісін толтыру символы.

Осы өрістерді басқару үшін **width, precision** және **fill** әдістері қолданылады:

int ios::width() — мәлімет шығару өрісі енінің мәнін қайтарады;

int ios:: width(int) — параметр мәніне сәйкес шығару өрісі енінің мәнін орнатады;

int ios::precision() — нақты сандарды шығару кезінде санды бейнелеу дәлдігінің мәнін қайтарады;

int ios::precision(int) — нақты сандарды шығару кезіндегі санды бейнелеу дәлдігінің мәнін орнатады, дәлдіктің бұрынғы (ескі) мәнін қайтарады;

char fill() — ағымдағы толтыру символын қайтарады;

char fill(char) — ағымдағы толтыру символының мәнін орнатады, смиволдың бұрынғы (ескі) мәнін қайтарады.

Кейбір жалаушаларды орнату алдында олармен бір сәтте қатар орнатыла алмайтын жалаушаларды алып тастау қажет. Ол үшін **setf** әдісінің екінші параметрі ретінде төменде көрсетілген **ios** класының *статикалық тұрақтыларын* қолданған ыңғайлы болып табылады:

```
adjustfield (left | right | internal)
basefield   (dec | oct | hex)
floatfield  (scientific | fixed)
```

Мәліметтерді жалаушалар мен әдістер көмегімен шығару кезіндегі форматтау мысалы:

```
#include <iostream.h>
int main(){
   long a = 1000, b = 077;
   cout.width(7);
   cout.setf(ios::hex | ios::showbase |ios::uppercase);
   cout << a;
   cout.width(7);
   cout << b << endl;
   double d = 0.12, c = 1.3e-4;
   cout.setf(ios::left);
   cout << d << endl;
   cout << c;
   return 0;
}
```

Программа жұмысының нәтижесінде бірінші жолда он алтылық жүйеде бас әріптермен **a** және **b** айнымалылары шығарылады, олардың әрқайсысына 7 позициядан бөлінеді (**width** функциясы тек бір ғана шығарылатын мәнге әсер етеді, сондықтан оны шақыртуды екі рет қайталау керек). **c** және **d** айнымалыларының мәндері өрістің сол жақ шетіне қарай тураланған:

```
0X3E8   0X3F
0.12
0.00013
```

Манипуляторлар

Манипуляторлар деп мәліметтерді форматтау үшін оларды орналастыру және шығарып алу операциялары тізбегіне қосуға болатын функцияларды атайды. Манипуляторлар *қарапайым* және *параметрленген* болып бөлінеді, бұлардың біріншісі аргументтерді көрсетуді талап етпейді. Манипуляторларды қолдану форматтау жалаушаларын орнату әдістерін қолдануға қарағанда айтарлықтай ыңғайлы болып табылады.

Қарапайым манипуляторлар

Төменде аргументтерін көрсетуді талап етпейтін манипуляторлар тізбекті түрде көрсетілген:

dec — мәліметтерді енгізу және шығару кезінде ондық санау жүйесінің жалаушасын орнатады;

oct — мәліметтерді енгізу және шығару кезінде сегіздік санау жүйесінің жалаушасын орнатады;

hex — мәліметтерді енгізу және шығару кезінде он алтылық санау жүйесінің жалаушасын орнатады;

ws — мәліметтерді енгізу кезінде бос орын символдарын шығарып алуды орнатады;

endl — мәліметтерді шығару кезінде ағымға жаңа жолға көшу символын қосады және буферді шығарады (босатады);

ends — мәліметтерді шығару кезінде ағымға нөлдік символды қосады;

flush — мәліметтерді шығару кезінде буферді босатады.

Санау жүйесін өзгерту оны келесі тікелей өзгертуге дейін әсер етеді.

Мысал:

```
cout << 13 << hex << ' ' << 13 << oct << ' ' << 13 << endl;
```

Егер келісім бойынша жалаушалардың басқа мәндері орнатылған болса, онда төмендегі нәтиже шығарылады:

```
13 d 15
```

Параметрленген манипуляторлар

Төменде аргументтерін көрсетуді талап ететін манипуляторлар тізімі көрсетілген. Оларды пайдалану үшін программаға **<iomanip>** тақырыптық файлын қосу керек.

setbase(int n) — санау жүйесінің негізін береді (n = 8, 16, 10 немесе 0). 0 үнсіз келісім бойынша орнатылған негіз болып табылады (8-дік немесе 16-лық сандар енгізілген сәттерден басқа жағдайларда ондық жүйе болады);

resetiosflags(long) — биттер мәні параметрде орнатылған ағым қалып-күйінің жалаушаларын алып тастайды;

setiosflags(long) — параметрдегі биттерінің мәні 1-ге тең ағым қалып-күйінің жалаушаларын орнатады;

setfill(int) — параметр мәніне тең коды бар символ-толтырушыны орнатады;

setprecision(int) — бекітілген нүктелі (**fixed** жалаушасы) түрдегі нақты сандар үшін санның бөлшек бөлігіндегі цифрлардың максималды санын немесе мантисса мен дәреже түрінде берілген сандар үшін (**scientific** жалаушасы) мәндерді сақтайтын цифрлардың жалпы санын орнатады;

setw(int) — мәлімет шығару өрісінің максималды енін орнатады.

Параметрленген манипуляторларды қолдану мысалы:

```
#include <iostream.h>
#include <iomanip.h>
int main(){
   double d[] = {1.234, -12.34567, 123.456789, -1.234,
   0.00001};
   cout << setfill('.') << setprecision(4)
        << setiosflags(ios::showpoint | ios::fixed);
   for (int i = 0; i <5; i++)
      cout << setw(12) << d[i] << endl;
   return 0;
}
```

Программа жұмысының нәтижесі:

```
......1.2340
....-12.3457
....123.4568
.....-1.2340
......0.0000
```

Ағымдармен мәлімет алмасу әдістері

Ағымдық кластардан мәліметтерді шығарып алу **>>** және қосу **<<** операцияларымен қатар, ағымнан мәліметтерді форматталмаған түрде оқу мен жазу әдістері анықталған (мұндайда мәліметтерді түрлендіру орындалмайды).

Төменде **istream** класында анықталған мәліметтерді оқу функциялары көрсетілген.

gcount() — соңғы форматталмаған мәлімет енгізу функциясы арқылы оқылған символдар санын қайтарады;

get() — ағымнан шығарылып алынған символдың кодын немесе **EOF** мәнін қайтарады;

get(c) — сілтемені мәлімет оқылған ағымға қайтарады да, шығарылып алынған символды **c**-ға жазады;

get(buf, num, lim='\n') — **num-1** символды (немесе **lim** символы кездескенге дейін) оқиды да, оларды **buf** символдық тіркесіне көшіреді. Тіркеске **lim** символының орнына

[1]Функцияға хабарласу кезінде келісім бойынша орнатылған параметрлерді көрсету міндетті еместігін еске түсірейік.

	тіркес соңы белгісі (**'\0'**) жазылады. lim символы ағымда қалады. Сілтемені үстіміздегі ағымға қайтарады;
getline(buf,num,lim='\n')	— **get** функциясына ұқсас, бірақ **buf** тіркесіне **lim** символын да көшіріп жазады;
ignore(num = 1,lim = EOF)	— **num** символ оқылып біткенше немесе **lim** параметрімен берілген ажыратқыш символ кездескенше символдарды оқиды және өткізіп отырады. Сілтемені үстіміздегі ағымға қайтарады;
peek()	— келесі символды ағымнан өшірмей қайтарады немесе файл соңына жеткен жағдайда **EOF** мәнін қайтарады;
putback(c)	— ағымға **c** символын орналастырады, ол ағымнан мәлімет шығарып алу кезінде үстіміздегі символға айналады;
read(buf,num)	— **num** символды (немесе символдар саны **num** шамасынан аз болса, барлық символдарды) **buf** символдық жиымына оқиды да, сілтемені жұмыс істеп тұрған ағымға қайтарады;
readsome(buf,num)	— **num** символды (немесе символдар саны **num** шамасынан аз болса, барлық символдарды) **buf** символдық жиымына оқиды да, оқылған символдар санын қайтарады;
seekg(pos)	— ағымдағы оқу позициясын **pos** мәніне орнатады;
seekg(offs, org)	— **org** параметрімен анықталатын позициялардың бірінен – **ios::beg** (файл басынан), **ios::cur** (ағымдағы позициядан) немесе **ios::end** (файл соңынан) бастап санай отырып, ағымдағы оқу позициясын **offs** байтқа жылжытады;
tellg()	— ағымды оқу кезінде үстіміздегі позицияны қайтарады;
unget()	— соңғы оқылған символды ағымға орналастырады да, сілтемені жұмыс істеп тұрған ағымға қайтарады.

ostream класында мәліметтерді форматталмаған түрде шығаруға арналған, жоғарыда көрсетілгендерге ұқсас функциялар анықталған:

flush()	— шығару ағымының құрамын физикалық құрылғыға жазады;
put(c)	— ағымға с символын шығарады да, сілтемені ағымға қайтарады;
seekp(pos)	— ағымдағы жазу позициясын **pos** мәніне орнатады;
seekp(offs, org)	— **org** параметрімен анықталатын позициялардың бірінен – **ios::beg** (файл басынан), **ios::cur** (ағымдағы позициядан) немесе **ios::end** (файл соңынан) бастап санай отырып, ағымдағы жазу позициясын **offs** байтқа жылжытып ығыстырады;
tellp()	— ағымның сол сәттегі жазу позициясын қайтарады;
write(buf, num)	— **buf** жиымынан **num** символды ағымға жазады және сілтемені ағымға қайтарады.

1-мысал. Программа сөз тіркестерін кіріс ағымынан символдық жиымға оқиды.

```
#include "iostream.h"
int main(){
   const int N = 20, Len = 100;
   char str[N][Len];
   int i = 0;
   while (cin.getline(str[i], Len, '\n') && i<N){
       // ...
       i++;
   }
   return 0;
}
```

2-мысал. Программа файлға (файлдық ағымдар келесі бөлімде қарастырылады) жылжымалы нүктелі санды және символдар тіркесін жазады, содан кейін оларды файлдан оқып, экранға шығарады:

```
#include <iostream.h>
#include <fstream.h>
#include <string.h>
int main(){
// Файлға жазу
   ofstream out("test");
   if(!out){
      cout << "Cannot open file 'test' for writing" << endl;
```

```
      return 1;
   }
   double num = 100.45;
   char str[ ] = "This is a test.";
   out.write(reinterpret_cast<char *>(&num),
             sizeof(double));
   out.write(str, strlen(str));
   out.close();
   // Файлдан оқу
   ifstream in("test", ios::in|ios::nocreate);
   if(!in){
      cout << "Cannot open file 'test' for reading"
           << endl;
      return 1;
   }
   double check_num;
   char check_str[60];
   in.read(reinterpret_cast<char *>(&check_num),
           sizeof(double));
   in.read(check_str, 60);
   int lstr = in.gcount(); // оқылған символдар саны
   check_str[lstr] = 0;    // нөл-символды тіркес
                           // соңына енгізу
   cout << check_num << ' ' << check_str << endl;
   in.close();
   return 0;
}
```

Бұл мысалдағы **write()** және **read()** функцияларының шақырылуындағы **reinterpret_cast<char *>** типін келтіру параметр символдық жиым болмайтын жағдайларда қажет болып табылады.

3-мысал. Төменде көрсетілген программада **test** файлы құрылады да, оған үш тіркес шығарылады.

```
#include <iostream.h>
#include <fstream.h>
#include <string.h>
int main(){
   // Файлға жазу
   ofstream out("test");
   if(!out) {
      cout << "Cannot open file 'test' for writing"
           << endl;
      return 1;    }
```

```
    char *str[ ] = {"This is the first line.",
                    "This is the second line.",
                    "This is the third line."};
    for (int i = 0; i<3; ++i){
       out.write(str[i], strlen(str[i]));
       out.put('\n');
    }
    out.close();
    // Файлдан оқу
    ifstream in("test", ios::in|ios::nocreate);
    if(!in){
       cout << "Cannot open file 'test' for reading"
            << endl;
       return 1;
    }
    char check_str[3][60];
    for (i = 0; i<3; ++i){
       in.get(check_str[i], 60);
       in.get();}
    // Тексеру үшін шығару
    for (i = 0; i<3; ++i)
       cout << check_str[i] << endl;
    in.close();
    return 0;
}
```

Мысалдағы **get(check_str[i],60)** функциясы орындалғаннан кейін тіркестерді ажыратушы символ **'\n'** кіріс ағымында қалады, сондықтан бір символды өткізіп жіберу үшін **get()** әдісін шақыру қажет. Баламалы тәсіл ретінде **get** функциясының орнына кіріс ағымынан шектеуіш-символды шығарып алатын **getline** функциясын қолдануға болады.

4-мысал. peek() және **putback()** функциялары әрбір нақты уақыт мезетінде енгізілетін ақпараттың типі белгісіз болған кезде басқаруды жеңілдетуге мүмкіндік береді. Келесі программа осыны көрсетеді. Онда файлдан (файлдық ағымдар келесі бөлімде қарастырылады) тіркестер немесе бүтін сандар оқылады. Тіркестер мен бүтін сандар кез келген ретпен орналасуы мүмкін.

```
#include <fstream.h>
#include <ctype.h>
#include <stdlib.h>
int main(){
char ch;
   // файлды дайындау
   ofstream out("test");
```

```
   if(!out) {
      cout << "Cannot open file 'test' for writing"
           << endl;
      return 1;
   }
   char str[80], *p;
   out << 123 << "this is a test" << 23;
   out << "Hello there!" << 99 << "bye" << endl;
   out.close();
   // файлды оқу
   ifstream in("test", ios::in|ios::nocreate);
   if(!in){
      cout << "Cannot open file 'test' for reading"
           << endl;
      return 1;
   }
   do{
      p = str;
      ch = in.peek();     // келесі символдың типін анықтау
      if(isdigit(ch)){
         // бүтін санды оқу:
         while(isdigit(*p = in.get())) p++;
         in.putback(*p);  // символды ағымға қайтару
         *p = '\0';       // тіркесті нөлмен аяқтаймыз
         cout << "Number: " << atoi(str);
      }
      else if(isalpha(ch)){ // тіркесті оқу
         while(isalpha(*p = in.get())) p++;
         in.putback(*p);  // символды ағымға қайтару
         *p = '\0';       // тіркесті нөлмен аяқтаймыз
         cout << "String: " << str;
      }
      else in.get();      // өткізіп жіберу
      cout << endl;
   } while(!in.eof());
   in.close();
   return 0;
}
```

Программа жұмысының нәтижесі:

```
Number: 123
String: this

String: is
```

```
String: a

String: test
Number: 23
String: Hello

String: there

Number: 99
String: bye
```

Ағымдар арқылы программаны қолданушылармен сұхбат ұйымдастыру кезінде буферлеу әрекетін ескеру қажет. Мысалы, енгізу әрекетін орындауға ұсыныс шығару кезінде оның кіріс ағымынан мәліметтердің оқылуынан бұрын пайда болатындығына кепілдік бере алмаймыз, өйткені ұсыныс тек шығару буфері толған жағдайда ғана экранда пайда болады:

```
cout << "x енгізіңіз";
cin >> x;
```

Бұл мәселені шешу үшін **basic_ios** құрамында **tie()** функциясы анықталған, ол **cin.tie(&cout)** түріндегі шақыру арқылы **istream** және **ostream** ағымдарын өзара байланыстырады. Осыдан кейін енгізу ағымынан жаңа символ алу қажет болған сайын шығару жолы тазартылады (яғни **cout.flush()** функциясы орындалады).

Бір программада ағымдармен қатар C кітапханасының **<cstdio>** немесе **<stdio.h>** файлдарында сипатталған функцияларын қолдану қажет етілмейді. Егер бұл белгілі бір себептермен қажет болып жатса, онда ағымдармен бірінші операцияны орындағанға дейін **ios_base** құрамында сипатталған, ортақ буферді қолдануды қамтамасыз ететін **sync_with_stdio()** функциясын шақыру қажет. **sync_with_stdio(false)** функциясын шақыру буферлерді ажыратады (бұл жұмыс өнімділігінің артуына алып келуі мүмкін).

Ағымдар қателіктері

ios базалық класында ағымның қалып-күйін биттер жиыны түрінде көрсететін **state** өрісі анықталған:

```
enum io_state{
   goodbit = 0x00,    // Қателер жоқ
   eofbit = 0x01,     // Файл соңына жетті
   failbit = 0x02,    // Форматтау немесе түрлендіру
                      // қателігі
   badbit = 0x04,     // Күрделі қате, ол туындағаннан
                  // кейін ағымды қолдану мүмкін болмайды
   hardfail = 0x08    // Жабдықтардың қателігі
};
```

Ағым қалып-күйін басқару үшін төменде көрсетілген әдістер мен операциялар қолданылады:

`int rdstate()`	— ағымның сол сәттегі қалып-күйін қайтарады;
`int eof()`	— егер **`eofbit`** жалаушасы орнатылған болса, нөлге тең емес мәнді қайтарады;
`int fail()`	— егер **`failbit`**, **`badbit`** немесе **`hardfail`** жалаушаларының бірі орнатылған болса, нөлге тең емес мәнді қайтарады;
`int bad()`	— егер **`badbit`** немесе **`hardfail`** мәндерінің бірі орнатылған болса, нөлге тең емес мәнді қайтарады;
`int good()`	— барлық қателік жалаушалары алып тасталған болса, нөлге тең емес мәнді қайтарады;
`void clear (int = 0)`	— параметр қателік күйі ретінде қабылданады, параметр жоқ болған жағдайда қателік күйі ретінде 0 орнатылады;
`operator void*()`	— кем дегенде бір қателік биті орнатылған болса, нөлдік нұсқауышты қайтарады;
`operator !()`	— кем дегенде бір қателік биті орнатылған болса, нөлдік емес нұсқауышты қайтарады.

Төменде ағым қалып-күйінің жалаушалары арқылы жиі пайдаланылатын операциялар келтірілген.

```
// flag жалаушасының орнатылуын тексеру:
if(stream_obj.rdstate() & ios::flag)
// flag жалаушасын алып тастау:
stream_obj.clear(rdstate() & ~ios::flag)
// flag жалаушасын орнату:
stream_obj.clear(rdstate() | ios::flag)
// flag жалаушасын орнату және барлық қалған
// жалаушаларлы алып тастау:
stream_obj.clear( ios::flag)
// Барлық жалаушаларды алып тастау:
stream_obj.clear()
```

Ағым 0 мәнімен салыстырылған сайын **`void*()`** операциясы жанамалы (тікелей емес) түрде шақырылады. Бұл келесідей циклдерді жазуға мүмкіндік береді:

```
while (stream_obj){
   // бәрі дұрыс, енгізу/шығаруды орындауға болады
}
```

Төменде келтірілген мысалда **`rdstate()`** функциясының жұмысы көрсетілген. Бұл программа команда жолында аты көрсетілетін мәтіндік

файлдың ішкі мазмұнын экранға шығарады. Қате табылған жағдайда функция ол туралы **CheckStatus()** арқылы хабарлайды.

```
#include <iostream.h>
#include <fstream.h>
void CheckStatus(ifstream &in);
int main(int argc, char *argv[ ]){
   if(argc != 2){
      cout << "Usage: <program_name> <file_name>"
           << endl;
      return 1;
   }
   ifstream in(argv[1], ios::in|ios::nocreate);
   if(!in){
      cout << "Cannot open file" << argv[1] << endl; return 1;
   }
   char c;
   while(in.get(c)){
      cout << c; CheckStatus(in);
   }
   CheckStatus(in);    // соңғы күйді қадағалау
   in.close();
   return 0;
}
void CheckStatus(ifstream &in){
   int i;
   i = in.rdstate();
   if(i & ios::eofbit)
      cout << "EOF is occured" << endl;
   else if(i & ios::failbit)
      cout << "Not fatal input/output error" << endl;
   else if(i & ios::badbit)
      cout << "Fatal input/output error" << endl;
}
```

Файлдық ағымдар

Әдетте файл ретінде сыртқы мәлімет жинақтауышта сақталған белгілі бір аты бар ақпаратты айтады, мысалы, қатты немесе иілгіш магниттік дискідегі мәліметтер жиыны. Логикалық тұрғыдан алғанда, файлды тізбектелген байттардың шектеулі саны ретінде көрсетуге болады, сондықтан дисплей, пернетақта және принтер сияқты құрылғылар да файлдың дербес бір түрі ретінде қарастырылады.

Мәліметтеріне қол жеткізу тәсіліне қарай файлдарды *тізбекті* және *кездейсоқ қол жеткізу* файлдары деп екіге бөлуге болады. Тізбекті файлға

мәлімет жазу мен оқу оның басынан бастап байтқа байт тіркеле орналастырылып жүргізіледі, ал екінші түрдегі файлдар мәліметтерді өзіміз көрсеткен позицияға жазу мен оқу мүмкіндігін береді.

Стандартты кітапханада файлдармен жұмыс істеуге арналған үш класс бар, олар:

ifstream — кіріс файлдық ағымдар класы;
ofstream — шығыс файлдық ағымдар класы;
fstream — екібағытты файлдық ағымдар класы.

Бұл кластар сәйкесінше **istream**, **ostream** және **iostream** кластарынан шыққан туынды кластар болып табылады, сондықтан олар асыра жүктелген **<<** және **>** операцияларын, форматтау жалаушаларын, манипуляторларды, әдістерді, ағымдар қалып-күйлерін және т.б. мұралайды.

Программада файлдарды қолдану кезінде төмендегі операциялар қажет етіледі:

- ағымды құру;
- ағымды ашу және оны файлмен байланыстыру;
- мәлімет алмасу (енгізу/шығару);
- файлды жабу.

Файлдық ағымдардың әрбір класында конструкторлар бар, солардың көмегімен осы класс объектілерін әртүрлі тәсілдермен құруға болады.

- Параметрсіз конструкторлар соларға сәйкес кластың объектісін оны файлмен байланыстырмай құрады:

```
ifstream();
ofstream();
fstream();
```

- Параметрлі конструкторлар өздеріне сәйкес класс объектісін құрады да, аты көрсетілген файлды ашады және оны объектімен байланыстырады:

```
ifstream(const char *name, int mode = ios::in);
ofstream(const char *name, int mode = ios::out |
ios::trunc);
fstream(const char *name, int mode = ios::in | ios::out);
```

Конструктодың екінші параметрі файлды ашу режимін (тәртібі) орнатады. Егер алдын ала келісім бойынша орнатылған мән программалаушының көңілінен шықпаса, онда **ios** класында анықталған биттік маскалардан олардың басқасын құруға да болады:

```
enum open_mode{
   in = 0x01,          // Оқу үшін ашу
   out = 0x02,         // Жазу үшін ашу
   ate = 0x04,         // Нұсқауышты файлдың соңына орнату
   app = 0x08,         // Файл соңына мәлімет қосу үшін ашу
   trunc = 0x10,       // Егер файл бар болса, оны жою
```

```
    nocreate = 0x20,      // Егер файл жоқ болса,
                          // қатені шығару
    noreplace = 0x40,     // Егер файл бар болса,
                          // қатені шығару
    binary = 0x80,        // Файлды екілік режимде ашу
};
```

10.2-кестеде **ios** класының биттік маскалары мен **<stdio.h>** файлында сипатталған файлды ашу режимдерінің (тәртіптерінің) арасындағы сәйкестік көрсетілген.

10.2-кесте. Файлды ашу режимдері

ios жалаушалары комбинациясы					**stdio эквиваленті**
binary	**in**	**out**	**trunc**	**app**	
		+			"w"
		+		+	"a"
		+	+		"w"
	+				"r"
	+	+			"r+"
	+	+	+		"w+"
+		+			"wb"
+		+		+	"ab"
+		+	+		"wb"
+	+				"rb"
+	+	+			"r+b"
+	+	+	+		"w+b"

Программада файл ашу үшін констркторды немесе соған сәйкес конструкторға тән параметрлерге ие болатын **open** әдісін қолдануға болады, мысалы:

```
// Конструкторды қолдану
ifstream inpf ("input.txt", ios::in|ios::nocreate);
if (!inpf){
   cout << "Файлды мәлімет оқу үшін ашу мүмкін емес";
   return 1;
}
ofstream f;
f.open("output.txt");        // open әдісін қолдану
if (!f){
   cout << "Файлды мәлімет жазу үшін ашу мүмкін емес";
   return 1;
}
```

Жазу мен оқу ағымдық кластардағы сияқты мәлімет оқу және оны шығарып алу операцияларының көмегімен немесе кластар әдістері арқылы орындалады.

Әдістерді қолдану мысалы (программа экранға файлдың ішкі мәліметін шығарып береді):

```
#include <fstream.h>
int main(){
   char text[81], buf[81];
   cout << "Файлдың атын енгізіңіз:";
   cin >> text;
   ifstream f(text, ios::in|ios::nocreate);
   if (!f){
      cout << "Файлды ашудағы қателік"; return 1;
   }
   while (!f.eof()){
      f.getline(buf, 81);
      cout << buf << endl;
   }
   return 0;
}
```

Ағымды жабу үшін **close()** әдісі анықталған, алайда оны деструктор жанамалы (нақты емес) түрде орындайтын болғандықтан, ағымды оның көріну аумағының соңына жеткенге дейін жабу қажет болған кезде тікелей шақыру қолданылады.

Тіркестік ағымдар

Тіркестік ағымдар ақпаратты жедел жады аймақтарынан дәл файлмен, консольмен немесе дисплеймен жұмыс істегендегідей түрде оқуға және жазуға мүмкіндік береді. Стандартты кітапханада тіркестік ағымдардың үш класы анықталған[1]:

istringstream — кіріс тіркестік ағымдары;
ostringstream — шығыс тіркестік ағымдары;
stringstream — екібағытты тіркестік ағымдар.

Бұл кластар **<sstream>**[2] тақырыптық файлында анықталады және олар сәйкесінше **istream**, **ostream** және **iostream** кластарынан шыққан туынды кластар болып табылады, сондықтан олар асыра жүктелген **<<** және **>>** операцияларын, форматтау жалаушаларын, манипуляторларды, әдістерді, ағымдардың қалып-күйлерін және т.б. мұралайды.

Мәліметтерді оқу және шығарып алу операциялары орындалатын жады аймақтары стандарт бойынша C++ тіркестері (**string** класы) ретінде

[1]Компиляторлардың ескі нұсқаларында бұл кластардың атаулары басқаша болуы мүмкін.

[2]Компиляторлардың ескі нұсқаларында тақырыптық файлдың атауы <strstrea.h> болуы мүмкін.

анықталады. Тіркестік класс 307 б. қарастырылады. Тіркестік ағымдар құрылады да, осы жады аймақтарымен келесі конструкторлар арқылы байланыстырылады:

```
explicit[1] istringstream(int mode = ios::in);
explicit istringstream(const string& name,
                       int mode =  ios::in);
explicit ostringstream(int mode = ios::out);
explicit ostringstream(const string& name,
                       int mode = ios::out);
explicit stringstream(int mode = ios::in | ios::out);
explicit stringstream(const string& name,
                      int mode = ios::in | ios::out);
```

Тіркестік ағымдар С кітапханасының **sscanf** және **sprintf** функцияларына ұқсас болып келеді және мәліметтер қандай да бір жады аймағына енгізіліп, содан кейін қажетті типтегі шамаларға оқылған кезде оларды түрлендіру үшін қолданылуы мүмкін. Бұл ағымдар, сонымен қатар, программаның модульдері арасында ақпарат алмасу үшін қолданылуы мүмкін.

Тіркестік ағымдарда тіркестің көшірмесін қайтаратын немесе оның мәнін орнататын **str** әдісі сипатталған:

```
string str() const;
void str(const string & s);
```

Тіркестік ағымның толып кетуін тексерудің қажеттілігі жоқ, өйткені тіркестің мөлшері динамикалық түрде өзгереді.

Төменде көрсетілген мысалда тіркестік ағым хабарлама қалыптастыру үшін қолданылады, хабарлама құрамында ағымдағы уақыт және параметр ретінде берілетін нөмір болады:

```
#include <sstream>
#include <string>
#include <iostream>
#include <ctime>
using namespace std;
string message( int i){
   ostringstream os;
   time_t t;
   time(&t);
   os << " time: " << ctime(&t) << " number: " << i << endl;
   return os.str();
}
int main(){
   cout << message(22); return 0;
}
```

[1] 213 б. қараңыз.

Қолданушы анықтаған ағымдар мен типтер

Ағымдардан мәлімет енгізу мен шығару мақсатында барлық стандартты типтер үшін асыра жүктелген оқу және шығарып алу әрекеттері, яғни **<<** және **>>** операциялары қолданылады. Мұндайда керекті операцияның таңдалуы нақты параметрлер типімен анықталады. Қолданушы анықтаған типтердегі шамаларды енгізу және шығару үшін осы операцияларды асыра жүктеу қажет. Бұл – сол жақ операндысы объект-ағым, ал оң жақ операндысы осы ағымнан шығарып алу немесе оған орналастыру талап етілетіндей бинарлы операция болып табылады. Қайтарылатын мән ағымға сілтеме болуы тиіс, осылайша стандартты типтердегі сияқты операциялар тізбегін ұйымдастыруға болады.

Мысалы, программада **MyClass** класы анықталсын делік:

```
class MyClass{
   int   x;
   float y;
   ...
}
```

Осы кластың объектілерін енгізу және шығару үшін **MyClass** класында келесі түрдегі операцияларды анықтау талап етіледі:

```
// Шығару:
friend ostream& operator << (ostream& out, MyClass& C){
   return out << "x = " << C.x <<  y = " << C.y;
}
// Енгізу:
friend istream& operator >> (istream& in, MyClass& C){
   cout << "x енгізіңіз: "; in >> C.x;
   cout << "y енгізіңіз: "; in >> C.y;
   return in;
}
```

Осыдан кейін программадағы енгізу және шығару операцияларында стандартты тип шамаларымен қатар **MyClass** класының объектілерін де қолдануға болады:

```
#include <iostream.h>
class MyClass{
      int   x;
      float y;
   public:
      MyClass(int nx = 1, float ny = 0.01){
         x = nx; y = ny;}
      friend ostream& operator << (ostream& out,
      MyClass& C){
         return out << "x = " << C.x << " y = " << C.y;
      }
```

```
      friend istream& operator >> (istream& in,
      MyClass& C){
         cout << "x енгізіңіз: "; in >> C.x;
         cout << "y енгізіңіз: "; in >> C.y;
         return in;
      }
};
int main(){
   MyClass C;
   cout << C << endl;
   MyClass C1(100, 100);
   cout << C1 << endl;
   MyClass C2;
   cin >> C2; cout << C2 << endl;
   return 0;
}
```

MyClass класының **C** экземпляры (данасы) келісім бойынша тағайындалған конструктордың параметрлері арқылы құрылады, сондықтан экранға келесі нәтиже шығарылады:

```
x = 1 y = 0.01
```

MyClass класының **C1** данасы 100, 100 параметрлерімен құрылады:

```
x = 100 y = 100
```

C2 экземпляры (данасы) құрылғаннан кейін **x** және **y** шамаларын енгізуді ұсынатын мәлімет шығарылады, осыдан соң барып пернетақтадан енгізілген мәндер экранға шығарылады.

Стандартты ағымдар қолданылған программаларды қарастыруға практикумның [12] 4-ші және 5-ші семинарлары арналған.

11-ТАРАУ

Тіркестер

Стандартты кластар жай ғана
C++ құрамына енгізілмейді.
Г. Шилдт

C++ тілінде сөз тіркестерін бейнелейтін «тіркес» стандартты мәліметтер типі жоқ. Оның орнына C++ тілі нөлсимволмен аяқталатын символдар жиымын сүйемелдейді. Кітапханада осындай жиымдармен жұмыс істеуге арналған **<string.h>** (**<cstring>**) тақырыптық файлында сипатталған және C тілінен мұраланған функциялар бар. Осы функциялар туралы «Тіркестермен және символдармен жұмыс істеу функциялары» бөлімінде, 97 б. жазылған. Олар жоғарғы тиімділікке қол жеткізуге мүмкіндік береді, бірақ қолдануға аса қолайсыз және сенімсіз болып келеді, өйткені символдардың тіркес шекарасынан шығу мүмкіндігі тексерілмейді.

Стандартты кітапхананың **string** мәліметтер типінде мұндай кемшіліктер жоқ, бірақ ол тиімділігі жағынан символдар жиымынан төмен қалуы мүмкін. Тіркестермен орындалатын негізгі әрекеттер операциялар мен әдістердің көмегімен атқарылады, ал тіркестің ұзындығы қажеттілігіне қарай динамикалық түрде өзгеріп отырады. Класты қолдану үшін программаға **<string>** тақырыптық файлын[1] қосу қажет. Енді мысал қарастырайық:

```
#include <cstring>
#include <string>
#include <iostream>
using namespace std;
int main (){
   char c1[80], c2[80], c3[80]; // Аяқтаушы нөлі
                                //бар тіркестер
   string s1, s2, s3;
   // Тіркестерді меншіктеу
   strcpy(c1, "old string one");
   strcpy(c2, c1);
   s1 = "new string one";
   s2 = s1;
   // Тіркестер конкатенациясы
   strcpy(c3, c1);
   strcpy(c3, c2);
   s3 = s1 + s2;
```

[1]Компиляторлардың ескі нұсқаларында **<string>** класын қосуға арналған тақырыптық файлдың атауы **cstring.h** немесе **bstring.h**, ал ескі стильдегі тақырыптық файлдардың атаулары **string.h** болуы мүмкін.

```
    // Тіркестерді салыстыру
    if (strcmp(c2, c3) < 0 ) cout << c2;
    else cout << c3;
    if (s2 < s3) cout << s2;
    else cout << s3;
}
```

Бұл мысалдан байқап отырғанымыздай, ескі стильдегі тіркестермен кез келген әрекеттерді орындау функцияларды қолдануды талап етеді және олар айтарлықтай көрнекті болмайды. Сонымен қатар, көшіру барысында қабылдаушы-тіркесте орын жеткілікті болуын тексеру керек, яғни ескі стильдегі тіркестермен жұмыс істеудің нақты коды ұзынырақ болуы тиіс.

string типті тіркестер ақпараттың жиым шекараларынан шығып кетуінен қорғалған және олармен кез келген құрамдас мәліметтер типімен жұмыс істегендей түрде, яғни операциялар көмегімен жұмыс істей беруге болады. Енді тіркестермен жұмыс істеудің негізгі ерекшеліктері мен тәсілдерін қарастырамыз.

Конструкторлар және тіркестерді меншіктеу

string класында бірнеше конструкторлар анықталған. Төменде жиі қолданылатын конструкторлардың тақырыптары қарапайым түрде көрсетілген:

```
string();
string(const char *);
string(const char *, int n);
string(string &);
```

Бірінші конструктор **string** типті бос объектіні құрады. Екіншісі **string** типті объектіні ескі стильдегі тіркес негізінде құрады, ал үшінші конструктор **string** типті объектіні құрады да, оған бірінші параметр ретінде көрсетілген тіркестен **n** символды көшіріп жазады. Соңғы конструктор көшіру конструкторы болып табылады, ол жаңа объектіні өзіне параметр ретінде берілген объект көшірмесі ретінде құрады.

string класында үш **меншіктеу операциясы** анықталған, олар:

```
string& operator=(const string& str);
string& operator=(const char* s);
string& operator=(char c);
```

Тақырыптарынан көрініп тұрғандай, тіркеске **string** типіндегі басқа бір тіркесті, ескі стильдегі тіркесті немесе жеке символды меншіктеуге болады екен, мысалы:

```
string s1;
string s2("Вася");
```

```
string s3(s2);
s1 = 'X';
s1 = "Вася";
s2 = s3;
```

Операциялар

Төменде **string** класының объектілері үшін қол жеткізуге (орындауға) болатын операциялар көрсетілген:

Операция	Әрекеті	Операция	Әрекеті
=	меншіктеу	>	үлкен
+	конкатенация	>=	үлкен немесе тең
==	теңдік	[ ]	индекстеу
! =	теңсіздік	<<	шығару
<	кіші	>>	енгізу
<=	кіші немесе тең	+=	қосу

Бұл операциялардың синтаксисі мен әрекеттері бұрын қарастырылғандардай сияқты айқын болып келеді. Тіркестердің өлшемдері автоматты түрде объект өзіне меншіктелген мәнді қабылдай алатындай болып орнатылады. Ескере кететін жай: **string** типті тіркестер үшін ескі стильдегі тіркестер жағдайында орындалғандай, тіркестің алғашқы элементі мен атауы арасындағы сәйкестік сақталмайды, яғни **&s[0]** және **s** бірдей емес.

Индекстеу операциясынан басқа, тіркес элементіне қол жеткізу үшін **at** функциясы анықталған:

```
string s("Вася");
cout << s.at(1);        // а символы шығарылады
```

Егер индекс тіркестің ұзындығынан артық болса, **out_of_range** аластамасы туындайды.

Тіркестермен толықтай жұмыс істеу үшін осы операциялар жеткілікті, ал тіркес бөліктерін өңдеу (мысалы, ішкі тіркесті іздеу, тіркеске кірістіру, символдарды өшіру) үшін **string** класында көптеген әдістер (функциялар) анықталған, олардың ішіндегі көбінесе кең қолданылатындары келесі бөлімде қарастырылады.

Функциялар

string класының функцияларын қарастыру ыңғайлы болуы үшін оларды келесідей санаттарға (категорияларға) бөлуге болады: тіркес бөліктерін меншіктеу және қосу, тіркестерді түрлендірулер, ішкі тіркестерді іздеу, салыстыру және тіркестердің сипаттамаларын алу.

Меншіктеу және тіркес бөліктерін қосу

Бір тіркестің ішкі бөлігін екінші бір тіркеске меншіктеу үшін **assign** функциясы қолданылады:

```
assign(const string& str);
assign(const string& str, size_type pos, size_type n);
assign(const char* s, size_type n);
```

Бұл функциялардың бірінші формасы **str** тіркесін оны шақырушы тіркеске меншіктейді, мұндайда функцияның әрекет етуі меншіктеу операциясына эквивалентті болып табылады:

```
string s1("Вася"), s2;
s2.assign(s1);              // s2 = s1 өрнегімен бірдей
```

Функциялардың екінші формасы **str** тіркесінің бір бөлігін оны шақырушы тіркеске **pos** позициясынан бастап меншіктейді[1]. Егер **pos** мәні тіркес ұзындығынан артық болса, онда **out_of_range** аластамасы туындайды. Шақыратын тіркеске **n** символ меншіктеледі, немесе, егер **pos+n** шамасы **str** тіркесінің ұзындығынан артық болса, онда **str** тіркесінің соңына дейінгі барлық символдар меншіктеледі.

Функциялардағы үшінші форма шақырушы тіркеске ескі типтегі **s** тіркесінің **n** символын меншіктейді.

Бір тіркестің ішкі бөлігін басқа бір тіркеске тіркеп қосу үшін **append** функциясы қолданылады, олар:

```
append(const string& str);
append(const string& str. size_type pos, size_type n);
append(const char* s, size_type n);
```

Функцияның бірінші формасы **str** тіркесін шақырушы тіркестің соңына тіркейді, мұндағы функцияның әрекет етуі конкатенация (**+**) операциясына ұқсас, яғни соған эквивалентті болып табылады.

Екінші формасы шақырушы тіркеске **str** тіркесінің бір бөлігін **pos** позициясынан бастап тіркейді. Егер **pos** мәні тіркестің ұзындығынан артық болса, **out_of_range** аластамасы туындайды. Шақыратын тіркеске **n** символ меншіктеледі, немесе, егер **pos+n** шамасы **str** тіркесінің ұзындығынан артық болса, **str** тіркесінің соңына дейінгі барлық символдар меншіктеледі. Егер нәтиженің ұзындығы тіркестің мүмкін болатын максималды ұзындығынан артық болатын болса, **length_error** аластамасы туындайды.

Үшінші форма шақырушы тіркеске ескі типтегі **s** тіркесінің **n** символын қосып тіркейді.

---

[1] **size_type** берілген модель үшін ең үлкен объектінің өлшемін сақтауға жеткілікті таңбасыз бүтін тип болып табылады.

Тіркестерді түрлендіру

Бір тіркеске екінші тіркестің белгілі бір бөлігін кірістіру үшін **insert** функциясы қолданылады, олар:

```
insert(size_type pos1, const string& str);
insert(size_type pos1, const string& str,
                          size_type pos2, size_type n);
insert(size_type pos, const char* s, size_type n);
```

Функцияның бірінші формасы шақырушы тіркеске оның **pos1** позициясынан бастап **str** тіркесін кірістіреді. Басқаша айтқанда, шақырушы тіркес жаңа тіркеске алмастырылады, бұл тіркестің алғашқы **pos1** символы шақырушы тіркес символдарынанан тұрады, олардың артынан **str** тіркесі толығымен орналасады, ал одан кейін шақырушы тіркестің қалған символдары жазылады. Егер **pos1** мәні тіркестің ұзындығынан артық болса, онда **out_of_range** аластамасы, ал егер нәтиженің ұзындығы тіркестің мүмкін болатын максималды ұзындығынан артық болатын болса, онда **length_error** аластамасы туындайды.

Функцияның екінші формасы шақырушы тіркеске **str** тіркесін шақырушы тіркестің **pos1** позициясынан бастап кірістіреді. Шақырушы тіркесті алмас-тыратын жаңа тіркес шақырушы тіркестің алғашқы **pos1** символдарынан, олардың соңынан жазылатын **str** тіркесінің **pos2** позициясынан басталатын **n** элементінен және олардан кейін орналасатын шақырушы тіркестің қалған символдарынан тұрады. Егер **n** саны **str** тіркесінің ұзындығынан артық болса, онда **str** тіркесі соңына дейін көшіріледі. Егер **pos1** немесе **pos2** мәндері сәйкес тіркесің ұзындығынан артық болса, онда **out_of_range** аластамасы, ал нәтиженің ұзындығы тіркестің мүмкін болатын максималды ұзындығынан артық болған жағдайда, **length_error** аластамасы туындайды.

Функцияның үшінші формасы шақырушы тіркеске ескі типтегі **s** тіркесінің **pos** позициясынан басталатын **n** символын кірістіреді.

Тіркестің бір бөлігін өшіру үшін **erase** функциясы қолданылады:

```
erase(size_type pos = 0, size_type n = npos);
```

Ол шақырушы тіркестің **pos** позициясынан басталатын **n** элементті өшіреді. Егер **pos** мәні көрсетілмесе, элементтер тіркестің басынан бастап өшіріледі. Егер **n** шамасының мәні көрсетілмеген болса, тіркес соңына дейін (тіркестің қалған бөлігі түгелдей) өшіріледі.

ЕСКЕРТУ

---

npos шамасы **string** класының статикалық мүшесі болып және **size_type** типінің ең үлкен оң саны (барлығы бірліктер биттік бейнелеуде) болып табылады.

Тіркесті толығымен тазартуды **clear** функциясының көмегімен жүзеге асыруға болады:

```
void clear();
```

Тіркестің бір бөлігін басқамен алмастыру үшін **replace** функциясы қолданылады:

```
replace(size_type pos1, size_type n1, const string& str);
replace(size_type pos1, size_type n1, const string& str,
                        size_type pos2, size_type n2);
```

Мұндағы **pos1** – шақырушы тіркестің алмастыру басталатын позициясы, **n1** – өшірілетін элементтер саны, **pos2** – шақырушы тіркеске кірістіру басталатын **str** тіркесінің ішкі позиция нөмірі, **n2** – **str** тіркесінің кірістірілетін элементтер саны. Егер **pos1** немесе **pos2** мәні соған сәйкес тіркес ұзындығынан артық болса, **out_of_range** аластамасы, ал нәтиженің ұзындығы тіркестің мүмкін болатын максималды ұзындығынан артық болған жағдайда, **length_error** аластамасы туындайды.

Алмастыру функциясының үшінші формасы шақырушы тіркестің қатарласа орналасқан **n1** символын ескі стильдегі **s** тіркесінің **n2** символдар тіркесіне алмастыруға мүмкіндік береді:

```
replace(size_type pos1, size_type n1, const char* s,
                                       size_type n2);
```

Екі тіркестің құрамындағы символдарды өзара алмастыру үшін **swap** функциясы қолданылады:

```
swap(string& s);
```

Тіркестің бір бөлігін ерекшелеп белгілеп алу үшін **substr** функциясы қолданылады:

```
string substr(size_type pos = 0, size_type n = npos) const;
```

Бұл функция ұзындығы **n**-ге тең болатын шақырушы тіркестің **pos** позициясынан басталатын ішкі тіркесті қайтарады. Егер **pos** мәні тіркестің ұзындығынан артық болса, **out_of_range** аластамасы туындайды. Егер **n** саны тіркестің ұзындығынан артық болатын болса, онда тіркестің қалған бөлігі толықтай қайтарылады.

Кейбір жағдайларда **string** типті объектілерді ескі стильдегі тіркестерге түрлендіру қажет болады. Бұл үшін **c_str** функциясы қолданылады:

```
const char* c_str() const;
```

Бұл функция нөл-символмен аяқталатын тіркеске тұрақтылық нұсқауышты қайтарады. Бұл тіркесті өзгертуге болмайды. Оған сілтеме жасайтын нұсқауыш бастапқы тіркеспен кез келген бір операция орындағаннан кейін ол дұрыс болмай қалуы мүмкін. **data** функциясы да осыған ұқсас түрде жұмыс жасайды, бірақ оның ерекшелігі – тіркестің соңына нөл-символды қоспайды.

```
const char* data() const;
```

Келесі **copy** функциясы **s** жиымына шақырушы тіркестің **pos** позициясынан бастап **n** элементін көшіріп жазады. Нөл-символ нәтижелік жиымға енгізілмейді. Функция көшірілген элементтер санын қайтарады:

```
size_type copy(char* s, size_type n, size_type pos = 0) const;
```

Тіркестердің құрамын өзгерту функцияларын қолдану мысалы:

```
#include <string>
#include <iostream>
using namespace std;
int main (){
   string s1("прекрасная королева"), s2("ле"), s3("корова");
   cout << "s1 = "<< s1 << endl;
   cout << "s2 = "<< s2 << endl;
   cout << "s3 = "<< s3 << endl;
   // insert функциясының қолданылуы:
   cout << "insert қолданудан кейін:" << endl;
   cout << "s3 = " << s3.insert(4, s2) << endl;
   cout << "s3 = " << s3.insert(7, "к") << endl;
   // erase функциясының қолданылуы:
   s1.erase(0,3);
   cout << "erase қолданудан кейін:" << endl;
   cout << "s1 = " << s1.erase(12,2) << endl;
   // replace функциясының қолданылуы:
   cout << "replace қолданудан кейін:" << endl;
   cout << "s1 = " << s1.replace(0,3, s3, 4,2) << endl;
}
```

Программаның орындалу нәтижесі:

```
s1 = прекрасная королева
s2 = ле
s3 = корова
insert қолданудан кейін:
s3 = королева
s3 = королевка
erase қолданудан кейін:
s1 = красная корова
replace қолданудан кейін:
s1 = лесная корова
```

Ішкі тіркестерді іздеу

Іздеу мақсаттары үшін **string** класында алуан түрлі функциялар қарастырылған. Төменде солардың негізгілері көрсетілген:

```
size_type find(const string& str, size_type pos = 0) const;
```

Бұл функция шақырушы тіркестің сол жақ шетіндегі **pos** позициясынан бастап **str** тіркесінің символдары тіркесін іздейді де, табылған алғашқы позиция нөмірін қайтарады, ал тіркес табылмаған жағдайда **npos** мәнін қайтарады.

```
size_type find(char c, size_type pos = 0) const;
```

Бұл функция да шақырушы тіркестің сол жақ шетіндегі **pos** позициясынан бастап **c** символын іздейді де, табылған алғашқы позиция нөмірін қайтарады, ал символ табылмаған жағдайда **npos** мәнін қайтарады.

```
size_type rfind(const string& str,
                size_type pos = npos) const;
```

Бұл функция шақырушы тіркестің оң жақ шетіндегі **pos** позициясына дейін **str** тіркесінің символдары тіркесін іздейді де, табылған алғашқы позиция нөмірін қайтарады, ал тіркес табылмаған жағдайда **npos** мәнін қайтарады.

```
size_type rfind(char c, size_type pos = npos) const;
```

Бұл функция да шақырушы тіркестің оң жақ шетіндегі **pos** позициясына дейін **c** символын іздейді де, табылған позиция нөмірін қайтарады, ал символ табылмаған жағдайда **npos** мәнін қайтарады.

```
size_type find_first_of(const string& str,
                        size_type pos = 0) const;
```

Бұл функция да шақырушы тіркестің сол жақ шетіндегі **pos** позициясынан бастап, **str** тіркесінің кез келген символын іздейді де, табылған позиция нөмірін қайтарады, ал символ табылмаған жағдайда **npos** мәнін қайтарады.

```
size_type find_first_of(char c, size_type pos = 0) const;
```

Бұл функция шақырушы тіркестің сол жақ шетіндегі **pos** позициясынан бастап, **c** символын іздейді де, табылған позиция нөмірін қайтарады, ал символ табылмаған жағдайда **npos** мәнін қайтарады.

```
size_type find_last_of(const string& str,
                       size_type pos = npos) const;
```

Бұл функция шақырушы тіркестің оң жақ шетіндегі **pos** позициясына дейін, **str** тіркесінің кез келген символын іздейді де, табылған позиция нөмірін қайтарады, ал символ табылмаған жағдайда **npos** мәнін қайтарады.

```
size_type find_last_of(char c, size_type pos = npos) const;
```

Бұл функция шақырушы тіркестің оң жақ шетіндегі **pos** позициясына дейін, **c** символын іздейді де, табылған позиция нөмірін қайтарады, ал символ табылмаған жағдайда **npos** мәнін қайтарады.

```
size_type find_first_not_of(const string& str,
                            size_type pos = 0) const;
```

Шақырушы тіркестің сол жақ шетіндегі **pos** позициясынан бастап **str** тіркесінің ешбір символы шақырушы тіркес символдарымен сәйкес келмейтін позицияны іздейді.

```
size_type find_first_not_of(char c,
                            size_type pos = 0) const;
```

Шақырушы тіркестің сол жақ шетіндегі **pos** позициясынан бастап, **c** символы шақырушы тіркес символдарымен сәйкес келмейтін позицияны іздейді.

```
size_type find_last_not_of(const string& str,
                           size_type pos = npos) const;
```

Шақырушы тіркестің оң жақ шетіндегі **pos** позициясына дейін **str** тіркесінің ешбір символы шақырушы тіркес символдарымен сәйкес келмейтін оң жақта орналасқан позицияны іздейді.

```
size_type find_last_not_of(char c,
                           size_type pos = npos) const;
```

Шақырушы тіркестің оң жақ шетіндегі **pos** позициясына дейін **c** символы шақырушы тіркес символдарымен сәйкес келмейтін оң жақта орналасқан позицияны іздейді.

Әрбір функция үшін берілген тіркесте ескі стильдегі ішкі тіркестерді іздеуге мүміндік беретін нұсқа болады.

Іздеу функцияларын қолдану мысалы:

```
#include <string>
#include <iostream>
using namespace std;
int main (){
   string s1("лесная королева"), s2("ле");
   cout << "s1 = " << s1 << endl;
   cout << "s2 = " << s2 << endl;
   int i = s1.find(s2);
   int j = s1.rfind(s2);
   cout << "s1 құрамындағы алғашқы s2 " << i << endl;
   cout << "s1 құрамындағы соңғы s2 " << j << endl;
   cout << "s1 құрамындағы алғашқы 'o' "
        << s1.find('o') << endl;
   cout << "s1 құрамындағы соңғы 'o' "
        << s1.rfind('o') << endl;
   cout << "s1 құрамындағы алғашқы "
        << s1.find_first_of("adcde") << endl;
   cout << "s1 құрамындағы соңғы "
        << s1.find_last_of("adcde") << endl;
   }
```

Программа жұмысының нәтижесі:

```
s1 = лесная королева
s2 = ле
s1 құрамындағы алғашқы s2 0
s1 құрамындағы соңғы s2 11
s1 құрамындағы алғашқы 'о' 8
s1 құрамындағы соңғы 'о' 10
s1 құрамындағы алғашқы 1
s1 құрамындағы соңғы 14
```

Тіркестердің бөліктерін салыстыру

Тіркестерді бүтіндей толық салыстыру үшін асыра жүктелген қатынастық операциялар қолданылады, ал егер тіркестердің бөліктерін салыстыру керек болса, **compare** функциясы қолданылады:

```
int compare(const string& str) const;
int compare(size_type pos1, size_type n1,
            const string& str) const;
int compare(size_type pos1, size_type n1, const string& str,
            size_type pos2, size_type n2) const;
```

Функцияның бірінші формасы екі тіркесті бүтіндей толығымен салыстырып, шақырушы тіркес лексикографиялық тұрғыдан **str** тіркесінен кіші болса, нөлден кіші мәнді, тіркестер бірдей болса, нөлге тең мәнді және шақырушы тіркес үлкен болған жағдайда нөлден үлкен мәнді қайтарады. Бұл форма С кітапханасындағы **strstr** тіркестерді салыстыру функциясының аналогы болып табылады.

Функцияның екінші формасы да осыған ұқсас әрекеттер орындайды, бірақ **str** тіркесімен шақырушы тіркестің **pos1** позициясынан басталатын қатар орналасқан **n1** симолдар санын салыстырады.

Функцияның үшінші формасы шақырушы тіркестің **pos1** позициясынан басталатын **n1** симолын **str** тіркесінің **pos2** позициясынан басталатын, ұзындығы **n2** болатын ішкі тіркесімен салыстырады.

Функциялардың осыған ұқсас формалары **string** типті тіркестерді ескі стильдегі тіркестермен салыстыру үшін қолданылады.

Тіркестерді салыстыру функциясын қолдану мысалы:

```
#include <string>
#include <iostream>
using namespace std;
int main (){
   string s1("лесная королева"),s2("ле"),s3("корова");
   cout << "s1 = " << s1 << endl;
   cout << "s2 = " << s2 << endl;
   cout << "s3 = " << s3 << endl;
```

```
    if (s2.compare(s3) > 0) cout << "s2 > s3 " << endl;
    if (s1.compare(7, 4, s3) < 0)
    cout << "1[7-10] < s3 "  << endl;
    if (s1.compare(7, 4, s3, 0,4 ) == 0)
    cout << "s1[7-10] == s3[0-3] " << endl;
}
```

Программаның орындалу нәтижесі:

```
s1 = лесная королева
s2 = ле
s3 = корова
s2 > s3
s1[7-10] < s3
s1[7-10] == s3[0-3]
```

Тіркестердің сипаттамаларын алу

string класында объектіге жұмсалған жады көлемі мен тіркестің ұзындығын алуға мүмкіндік беретін бірқатар функция-мүшелер анықталған:

```
size_type size() const;    // Тіркес элементтерінің саны
size_type length() const;  // Тіркес элементтерінің саны
size_type max_size() const;  // Тіркестің максималды
                             // ұзындығы
size_type capacity() const;  // Тіркестің алатын
                             // жады көлемі
bool empty() const;          // Тіркес бос болса,
                             // ақиқат мәнін қайтарады
```

Тіркестермен жұмыс практикумның [12] бесінші семинарында қарастырылады.

12-ТАРАУ

Контейнерлік кластар

Контейнерлік кластар – бұл белгілі бір жолмен ұйымдастырылған мәліметтерді сақтауға арналған кластар. Контейнерлерге мысал ретінде жиымдарды, сызықтық тізімдерді немесе стектерді алуға болады. Контейнердің әрбір типі үшін мұнда сақталатын нақты мәліметтердің типінен тәуелсіз болатын және оның элементтерімен жұмыс істеуге арналған әдістер анықталған, сондықтан контейнердің бір түрін әртүрлі типтегі мәліметтерді сақтау үшін қолдануға болады. Бұл мүмкіндік класс шаблондарының көмегімен жүзеге асырылған, сондықтан контейнерлік кластармен қатар, келесі бөлімдерде қарастырылатын алгоритмдер мен итераторлар енгізілген C++ кітапханасының бөлігі шаблондардың стандартты кітапханасы (STL — Standard Template Library) деп аталады.

Контейнерлерді қолдану программа сенімділігін, бір платформадан екіншісіне көшірілу мүмкіндігін және әмбебаптылығын арттыруға, сонымен қатар оларды құру мерзімін қысқартуға мүмкіндік береді.

Әрине, мұндай артықшылықтар тегін берілмейді: конейнерлік кластарды қолданудың әмбебаптылығы мен қауіпсіздігі программаның жылдамдығына әсер етпеуі мүмкін емес. Компилятордың да нұсқаларына байланысты жылдамдықтың да төмендеуі әжептеуір болуы да мүмкін. Сонымен қатар, контейнерлерді тиімді пайдалану үшін кітапхананы саналы түрде меңгеруге де көп еңбек жұмсау керек.

STL құрамында программа жазуда қолданылатын негізгі мәлімет құрылымдарын жүзеге асыратын контейнерлер бар, олар: *векторлар, екіжақты кезектер, тізімдер мен олардың алуан түрлері, сөздіктер және жиындар.* Контейнерлерді тізбектік және ассоциативті деп екі типке бөлуге болады.

Тізбектік контейнерлер біртипті шамалардың шекті санын үздіксіз тізбек түрінде сақтауды қамтамасыз етеді. Оларға векторлар (**`vector`**), екіжақты кезектер (**`deque`**) және тізімдер (**`list`**), сонымен қатар адаптерлер, яғни контейнерлер нұсқалары – стектер (**`stack`**), кезектер (**`queue`**) және басымдылықтары бар кезектер (**`priority_queue`**) жатады.

Контейнердің әрбір түрі мәліметтерге қолданылатын өзіндік әрекеттер жиынтығын орындауды қамтамасыз етеді. Контейнердің түрін таңдау программадағы мәліметтермен орындалатын әрекеттерге байланысты болады. Мысалы, тізбектің ортасына элементтерді жиі кірістіру немесе оларды өшіру қажет болса, тізімді қолданған жөн, ал егер элементтерді негізінен тізбек соңына немесе басына қосу орындалатын болса, онда екіжақты кезекті қолданған дұрыс болып табылады.

Ассоциативті контейнерлер мәліметтерге кілт арқылы жылдам қол жеткізуді қамтамасыз етеді. Мұндай контейнерлер теңдестірілген бұтақтар негізінде құрылған. Ассоциативті контейнерлердің бес түрлі типі бар: сөздіктер (**map**), дубликатты (көшірмелі) сөздіктер (**multimap**), жиындар (**set**), дубликатты (көшірмелі) жиындар (**multiset**) және биттік жиындар (**bitset**).

Программалаушы стандартты кітапханадағы контейнерлік кластардың негізінде өзіндік контейнерлік кластарын құра алады.

STL кітапханасының орталық ұғымы шаблон болып табылады, сондықтан осы тараудың материалын оқуға кірісуден бұрын бұл ұғымның оқырмандар үшін жұмбақ еместігіне көз жеткізу керек (88 б. «Функциялардың шаблондары» бөлімін және 228 б. «Класс шаблондары» бөлімін қараңыз). Сонымен қатар, атаулар кеңістіктері (107 б.), функцияларды асыра жүктеу (86 б.) және операцияларды асыра жүктеу (205 б.) түсініктерімен таныс болу керек.

Контейнерлік кластар оларды пайдалану кезінде стандартталған интерфейсті қамтамасыз етеді. Әртүрлі контейнерлер үшін аттары бірдей операциялардың мағынасы да бірдей, негізгі операцияларды контейнердің барлық типтерінде қолдануға болады. Стандарт тек контейнерлердің интерфейстерін анықтайды, сондықтан олардың әртүрлі жүзеге асырулары тиімділігі бойынша бір-бірінен айтарлықтай өзгеше болуы мүмкін.

Кез келген дерлік контейнерлік класта төменде көрсетілген типтердің өрістері анықталған:

Өріс	**Түсініктемесі**
value_type	Контейнер элементінің типі
size_type	Элементтер индекстерінің, санауыштарының және т.б. типтері
iterator	Итератор
const_iterator	Тұрақтылық итератор
reverse_iterator	Кері итератор
const_reverse_iterator	Тұрақтылық кері итератор
reference	Элементке сілтеме
const_reference	Элементке тұрақтылық сілтеме
key_type	Кілт типі (ассоциативті контейнерлер үшін)
key_compare	Салыстыру критериінің типі (ассоциативті контейнерлер үшін)

Итератор элементке нұсқауыштың аналогы болып табылады. Ол контейнерді тура немесе кері бағытта қарап шығу үшін қолданылады. Итератордан талап етілетін бар әрекет – контейнер элементіне сілтеме жасай алуы және оның келесі элементіне көшу операциясын жүзеге асыра алу мүмкіндігі. Тұрақтылық итераторлар контейнердің сәйкес элементтерінің мәндері

өзгермейтін кезде қолданылады (итераторлар туралы толығырақ «Итераторлар» бөлімінде, 356 б. айтылады).

Итераторлар көмегімен элементтерге қол жеткізу үшін қолданылатын нақты мәліметтер типтерін ескерместен контейнерлерді қарап шығуға болады. Ол үшін әрбір контейнерде төмендегі кестеде көрсетілген бірнеше әдістер анықталған.

Әдіс	**Түсініктемесі**
`iterator begin(), const_iterator begin() const`	Бірініш элементке нұсқайды
`iterator end(), const_iterator end() const`	Соңғы элементтен кейін орналасқан элементке нұсқайды
`reverse_iterator rbegin(), const_reverse_iterator rbegin() const`	Кері тізбектегі алғашқы элементке нұсқайды
`reverse_iterator rend(), const_reverse_iterator rend() const`	Кері тізбектегі соңғы элементтен кейін орналасқан элементке нұсқайды

Әрбір контейнерде осы типтер мен әдістер олардың жүзеге асырылуына тәуелді тәсілмен анықталады.

Барлық контейнерлерде олардың өлшемі туралы мәлімет алуға мүмкіндік беретін әдістер анықталған:

Әдіс	**Түсініктемесі**
`size()`	Элементтер саны
`max_size()`	Контейнердің максималды өлшемі (миллиард элемент шамасында)
`empty()`	Контейнердің бос екендігін көрсететін бульдік функция

Контейнерлердің басқа өрістері мен әдістерін қажеттілігі туған кезде қарастырамыз.

STL кітапханасы 13 тақырыптық файлда анықталады, олар:

```
algorithm deque   functional iterator list    map
memory    numeric queue set  stack    utility vector
```

Тізбекті контейнерлер

Веторлар (**vector**), екіжақты кезектер (**deque**) және тізімдер (**list**) операциялардың әртүрлі жиынтығын сүйемелдейді, олардың ішінде өзара бірбіріне сәйкес келетін бірдей операциялар да бар. Олар әртүрлі тиімділікпен жүзеге асырылуы мүмкін:

Операция	Әдіс	vector	deque	list
Басына кірістіру	`push_front`	-	+	+
Басынан жою	`pop_front`	-	+	+
Соңына кірістіру	`push_back`	+	+	+
Соңынан жою	`pop_back`	+	+	+
Кездейсоқ орынға кірістіру	`insert`	(+)	(+)	+
Кездейсоқ орыннан жою	`erase`	(+)	(+)	+
Элементке кездейсоқ түрде қол жеткізу	`[ ], at`	+	+	-

Кестедегі **+** белгісі соған сәйкес операция контейнердегі элементтер санына (**n**) тәуелсіз түрде тұрақты уақыт ішінде жүзеге асатынын білдіреді. (**+**) белгісі соған сәйкес операция **n** шамасына пропорционалды уақытта орындалатынын білдіреді. Егер **n** шамасы аз болса, **+** белгісімен берілген операциялардың орындалу уақыты (**+**) белгісімен көрсетілген операциялардың уақытынан артық болуы мүмкін, бірақ элементтер саны көп болған жағдайда соңғысы өте қымбат болып кетуі мүмкін.

Кестеден көріп отырғанымыздай, мұндай операцияларға кезектің және вектордың кездейсоқ элементтерін кірістіру және жою әрекеттері жатады, өйткені мұндайда осы элементтен кейін орналасқан барлық элементтерді жаңа орынға көшіріп жазу қажет болады.

Сонымен, *вектор* – бұл элементтерге кездейсоқ қол жеткізуді, соңына қосуды және соңынан жоюды тиімді жүзеге асыратын құрылым.

Екіжақты кезек элементтерге кездейсоқ қол жеткізуді, екі шетіне (жағына) де қосуды және екі шетінен де (жағынан) жоюды тиімді жүзеге асырады.

Тізім элементтерді кездейсоқ орындарға кірістіруді және жоюды жүзеге асырады, бірақ өз элементтеріне кездейсоқ түрде қол жеткізе алмайды.

Вектормен жұмыс жасау мысалы. Файлдың ішінде бүтін сандардың кездейсоқ мөлшері берілген. Программада олар векторға оқылады да, сол реттілікпен экранға шығарылады.

```
#include <fstream>
#include <vector>
using namespace std;
int main(){
   ifstream in ("inpnum.txt");
   vector<int> v;
   int x;
```

```
    while ( in >> x, !in.eof())
       v.push_back(x);
    for (vector<int>::iterator i = v.begin();
       i != v.end(); ++i)
    cout << *i << " ";
}
```

Файл құрамында бүтін сандар болғандықтан, **vector** шаблонының соған сәйкес специализациясы – **vector<int>** қолданылады. **V** векторын құру үшін келсім бойынша берілген конструктор қолданылады. Файл соңына дейін цикл ұйымдастырылады да, онда файлдан кезекті бүтін сан оқылады. Ол сан **push_back** әдісінің көмегімен векторға енгізіледі, вектор өлшемі автоматты түрде өзгереді[1].

Вектор бойымен оны толық қарап шығу үшін соған сәйкес типтің итераторы ретінде **i** айнымалысы енгізіледі (**::** операциясы көріну аймағына қол жеткізуді білдіретінін еске салайық, яғни бұл жерде «шаблонның нақты специализациясына арналған итератор» типті **i** айнымалысы жарияланады). Осы итератордың көмегімен контейнердің барлық элементтеріне біріншісінен бастап рет-ретімен қол жеткізу орындалады. **begin()** әдісі нұсқауышты бірініші элементке, ал **end()** әдісі соңғыдан кейінгі элементке қайтарады. Жүзеге асыру жолы бұл нұсқауыштың анықталғандығына кепілдік береді.

Ағымдағы мәнді шекаралық мәнмен салыстыруды **!=** операциясының көмегімен орындаған дұрыс, себебі **<** немесе **<=** операциялары берілген тип үшін анықталмауы мүмкін. Инкремент операциясы (**i++**) осыдан кейін итератор контейнердің қарап шығу реттілігіндегі келесі элементіне нұсқайтындай етіп жүзеге асырылған. Вектор элементіне қол жеткізу қарапайым нұсқауыштардағы сияқты, адрессіздендіру операциясының көмегімен орындалады.

Бұл берілген мысалда **vector** сөзін жай ғана **deque** немесе **list** сөзіне алмастыру арқылы вектордың орнына кез келген тізбекті контейнерді қолдануға болатын еді. Мұндайда мәліметтердің ішкі көрінісі (бейнесі) және қолжетімді операциялар жиынтығы өзгеретін еді, ал программаның орындалу тәртібінде ешқандай өзгерістер болмайтын еді.

Алайда, **for** цикінің орнына индекс бойынша қол жеткізу операциясы **[ ]** қолданылған

```
for (int i = 0; i<v.size(); i++) cout << v[i] << " ";
```

тізбегін қоятын болсақ, программа **list** типті контейнер үшін жұмыс істемейді, себебі осы контейнерде бұл операция анықталмаған.

[1]Вектордың өлшемі әрбір элементті қосқан сайын өзгеріп отырмайды, бұл тиімді болмас еді. Ол белгілі бір алгоритм бойынша өзгереді, бұл алгоритмді басқаруға болады (325 б. қараңыз)

Векторлар (vector)

Векторды құру үшін келесі конструкторларды қолдануға болады (қарапайым жазбалары келтірілген):

```
explicit vector();                                    //1
explicit vector(size_type n, const T& value = T());  //2
template <class InputIter>                            //3
   vector(InputIter first, InputIter last);
vector(const vector<T>& x);                           //4
```

Мысалдағы **explicit** түйінді сөзі объектіні құру кезінде басқа типтің мәнін меншіктегенде жанамалы түрде орындалатын түрлендіруге тыйым салу үшін қолданылады (213 б. де қараңыз).

1-конструктор келісім бойынша берілген конструктор болып табылады.

2-конструктор ұзындығы **n** болып келетін ветор құрады да, оны бірдей элементтермен – **value** көшірмелерімен толтырады.

Вектор өлшемін өзгерту қымбатқа түсетіндіктен, оны құру кезінде бастапқы өлшемін беру өте ыңғайлы болып саналады. Мұндайда құрамдас типтер үшін әрбір элементті **value** мәнімен инициалдау орындалады. Егер ол көрсетілмесе, ауқымды (глобалды) векторлардың элементтері нөлмен инициалданады.

Егер вектор элементінің типін қолданушы анықтаған болса, оның бастапқы мәні осы тип үшін келісім бойынша берілген конструктордың көмегімен қалыптасады. Екінші параметрдің орнына параметрлері бар конструкторды шақыруға болады, осылайша қажетті қасиеттерге ие болатын элементтер векторы құрылады (төмендегі мысалды қара).

ЕСКЕРТУ

Кез келген контейнер элементтері оған кірістірілетін объектілердің көшірмелері болып табылады. Сондықтан олар үшін көшіру конструкторы және меншіктеу операциясы анықталуы керек.

3-конструктор итераторлар көмегімен көрсетілген элементтер диапазонын көшіру арқылы вектор құрады. Итераторлар типі «оқу үшін» болуы тиіс.

4-конструктор көшіру конструкторы болып табылады.

Конструкторларға мысалдар:

```
// Бірге тең 10 элементтен тұратын вектор құрылады:
vector <int> v2 (10, 1);
// v1 векторына тең вектор құрылады:
vector <int> v4 (v1);
// v1 векторының алғашқы екі элементіне тең
// екі элементтен тұратын вектор құрылады:
vector <int> v3 (v1.begin(), v1.begin() + 2);
```

```
// monstr класының 10 объектісінен тұратын вектор
// құрылады (197 б. қараңыз)
// (келісім бойынша берілген конструктор жұмыс істейді):
vector <monstr> m1 (10);
// monstr класының атауы тағайындалған 5 объектісінен
// тұратын вектор құрылады
// (char* параметрі бар конструктор жұмыс істейді):
vector <monstr> m2 (5, monstr("Вася"));
```

Қарастырылып отырған **vector** шаблонында меншіктеу операциясы және **көшіру** функциясы анықталған:

```
vector<T>& operator=(const vector<T>& x);
void assign(size_type n, const T& value);
template <class InputIter>
   void assign(InputIter first, InputIter last);
```

Бұл жерде **T** арқылы вектор элементтерінің типі көрсетілген. Стандартты мәліметтер типтері немесе тіркестер сияқты, векторларды да бірбіріне меншіктеуге болады. Меншіктеуден кейін вектордың өлшемі жаңа мәнге тең болады да, барлық ескі элементтер өшіріледі.

Бірінші формадағы **assign** функциясы өз әрекеті бойынша 2-конструкторға ұқсас, бірақ ол бұрыннан бар объектіге қолданылады. Екінші формадағы **assign** функиясы 3-конструктор сияқты, шақырушы вектор элементтеріне **first** және **last** итераторларымен анықталатын диапазоннан мәндерді меншіктеуге арналған, мысалы:

```
vector <int> v1, v2;
// v1 векторының алғашқы 10 элементіне 1 мәні
// меншіктеледі:
v1.assign(10,1);
// v2 векторының алғашқы 3 элементіне v1[5], v1[6],
// v1[7] мәндері меншіктеледі:
v2.assign(v1.begin() + 5, v1.begin() + 8);
```

Берілген **vector** класының итераторлары 12-кестеде көрсетілген.

Вектор элементтеріне қол жеткізу келесі операциялар мен әдістердің көмегімен жүзеге асырылады:

```
reference       operator[](size_type n);
const_reference operator[](size_type n) const;
const_reference at(size_type n) const;
reference       at(size_type n);
reference       front();
const_reference front() const;
reference       back();
const_reference back() const;
```

Мысалдағы **[ ]** операциясы вектор элементіне индексі бойынша, оның вектор шекарасынан шығып кетуін тексермей, қол жеткізуді жүзеге асырады. Мұндай тексеруді **at** функциясы орындайды да, вектор шекарасынан шығып кеткен жағдайда, **out_of_range** аластамасын тудырады. Әрине, **at** функциясы **[ ]** операциясына қарағанда баяу жұмыс атқарады, сондықтан диапазон нақты анықталған кезде төмендегі операцияны қолданған жөн:

```
for (int i = 0; i<v.size(); i++) cout << v[i] << " ";
```

Кері жағдайда ерекше жағдайларды өңдейтін **at** функциясы қолданылады:

```
try{
   //...
   v.at(i) = v.at(...);
}
catch(out_of_range) { ... }
```

Қол жеткізу операциялары тұрақтылық объектіге қолданылуына немесе қолданылмауына байланысты элементке сілтеменің мәнін (**reference**) немесе тұрақтылық сілтеме мәнін (**const_reference**) қайтарады.

Берілген **front** және **back** әдістері сәйкесінше вектордың алғашқы және соңғы элементтеріне сілтемелер қайтарады (бұл **begin** мен **end** функцияларымен бірдей емес, **begin** – алғашқы элементке нұсқауыш және **end** – соңғы элементтен кейінгі элементке нұсқауыш). Мысал:

```
vector <int> v(5, 10):
v.front() = 100;  v.back() = 100;
cout << v[0] << " " << v[v.size() - 1];
// Нәтиже: 100 100
```

Вектор орналасатын жедел жады көлемін **capacity** функциясы анықтайды:

```
size_type capacity() const;
```

Векторға жады динамикалық түрде бөлінеді, бірақ әрбір уақыт мезетінде бір элементке емес (бұл ресурсты артық жұмсау болып табылар еді), бірден элементтер тобына бөлінеді, мысалы 256 немесе 1024. Жадыны қайта бөлу элементтердің осы саннан артып кетуі кезінде ғана орындалады, мұндайда бөлінген кеңістік көлемі екі еселенеді. Жады қайта бөлінгеннен кейін вектор элементтеріне сілтеме жасайтын кез келген итераторлар жарамсыз болып табылады, өйткені вектор жадының басқа бөлігіне ауыстырылуы мүмкін және онымен байланысқан сілтемелер автоматты түрде жаңартылады деп күтудің қажеті жоқ.

Сонымен қатар, жадыны бөлетін **reserve** функциясы бар, ол векторды сақтау үшін жадының қажетті көлемін тағайындауға мүмкіндік береді:

```
void reserve(size_type n);
```

Функцияны қолдану мысалы:

```
vector <int> v;
v.reserve(1000);      // 1000 элементке жады бөлу
```

Осы функция орындалғаннан кейін **capacity** функциясының мәні кемінде **n** шамасына тең болады. Қарастырылған **reserve** функциясын вектордың өлшемі алдын ала белгілі болған кезде ғана қолдану тиімді болып саналады.

Вектордың өлшемдерін өзгерту үшін **resize** функциясы қолданылады:

```
void resize(size_type sz, T c = T());
```

Бұл функция берілетін **sz** мәні **size()** мәнінен үлкен немесе кіші болуына байланысты вектордың өлшемін арттырады немесе кемітеді. Екінші параметр вектордың барлық жаңа элементтеріне меншіктелетін мәнді тағайындайды. Олар вектордың соңына орналастырылады. Егер жаңа өлшем **size()** мәнінен кіші болса, вектордың соңынан **size() – sz** элемент жойылады.

Vector класының **объектілерін өзгертуге арналған келесі әдістер** анықталған:

```
void push_back(const T& value);
void pop_back();
iterator insert(iterator position, const T& value);
void insert(iterator position, size_type n, const T& value);
template <class InputIter> void insert(iterator position,
                 InputIter first, InputIter last);
iterator erase(iterator position);
iterator erase(iterator first, iterator last);
void swap();
void clear();                 // Векторды тазарту
```

Мысалда қарастырылған **push_back** функциясы вектордың соңына элемент қосады, ал **pop_back** функциясы вектор соңындағы элементті жояды.

Келесі **insert** функциясы векторға элементті кірістіру үшін қолданылады. Функцияның бірінші формасы **value** элементін бірінші параметрмен (итератормен) берілген позицияға кірістіреді де, кірістірілген элементке сілтеме жасайтын итераторды қайтарады. Функцияның екінші формасы векторға бірдей **n** элементті кірістіреді. Функцияның үшінші формасы сәйкес келетін типтегі элементтердің кез келген диапазонымен берілетін бірнеше элементті кірістіруге мүмкіндік береді, мысалы:

```
vector <int> v(2), v1(3,9);
int m[3] = {3, 4, 5};
v.insert(v.begin(), m, m + 3);  // v құрамы: 3 4 5 0 0
v1.insert(v1.begin() + 1, v.begin(), v.begin() + 2);
// v1 құрамы: 9 3 4 9 9
```

Векторға элемент кірістіруге жаңа позицияға ығыстырылатын элементтердің санына пропорционал уақыт кетеді. Бұл кезде вектордың жаңа өлшемі бұрынғы алып тұрған жады көлемінен артық болса, жадыны қайта бөлу орындалады. Бұл – индекс бойынша қол жеткізу жеңілдігінің құны. Егер кірістіру кезінде жадыны қайта бөлу орындалмаса, онда барлық итераторлар өздерінің мәндерін сақтайды. Кері жағдайда олар жарамсыз болып қалады.

erase функциясы вектордың бір элементін жою үшін (функцияның бірінші формасы) немесе итераторлар көмегімен берілген диапазонын (екінші форма) жою үшін қолданылады:

```
vector <int> v;
for (int i = 1; i<6; i++)  v.push_back(i);
// v құрамы: 1 2 3 4 5
v.erase(v.begin());                  // v құрамы: 2 3 4 5
v.erase(v.begin(), v.begin() + 2); // v құрамы: 4 5
```

Мұнда үшінші параметр арқылы соңғы жойылатын элемент емес, одан кейін орналасқан элемент берілетініне назар аударыңыз.

Кірістіру кезіндегі сияқты, **erase** функциясының әрбір шақырылуы жаңа позицияға ығыстырылып жылжытылатын элементтердің санына пропорционал уақытты қажет етеді. Жою орнынан «оңға қарай» орналасқан барлық сілтемелер мен итераторлар жарамсыз болып қалады.

swap функциясы типтері бірдей, бірақ өлшемдері бірдей бола бермейтін (ол міндетті емес) екі вектор элементтерін алмастыру үшін қолданылады:

```
vector <int> v1, v2;
...
v1.swap(v2);    // v2.swap(v1) өрнегіне эквивалентті;
```

Векторлар үшін **==**, **!=**, **<**, **<=** және **=** сияқты **салыстыру операциялары** анықталған. Егер екі вектордың өлшемдері мен барлық бір-біріне сәйкес элементтер жұбы тең болатын болса, онда олар өзара тең болып есептеледі. Егер бір вектор элементтерінің ішіндегі екінші вектордың сәйкес элементіне тең емес алғашқы элементі одан кіші болса (яғни, салыстыру лексикографиялық түрде орындалады), бір вектор екіншісінен кіші болып табылады. Мысал:

```
#include <vector>
using namespace std;
vector <int> v7, v8;
int main(){
   for (int i = 0; i<6; i++) v7.push_back(i);
   cout << "v7: ";
   for (int i = 0; i<6; i++) cout << v7[i] << " ";
   cout << endl;
   for (int i = 0; i<3; i++) v8.push_back(i+1);
   cout << "v8: ";
   for (int i = 0; i<3; i++) cout << v8[i] << " ";
```

```
    cout << endl;
    if (v7 < v8 ) cout << " v7 < v8" << endl;
    else cout << " v7 > v8" << endl;
}
```

Программа жұмысының нәтижесі:

```
v7: 0 1 2 3 4 5
v8: 1 2 3
v7 < v8
```

Векторлармен нәтижелі жұмыс істеу үшін стандартты кітапханада алгоритмдер деп аталатын функция шаблондары анықталған. Олардың құрамында мәндерді іздеу, элементтерді сұрыптау, кірістіру, алмастыру, жою және де басқа операциялар қарастырылған. Алгоритмдер «Алгоритмдер» бөлімінде, 373 б. сипатталады.

Логикалық мәндер векторлары (vector <bool>)

vector <bool> шаблонының специализациясы жадының орналасуын тиімділеу үшін анықталған, өйткені логикалық мәндер векторын оның элементі 1 бит орын алатындай етіп жүзеге асыруға болады. Бұл кезде жеке биттердің адрестелуі программалық жолмен орындалады. Мұндай вектордың итераторы нұсқауыш бола алмайды. Басқа жақтарынан алғанда логикалық мәндер векторлары қарапайым векторларға ұқсас және дәл сондай операциялар мен әдістердің жиынтығын жүзеге асырады. Оларға қосымша битті және векторды толықтай (**flip**) терістеу (инвертирлеу) әдістері анықталған.

Логикалық мәндер векторының элементіне сілтеуіш **reference** класы түрінде жүзеге асырылған, ол элементке қарапайым сілтемені модельдейді:

```
class reference{
      friend class vector;
      reference();
   public:
      ~reference();
      operator bool() const;
      reference& operator=(const bool x);
      reference& operator=(const reference& x);
      void flip();
};
```

Мысал (пернетақтадан векторға 0 немесе 1 болатын 10 мән енгізіледі, содан кейін олар экранға шығарылады).

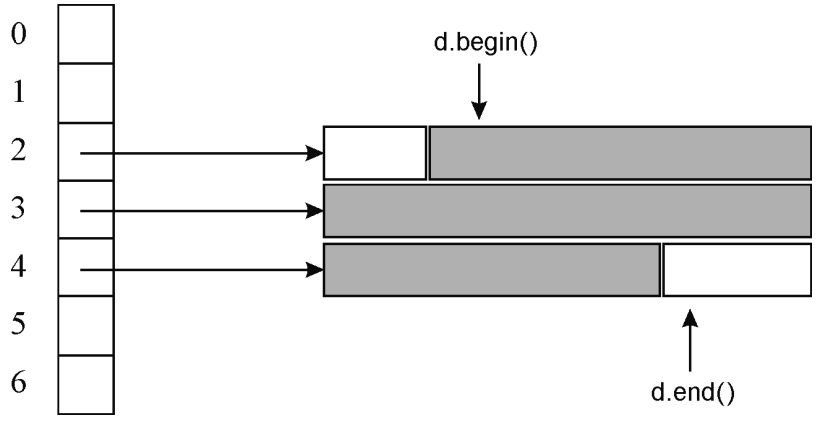
```
#include <vector>
#include <iostream>
using namespace std;
vector <bool> v (10);
int main(){
```

```
    for(int i = 0; i<v.size(); i++) cin >> v[i];
    for (vector <bool>:: const_iterator p = v.begin();
    p!=v.end(); ++p)
    cout << *p;
}
```

Екіжақты кезектер (deque)

Екіжақты кезек – вектормен қатар, элементтерге кездейсоқ қол жеткізуді сүйемелдейтін және тұрақты уақыт ішінде кезектің екі жағынан да элементтерді кірістіру мен жоюды қамтамасыз ететін тізбекті контейнер. Кезектің ішіндегі элементтермен орындалатын осындай операциялар орны ауыстырылатын элементтер санына пропорционал уақыт алады. Мұнда жадыны бөлу автоматты түрде орындалады.

Кезекті ұйымдастыру схемасын қарастырайық (12.1-сурет). Элементтеріне тұрақты уақыт ішінде кездейсоқ түрде қол жеткізуді қамтамасыз ету үшін кезек блоктарға бөлінген, олардың әрқайсысына қол жеткізу нұсқауыш арқылы жүзеге асырылады. Суреттегі боялған аймақтар кезектің бос емес элементтерін көрсетеді. Егер басына немесе соңына элементті қосу кезінде блок толып кетсе, келесі блокқа жады бөлінеді (мысалы, 4-блок толған соң 5 -блокқа, ал 2-блок толған соң 1-блокқа жады бөлінеді). Блоктардың ішіндегі шеткісін толтырған кезде тек ортаңғы элементтер қолданылатындай түрде нұсқауыштар жиымына жады қайта үлестіріледі. Ол көп уақыт алмайды. Осылайша, кезек элементтеріне қол жеткізу тұрақты уақыт кезеңінде орындалады, алайда бұл уақыт векторға қарағанда көбірек болады.



12.1-сурет. Екіжақты кезектің ұйымдастырылуы

Екіжақты кезекті құру үшін вектор конструкторларына ұқсайтын келесі конструкторларды қолдануға болады (қарапайым жазба түрі келтірілген):

```
explicit deque();                                          //1
explicit deque(size_type n, const T& value = T());  //2
template <class InputIter>                                 //3
   deque(InputIter first, InputIter last);
deque(const vector<T>& x);                                 //4
```

1-конструктор келісім бойынша берілген конструктор болып табылады.

2-конструктор ұзындығы **n** болатын кезек құрады да, оны бірдей элементтермен – **value** көшірмелерімен толтырады.

3-конструктор итераторлардың көмегімен көрсетілген элементтер диапазонын көшіру арқылы кезек құрады. Итераторлардың типі «оқу үшін» болуы тиіс.

4-конструктор көшіру конструкторы болып табылады.

Конструкторлар мысалдары:

```
// бірге тең 10 элементтен тұратын кезек құрылады:
deque <int> d2 (10, 1);
// v1 кезегіне тең кезек құрылады:
deque <int> d4 (v1);
// өткен бөлімдегі v1 векторының алғашқы екі элементіне
// тең екі элементтен тұратын кезек құрылады:
deque <int> d3 (v1.begin(), v1.begin() + 2);
// monstr класының 10 объектісінен тұратын кезек
// құрылады (197 б. қараңыз)
// (келісім бойынша берілген конструктор жұмыс істейді):
deque <monstr> m1 (10);
// monstr класының атаулары берілген 5 объектісінен
// тұратын кезек құрылады
// (char* параметрлі конструктор жұмыс істейді):
deque <monstr> m2 (5, monstr("Вася кезекте"));
```

Қарастырылып отырған **deque** шаблонында **меншіктеу** операциясы, **көшіру** функциялары, **итераторлар, салыстыру операциялары, элементтерге қол жеткізу** және **объектілерді өзгерту** операциялары мен функциялары анықталған, олар вектордың осыларға сәйкес операциялары мен функцияларына ұқсас болып келеді.

Вектор жағдайындағы сияқты, кірістіру мен жою элементтердің санына пропорционал уақытта орындалады. Егер осы операциялар кезектің ішкі элементтерімен орындалатын болса, кезек элементтеріне сілтемелер мен итераторлардың барлық мәндері жарамсыз болады. Кез келген шетіне қосу операцияларынан кейін итераторлардың барлық мәндері жарамсыз болып шығады, ал кезек элементтеріне сілтемелердің мәндері сақталады. Кез келген шетінен таңдау операцияларынан кейін тек осы элементтермен байланысты итераторлар мен сілтемелердің мәндері жарамсыз болады.

Аталғандармен қатар, кезектің басына қосу және таңдау функциялары анықталған:

```
void push_front(const T& value);
void pop_front();
```

Таңдау кезінде элемент кезектен жойылады.

Кезек үшін **capacity** және **reserve** функциялары анықталмаған, алайда **resize** және **size** функциялары бар.

Кезектерге «Алгоритмдер» деп аталатын 14-тарауда, 373 б. сипатталған стандартты кітапхана алгоритмдерін қолдануға болады.

Тізімдер (list)

Тізім өз элементтеріне кездейсоқ түрде қол жеткізуді сүйемелдемейді, бірақ элементтерді кірістіру мен жою операциялары тұрақты уақыт кезеңінде орындалады. STL кітапханасында **list** класы әрбір түйінінде өзінен кейінгі және алдыңғы элементтерге сілтемелер беретін екібайланысты тізім түрінде жүзеге асырылған. Сондықтан кезектің итераторлары үшін инкремент және декремент операциялары тұрақты уақытта орындалады, ал ондағы **n** түйінге ығыстыру әрекеті осы **n** шамасына пропорционал уақытты қажет етеді.

Кірістіру және жою операцияларын орындағаннан кейін барлық итераторлар мен сілтемелердің мәндері жарамды болып қалады.

Тізім векторлар мен кезектерге қолданылатын операцияларға ұқсас болып келетін **конструкторларды, меншіктеу операциясын, көшіру функциясын, салыстыру операциялары** және **итераторларды** сүйемелдейді.

Тізімдер үшін оның **элементтеріне қол жеткізу** келесі әдістермен шектеледі:

```
reference       front();
const_reference front() const;
reference       back();
const_reference back()  const;
```

Элементтерді тізім басына және соңына енгізу үшін кезектің осыларға сәйкес әдістеріне ұқсас әдістер анықталған:

```
void push_front(const T& value);
void pop_front();
void push_back(const T& value);
void pop_back();
```

Сонымен қатар, дәл векторлар мен кезектердегі сияқты, **list** объектілерін өзгертуге арналған қалған барлық әдістер де жұмыс істейді:

```
iterator insert(iterator position, const T& value);
void insert(iterator position, size_type n, const T& value);
template <class InputIter> void insert(iterator position,
                InputIter first, InputIter last);
iterator erase(iterator position);
iterator erase(iterator first, iterator last);
```

```
void swap();
void clear();
```

Тізім үшін **capacity** функциясы анықталмаған, өйткені элементтерге жады қажет болған кезде ғана бөлінеді. Тізім соңынан элементтерді жою немесе қосу арқылы (екіжақты кезекке ұқсас түрде) оның өлшемін өзгертуге болады:

```
void resize(size_type sz, T c = T());
```

Жоғарыда аталғандармен қатар, тізімдер үшін бірқатар спецификалық әдістер анықталған. **Тізімдердің тіркесуі** (**splice**) элементтерді бір тізімнен екінші тізімге жадыны қайта бөлмей-ақ, тек нұсқауыштарды өзгерту есебінен көшіру үшін қолданылады:

```
void splice(iterator position, list<T>& x);
void splice(iterator position, list<T>& x, iterator i);
void splice(iterator position, list<T>& x,
            iterator first, iterator last);
```

Екі тізім де бір типтегі элементтерден тұруы керек. Функцияның бірінші формасы шақырушы тізімге орны бірінші параметрмен нұсқалған элементтің алдына тізімнің екінші параметрмен нұсқалған барлық элементтерін кірістіреді, мысалы:

```
list <int> L1, L2;
... // тізімдердің қалыптастырылуы
L1.splice(L1.begin() + 4, L2);
```

Екінші тізім бос қалады. Тізімді өз-өзіне кірістіруге болмайды.

Функцияның екінші формасы орны үшінші параметр арқылы анықталған элементті **x** тізімінен оны шақырушы тізімге тасымалдайды. Элементті бір тізімнің ішінде ғана тасымалдауға болады.

Функцияның үшінші формасы жоғарыдағыға ұқсас түрде тізімнен тізімге бірнеше элементті тасымалдайды. Олардың диапазоны функцияның үшінші және төртінші параметрлерімен беріледі. Егер бір тізім үшін бірінші параметр үшінші және төртінші параметрлер аралығындағы диапазонда орналасатын болса, онда нәтижие анықталмаған болып саналады. Мысалы:

```
#include <list>
using namespace std;
int main(){
   list<int> L1;
   list<int>::iterator i, j, k;
   for (int i = 0; i<5; i++) L1.push_back(i + 1);
   for (int i = 12; i<14; i++) L1.push_back(i);
   cout << "Бастапқы тізім: ";
   for (i = L1.begin(); i != L1.end(); ++i)
      cout << *i << " ";
```

```
        cout << endl;
        i = L1.begin(); i++;
        k = L1.end();
        j = --k; k++; j--;
        L1.splice( i, L1, j, k);
        cout << "Тіркестіруден кейінгі тізім: ";
        for ( i = L1.begin(); i != L1.end(); ++i)
           cout << *i << " ";
}
```

Программаның жұмыс істеу нәтижесі:

```
Бастапқы тізім: 1 2 3 4 5 12 13
Тіркестіруден кейінгі тізім: 1 12 13 2 3 4 5
```

Тасымалданған элементтер қаралайтылған қаріппен ерекшеленген. Тізімдер итераторлары үшін қосу және азайту операциялары анықталмағанына назар аударыңыздар, яғни **`j = k – 1`** деп жазуға болмайды, сондықтан тізімдер итераторлары үшін рұқсат етілген инкремент және декремент операцияларын қолдануға тура келді. Жалпы жағдайда тізімнен элементті іздеу үшін **`find`** функциясы қолданылады (376 б. қара).

Элементті мәні бойынша **жою** үшін **`remove`** функциясы қолданылады:

```
void remove(const T& value);
```

Егер тізімде **`value`** мәнді бірнеше элемент бар болса, онда олардың барлығы да жойылады.

Тізімнен белгілі бір шартты қанағаттандыратын элементтерді жоюға болады. Ол үшін **`remove_if`** функциясы қолданылады:

```
template <class Predicate> void remove_if(Predicate pred);
```

Параметр ретінде тізімнің элементіне қойлатын шартты тағайындайтын класс-предикат алынады. Предикаттар туралы 365 б. қараңыз.

Тізім элменттерін реттеу үшін **`sort`** әдісі қолданылады:

```
void sort();
template <class Compare> void sort(Compare comp);
```

Бірінші жағдайда тізім элементтердің өсу ретімен (элементтер үшін **`<`** операциясының анықтамасына сәйкес) сұрыпталып, екінші жағдайда **`Compare`** функционалды объектісіне сәйкес сұрыпталады (функционалды объектілер туралы 211 б. «Функцияны шақыру операциясының асыра жүктелуі» бөлімінде жазылған). Егер функционалды объектіге берілетін екі мән сұрыпталу кезінде бұрынғы ретімен қалуы керек болса, онда оның мәні **`true`**, кері жағдайда – **`false`** болады.

Бірдей мәнді элементтердің орналасу реттілігі сақталады. Сұрыпталу уақыты $N \cdot \log_2 N$ шамасынаа пропорционал болады, мұндағы N – тізімдегі элементтердің саны.

unique әдісі тізімде қатар орналасқан бірдей элементтердің әрбір серия-сынан тек бірінші элементті ғана қалдырады. Әдістің бірінші формасының форматы келесідей болады:

```
void unique();
```

unique әдісінің екінші формасы параметр ретінде бинарлы предикат-ты (365 б. қараңыз) қолданады, бұл тізімнен элементтерді жоюдың өзіндік критериін тағайындауға мүмкіндік береді. Егер критерий орындалса, преди-кат мәні **true**, кері жағдайда – **false** болады. Предикат аргументтерінің типі тізімнің элементтерінің типімен сәйкес келеді:

```
template <class BinaryPredicate>
   void unique(BinaryPredicate binary_pred);
```

Тізімдерді біріктіру үшін **merge** әдісі қолданылады:

```
void merge(list<T>& x);
template <class Compare>
   void merge(list<T>& x, Compare comp);
```

Екі тізім де сұрыпталған түрде (реттелген) болуы керек (бірінші жағдайда элементтер үшін **<** операциясының анықтамасына сәйкес, екіншісінде **Compare** функционалды объектісіне сәйкес болады). Нәтижесі – реттелген тізім болады. Егер шақыратын тізімдегі және тізім-параметрдегі элементтер бірдей келсе, алдымен шақыратын тізімдегі элементтер орналасады.

reverse әдісі тізім элементтерінің орналасу ретін кері бағытқа өзгерту үшін қолданылады (жұмыс уақыты элементтер санына пропорционал):

```
void reverse();
```

Тізіммен жұмыс істеу мысалы:

```
#include <fstream>
#include <list>
using namespace std;
void show (const char *str, const list<int> &L){
   cout << str << ":" << endl;
   for (list<int>::const_iterator i = L.begin();
   i != L.end(); ++i)
      cout << *i << " ";
   cout << endl;
}
int main(){
   list<int> L;
   list<int>::iterator i;
   int x;
   ifstream in("inpnum");
   while ( in >> x, !in.eof()) L.push_back(x);
   show("Бастапқы тізім", L);
```

```
    L.push_front(1);
    i = L.begin();
    L.insert(++i, 2);
    show("Тізім басына 1 және 2 сандарын кірістіруден
          кейін", L);
    i = L.end(); L.insert(--i, 100);
    show("Соңғы элементтің алдына 100 санын кірістіруден
          кейін", L);
    i = L.begin(); x = *i; L.pop_front();
    cout << "Басынан " << x << " өшірілген" << endl;
    i = L.end(); x = *--i; L.pop_back();
    cout << "Соңынан " << x << " өшірілген" << endl;
    show("Өшіруден кейінгі тізім:", L);
    L.remove(76);
    show("Мәні 76 болатын элементтерді өшіруден кейін", L);
    L.sort();
    show("Сұрыптаудан кейін:", L);
    L.unique();
    show("unique әдісінен кейін:", L);
    list<int> L1 (L);
    L.reverse();
    show("reverse әдісінен кейін:", L);
}
```

Программаның орындалу нәтижесі:

```
Бастапқы тізім:
56 34 54 0 76 23 51 11 51 11 76 88
Тізім басына 1 және 2 сандарын кірістіруден кейін:
1 2 56 34 54 0 76 23 51 11 51 11 76 88
Соңғы элементтің алдына 100 санын кірістіруден кейін:
1 2 56 34 54 0 76 23 51 11 51 11 76 100 88
Басынан 1 өшірілген
Соңынан 88 өшірілген
Өшіруден кейінгі тізім:
2 56 34 54 0 76 23 51 11 51 11 76 100
Мәні 76 болатын элементтерді өшіруден кейін:
2 56 34 54 0 23 51 11 51 11 100
Сұрыптаудан кейін:
0 2 11 11 23 34 51 51 54 56 100
unique әдісінен кейін:
0 2 11 23 34 51 54 56 100
reverse әдісінен кейін:
100 56 54 51 34 23 11 2 0
```

Тізімдерге «Алгоритмдер» бөлімінде сипатталған стандартты кітапхана алгоритмдерін қолдануға болады (373 б. қараңыз).

Стектер (stack)

Стекте (анықтамасын «Стектер» бөлімінен, 129 б. қараңыз) оның өлшемін өзгертетін тек екі операцияға рұқсат берілгені белгілі, олар – стектің төбесіне элемент қосу және төбесінен таңдау. Стекті қарастырылған контейнерлердің кез келгенінің – вектордың, екіжақты кезектің немесе тізімнің негізінде жүзеге асыруға болады. Осылайша, стек контейнердің жаңа типі емес, бұрыннан бар типінің нұсқасы болып табылады, сондықтан оны контейнердің *адаптері* деп атайды. Басқа адаптерлер (кезектер және басымдылықтары бар кезектер) келесі бөлімдерде қарастырылады.

STL кітапханасында стек келісім бойынша екіжақты кезектің негізінде анықталған:

```
template <class T, class Container = deque<T> >
class stack {
   protected:
      Container c;
   public:
      explicit stack(const Container& = Container());
      bool        empty() const     {return c.empty();}
      size_type   size() const      {return c.size();}
      value_type& top()             {return c.back();}
      const value_type& top() const {return c.back();}
      void push(const value_type& x) {c.push_back(x);}
      void pop()                    {c.pop_back();}
};
```

Келтірілген сипаттамадан (ол қысқартылып берілген) **push** стекке енгізу әдісі **push_back** соңына енгізу әдісіне, **pop** стектен таңдау әдісі **pop_back** соңынан таңдау әдісіне сәйкес келетіні көрініп тұр, сонымен қатар, стектің төбесіндегі элементтің мәнін алу немесе өзгерту үшін **top** әдісі қосылған. **stack** класының конструкторына параметр ретінде базалық контейнерге сілтеме беріледі, ол **c** қорғалған мәліметтер өрісіне көшіріледі.

Стекпен жұмыс істегенде итераторларды қолдануға болмайды және стектің ортасындағы элементтің мәнін одан жоғары элементтердің барлығын таңдап алу арқылы ғана пайдалануға болады. Жоғарыда қарастырылған барлық контейнерлер сияқты, стек үшін де салыстыру операциялары анықталған (327 б. қараңыз).

Стекті қолдану мысалы (программа файлдан сандарды енгізеді де, оларды экранға кері бағытта шығарады):

```
#include <fstream>
#include <vector>
#include <stack>
using namespace std;
int main(){
```

```
   ifstream in ("inpnum");
   stack <int, vector<int> > s;
   int x;
   while ( in >> x, !in.eof()) s.push(x);
   while (! s.empty()){
      x = s.top(); cout << x << " ";
      s.pop();
   }
}
```

inpnum файлының құрамы:

```
56 34 54 0 76 23 51 11 51 11 76 88
```

Программа жұмысының нәтижесі:

```
88 76 11 51 11 51 23 76 0 54 34 56
```

Кезектер (queue)

Кезекте (анықтамасын «Кезектер» бөлімінен, 130 б. қараңыз) оның өлшемін өзгертетін екі операцияға рұқсат берілгені белгілі, олар – кезек соңына элементті қосу және басынан таңдау. Кезек – екіжақты кезектің немесе тізімнің негізінде жүзеге асыруға болатын адаптер (вектор жарамайды, өйткені онда басынан таңдау операциясы жоқ) болып табылады.

STL кітапханасында кезек келісім бойынша екіжақты кезектің негізінде анықталған:

```
template <class T, class Container = deque<T> >
class queue {
   protected:
      Container c;
   public:
      explicit queue(const Container& = Container());
      bool        empty() const       {return c.empty();}
      size_type   size() const        {return c.size();}
      value_type& front()             {return c.front();}
      const value_type& front() const {return c.front();}
      value_type& back()              {return c.back();}
      const value_type& back() const  {return c.back();}
      void push(const value_type& x)  {c.push_back(x);}
      void pop()                      {c.pop_front();}
};
```

Мысалдағы **front** және **back** әдістері сәйкесінше кезектің басында және соңында орналасқан элементтердің мәндерін алу үшін қолданылады (бұл кезде элементтер кезекте сақталады).

Кезекпен жұмыс істеу мысалы (программа файлдан кезекке сандар енгізеді және кезек босаған кезге дейін одан таңдауды орындайды):

```
#include <fstream>
#include <list>
#include <queue>
using namespace std;
int main(){
   ifstream in ("inpnum");
   queue <int, list<int> > q;
   int x;
   while ( in >> x, !in.eof()) q.push(x);
   cout << "q.front(): " << q.front() << " ";
   cout << "q.back(): " << q.back() << endl;
   while (! q.empty()){
      q.pop();
      cout << "q.front(): " << q.front() << " ";
      cout << "q.back(): " << q.back() << endl;
   }
}
```

inpnum файлының құрамы:

```
56 34 54 0 76 23 51 11 51 11 76 88
```

Программа жұмысының нәтижесі:

```
q.front(): 56  q.back(): 88
q.front(): 34  q.back(): 88
q.front(): 54  q.back(): 88
q.front(): 0   q.back(): 88
q.front(): 76  q.back(): 88
q.front(): 23  q.back(): 88
q.front(): 51  q.back(): 88
q.front(): 11  q.back(): 88
q.front(): 51  q.back(): 88
q.front(): 11  q.back(): 88
q.front(): 76  q.back(): 88
q.front(): 88  q.back(): 88
q.front(): 0   q.back(): 0
```

Стектерге және кезектерге «Алгоритмдер» бөлімінде сипатталған стандартты кітапхана алгоритмдерін қолдануға болады (373 б. қараңыз).

Басымдылықтары бар кезектер (priority_queue)

Басымдылықтары бар кезекте әрбір элементке кезектен таңдау реттілігін анықтайтын басымдық сәйкес келеді. Келісім бойынша ол **<** операциясының көмегімен анықталады; осылайша, кезектен әр жолы ең үлкен (максималды) элемент таңдалып алынады.

Басымдылықтары бар кезекті жүзеге асыру үшін элементтерге кездейсоқ қол жеткізуді рұқсат ететін контейнерді, мысалы, векторды немесе екіжақты кезекті қолдануға болады. Контейнердің типі шаблонның екінші параметрі ретінде беріледі (әдеттегідей, біріншісі – элементтердің типі). Үшінші параметр ретінде басымдылықты анықтайтын функция немесе функционалды объект (211 және 363 б. қараңыз) көрсетіледі:

```
template <class T, class Container = vector<T>,
   class Compare=less<typename Container::value_type> >
class priority_queue {
   protected:
      Container c;
      Compare comp;
   public:
      explicit priority_queue(const Compare& x = Compare(),
         const Container& = Container());
      template <class InputIter>
         priority_queue(InputIter first, InputIter last,
         const Compare& x = Compare(),
         const Container& = Container());
         bool empty() const           {return c.empty();}
         size_type size() const       {return c.size();}
         const value_type& top() const {return c.front();}
         void push(const value_type& x);
         void pop();
};
```

Басымдылықтары бірдей элементтер үшін басымдылықтары бар кезек қарапайым кезек болып табылады. Стектер ортасындағы сияқты, мұнда да негізгі әдістерге **push**, **pop** және **top** әдістері жатады.

Қарапайым мысал:

```
#include <iostream>
#include <vector>
#include <functional>
#include <queue>
using namespace std;
int main(){
   priority_queue <int, vector<int>, less<int> > P;
   int x;
   P.push(13); P.push(51); P.push(200); P.push(17);
   while (!P.empty()){
      x = P.top();
      cout << x << " элементі таңдалды" << endl;
      P.pop();
   }
}
```

Программа жұмысының нәтижесі:

```
200 элементі таңдалды
51 элементі таңдалды
17 элементі таңдалды
13 элементі таңдалды
```

Бұл мысалда шаблонның үшінші параметрі **<functional>** тақырыптық файлында («Функционалдық объектілер» бөлімін, 363 б. қараңыз) анықталған шаблон болып табылады. Ол «кіші» екендігін тексеретін салыстыру операциясын береді. Стандартты **greater<тип>**, **greater_equal<тип>**, **less_equal<тип>** шаблондарын беруге болады. Егер кезектен таңдаудың басқа бір реттілігін анықтау қажет болса, онда өзіндік функционалды объект енгізіледі. Төменде берілген мысалда таңдау сан цифрларының ең кіші қосындысы бойынша орындалады:

```
#include <iostream>
#include <vector>
#include <functional>
#include <queue>
using namespace std;
class CompareSum{
   public:
      bool operator()(int x, int y){
         int sx = 0, sy = 0;
         while (x){sx += x % 10; x /= 10;}
         while (y){sy += y % 10; y/=10;}
         return sx > sy ;
      }
};
int main(){
   priority_queue <int, vector<int>, CompareSum > P; int x;
   P.push(13); P.push(51); P.push(200); P.push(17);
   while (!P.empty()){
      x = P.top();
      cout << x << " элементі таңдалды" << endl;
      P.pop();
   }
}
```

Программа жұмысының нәтижесі:

```
200 элементі таңдалды
13 элементі таңдалды
51 элементі таңдалды
17 элементі таңдалды
```

Ассоциативті контейнерлер

Жоғарыда көрсетілгендей, ассоциативті контейнерлер, көбінесе теңдестірілген іздеу бұтақтарының негізінде құрылуының арқасында мәліметтерге жылдам қол жеткізуді қамтамасыз етеді (контейнердің жүзеге асырылуы емес, тек оның интерфейсі ғана стандартқа сәйкес анықталады).

Ассоциативті контейнерлердің бес типі бар, олар: сөздіктер (**map**), дубликатты сөздіктер (**multimap**), жиындар (**set**), дубликатты жиындар (**multiset**) және биттік жиындар (**bitset**). Сөздіктерді *ассоциативті жиымдар* немесе *бейнелер* деп те атайды.

Сөздік жұпталған мәндер негізінде құрылады, жұптардың алғашқысы элементті анықтауға арналған кілт, ал екіншісі – элементтің өзі болып табылады. Кілт элементпен ассоциацияланған (байланысқан) деп айтуға болады, контейнерлердің бұлай аталуы да осыған орай қалыптасқан. Мысалы, ағылшын-қазақ сөздігінде кілт ретінде ағылшын сөзі, ал элемент ретінде қазақ сөзі алынады. Қарапайым жиымды да сөздік ретінде қарастыруға болады, мұндағы кілттің рөлін элементтің нөмірі атқарады. STL кітапханасында сипатталған сөздіктерде кілт ретінде кездейсоқ типтегі мән қолданылуы мүмкін. Ассоциативті контейнерлер **<map>** және **<set>** тақырыптық файлдарында сипатталған.

«Кілт–элемент» жұбын сақтау үшін **<untility>** тақырыптық файлында сипатталған **pair** шаблоны қолданылады:

```
template <class T1, class T2> struct pair{
   typedef T1 first_type;
   typedef T2 second_type;
   T1 first;
   T2 second;
   pair();
   pair(const T1& x, const T2& y);
   template <class U, class V> pair(const pair<U, V> &p);
};
```

pair шаблонының екі параметрі бар, олар жұп элементтерінің типтерін көрсетеді. Бірінші элементтің атауы – **first**, екіншісінің атауы – **second**. Екі конструктор анықталған: біріншісі элементтерді инициалдау үшін екі мәнді қабылдауы керек, екіншісі (көшіру конструкторы) – басқа жұпқа сілтемені қабылдауы тиіс. Жұптың келісім бойынша анықталған конструкторы жоқ, яғни объектіні құру кезінде оған мәнді нақты түрде меншіктеу керек.

Жұптар үшін теңдікке тексеру және кіші екендігін салыстыру операциясы анықталған (басқа барлық қатынас операциялары STL кітапханасында осы екі операция негізінде автоматты түрде жасалады). Егер **p1.first < p2.first** не-

месе **p1.first == p2.first && p1.second < p2.second** болса, **p1** жұбы **p2** жұбынан кіші болып саналады.

Жұпқа мәнді меншіктеу үшін **make_pair** функциясын қолдануға болады:

```
template <class T1, class T2>
   pair<T1, T2> make_pair(const T1& x, const T2& y);
```

Жұптарды қалыптастыру мысалы:

```
#include <iostream>
#include <utility>
using namespace std;
int main(){
   pair<int, double> p1(10, 12.3), p2(p1);
   p2 = make_pair(20, 12.3);
   // p2 = pair <int, double >(20, 12.3) өрнегіне эквивалентті
   cout << "p1: " << p1.first << " " << p1.second << endl;
   cout << "p2: " << p2.first << " " << p2.second << endl;
   p2.first -= 10;
   if (p1 == p2) cout << "p1 == p2\n";
   p1.second -= 1;
   if (p2 > p1) cout << "p2 > p1\n";
}
```

Программаның орындалу нәтижесі:

```
p1: 10 12.3
p2: 20 12.3
p1 == p2
p2 > p1
```

<utility> тақырыптық файлы **<map>** немесе **<set>** файлдарын қолдану кезінде автоматты түрде қосылады.

Сөздіктер (map)

Сөздікте (**map**), дубликатты сөздіктерге (**multiset**) қарағанда, барлық кілттер қайталанбайтын болуы тиіс. Сөздіктегі элементтер сұрыпталған түрде сақталады, сондықтан кілттер үшін «кіші» қатынасы анықталуы тиіс. Сөздіктің шаблонында үш параметр болады: кілттің типі, элементтің типі, «кіші» қатынасын анықтайтын функционалды объектінің типі (функционалдық объектілер 211 б. қарастырылған):

```
template <class Key, class T, class Compare= less<Key> >
class map{
   public:
      typedef pair <const Key, T> value_type;
      explicit map(const Compares comp = Compare());
      template <class InputIter> map(InputIter first,
```

```
        InputIter last, const Compares comp = Compare());
    map(const map <Key, T, Compare>& x);
        ...
};
```

Келтірілген сипаттамада (ол қысқартылған түрде берілген) көрсетілгендей, **value_type** сөздік элементтерінің типі **Key** және **T** типті элементтердің жұбы ретінде анықталады.

Бірінші конструктор көрсетілген функционалды объектіні қолдана отырып, бос сөздік құрады. Екінші конструктор сөздікті құрады да, оған көрсетілген итераторлар диапазонымен анықталатын элементтерді жазады. Егер элементтер сұрыпталып реттелген болса, осы конструктордың жұмыс істеу уақыты жазылатын элементтердің санына пропорционал, ал кері жағдайда элементтер санының квадратына тең болады. Үшінші конструктор көшіру конструкторы болып табылады.

Барлық контейнерлердегі сияқты, сөздік үшін де деструктор, меншіктеу операциясы және қатынас операциясы анықталған. Сөздіктің итераторлары 12-кестеде көрсетілген. Элементтерге кілті бойынша қол жеткізу үшін **[ ]** операциясы анықталған:

```
T & operator[](const Key & x);
```

Бұл операцияның көмегімен элементтердің мәнін алып қана қоймай, сонымен қатар сөздікке жаңа мәндерді қосуға болады. Дәстүрлерден бас тартпай, сөздікке мысал ретінде телефондық кітапшаны келтірейік, мұнда кілт ретінде адамдардың тегі, ал элемент ретінде олардың телефон нөмірі алынады:

```
#include <fstream>
#include <iostream>
#include <string>
#include <map> using namespace std;
typedef map <string, long, less <string> > map_sl; // 1
int main(){
   map_sl m1;
   ifstream in("phonebook");
   string str;
   long num;
   while (in >> num, !in.eof()){     // Нөмірді оқу
      in.get();                      // Бос орынды қалдыру
      getline(in, str);              // Тегін оқу
      m1[str] = num;                 // Сөздікке енгізу
      cout << str << " " << num << endl;
   }
   m1["Petya P."] = 2134622;         // Сөздікті толықтыру
   map_sl :: iterator i;
   cout << "m1:" << endl;            // Сөздікті шығару
```

```
    for (i = m1.begin(); i != m1.end(); i++)
    cout << (*i).first << " " << (*i).second << endl;
    i = m1.begin(); i++;       // Екінші элементті шығару
    cout << "Екінші элемент: ";
    cout << (*i).first << " " << (*i).second << endl;
    cout << "Vasia: " << m1["Vasia"] << endl;
    // Элементті кілті бойынша шығару
    return 0;
}
```

Программаның оқылуын жақсарту үшін сөздік типінің қысқаша белгіленуі енгізілген (`// 1` түрінде белгіленген оператор). Әрбір адам туралы мәліметтер **phonebook** файлының бір жолында орналасқан: алдымен телефон нөмірі, одан кейін бос орын арқылы адамның аты-жөні жазылады:

```
1001002 Petya K.
3563398 Ivanova N.M.
1180316 Vovochka
2334476 Vasia
```

Сөздіктің итераторлары үшін инкремент және декремент операцияларын қолдануға болады, бірақ **+** және **–** операцияларына тыйым салынған. Төменде программа жұмысының нәтижиесі көрсетілген (сөздіктің сұрыпталып реттелген түрде шығарылатынына назар аударыңыз):

```
Petya K.  1001002
Ivanova N.M.  3563398
Vovochka  1180316
Vasia  2334476
m1:
Ivanova N.M.  3563398
Petya K.  1001002
Petya P.  2134622
Vasia  2334476
Vovochka  1180316
Екінші элемент: Petya K.  1001002
Vasia:  2334476
```

Сөздікте элементтерді **іздеу** үшін төмендегі функциялар анықталған:

```
iterator        find(const key_type& x);
const_iterator  find(const key_type& x) const;
iterator        lower_bound(const key_type& x);
const_iterator  lower_bound(const key_type& x) const;
iterator        upper_bound(const key_type& x);
const_iterator  upper_bound(const key_type &x) const;
size_type       count(const key_type& x) const;
```

Іздеу сәтті орындалған жағдайда **find** функциясы табылған элементке итераторды немесе кері жағдайда **end()** мәнін қайтарады.

Келесі **upper_bound** функциясы итераторды кілті **x** шамасынан кіші болмайтын алғашқы элементке қайтарады, ондай элемент жоқ болса, **end()** мәнін қайтарады (егер **x** кілті бар элемент сөздікте бар болса, сол элементке итератор қайтарылады).

Ал **lower_bound** функциясы итераторды кілті **x** шамасынан үлкен алғашқы элементке немесе ондай элемент жоқ болса, **end()** мәнін қайтарады.

Жоғарыда көрсетілген мысалға келесі операторларды қосайық:

```
getline(cin, str);
if (m1.find(str) != m1.end())
   cout << m1 [str];
else{
   cout << (*m1.upper_bound(str)).first << " ";
   cout << (*m1.lower bound(str)).first << " ";
}
```

Егер пернетақтадан сөздікте бар аты-жөнді енгізсе, онда соған сәйкес келетін телефон нөмірі шығарылады, кері жағдайда – алфавит бойынша енгізілген атыжөннен кейін орналасатын алғашқы адамның тегі екі рет шығарылады, мысалы:

```
Petya M.    // Астын сызуу арқылы қолданушының енгізген
            // мәліметі белгіленген
Petya P. Petya P.
```

count функциясы кілті **x** шамасына тең болатын элементтердің санын қайтарады (ондай элементтер саны 0 немесе 1 болуы мүмкін).

Элементтерді **кірістіру** және **жою** үшін төмендегі функциялар анықталған:

```
pair<iterator, bool> insert(const value_type& x);
iterator insert(iterator position, const value_type& x);
template <class InputIter>
void      insert(InputIter first, InputIter last);
void      erase(iterator position);
size_type erase(const key_type& x);
void      erase(iterator first, iterator last);
void      clear();
```

Функцияның бірінші формасы сөздікке «кілт–мәні» жұбын кірістіру үшін қолданылады. Функция кірістірілген мәнді көрсететін итератордан және операция нәтижесінің бульдік көрсеткішінен тұратын жұпты қайтарады, егер берілген кілті бар жазба сөздікте жоқ болса, бульдік көрсеткіш **true** мәніне (тек осы жағдайда ғана кірістіру орындалады), кері жағдайда (итератор бар жазбаға сілтеме жасайды) **false** мәніне тең болады. Функцияның жұмыс

істеу уақыты сөздіктегі элементтер санынының логарифміне пропорционал болып табылады.

Сонымен, сөздікте бар жазбаны кірістіру функциясының көмегімен түзетуге болмайды. Бұл әрекет индекс бойынша қол жеткізу операциясы арқылы орындалады.

Осының алдындағы листингте типі сипатталған сөздікке кірістірудің бірнеше мысалы төменде берілген:

```
map_sl m2;                    // Бос сөздікті құру
m2.insert(map_sl::value_type("Lena", 3157725));
str = "Anna";
num = 5536590;
m2.insert(make_pair(str, num));
num = 5530000;
// Сөздікте бар жазбаны кірістіруге талпыну:
m2.insert(make_pair(str, num));
i = m1.begin();
m2.insert(*i);           // m2 сөздігіне m1 сөздігінің
                         // алғашқы элементін кірістіру
m2["Lena"] = 2222222;          // Элементті түзету
for (i = m2.begin(); i != m2.end(); i++)  // Сөздікті
                                          // шығару
cout << (*i).first << " " << (*i).second << endl;
```

Программа орындалуының нәтижесі:

```
Anna  5536590
Ivanova N.M.  3563398
Lena  2222222
```

Берілген **insert** функциясының екінші формасы кірістіру үрдісін жылдамдату үшін қолданылады. Осы мақсатта оған бірінші параметр ретінде кірістіру орнын іздеу басталатын сөздіктегі позиция беріледі[1]. Кірістіру сөздікте **x** мәні болмаған жағдайда ғана орындалады. Функция итераторды **x** құрамындағы кілті бар сөздік элементіне қайтарады.

Мысалы, элементтер сөздікке өсу ретімен орналастырылатыны белгілі болса, кірістіру функциясына бірінші параметр ретінде осының алдындағы элементтің позициясын беруге болады (мұндайда кірістіру уақыты тұрақты болады):

```
#include <iostream>
#include <string>
#include <map>
using namespace std;
typedef map <string, long, less <string> > map_sl;
typedef pair <string, long > pair_sl;
```

[1]Егер көрсетілген позиция элемент кірістірілетін орыннан кейін орналасса, кірістіру бұл жағдайда да дұрыс орындалады

```
int main(){
   pair_sl p[3] = { pair_sl("Anna", 123123),
                    pair_sl("Maria", 234234),
                    pair_sl("Teresa", 345345)};
   map_sl m1;
   map_sl :: iterator i = m1.begin();
   for (int k = 0; k<3; k++)
      i = m1.insert(i, p[k]);          // sic!
   for (i = m1.begin(); i != m1.end(); i++)
      cout << (*i).first << " " << (*i).second << endl;
   return 0;
}
```

Ал **insert** функциясының үшінші формасы итераторлар диапазонымен анықталатын элементтер тобын кірістіру үшін қолданылады. Элементтерді өшіру және сөздікті тазарту функциялары басқа контейнерлердің осы аттас функцияларына ұқсас жұмыс істейді: **erase** функциясының бірінші формасы сөздіктен итератор арқылы берілген позициядағы элементті, екіншісі элементті кілті бойынша, ал үшіншісі элементтер диапазонын өшіреді.

Сөздікке кірістіру операциялары олармен байланысқан итераторлар мен сілтемелерді жарамсыз етіп тастамайды, ал өшіру операциялары тек өшірілетін элементтермен байланысқан итераторлар мен сілтемелердің күшін жояды.

Екі сөздіктің барлық элементтерін өзара алмастыру үшін **swap** функциясы қолданылады:

```
template <class Key, class T, class Compare>
   void swap(map<Key, T, Compare>& x,
             map<Key, T, Compare>&y);
```

Ал **equal_range** функциясы өзіне берілген **x** мәні үшін (**lower_bound(x)**, **upper_bound(x)**) итераторлар жұбын қайтарады:

```
pair<iterator,iterator> equal_range(const key_type& x);
pair<const_iterator, const_iterator>
   equal_range(const key_type& x) const;
```

Егер берілген кілті бар элемент табылса, функцияны шақырған соң екі итератор да осы элементке, кері жағдайда одан үлкен алғашқы элементке нұсқап тұрады.

Дубликатты сөздіктер (multimap)

Жоғарыда айтылғандай, дубликатты сөздіктер (**multimap**) кілттері бірдей элементтерді сақтауды қолдайды. Сондықтан олар үшін индекс бойынша қол жеткізу **[ ]** операциясы анықталмаған, ал **insert** функциясының көмегімен қосу операциясы кез келген жағдайда сәтті орындалады. Функция итераторды кірістірілген элементке қайтарады.

Кілттері бірдей элементтер сөздікте енгізілу ретімен сақталады. Элементті кілті бойынша өшіру кезінде **erase** функциясы өшірілген элементтер санын қайтарады. Ал **equal_range** функциясы кілті берілген элементтің барлық орналасқан нұсқаларын анықтайтын итераторлар диапазонын қайтарады. Келесі **count** функциясы 1-ден үлкен мән қайтаруы мүмкін. Басқа тұрғылардан алғанда, дубликатты сөздіктер қарапайым сөздіктерге ұқсас болып келеді.

Жиындар (set)

Жиын – бұл тек кілттердің мәнін сақтайтын ассоциативті контейнер, яғни мұнда **value_type** типі **Key** типіне сәйкес келеді. Кілттердің мәндері қайталанбауы тиіс. Жиын шаблонының екі параметрі бар: кілттің типі және «кіші» қатынасын анықтайтын функционалдық объектінің типі:

```
template <class Key, class Compare = less<Key> >
class set{
public:
   typedef Key key_type;
   typedef Key value_type;
   explicit set(const Compare& comp = Compare());
   template <class InputIter>
      set(InputIter first, InputIter last,
         const Compare& comp = Compare());
   set(const set<Key, Compare>& x);
   pair<iterator,bool> insert(const value_type& x);
   iterator insert(iterator position, const value_type& x);
   template <class InputIter>
      void insert(InputIter first, InputIter last);
   void      erase(iterator position);
   size_type erase(const key_type& x);
   void      erase(iterator first, iterator last);
   void      swap(set<Key, Compare>&);
   void clear();
   iterator  find(const key_type& x) const;
   size_type count(const key_type& x) const;
   iterator  lower_bound(const key_type& x) const;
   iterator  upper_bound(const key_type& x) const;
   pair<iterator,iterator>
      equal_range(const key_type& x) const;
   ...
};
```

Қысқартылған түрде берілген сипаттамадан жиынның интерфейсі сөздіктің интерфейсіне ұқсас екенін байқауға болады. Төменде бүтін сандар жиыны құрылатын қарапайым мысал берілген:

```
#include <iostream>
#include <set>
using namespace std;
typedef set<int, less<int> > set_i;
set_i::iterator i;
int main(){
   int a[4] = {4, 2, 1, 2};
   set_i s1;               // Бос жиын құрылады
   set_i s2(a, a + 4);     // Жиын жиымды көшіру арқылы
                           // құрылады
   set_i s3(s2);     // Көшіру конструкторы жұмыс істейді
   s2.insert(10);    // Элементтерді кірістіру
   s2.insert(6);
   for ( i = s2.begin(); i != s2.end(); i++) // Шығару
      cout << *i << " ";
   cout << endl;
   // equal_range нәтижесін сақтауға арналған айнымалы:
   pair <set_i::iterator, set_i::iterator > p;
   p = s2.equal_range(2);
   cout << *(p.first)<< " " << *(p.second) << endl;
   p = s2.equal_range(5);
   cout << *(p.first)<< " " << *(p.second) << endl;
   return 0;
}
```

Программаның орындалу нәтижесі:

```
1 2 4 6 10
2 4
6 6
```

Сөздіктердегі сияқты, жиында да элементтер сұрыпталған түрде сақталады. Қайталанатын элементтер жиынға енгізілмейді.

Жиындармен жұмыс істеу үшін стандартты кітапханада 395 б. сипатталатын алгоритмдер анықталған.

Дубликатты жиындар (multiset)

Дубликатты жиындарда кілттер қайталануы мүмкін, сондықтан элементті кірістіру операциясы әрқашан сәтті орындалады және **insert** функциясы итераторды кірістірілген элементке қайтарады. Кілттері бірдей элементтер жиында енгізілу ретімен сақталады. Ал **find** функциясы итераторды алғашқы табылған элементке қайтарады немесе берілген кілті бар элемент табылмаған жағдайда, **end()** мәнін қайтарады.

Мәндері бірдей кілттермен жұмыс істеу кезінде **multiset** контейнерінде **count**, **lower_bound, upper_bound** және **equal_range** функцияларын жиі қолданады, олардың әрекеттері дубликатты сөздіктердегі функцияларға ұқсас болып табылады.

Биттік жиындар (bitset)

Биттік жиын биттердің ұзын тізбектерін бейнелеу мен өңдеуге арналған шаблон болып табылады[1]. Нақты алғанда **bitset** – бұл биттік жиым, ол үшін кездейсоқ қол жеткізу, жеке биттерді және жиымды толығымен өзгерту операциялары қамтамасыз етіледі. Биттер оңнан солға қарай нөлден бастап нөмірленеді.

Биттік жиынның шаблоны **<bitset>** тақырыптық файлында анықталған. Шаблонның параметрі ретінде биттік тізбектің ұзындығы алынады, ол тұрақты болуы тиіс:

```
template<size_t N> class bitset {...};
```

Жеке биттті адрестеу үшін **bitset** контейнерінде **reference** класы енгізілген:

```
class reference {
   friend class bitset;
   reference();
   public:
      ~reference();
      reference& operator=(bool x); // b[i] = x; үшін
      reference& operator=(const reference&);
      // b[i] = b[j]; үшін
      bool operator~() const;    // b[i] инверсиясы
      operator bool() const;     // x = b[i]; үшін
      reference& flip();         // b[i] инверсиясы үшін
};
```

Конструкторлар тек нөлдерден, **long** типті мәннен немесе **string**[2] типті тіркестен тұратын биттік жиынды құруға мүмкіндік береді:

```
bitset();                                  // 1
bitset(unsigned long val);                 // 2
explicit bitset(const string& str,         // 3

   string::size_type pos = 0,
   string::size_type n = string::npos);
```

[1]Орналастыруға **int** типті элемент жеткіліксіз, бірақ **long** типті элемент жеткілікті болатын биттік тізбектерді жүзеге асырылуына байланысты бүтін сандарға қолданылатын биттік операциялардың көмегімен өңдеу тиімдірек болуы мүмкін (25 б. қараңыз). Бір сөзге сыятын қысқа тізбектер биттік жиынның көмегімен айтарлықтай тиімді өңделуі мүмкін.

[2]Конструкторлардың сипаттамалары қысқартылып берілген.

Бірінші конструктор нөлдерден тұратын биттік жиынды құрады, екіншісі **long** типті мән қабылдайды және жиындағы әрбір битті осы мәннің ішкі көрінісінің (бейнесінің) сәйкес битімен инициалдайды. Үшінші конструктор нөлдер мен бірлерден құралуы тиіс тіркесті қабылдайды (олай болмаса, **invalid_argument** аластамасы туындайды) және жиынның әрбір битін тіркес символының мәніне сәйкес түрде инициалдайды. Конструктордың екінші және үшінші параметрлері тіркестің басталу позицияларын және инициалдау үшін қолданылатын символдар санын тағайындайды. Келісім бойынша тіркес толығынан пайдаланылады.

Биттік жиындарды құру мысалдары:

```
bitset <100> b1;                        // жүз нөл бар
bitset <16> b2 (0xf0f);                 // 0000111100001111
bitset <16> b3 ("0000111100001111");    // 0000111100001111
bitset <5> b4 ("00110011", 3);          // 10011
bitset <3> b5 ("00110101", 1, 3);       // 011
```

Биттік жиындармен келесі **операцияларды** орындауға болады:

```
bool operator==(const bitset<N>& rhs) const;
bool operator!=(const bitset<N>& rhs) const;
bitset<N>& operator&=(const bitset<N>& rhs);
bitset<N>& operator|=(const bitset<N>& rhs);
bitset<N>& operator^=const bitset<N>& rhs);
bitset<N> operator<<(size_t pos) const;
bitset<N> operator>>(size_t pos) const;
bitset<N>& operator<<=(size_t pos);
bitset<N>& operator>>=(size_t pos);
bitset<N>& set();
bitset<N>& set(size_t pos, int val = true);
bitset<N>& reset();
bitset<N>& reset(size_t pos);
bitset<N> operator~() const;
bitset<N>& flip();
bitset<N>& flip(size_t pos);
reference operator[](size_t pos);      // b[i];
```

Жиындарды теңдік (**==**) және теңсіздік (**!=**) арқылы салыстыруға болады. Ал **<<** және **>>** операциялары сәйкесінше оңға және солға қарай **pos** битке ығыстырылған биттік жиынтықтарды құрады. Ығыстыру кезінде босатылатын позициялар нөлмен толтырылады. Келесі **set** операциясы жиынның барлық биттеріне 1 мәнін, **reset** операциясы 0 мәнін орнатады. Ал **~** операциясы қосымша жиынтықты құрады. Әрбір биттің немесе **pos** параметрі арқылы берілген биттің мәнін **flip** көмегімен терістеуге (инверсиялауға) болады.

Жеке битке индексация операциясының көмегімен қол жеткізуге болады. Егер индексация мәні жиынтық шекарасынан шығатын болса, **out_of_range** аластамасы туындайды.

Биттік жиынның **bitset** шаблонында ұзын бүтінге және тіркеске түрлендіру, сонымен қатар жиын мәндерін талдау әдістері анықталған:

```
unsigned long to_ulong() const; // unsigned long
string to_string() const        // string типінде
size_t count() const;           // биттік 1 саны
size_t size() const;            // биттер саны
bool test(size_t pos) const;  // true, b[pos]== 1 болса
bool any() const;             // true, кем дегенде бір
                              // бит бірге тең болса
bool none() const;            // true, егер бірде-бір бит
                              // бірге тең емес болса
```

Сонымен қатар, қарапайым енгізу және шығару операциялары **<<** және **>>** анықталған. Жиын биттері '0' және '1' символдарының көмегімен солдан оңға қарай шығарылады, ең үлкен бит сол жақта болып есептеледі.

Биттік жиында итераторлар анықталмаған, сондықтан ол нағыз контейнер болып табылмайды, өйткені ол контейнерлердің стандартты интерфейсін толық қамтамасыз етпейді.

Контейнерлерді қолдану мысалы

Төменде көрсетілген программа берілген мәтіндік файл үшін нұсқауышты, яғни онда кездесетін сөздердің алфавит бойынша реттелген тізімін қалыптастырады, бұл сөздердің әрқайсысы үшін ол орналасқан жолдардың нөмірлері жазылған. Егер сөз жолда бір реттен артық кездесетін болса, жолдың нөмірі бір рет шығарылады.

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <set>
#include <map>
using namespace std;
typedef set<int, less<int> > set_i;
typedef map<string, set_i, less<string> > map_ss;
bool wordread(ifstream &in, string &word, int &num){
   char ch;
   // Бірінші әріпке дейін өткізу:
   for (;;){
      in.get(ch);
```

```
      if (in.fail()) return false;
      if (isalpha(ch) || ch == '_') break;
      if (ch == '\n') num++;
   }
   word = ""
   // Сөздің соңын іздеу:
   do{
      word += tolower(ch);
      in.get(ch);
   }while (!in.fail() && (isalpha(ch) || ch == '_'));
   if (in.fail()) return false;
   in.putback(ch);       // Егер символ - '\n'
   return true;
}
int main(){
   map_ss m;
   map_ss::iterator im;
   set_i::iterator is, isbegin, isend;
   string word;
   int num = 1;
   ifstream in ("some_file");
      if (!in){cout << "Cannot open input file.\n";
         exit(1);
      }
   while (wordread(in, word, num)){
      im = m.find(word);
      if (im == m.end())
         im = m.insert(map_ss::value_type(word,
            set_i())).first;
      (*im).second.insert(num);
   }
   for (im = m.begin(); im != m.end(); im++){
      cout << setiosflags(ios::left) << setw(15)
           << (*im).first.c_str();
      isbegin = (*im).second.begin();
      isend = (*im).second.end();
      for (is = isbegin; is != isend; is++)
          cout << " " << *is;
      cout << endl;
   }
   return 0;
}
```

Some_file кіріс файлында келесі мәтін бар делік:

```
class value_compare:
   public binary_function <value_type, value_type,
      bool> {
   friend class map; protected:
   Compare comp;
   value_compare(Compare c) : comp(c) {} public:
   bool operator() (const value_type& x,
   const value_type& y) const {
      return comp(x.first, y.first);}
};
```

Мұнда программа экранға келесі нәтижені шығарады:

```
binary_function 2
bool            2 6
c               5
class           1 3
comp            4 5 8
compare         4 5
const           7
first           8
friend          3
map             3
operator        6
protected       3
public          2 5
return          8
value_compare   1 5
value_type      2 7
x               7 8
y               7 8
```

Программаның жұмысын толығырақ қарастырайық. Мысалдағы **wordread** функциясы кіріс файлдан кезекті сөзді оқиды және ағымдағы жолдың нөмірін қалыптастырады, мұнда сөздің құрамына алфавиттік-цифрлық символдар мен астын сызу белгісі болуы мүмкін деп есептеледі.

Құрылатын нұсқауыш «жол – жиын» жұптарынан тұратын сөздікте сақталады. Жол кілт болып табылады, мұнда кіріс файлдан алынған бір сөз сақталады. Бүтін сандар жиыны осы сөз кездесетін жолдардың нөмірлерін сақтайды. Жиын да, сөздік те біздің қатысуымызсыз жылдам іздеуді және элементтерді дубликаттарсыз (көшірмелерсіз) реттелген түрде сақтауды қамтамасыз етеді.

Әрбір сөз үшін **find** функциясының көмегімен оның сөздікте бар екендігі тексеріледі. Егер жоқ болса (функция **end()** қайтарды), сөздікке **insert** функциясының көмегімен осы сөзден және бос жиыннан (**set_i()**

конструкторын шақыру) тұратын жұп қосылады. Осы **insert** функциясы бірінші элементі (**first**) кірістірілген элементке итератор болатын жұпты қайтарады. Ол **im** айнымалысына меншіктеледі.

Келесі операторда **insert** функциясының көмегімен жиынға жаңа элементті қосу орындалады, ол сөздіктің екінші элементі (**second**) болып табылады.

Осыдан кейін сөздікті шығару орындалады. Әрбір сөз үшін циклде жолдар нөмірлерінен тұратын жиынның элементтері шығарылады.

Контейнерлік кластарға практикумның [12] алтыншы семинары арналған.

13-ТАРАУ

Итераторлар және функционалдық объектілер

Итераторлар мен функционалдық объектілер стандартты кітапханада кеңінен қолданылады. Итератор түсінігі осының алдындағы бөлімде контейнерлік кластарды қарастыру кезінде енгізілді, функционалдық объектілермен басымдылықтары бар кезектерді қарастыру кезінде танысқан болатынбыз. Оларды осыдан ары «Алгоритмдер» бөлімінде (373 б.) сипаттаалатын кітапхана алгоритмдерінде қолдану үшін толығырақ қарастырайық.

Итераторлар

Мәліметтерді кез келген бір абстрактылы тізбек түрінде елестетейік. Бізге оның ұйымдастырылу тәсіліне және мәліметтер типіне тәуелсіз түрде, жеке элементтер бойынша тізбекті қарап шығу және оның әрбір элементіне қол жеткізу құралдары қажет болады. Итератор осы қарап шығу құралдарын ұйымдастыруды қамтамасыз етеді.

Итератор – бұл нұсқауыш ұғымының әртүрлі мәліметтер құрылымдарымен стандартты тәсіл арқылы жұмыс істеу үшін жалпы түрде қарастырылуы. Әртүрлі құрылымдағы мәліметтермен дұрыс әрі тиімді жұмыс істейтін алгоритмдерді жүзеге асыру мүмкін болуы үшін стандарт интерфейсті ғана емес, итераторлардың көмегімен қол жеткізу уақытына қойылатын талаптарды да анықтайды.

Итератор «нұсқауыш» ұғымының жалпылама ұғымы болғандықтан, олардың семантикасы бірдей болады және параметр ретінде итераторларды қабылдайтын барлық функциялар да қарапайым нұсқауыштармен осылай жұмыс істей алады.

Стандартты кітапханада итераторлар контейнерлік кластармен, ағымдармен және ағым буферлерімен жұмыс істеу үшін қолданылады.

Итераторларда «ағымдағы нұсқалған элемент» және «келесі элементке нұсқау» ұғымдары қолданылады. Тізбектің ағымдағы элементіне қол жеткізу, қарапайым нұсқауыштардағы сияқты, `*` және `->` операцияларының көмегімен орындалады. Ал келесі элементке көшу инкремент `++` операциясы арқылы жүзеге асырылады. Сонымен қатар, барлық итераторлар үшін меншіктеу, теңдікке және теңсіздікке тексеру операциялары анықталған.

Мәліметтер әртүрлі жолдармен – мысалы, жиым, тізім, вектор немесе бұтақ түрінде ұйымдастырылуы мүмкін. Тізбектің жеке бір түрі үшін әртүрлі операциялар жиынтығын сүйемелдейтін итератордың өзіндік типі болуы тиіс. Қамтамасыз етілетін операцияларының жиынтығына сәйкес итераторлар

13.1-кестеде сипатталған бес категорияға, яғни санаттарға бөлінеді. Кестеде келтірілген операциялар тұрақты уақыт ішінде орындалады.

Айталық, **i** және **j** – бір түрдегі итераторлар, **x** – тізбек элементімен бірдей типтегі элемент, **n** – бүтін шама болсын. Онда келесі өрнектерді жазуға болады:

```
i++   ++i   i = j   i == j   i != j
```

13.1-кесте. Итераторлардың категориялары

Итератор категориясы	**Операциялар**	**Контейнерлер**
кіріс (**input**)	`x = *i`	барлығы
шығыс (**output**)	`*i = x`	барлығы
тура (**forward**)	`x = *i, *i = x`	барлығы
екібағытты (**bidirectional**)	`x = *i, *i = x,` `--i, i--`	барлығы

кездейсоқ қол жеткізілетін (**random access**)	`x = *i, *i = x,`
	`-- i , i--`
	`i + n, i - n, i += n, i -= n`
	`i < j, i > j, i <= j, i >= j`
		list класынан басқаларының барлығы

Кестеден байқап отырғанымыздай, *тура* итератор кіріс және шығыс итераторларының барлық операцияларын сүйемелдейді және олар кіріс немесе шығыс итераторлары қажет болған барлық жерлерде қолданылуы мүмкін. *Екібағытты* итератор тура итератордың барлық операцияларын сүйемелдейді, декрементті де қолдайды және тура итератор қолданылатын барлық жерлерде пайдаланылуы мүмкін. *Кездейсоқ қол жеткізілетін* итератор екібағытты итератордың барлық операцияларын сүйемелдейді, сонымен қатар, тізбектің кездейсоқ элементіне көшуді және операторларды салыстыру әрекеттерін де орындайды.

Итераторлар иерархия құрады деп айтуға болады, оның жоғарғы деңгейінде кездейсоқ қол жеткізу итераторлары орналасады. Итератордың деңгейі неғұрлым жоғары болған сайын, ол құрамында жұмыс істейтін контейнерге соғұрлым жоғары функционалдық талаптар қойылады. Мысалы, тізімдер үшін кездейсоқ қол жеткізу итераторларын қолдануға болмайды, өйткені тізім итератордың қажетті операциялар жиынтығын сүйемелдемейді.

Итерациялық кластар мен функциялар **<iterator>** тақырыптық файлында сипатталған. Стандартты контейнерлерді қолдану кезінде бұл файл автоматты түрде іске қосылады.

Итераторлар тұрақтылық болуы мүмкін. Тұрақтылық итераторлар контейнердің соған сәйкес элементтерінің мәндерін өзгерту қажеттілігі болмаған кезде қолданылады.

Итератор *жарамды* (ол қандай да бір элементке нұсқап тұрса) немесе *жарамсыз* болуы мүмкін. Итератор келесі жағдайларда жарамсыз болып қалуы мүмкін:

- итератор инициалданған жоқ;
- ол байланысқан контейнер өлшемдері өзгерген немесе ол жойылған;
- итератор тізбектің соңына нұсқайды.

Тізбектің соңы оның соңғы элементінен кейін орналасқан элементке нұсқауыш түрінде бейнеленеді. Мұндай нұсқауыш әрқашанда бар болады. Осындай тәсілді қолдану бос тізбекті ерекше жағдай ретінде қарастырмауға мүмкіндік береді. «Нөлдік итератор» ұғымы болмайды.

Итераторлар өздері нұсқайтын объектілермен жұмыс істеу үшін (мысалы, контейнер элементінің мәнін алу үшін) қолданылатындықтан, соларға сәйкес типтерді анықтау қажет. Ол үшін **<iterator>** тақырыптық файлында **iterator_traits** шаблоны (**trait** ағылшын тілінен аударғанда — ортақ сипаты) анықталған.

```
template<class Iter> struct iterator_traits{
   typedef typename Iter::difference_type difference_type;
   typedef typename Iter::value_type value_type;
   typedef typename Iter::pointer pointer;
   typedef typename Iter::reference reference;
   typedef typename Iter::iterator_category
                          iterator_category;
};
```

Мысалдағы **typename** түйінді сөзі компилятор **Iter** сөзін типтің атауы ретінде тануы үшін қажет. **iterator_category** – бұл итератордың типі, ол итератордың қандай операцияларды қолдайтынын анықтайды. **difference_type** типі екі итератордың арасындағы айырманы өрнектеу қызметін атқарады.

Тақырыптық файлда келесідей итераторлар типтері сипатталған:

```
struct input_iterator_tag{};
struct output_iterator_tag{};
struct forward_iterator_tag:
   public input_iterator_tag{};
struct bidirectional_iterator_tag:
   public forward_iterator_tag{};
struct random_access_iterator_tag:
   public bidirectional_iterator_tag{};
```

Итератордың типін итераторды қолданатын функцияның шаблонының параметрлер тізімінде көрсетуге болады, бұл әрекет компиляция кезеңінде барынша тиімді жүзеге асыруды қамтамасыз ететін итератордың типін таңдау үшін жасалады, өйткені итератордың алуан түрлері үшін бір алгоритм әртүрлі тиімділікпен жүзеге асырылуы мүмкін.

Қарапайым нұсқауыштар үшін олардың тиімділігін арттыру мақсатында **iterator_traits** шаблонының специализациялары анықталған (шаблондардың специализациялары туралы 238 б. жазылған):

```
template<class T> struct iterator_traits<T*>{
   typedef ptrdiff_t difference_type;
   typedef T value_type;
   typedef T* pointer;
   typedef T& reference;
   typedef random_access_iterator_tag iterator_category;
};
```

Екі нұсқауыштың айырмасы **<cstddef>** файлында анықталған стандартты **ptrdiff_t** типінде болады. Осыған ұқсас шаблон специализациясы **const T*** типі үшін де енгізілген.

Тек кездейсоқ қол жеткізу итераторлары ғана **+** және – операцияларын сүйемелдейтін болғандықтан, кітапханада **distance** және **advance** функциялары сипатталған. **distance** функциясы контейнер элементтерінің арасындағы қашықтықты анықтау үшін қолданылады:

```
distance(InputIterator first, InputIterator last);
```

Бұл екі итератордың арасындағы айырманы көрсететін **difference_type** типті мән қайтарады. Бұл функция кездейсоқ қол жеткізу итераторларынан басқа барлық итераторлар үшін **++** операциясының көмегімен контейнер элементтерінің арасындағы қашықтыққа пропорционал уақыт ішінде орындалады, яғни ол айтарлықтай ресурс көлемін жұмсайды. **advance** функциясы да осыған ұқсас жұмыс түрін атқарады, ол **i += n** операциясын жүзеге асыру үшін қолданылады:

```
void advance(InputIterator& i, Distance n);
```

Мұндағы **n** шамасы тек екібағытты және кездейсоқ қол жеткізу итераторлары үшін теріс мәнді болуы мүмкін.

Кері итераторлар

Екібағытты және кездейсоқ қол жеткізу итераторлары үшін олардың *итераторлардың адаптерлері* деп аталатын түрлері анықталған. Тізбекті кері бағытта қарап шығатын адаптер **reverse_iterator** деп аталады:

```
template <class It>
   class reverse_iterator : public
   iterator<typename
            iterator_traits<It>::iterator_category,
   typename iterator_traits<It>::value_type,
   typename iterator_traits<It>::difference_type,
   typename iterator_traits<It>::pointer,
```

```
    typename iterator_traits<It>::reference>{
    protected:
       It current;
    public:
       typedef It iterator_type;
       reverse_iterator();
       explicit reverse_iterator(It x);
       template <class U> reverse_iterator(const
          reverse_iterator<U>& u);
       It base() const;
       reference operator*() const;
       pointer operator->() const;
       reverse_iterator& operator++();
       reverse_iterator operator++(int);
       reverse_iterator& operator--();
       reverse_iterator operator--(int);
       reverse_iterator operator+ (difference_type n) const;
       reverse_iterator& operator+=(difference_type n);
       reverse_iterator operator- (difference_type n) const;
       reverse_iterator& operator-=(difference_type n);
       reference operator[](difference_type n) const;
};
```

Кері итератордың ағымдағы итератор – **current** деп аталатын қорғалған мәліметтер өрісі бар. Инкремент операциясы осы итератордың декременті арқылы жүзеге асырылады:

```
template <class It>
   reverse_iterator<It>& reverse_iterator<It> operator++(){
      --current;
      return *this;
}
```

Итератор контейнердің барлық элементтеріне, оның ішінде соңғыдан кейін орналасқан элементіне де нұсқай алады, бірақ кері итератор үшін мұндай элементтің орнында алғашқы элементтің алдындағысы болуы тиіс, ал ондай элемент жоқ. Сондықтан **current**, шын мәнінде, кері итератор нұсқайтан элементтен кейін орналасқан элементке сілтеме жасайды. Тура және кері итераторлар арасындағы қатынас келесі өрнекпен беріледі:

```
&*(reverse_iterator(i)) == &*(i - 1).
```

Кері итераторлар үшін қарапайым қатынас операциялары **==**, **!=**, **<**, **<=** және **=** сүйемелденеді.

Контейнерлік кластардағы кері итерторлар олардың элементтерін кері бағытта қарап шығу үшін сипатталған. Сонымен қатар, **reverse_iterator**

мәнін қайтаратын **rbegin()** және **rend()** әдістері сипатталған. Мысалы, векторды кері бағытта қарап шығу үшін келесі циклді қолдануға болады:

```
vector <int> v;
...
for (vector <int> reverse_iterator i = v.rbegin();
   i != v.rend; ++i)
   cout << *i << " ";
```

Егер контейнер **const** ретінде (мысалы, функцияға берілетін параметрлер тізімінде) жарияланған болса, онда **const** префиксі бар итераторды – **const_reverse_iterator** қолдану қажет.

Кірістіру итераторлары

Кірістіру итераторлары да, кері итераторлар сияқты, итераторлардың адаптерлері болып табылады. Олар контейнердің басына, соңына немесе кездейсоқ жеріне жаңа элементтер қосу үшін қолданылады. Стандартты кітапханада шығыс итераторлары негізінде құрылған кірістіру итераторлары кластарының үш түрлі шаблоны анықталған, олар: **back_insert_iterator**, **front_insert_iterator**, **insert_iterator**.

Сонымен қатар, үш кірістіру функциясы да анықталған:

```
template <class C> back_insert_iterator<C>
   back_inserter(C& x);
template <class C> front_insert_iterator<C>
   front_inserter(C& x);
template <class C, class Iter> insert_iterator<C>
   inserter(C& x, Iter i);
```

Мұндағы **C** – ішіне элементтерді кірістіру қажет етілетін контейнер. Келесі **back_inserter** функциясы элементтерді контейнердің соңына, **front_inserter** контейнердің басына кірістіреді, ал **inserter** функциясы оның аргументитераторы сілтеме жасап тұрған элементтің алдына кірістіреді. Бұл функциялар өздеріне сәйкес типтегі итераторларды қайтарады және көбінесе стандартты алгоритмдердің аргументтері ретінде қолданылады (385 б.), мұнда олар жұмыс тәртібін – контейнер элементтерін алмастыру немесе кірістіру тәртіптерінің бірін анықтайды.

Ағымдық итераторлар

Келесі бөлімде қарастырылатын стандартты алгоритмдер енгізу/шығару ағымдарын тікелей қолдану мүмкіндігіне ие болуы үшін ағымдық итераторлар енгізілген. Ағымдар тізбектер түрінде бейнеленеді. Ағымдық итераторлар кластарының екі шаблоны анықталған, олар: кіріс ағымының итераторы **istream_iterator** және шығыс ағымының итераторы **ostream_iterator**.

Кіріс ағымының итераторы өзі арналып құрылған ағымнан элементтерді оқиды, осыдан кейін оларды итератор үшін үйреншікті тәсілмен, яғни адрессіздендіру операциясы арқылы пайдалануға болады. Мысалы, **temp** атты файлдан бүтін санды оқу үшін келесі программа үзіндісін қолдануға болады:

```
istream in("temp");
istream_iterator<int> i(in);
int buf = *i;
```

Кіріс ағымынан кезекті мән инкремент операциясының орындалуы кезінде оқылады, мысалы:

```
++i;
int buf1 = *i;
```

Кіріс ағымының соңына жеткен кезде итератор енгізу операциясының соңы мәнін қабылдайды. Итератор конструкторында да келісім бойынша осы мән болады, сондықтан файлдан мәлімет оқу циклін келесідей түрде ұйымдастыруға болады:

```
while ( i != istream_iterator<int> () )
   cout << *i++ << " ";
```

Кіріс ағымының итераторлары үшін теңдікке және теңсіздікке тексеру операциялары анықталған. Енгізу операциясының соңына тең барлық итераторлар да өзара тең болады. Енгізу операциясының соңына жетпеген (соңына тең емес) итераторларды олардың барлығы да бір ағым үшін құрылған жағдайда бір-бірімен салыстыруға болады.

Кіріс ағымы итераторларының ерекшелігі олардың инкремент операциясынан кейін теңдікті сақтамауында болып табылады, яғни **i == j** болса, **++i == ++j** болуы міндетті емес. Сондықтан оларды тек бір рет қана орындалатын алгоритмдерде қолдануға ұсыныс беріледі. Егер алгоритмнің бірнеше рет орындалуы талап етілсе, онда жедел жадындағы мәліметтер құрылымдарын қолданған дұрысырақ болып саналады.

Шығыс ағымының итераторы элементтерді **<<** операциясының көмегімен өзі үшін арнайы құрылған шығыс ағымына жазады. Егер конструктордың екінші аргументі символдар тіркесі болса, ол әрбір шығарылатын мәннен кейін жазылады:

```
ostream_iterator<int> os(cout, " кг");
*os = 100;          // 100 кг шығарылады
++os; *os = 2;      // 2 кг шығарылады
```

Меншіктеу операциясының жүзеге асырылуын қарастырайық:

```
ostream_iterator& operator=(const T& value){
   *out_stream << value;
   if(delim != 0) *out_stream << delim; return (*this);
}
```

Мұндағы **out_stream** – итератор конструкторының бірінші параметрі ретінде берілген шығыс ағымына сілтеме, **delim** – конструктордың екінші параметрінде көрсетілген тіркес.

Ағымдық итераторларды кітапхана алгоритмдерімен бірге қолдану мысалы 380 б. келтірілген.

Функционалдық объектілер

Функционалдық объект деп функцияны шақыру операциясы анықталған класты (211 б. қараңыз) айтады. Функционалдық объектілермен басымдылықтары бар кезектерді (339 б.) қарастыру кезінде танысқан болатынбыз. Көбінесе бұл объектілер объектілерді салыстырудың қолданушылық критерилерін немесе оларды өңдеу тәсілдерін тағайындауға арналған стандартты алгоритмдердің параметрлері ретінде қолданылады.

Параметр ретінде функционалдық объектіні қолдануға болатын алгоритмдерде функцияға нұсқауышты да қолдануға болады. Мұндайда **()** операциясын құрамдас операция ретінде анықтауға болатындықтан, функционалдық объектіні қолдану тиімдірек болып шығуы мүмкін.

Стандартты кітапханада оны тиімді түрде қолдануға және кеңейтуге қажетті көптеген функционалдық объектілер бар. Олар **<functional>** тақырыптық файлында сипатталған. Осындай объектілердің ішінен **bool** типті мән қайтаратын объектілерді ерекшелеп айтуға болады. Мұндай объектілер *предикаттар* деп аталады. Предикат деп **bool** мәнін қайтаратын қарапайым функцияларды да атайды.

Кітапханада аргументтер типтері үшін стандартты атауларды енгізетін базалық кластар ретінде унарлы және бинарлы функция шаблондары анықталған:

```
template <class Arg, class Result>
   struct unary_function{
      typedef Arg    argument_type;
      typedef Result result_type;
};
template <class Arg1, class Arg2, class Result>
   struct binary_function{
      typedef Arg1   first_argument_type;
      typedef Arg2   second_argument_type;
      typedef Result result_type;
};
```

Кітапхананың функционалдық объектілері осы базалық объект-функциялардың ұрпақтары болып табылады. Аргументтер мен нәтиже үшін типтердің стандартты атауларын анықтау функционалдық объектілерді адаптерлермен және кітапхананың басқа да құралдарымен бірге пайдалану үшін қажет.

Функцияның адаптері деп аргумент ретінде функцияны қабылдап, одан басқа функция қалыптастыратын функцияны атайды. Функцияның орнында функционалдық объект те болуы мүмкін.

Стандарты кітапханада бірнеше адаптер типтерінің сипаттамалары бар:

- екі аргументі бар функционалдық объектіні бір аргументі бар объект сияқты қолдану үшін пайдаланылатын *байланыстырғыштар;*
- предикат мәнін инверсиялау (терістеу) үшін қажетті *терістеуіштер*;
- *функцияға нұсқауыштардың адаптерлері;*
- алгоритмдерде әдістерді қолдану үшін қажет болатын *әдістер адаптерлері.*

Осы аталған адаптерлердің синтаксисі мен қолданылуы келесі бөлімдерде қарастырылады.

Арифметикалық функционалдық объектілер

Стандартты кітапханада C++ тілінде анықталған барлық арифметикалық операциялар үшін функционалдық объектілердің шаблондары анықталған.

13.2-кесте. Арифметикалық функционалдық объект

Атауы	Типі	Нәтижесі
`plus`	бинарлы	`x + y`
`minus`	бинарлы	`x - y`
`multiplies`	бинарлы	`x * y`
`divides`	бинарлы	`x / y`
`modulus`	бинарлы	`x % y`
`negate`	унарлы	`-x`

Төменде **plus** объектісінің шаблоны берілген (қалған объектілер осыған ұқсас жолмен сипатталады):

```
template <class T> struct plus :
   binary_function<T, T, T>{
      T operator()(const T& x, const T& y) const{
         return x + y;
   }
};
```

Функционалдық объектілер көбінесе стандартты кітапхана алгоритмдерімен бірге қолданылатын болғандықтан, арифметикалық функционалдық объектілердің қолданылу мысалдары «Алгоритмдер» бөлімінде, 387 және 402 беттерде берілген.

Предикаттар

Стандартты кітапханада C++ тілінде анықталған салыстыру операциялары мен логикалық операциялар үшін функционалдық объектілердің шаблондары анықталған. Олар **bool** типті мән қайтарады, яғни предикаттар болып табылады.

13.3-кесте. Стандартты кітапхана предикаттары

Атауы	Типі	Нәтижесі
equal_to	бинарлы	**x == y**
not_equal_to	бинарлы	**x != y**
greater	бинарлы	**x > y**
less	бинарлы	**x < y**
greater_equal	бинарлы	**x >= y**
less_equal	бинарлы	**x <= y**
logical_and	бинарлы	**x && y**
logical_or	бинарлы	**x || y**
logical_not	унарлы	**!x**

Төменде **equal_to** объектісінің шаблоны берілген (қалған объектілер осыған ұқсас түрде сипатталған):

```
template <class T> struct equal_to :
   binary_function<T, T, bool>{
      bool operator()(const T& x, const T& y) const{
         return x == y;
   }
};
```

Программалаушы объектілерді салыстыру критерилерін анықтау үшін өзіндік предикаттарын сипаттай алады. Бұл әрекет контейнер қолданушы анықтайтын типтегі элементтерден тұратын кезде қажет. Мысалы, 197 б. сипатталған **monstr** класы үшін екі объектіні **ammo** өрісінің мәні бойынша салыстыратын предикат төмендегідей түрде болады:

```
struct monstr_less_ammo:
   public binary_function<monstr, monstr, bool>{
      bool operator ()(monstr & ml, monstr & m2){
         return m1.get_ammo() < m2.get_ammo();
      }
};
```

Предикаттарды қолдану мысалдары 367, 391, 394 беттерде берілген.

Терістеуіштер

Қарастырылатын **not1** және **not2** терістеуіштері сәйкесінше қарама-қарсы унарлы және бинарлы предикаттарды алу үшін қолданылады. Терістеуіштерді қолдану өте жеңіл. Мысалы, **less<int>()** предикатының инверсиясын алу үшін, **not2(less<int>())** өрнегін жазу керек. Ол **greater_equal<int>** өрнегіне эквивалентті болып табылады.

Терістеуіштер қолданушылар тағайындаған предикаттарды инверсиялау (терістеу) үшін қолданылады, өйткені стандартты предикаттар үшін кітапханада оларға сәйкес қарама-қарсы объектілер бар.

Терістеуіштердің жүзеге асырылуын қарастырайық. Кітапханада **unary_function** және **binary_function** объектілерінің өздеріне сәйкес ұрпақтары болып табылатын **unary_negate** және **binary_negate** сияқты функционалдық объектілері сипатталған. Бұл кластар конструктордың параметрі ретінде қажетті тип предикаты болып табылатын объектіні қабылдайды:

```
template <class Predicate>
   class unary_negate: public
   unary_function<typename
                  Predicate::argument_type, bool> {
      public:
         explicit unary_negate(const Predicate& pred);
         bool operator() (const typename
            Predicate::argument_type& x) const;
};
```

Мысалдағы **()** операциясы предикаттың инверсиясын (терістеуін) қайтарады. **not1** және **not2** предикатты қабылдайтын және функционалдық объектіні қайтаратын функциялардың шаблондары ретінде анықталған:

```
template <class Predicate>
   unary_negate<Predicate> not1(const Predicate& pred);
template <class Predicate>
   binary_negate<Predicate> not2(const Predicate& pred);
```

Байланыстырғыштар

Бинарлы предикаттардың көмегімен әртүрлі екі объектіні салыстыруға болады. Көбінесе объектіні басқа объектімен емес, тұрақтымен салыстыру қажет етіледі. Осы мақсатта дәл сол предикатты қолдану үшін оның екі аргументінің біреуін тұрақтымен байланыстыру керек. Ол үшін нақты мәнмен бинарлы функцияның бірінші және екінші аргументтерін байланыстыруға мүмкіндік беретін сәйкесінше **bindlst** және **bind2nd** байланыстырғыштары қолданылады.

Байланыстырғыштар стандартты кітапханада бірінші параметр ретінде екі аргументі бар **f** функционалдық объектісін, ал екінші параметр ретінде онымен түйістірілетін **value** мәнін қабылдайтын функциялардың үлгілері ретінде жүзеге асырылған. Функцияны шақыру нәтижесі функционалдық объект болып табылады, ол **f** кіріс объектісінен оның екінші немесе бірінші аргументіне **value** мәнін «орналастыру» арқылы құрылады.

Қайтарылатын функционалдық объектінің типін сипаттау үшін кітапханада **binder2nd** және **binderlst** кластарының шаблондары анықталған. Төменде байланыстырғыштардың шаблондары көрсетілген:

```
template <class Op, class T>
   binder2nd<Op> bind2nd(const Op& op, const T& x);
template <class Op, class T>
   binderlst<Op> bindlst(const Op& op, const T& x);
```

Мұндағы **Op** – функционалдық объектінің типі, **T** – түйісетін мәннің типі.

Мысалы, бүтін сандық жиымның 40-тан кіші элементтерінің санын анықтау керек болсын делік:

```
#include <iostream>
#include <functional>
#include <algorithm>
using namespace std;
int main(){
   int m[8] = {45, 65, 36, 25, 674, 2, 13, 35};
   cout << count_if(m, m + 8, bind2nd(less<int>(),40));
   return 0;
}
```

Бұл программада элементтер санын есептеу үшін стандартты кітапхананың **count_if** алгоритмі (375 б. қараңыз) қолданылады. Оны пайдалану үшін **<functional>** тақырыптық файлын іске қосу керек. **count_if** үшін алғашқы екі параметр ретінде өңделуші тізбектің басы мен соңын анықтайтын итераторлар алынуы керек. Жиымтың итераторлары ретінде оның элементіне нұсқауыштар қолданылады. **count_if** функциясының үшінші параметрі бинарлы функция немесе функционалдық объект болуы керек. **bind2nd** функциясы **x < y** салыстыру шартын **x < 40** шартына айналдырады.

Функцияларға нұсқауыштар адаптерлері

Байланыстырғыштарды қарапайым функция нұсқауыштарына қолдану үшін адаптерлер деп аталатын арнайы түрлендіргіштер қажет. Стандартты кітапхана екі функционалдық объектіні – **pointer_to_unary_function** унарлы функцияға нұсқауышты және **pointer_to_binary_function** бинарлы функцияға нұсқауышты анықтайды, сонымен қатар өздеріне пара-

метр ретінде берілген функцияға нұсқауышты функционалдық объектіге түрлендіретін бір және екі аргументі бар екі **ptr_fun** функция-адаптерлерін де анықтайды.

Төменде унарлы және бинарлы функциялар үшін функция және класс шаблонының сипаттамасы берілген:

```
template <class Arg, class Result>
   class pointer_to_unary_function :
      public unary_function<Arg, Result>{
      public:
      explicit pointer_to_unary_function(Result
         (*f)(Arg));
      Result operator()(Arg x) const;    // f(x) қайтарады
   };
template <class Arg, class Result>
   pointer_to_unary_function<Arg, Result>
      ptr_fun(Result (*f)(Arg));
template <class Arg1, class Arg2, class Result>
   class pointer_to_binary_function : public
      binary_function<Arg1, Arg2, Result>{
      public:
      explicit pointer_to_binary_function(Result (*f)
         (Arg1, Arg2));
      Result operator()(Arg1 x, Arg2 y) const;
   };
template <class Arg1, class Arg2, class Result>
   pointer_to_binary_function<Arg1, Arg2, Result>
      ptr_fun(Result (*f)(Arg1, Arg2));
```

Функция адаптерін қолдану мысалы:

```
#include <iostream>
#include <functional>
#include <algorithm>
using namespace std;
struct A{
   int x, y;
};
bool lss(A a1, A a2){return a1.x < a2.x;}
int main(){
   A ma[5] = {{2, 4}, {3, 1}, {2, 2}, {1, 2}, {1, 2}};
   A elem = {3, 0};
   cout << count_if(ma, ma + 5, bind2nd(ptr_fun(lss),
   elem));
   return 0;
}
```

Мысалдағы **lss** функциясы **A** құрылымының элементтерін салыстыру ережесін анықтайды. Стандартты кітапхананың **count_if** алгоритмі үшінші параметр арқылы берілген шартты қанағаттандыратын **ma** құрылымындағы элементтер санын есептейді. Бұл параметр функционалдық объект болып табылады, ол **lss** функциясынан **ptr_fun** адаптерінің көмегімен алынған функционалдық объектіден және функцияның екінші параметрінің орнына қойылатын айнымалыдан **bind2nd** байланыстырғышы арқылы құрылады. Осының нәтижесінде **A** құрылымының **x** өрісі үштен кем болатын элементтерінің саны есептеледі (нәтижесі – 4).

Екінші мысал ретінде бұдан бұрын енгізілген (197 б. қараңыз) **monstr** класының объектілерінен тұратын векторды қарастырамыз. Төменде келтірілген программа стандартты кітапхананың **count_if** алгоритмінің (375 б. қараңыз) көмегімен вектордағы денсаулықтары нашарлаған монстрлар (құбыжықтар) санын анықтайды (бұл өкінішті жағдай өріс мәні **health < 30** болғанда орын алады делік). Қысқаша болуы үшін кластың тек қажетті әдістері ғана көрсетілген:

```
#include <iostream>
#include <vector>
#include <functional>
#include <algorithm>
using namespace std;
enum color {red, green, blue};
class monstr{
      int health, ammo;
      color skin;
      char *name;
   public:
      monstr(int he = 100, int am = 10);
      monstr(color sk);
      monstr(char * nam);
      monstr(const monstr &M);
      ~monstr() {delete [] name;}
      operator int(){return health;}
      int get_health(){;return health;}
      friend ostream&
         operator <<(ostream & out, monstr & m){
         return out << "monstr: " << " ammo = "
                    << m.ammo << " health = "
                    << m.health <<endl;}
};
```

```
monstr::monstr(int he, int am):
   health (he), ammo (am), skin (red), name (0){}

monstr::monstr(const monstr &M){
   if (M.name){ name = new char [strlen(M.name) + 1];
      strcpy(name, M.name);}
   else name = 0;
   health = M.health; ammo = M.ammo; skin = M.skin;
}

monstr::monstr(color sk){
   switch (sk){
   case red :
      health = 1; ammo = 10; skin = red; name = 0; break;
   case green:
     health = 2; ammo = 20; skin = green; name = 0; break;
   case blue:
      health = 3; ammo = 40; skin = blue; name = 0; break;
}}

monstr::monstr(char * nam){
   name = new char [strlen(nam) + 1]; strcpy(name, nam);
   health = 200; ammo = 10; skin = red;
}

bool less_health (monstr m1, monstr m2){
   return m1.get_health() < m2.get_health() ;}

int main(){
   vector <monstr> m;
   monstr M (10, 30);
   m.push_back(M);
   m.push_back(monstr("Vasia"));
   m.push_back(monstr(red));
   cout << " MONSTRY:" << endl;
   for (int i = 0; i<m.size(); i++) cout << m[i] << " ";
   cout << endl;
   cout << " COUNT_IF: ";
   cout << count_if(m.begin(), m.end(),
           bind2nd(ptr_fun(less_health), 20));
   return 0;
}
```

Мысалда берілген **less_health** бульдік функциясы өзіне берілген **monstr** класынаң екі аргументінің **health** өрістерін салыстырады. **health** өрісін тұрақтымен салыстыру үшін онымен функцияның екінші аргументін байланыстыру керек. **bind2nd** байланыстырғышы бірінші аргумент ретінде **ptr_fun** адаптерінің көмегімен қалыптастырылатын функционалдық объектіні қабылдауы керек. Программа жұмысының нәтижесі (ауру монстрлар саны – 2):

```
MONSTRY:
monstr: ammo = 30 health = 10
  monstr: ammo = 10 health = 200
  monstr: ammo = 10 health = 1
COUNT_IF: 2
```

Әдістер адаптерлері

Контейнерлерде мәліметтердің қолданушы анықтаған кластарының объектілерін сақтау кезінде контейнердің барлық элементтеріне кластың бір ғана әдісін қолдану мәселесі жиі туындайды. Кітапханада контейнер элементтерін қарап шығуға арналған **for_each** алгоритмі бар. Оған контейнердің әрбір қаралатын элементі үшін шақырылуы тиіс функцияға нұсқауыш беруге болады, мысалы:

```
#include <iostream>
linclude <algorithm>
using namespace std;
void show(int a){ cout << a << endl;}
int main(){
   int m[4] =   {3, 5, 9, 6};
   for_each(m, m + 4, show);
   return 0;
}
```

Егер қарапайым жиымның орнына кез келген бір кластың объектілерінен тұратын контейнерді қолданатын болсақ, онда осыған ұқсас программада **show** функциясының орнына класс әдісінің шақырылуын жазу мүмкін болмайды, өйткені әдісті нақты класс объектісін көрсету арқылы шақыру қажет.

Әдістер адаптерлері кластардың әдістерін стандартты алгоритмдердің аргументтері ретінде қолдануға мүмкіндік береді. Терістеуіш пен байланыстырғыш сияқты, әдіс адаптері де функцияны қабылдап, одан басқа функция қалыптастырады. Стандартты кітапханада аргументтерінің саны әртүрлі болып келетін әдістер үшін бірнеше адаптер анықталған:

13.4-кесте. Кластар әдістерінің адаптерлері

Атауы	Объектінің типі	Әрекеті
mem_fun	**mem_fun_t**	Нұсқауыш арқылы аргументсіз әдісті шақырады
mem_fun	**const_mem_fun_t**	Нұсқауыш арқылы аргументсіз тұрақтылық әдісті шақырады
mem_fun	**mem_fun1_t**	Нұсқауыш арқылы унарлы әдісті шақырады
mem_fun_ref	**mem_fun_ref_t**	Сілтеме арқылы аргументсіз әдісті шақырады
mem_fun_ref	**const_mem_fun_ref_t**	Сілтеме арқылы аргументсіз тұрақтылық әдісті шақырады
mem_fun_ref	**mem_fun1_ref_t**	Сілтеме арқылы унарлы әдісті шақырады
mem_fun_ref	**const_mem_fun1_ref_t**	Сілтеме арқылы унарлы тұрақтылық әдісті шақырады
mem_fun_ref	**const_mem_fun1_t**	Нұсқауыш арқылы унарлы тұрақтылық әдісті шақырады

Мысал ретінде **monstr** класында **dead** әдісін анықтайық, ол монстрдың (құбыжықтың) тірі немесе өлі екенін анықтайды:

```
bool dead(){return !health;}
```

Жоғарыда аталған **count_if** алгоритмінің көмегімен

```
vector <monstr> ostrich (100);
```

түрінде сипатталған вектордағы шығындар санын келесі оператор арқылы есептеуге болады:

```
cout << count_if(ostrich.begin(), ostrich.end(),
        mem_fun_ref(&monstr::dead));
```

14-ТАРАУ

Алгоритмдер

STL кітапханасының алгоритмдері контейнерлермен және басқа да тізбектермен жұмыс істеуге арналған. Әрбір алгоритм шаблон немесе функция шаблондарының жиынтығы түрінде жүзеге асырылған, сондықтан олар тізбектердің алуан түрлерімен және әртүрлі типтегі мәліметтермен жұмыс істей алады. Алгоритмді қолданушының нақты талаптарына сәйкес оны баптау үшін функционалдық объектілер (363 б. қараңыз) қолданылады.

Стандартты кітапхананың басқа құралдары сияқты, стандартты алгоритмдерді қолдану әрекеті де программалаушыны тізбектерді өңдеу циклдерін жазудан, оларды жөндеп түзетуден және құжаттаудан құтқарады, осы арқылы программадағы қателіктер саны азаяды, оның құрылу уақыты қысқарады және ол оқуға жеңіл әрі ықшамды болады.

Стандартты алгортимдердің жариялау **<algorithm>** тақырыптық файлында, ал стандартты функционалдық объектілердің жарияланымдары **<functional>** файлында орналасқан.

Барлық STL алгоритмдерін төрт категорияға бөліп қарастыруға болады:

- тізбектерді модификацияламай орындалатын операциялар;
- тізбектермен орындалатын модификациялаушы операциялар;
- сұрыптаумен байланысты алгоритмдер;
- пирамидалармен және жиындармен жұмыс істеу алгоритмдері.

Сонымен қатар, кітапханада жалпыланған түрдегі сандық алгоритмдер бар, олардың жарияланымдары **<numeric>** файлында орналасады («Сандық есептеулерге арналған құралдар» атты 15-тарауды қараңыз, 401 б.).

Алгоритмге параметр ретінде өңделетін тізбектің басы мен соңын анықтайтын итераторлар беріледі. Итераторлардың түрі осы алгоритмді қолдануға болатын контейнерлер типін анықтайды. Мысалы, сұрыптау алгоритмі (**sort**) өз жұмысы үшін кездейсоқ қол жеткізу итераторын талап етеді, сондықтан ол **list** контейнерімен жұмыс істмейді. Алгоритмдер тізбектің шекараларынан шығып кету мүмкіндігін тексермейді.

Келесі бөлімдердің басында келтірілген кестелер STL стандартты алгоритмдерінің мүмкіндіктері туралы түсінік береді. Ары қарай әрбір алгоритмнің сипаттамасы берілген. Қолданушы анықтаған мәліметтер типтерін қамтитын тізбектер үшін өзіндік критерийлерді тағайындауға болатынын ескерген жөн.

Алгоритм шаблондарының параметрлерін сипаттау кезінде келесідей қысқартулар қолданылады:

In – оқуға арналған итератор;

Out – жазуға арналған итератор;

For – тура итератор;
Bi – екібағытты итератор;
Ran – кездейсоқ қол жеткізу итераторы;
Pred – унарлы предикат (шарт);
BinPred – бинарлы предикат;
Comp – салыстыру функциясы;
Op – унарлы операция;
BinOp – бинарлы операция.

Тізбектерді модификацияламайтын операциялар

Бұл категорияның алгоритмдері тізбекті өзгертпей қарап шығады. Олар тізбек туралы ақпарат алу үшін немесе элементтің орналасу жағдайын анықтау үшін қолданылады.

14.1-кесте. Тізбектерді модификацияламайтын операциялар

Алгоритм	Орындалатын функция
adjacent_find	Көршілес мәндер жұбын табу
count	Мәннің тізбекке ену санын есептеу
count_if	Тізбекте шарттың орындалу санын есептеу
equal	Екі тізбек элементтерінің жұп бойынша теңдігі
find	Мәннің тізбекке алғашқы енгізілуін табу
find_end	Бір тізбектің басқа тізбекке соңғы енгізілуін табу
find_first_of	Бір тізбектегі бірінші мәнді басқа тізбектен табу
find_if	Тізбектегі шартқа алғашқы сәйкестікті табу
for_each	Тізбектің әрбір элементі үшін функцияны шақыру
mismatch	Екі тізбектегі бірдей емес алғашқы элементті табу
search	Бір тізбектің басқа тізбекке алғашқы енгізілуін табу
search_n	Бір тізбектің басқа тізбекке **n**-ші рет енгізілуін табу

Енді осы алгоритмдерді толығырақ қарастырайық.

adjacent_find

adjacent_find алгоритмі көршілес мәндер жұбын табуды орындайды.

```
template<class For> For adjacent_find(For first, For last);
template<class For, class BinPred> For
   adjacent_find(For first, For last, BinPred pred);
```

Алгоритмнің бірінші формасы тізбекті контейнерде көршілес орналасқан бірдей мәндер жұбын табады да, итераторды олардың алғашқысына немесе

тізбектің соңына (соңғы элементтен кейін орналасқан элементке итератор) қайтарады.

Екінші форма **pred** предикаты арқылы функция немесе функционалдық объект түрінде берілген шартты қанағаттандыратын көршілес элементтерді табады.

Мысал (программа бүтін сан түрінде берілген жиымтың ең сол жағында орналасқан бірдей элементтер жұбын және құрылымдағы өрістер қосындысы тең болатын элементтер жұбын табады):

```
#include <algorithm>
using namespace std;
struct A{ int x, y;};
bool f(A &a1, A& a2){
   return a1.x + a1.y == a2.x + a2.y;}
int main (){
   int m[8] = {45, 60, 60, 25, 25, 2, 13, 35};
   cout << *(adjacent_find(m, m + 8));
   // Нәтижесі: 60
   A ma[5] = {{2,4}, {3,1}, {2,2}, {1,2}, {1,2}};
   cout << (*adjacent_find(ma, ma + 5, f)).x << endl;
   // Нәтижесі: 3
   return 0;
}
```

count, count_if

count алгоритмі белгілі бір мәннің тізбекке ену нұсқаларының санын есептеуді орындайды:

```
template<class In, class T>
   typename iterator_traits<In>::difference_type
      count(In first, In last, const T& value);
```

Алгоритмнің бұл формасы берілген **value** мәнінің тізбекті контейнердегі енгізілулер санын анықтайды. Нәтиженің типі ретінде екі итератор арасындағы айырма – **difference_type** (358 б. қараңыз) типі қабылданады.

count_if алгоритмі тізбекте шарттың орындалу санын есептейді:

```
template<class In, class Pred>
   typename iterator_traits<In>::difference_type
      count_if(In first, In last, Pred pred);
```

Алгоритмнің бұл формасы тізбекті контейнерде функция немесе функционалдық объект түрінде **pred** предикаты арқылы қойылған шартты қанағаттандыратын элементтер санын анықтайды.

count_if функциясының қолданылу мысалы 375 б. көрсетілген.

equal

equal алгоритмі екі тізбек элементтерін жұбтары бойынша салыстыруды орындайды. Қолданушы нені теңдік деп санауға болатынын анықтайтын предикатты тағайындай алады:

```
template <class In1, class In2>
   bool equal (In1 first1, In1 last1, In2 first2);
template <class In1, class In2, class BinPred>
   bool equal(In1 first1, In1 last1,
      In2 first2, BinPred pred);
```

find, find_if

find алгоритмдер топтамасы тізбекте элемент іздеуді жүзеге асырады.
find алгоритмі берілген **value** мәнін іздеуді орындайды:

```
template<class In, class T>
   In find(In first, In last, const T& value);
```

find_if алгоритмі берілген **pred** предикатына сәйкес мәнді іздеуді жүзеге асырады:

```
template<class In, class Pred>
   In find_if(In first, In last, Pred pred);
```

Бұл алгоритмдер іздеу сәтті өткен жағдайда итераторды ең сол жақтан табылған мәнге, ал кері жағдайда тізбектің соңына қайтарады.

Мысал ретінде элементтері файлдан оқылатын вектордағы мәндерді табуды қарастырайық (предикат ретінде функционалдық объект қолданылады):

```
#include <fstream>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
class In_10_50{
   public:
      bool operator()(int x) {return x > 10 && x < 50;}
};
int main(){
   ifstream in ("inpnum");
   vector<int> v;
   int x;
   while ( in >> x, !in.eof()){
      v.push_back(x);}
   for (int i = 0; i<v.size(); i++) cout << v[i] << " ";
```

```
    cout << endl;
    // 51-ге тең элементті іздеу:
    cout << *find(v.begin(), v.end(), 51) << endl;
    // 10<x<50 шартын қанағаттандыратын элементті іздеу:
    cout << *find_if(v.begin(), v.end(), In_10_50())
         << endl;
    return 0;
}
```

Программа жұмысының нәтижесі:

```
56 34 54 0 76 23 51 11 51 11 76 88
51
34
```

find_first_of

find_first_of алгоритмі екінші тізбек элементінің бірінші тізбекке алғаш енгізілуін іздеп табады:

```
templ1ate<class For1, class For2>
   For1 find_first_of(For1 first1, For1 last1,
       For2 first2, For2 last2);
template<class For1, class For2, class BinPred>
   For1 find_first_of(For1 first1, For1 last1,
       For2 first2, For2 last2, BinPred pred);
```

Тізбектердің шекаралары итераторлар көмегімен беріледі. Алгоритмнің бірінші формасы кез келген элементтің енгізілуін іздейді, ал екінші формасы бірінші және екінші тізбектердің сәйкес элементтерін талдайтын бинарлы предикат орындалатын объектінің енгізілуін іздейді. Іздеу сәтсіз орындалған жағдайда **last1** мәні қайтарылады.

find_end

find_end алгоритмі бірінші тізбекке екінші тізбектің алғашқы енгізілуін табады (предикатты талдай отырып немесе талдаусыз) және итераторды соңғы сәйкес келетін элементке қайтарады:

```
template<class For1, class For2>
   For1 find_end(For1 first1, For1 last1,
       For2 first2, For2 last2);
template<class For1, class For2, class BinPred>
   For1 find_end(For1 first1, For1 last1,
       For2 first2, For2 last2, BinPred pred);
```

Іздеу сәтсіз орындалған жағдайда **last1** мәні қайтарылады.

for_each

Алгоритм тізбектің әрбір элементі үшін тағайындалған функцияны қайтарады:

```
template<class In, class Function>
   Function for_each(In first, In last, Function f);
```

Алгоритмнің қолданылу мысалы 371 б. терілген.

mismatch

mismatch алгоритмі екі тізбектің өзара сәйкес тең емес алғашқы элементтер жұбын іздейді және итераторларды осы жұпқа қайтарады:

```
template<class In1, class In2>
   pair<In1, In2> mismatch(In1 first1, In1 last1, In2 first2);
template <class In1, class In2, class BinPred>
   pair<In1, In2> mismatch(In1 first1, In1 last1,
        In2 first2, BinPred pred);
```

Екінші тізбектің ұзындығы бірінші тізбектің ұзындығынан артық немесе оған тең болып есептеледі. Қолданушы нені сәйкессіздік ретінде есептеу керектігін анықтайтын предикатты тағайындай алады.

search, search_n

search алгоритмі бірінші тізбекке екінші тізбектің алғашқы енгізілуін табады (предикатты талдай отырып немесе талдаусыз) және сәйкес келетін (тең болатын) алғашқы элементке итераторды қайтарады:

```
template<class For1, class For2>
   For1 search(For1 first1, For1 last1,
       For2 first2, For2 last2);
template<class For1, class For2, class BinPred>
   For1 search(For1 first1, For1 last1,
       For2 first2, For2 last2, BinPred pred);
```

Іздеу сәтсіз орындалған жағдайда **last1 мәні** қайтарылады.

search_n алгоритмі тізбектің құрамындағы **value** мәнінің кем дегенде **n** санынан тұратын ішкі тізбекті табады (предикатты талдай отырып немесе талдаусыз) және итераторды соған сәйкес келетін алғашқы элементке қайтарады:

```
template<class For, class Size, class T>
   For search_n(For first, For last,
         Size count, const T& value);
template <class For, class Size, class T, class BinPred>
   For1 search_n(For first, For last,
         Size count, const T& value, BinPred pred);
```

Тізбектермен орындалатын модификациялаушы операциялар

Бұл категорияның алгоритмдері өздері жұмыс істейтін тізбекті қандай да бір жолмен өзгертеді. Олар тізбек элементтерін көшіру, өшіру, алмастыру және олардың орналасу реттілігін өзгерту үшін қолданылады.

14.2-кесте. Тізбектермен орындалатын модификациялаушы операцилар

Алгоритм	**Орындалатын функция**
copy	Тізбекті алғашқы элементінен бастап көшіру
copy_backward	Тізбекті соңғы элементінен бастап көшіру
fill	Барлық элементтерді берілген мәнге алмастыру
fill_n	Алғашқы **n** элементті берілген мәнге алмастыру
generate	Барлық элементтерді операция нәтижесіне алмастыру
generate_n	Алғашқы **n** элементті операция нәтижесіне алмастыру
iter_swap	Итераторлармен берілген екі элементтің орнын өзара алмастыру
random_shuffle	Кездейсоқ бірқалыпты үлестірімге сәйкес элементтердің орындарын ауыстыру
remove	Мәні берілген элементтердің орындарын ауыстыру
remove_copy	Мәні берілген элементтердің орындарын ауыстыра отырып, тізбектің көшірмесін алу
remove_copy_if	Предикат орындалған кезде элементтердің орындарын ауыстыра отырып, тізбекті көшіру
remove_if	Предикат орындалған кезде элементтердің орындарын ауыстыру
replace	Мәндері берілген элементтерді алмастыру
replace_copy	Мәндері берілген элементтерді алмастыра отырып, тізбекті көшіру
replace_copy_if	Предикат орындалған кезде элементтерді алмастыра отырып, тізбекті көшіру
replace_if	Предикат орындалған кезде элементтерді алмастыру
reverse	Элементтердің орналастыру ретін керіге өзгерту
reverse_copy	Тізбекті кері ретпен көшіру
rotate	Тізбек элементтерінің орнын циклдік түрде ауыстыру
rotate_copy	Циклдік түрде элементтердің көшірмесін алу
swap	Екі элементтің орындарын өзара алмастыру
swap_ranges	Екі тізбек элементтерінің орындарын өзара алмастыру

Алгоритм	Орындалатын функция
transform	Берілген операцияны тізбектің әрбір элементімен орындау
unique	Көршілес тең элементтерді өшіру
unique_copy	Көршілес тең элементтерді өшіре отырып, тізбектің көшірмесін алу

Осы алгоритмдерді толығырақ қарастырайық.

copy, copy_backward

copy алгоритмі шекаралары **first** және **last** итераторлары арқылы берілетін тізбекті алғашқы элементінен бастап шығыс тізбекке көшіреді, шығыс тізбек үшін **result** басының итераторы тағайындалады:

```
template<class In, class Out>
   Out copy(In first, In last, Out result);
```

copy_backward алгоритмі берілген тізбектің соңғы элементінен бастап көшіруді орындайды. Үшінші параметр қабылдаушы тізбектің соңғы элементінен кейін орналасатын элементті көрсету керек, өйткені оның мәні әрбір элементті көшіру операциясынан бір қадам бұрын кемиді:

```
template<class Bi1, class Bi2>
   Bi2 copy_backward(Bi1 first, Bi1 last, Bi2 result);
```

Тізбектер бір-бірімен қиылысуы мүмкін. Көшіру кезінде шығыс тізбектің шекараларынан шығып кетпеуін қадағалап отыру керек.

Тізбектің көшірмесін алудың мысалы:

```
#include <iostream>
#include <algorithm>
using namespace std;
int main(){
   int b[4], a[5] = {1, 2, 3, 4, 5}, i;
   copy (a + 1, a + 5, b);
   for (i = 0; i < 4; i++) cout << b[i];  // 2 3 4 5
   cout << endl;
   copy (a + 1, a+5, a);
   for (i = 0; i < 5; i++) cout << a[i];  // 2 3 4 5 5
   cout << endl;
   copy_backward (b, b + 3, b + 4);
   for (i = 0; i < 4; i++) cout << b[i];  // 2 2 3 4
   cout << endl;
   return 0;
}
```

copy алгоритмін тізбекті енгізу және шығару үшін де қолдануға болады. Ол үшін үшінші параметр ретінде ағымдық итератор беріледі (361 б.):

```
#include <iostream>
#include <algorithm>
using namespace std;
int main(){
   const int N = 5;
   int a[N] = {1, 2, 3, 4, 5};
   copy (a, a + N, ostream_iterator<int>(cout, " "));
   cout << endl;
   return 0;
}
```

fill, fill_n

fill алгоритмі **first** және **last** итераторларының көмегімен анықталған тізбектің барлық элементтерін берілген **value** мәніне алмастырады. **fill_n** алгоритмі **n** элементті берілген мәнге алмастыруды жүзеге асырады:

```
template<class For, class T>
   void fill(For first, For last, const T& value);
template<class Out, class Size, class T>
   void fill_n(Out first, Size n, const T& value);
```

Бүтін сандардан тұратын жиымды толтыру мысалын қарастырайық:

```
#include <iostream>
#include <algorithm>
using namespace std;
int main(){
   int a[5], i;
   fill (a, a + 5, 1);
   for (i = 0; i < 5; i++) cout << a[i]; // 1 1 1 1 1
   cout << endl;
   fill_n (a + 2, 2, 0);
   for (i = 0; i < 5; i++) cout << a[i]; // 1 1 0 0 1
   cout << endl; return 0;
}
```

Мұнда біздер тізімдер үшін **fill (a, a + 5, 1);** сияқты өрнекті қолдана алмайтынымызға назар аударыңыз, өйткені тізім итераторлары үшін қосу операциясы анықталмаған. Егер толтырылатын тізбек соңына тағайындалатын итератор белгісіз болса, алгоритмнің екінші формасын қолдануға болады.

generate, generate_n

generate алгоритмі барлық элементтерді операция нәтижесіне алмастыруды орындайды. Осы арқылы контейнерді бірдей мәндермен емес, үшінші параметр арқылы берілген функция арқылы немесе **gen** функционалдық объект көмегімен есептелген мәндермен толтыруға мүмкіндік аламыз.

```
template<class For, class Generator>
   void generate(For first, For last, Generator gen);
template<class Out, class Size, class Generator>
   void generate_n(Out first, Size n, Generator gen);
```

Қарапайым мысал:

```
#include <iostream>
#include <algorithm>
using namespace std;
int f(){
   static int i = 1;
   return (++i) * 3;
}
int main(){
   int a[5], i;
   generate(a, a + 5, f);
   for (i = 0; i<5; i++)
      cout << a[i] << " ";    // 6 9 12 15 18
   return 0;
}
```

iter_swap, swap, swap_ranges

iter_swap алгоритмі итераторлармен берілген екі элементтің орындарын өзара алмастырады:

```
template<class For1, class For2>
   void iter_swap(For1 a, For2 b);
```

swap алгоритмі екі элементтің орындарын өзара алмастырады:

```
template<class T> void swap(T& a, T& b);
```

Келесі **swap_ranges** алгоритмі берілген екі диапазондағы элементтердің орындарын өзара алмастырады (екінші диапазонның басталатын орны ғана көрсетілген):

```
template<class For1, class For2>
   For2 swap_ranges(For1 first1, For1 last1, For2 first2);
```

random_shuffle

random_shuffle алгоритмі кездейсоқ бірқалыпты үлестірімге сәйкес элементтердің орындарын алмастыруды жүзеге асырады. Алгоритмнің үшінші параметрі ретінде кездейсоқ сандардың генераторын беруге болады. Осылайша программаны әрбір іске қосқан кезде әртүрлі нәтижелер алу мүмкіндігіне ие боламыз. Генератор **int** типті **n** аргументін қабылдайтын, 0 мен **n** аралығындағы бүтін санды қайтаратын функция немесе функционалдық объект болуы мүмкін.

```
template<class Ran>
   void random_shuffle(Ran first, Ran last);
template<class Ran, class RandomNumberGenerator>
   void random_shuffle(Ran first, Ran last,
      RandomNumberGenerator& rand);
```

Генератор енгізілген мысалда **<time.h>** тақырыптық файлында жарияланған **rand** функциясы қолданылған:

```
#include <iostream>
#include <algorithm>
#include <time.h>
using namespace std;
struct random_gen{
   random_gen(){srand((unsigned int)time(NULL)); }
   int operator()(int n){return rand() % n;}
};
int main(){
   int a[5] = {1, 2, 3, 4, 5}, i;
   random_shuffle(a, a + 5, random_gen());
   for (i = 0; i < 5; i++)
      cout << a[i] << " ";      // 5 3 4 1 2
   cout << endl;
   return 0;
}
```

remove, remove_if, remove_copy, remove_copy_if

remove тектес алгоритмдер тізбектің элементтерін берілген **value** мәні бойынша немесе **pred** предикаты бойынша тізбектің соңына жылжытады. Бұл кезде тізбектің қалған элементтері өзара орналасу реттіліктерін сақтай отырып, тізбектің бас жақ бөлігіне қарай ығыстырылады. Алгоритм олардың орналасу шекарасын қайтарады. Шекарадан кейін орналасқан элементтер өшірілмейді, тізбектің өлшемі өзгертілмейді. **copy** сөзі бар алгоритмнің формалары өңдеудің алдында тізбекті **Out** итераторы арқылы берілген орынға көшіреді де, тізбектің көшірмесін өңдейді.

```
template<class For, class T>
   For remove(For first, For last, const T& value);
template<class For, class Pred>
   For remove_if(For first, For last, Pred pred);
template<class In, class Out, class T>
   Out remove_copy(In first, In last,
      Out result, const T& value);
template<class In, class Out, class Pred>
   Out remove_copy_if(In first, In last,
      Out result, Pred pred);
```

Мәндері екіге тең вектор элементтерін өшіру үшін **remove** алгоритмін қолдану мысалы:

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int main(){
   vector<int> a;
   int i;
   for (i = 0; i < 5; i++) a.push_back(i);
   for (i = 0; i < 5; i++) a.push_back(i);
   for (i = 0; i < a.size(); i++) cout << a[i];
   cout << endl;
   vector<int>::iterator k,
               p = remove(a.begin(), a.end(), 2);
   for (i = 0; i < a.size(); i++) cout << a[i];
   cout << endl;
   for (k = a.begin(); k != p; k++) cout << *k;
   return 0;
}
```

Программа жұмысының нәтижесі:

```
0123401234
0134013434
01340134
```

Мәндері 10 және 50 аралығында жататын вектор элементтерін өшіру үшін **remove_if** алгоритмін **erase** әдісімен қатар қолдану мысалы:

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
bool In_10_50 (int x) {return x > 10 && x < 50;}
int main(){
   vector<int> a;
   int i;
   for (i = 1; i<10; i++) a.push_back(i*10);
   for (i = 0; i<a.size(); i++) cout << a[i] << " ";
   cout << endl;
   vector<int>::iterator new_end = remove_if(a.begin(),
               a.end(), In_10_50);
   a.erase(new_end, a.end());
   for (i = 0; i<a.size(); i++) cout << a[i] << " ";
   cout << endl;
   return 0;
}
```

Программаның орындалу нәтижесі:

```
10 20 30 40 50 60 70 80 90
10 50 60 70 80 90
```

replace, replace_if, replace_copy, replace_copy_if

replace тектес алгоритмдер берілген мәні бар элементтерді немесе предикатқа сәйкес элементтерді алмастыруды орындайды. Алгоритмнің **copy** сөзі бар формалары өңдеу алдында тізбекті **Out** итераторы арқылы берілген орынға көшіреді де, тізбектің көшірмесін өңдейді.

```
template<class For, class T>
   void replace(For first, For last,
      const T& old_value, const T& new_value);
template<class For, class Pred, class T>
   void replace_if(For first, For last,
      Pred pred, const T& new_value);
template<class In, class Out, class T>
   Out replace_copy(In first, In last, Out result,
      const T& old_value, const T& new_value);
template<class Iterator, class Out, class Pred, class T>
   Out replace_copy_if(Iterator first, Iterator last,
      Out result, Pred pred, const T& new_value);
```

Мысал қарастырайық, мұнда **a** векторының мәндері 10 мен 50 аралығында жататын элементтерін 33 мәніне алмастыра отырып, оны жаңа **b** векторына көшіру орындалады (кірістіру итераторы қолданылады):

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
bool In_10_50(int x){
   return x >10 && x < 50;
}
int main(){
   vector<int> a, v;
   vector<int>::iterator i;
   for (int k = 1; k < 10; k++)a.push_back(k * 10);
   for (i = a.begin(); i != a.end(); i++)
      cout << *i << " ";
   cout << endl;
   replace_copy_if(a.begin(), a.end(),
   inserter(v, v.begin()), In_10_50, 33);
   for (i = v.begin(); i != v.end(); i++)
      cout << *i << " ";
```

```
    cout << endl;
    return 0;
}
```

Программаның жұмыс істеу нәтижесі (бірінші жолда бастапқы вектор, екінші жолда нәтижелік вектор):

```
10 20 30 40 50 60 70 80 90
10 33 33 33 50 60 70 80 90
```

reverse, reverse_copy

reverse алгоритмі тізбек элементтерінің орналасу реттілігін кері бағытқа өзгертеді, ал **reverse_copy** бастапқы тізбекті кері бағытта орналасқан нәтижелік тізбекке көшіруді жүзеге асырады:

```
template<class Bi>
   void reverse(Bi first, Bi last);
template<class Bi, class Out>
   Out reverse_copy(Bi first, Bi last, Out result);
```

rotate, rotate_copy

rotate алгоритмі тізбек элементтерінің орындарын циклдік түрде біртіндеп ығыстыруды жүзеге асырады, ал **rotate_copy** тізбекті циклдік түрде көшіреді:

```
template<class For>
   void rotate(For first, For middle, For last);
template<class For, class Out>
   Out rotate_copy(For first, For middle,
      For last, Out result);
```

Біртіндеп орын ауыстыру, яғни ығыстыру екінші параметр арқылы нұсқалған элемент тізбектегі алғашқы орынға орналасқанға дейін жүзеге асырылады. Бірінші және үшінші параметрлер өңделіп жатқан тізбектің басы мен соңын көрсетеді. Мысалы, int **a[5] = {1,2,3,4,5}** бүтін сандық жиымы үшін **rotate(a, a+2, a+5)** функциясын шақыру элементтердің **34512** ретімен орналасуына алып келеді. Егер осыдан кейін мынадай **rotate(a, a+3, a+5)** функциясын шақырсақ, жиым бастапқы қалпына қайтып оралады.

transform

transform алгоритмі берілген операцияны тізбектің әрбір элементі үшін орындайды. Алгоритмнің бірінші формасы **op** функциясы немесе функционалдық объектісі арқылы берілген унарлы операцияны орындайды да, нәтижені **result** итераторының көмегімен берілген орынға орналастырады:

```
template<class In, class Out, class Op>
   Out transform(In first, In last, Out result, Op op);
```

Алгоритмнің екінші формасы екі тізбектің өзара сәйкес орындарындағы элементтер жұбымен бинарлы операция орындайды да, нәтижені **result** итераторы арқылы берілген орынға орналастырады:

```
template<class In1, class In2, class Out,
   class BinaryOperation> Out transform(In1 first1,
      In1 last1, In2 first2, Out result,
         BinaryOperation binary_op);
```

Функционалдық объект стандартты немесе қолданушы тағайындаған болуы мүмкін. Қолданушы тағайындаған функционалдық объект **unary_ function** немесе **binary_function** функциясының (363 б. қараңыз) ұрпағы болуы тиіс.

Төменде көрсетілген мысалда **transform** алгоритмін бірінші шақыру кезінде **a** жиымын $a_i = a_i^2 - b_i^2$ формуласы бойынша түрлендіру орындалады, ал оны екінші шақыру кезінде **negate** стандартты функционалдық объектісінің көмегімен **b** жиымы элементтерінің таңбасы өзгертіледі.

```
#include <iostream>
#include <algorithm>
#include <functional>
using namespace std;
struct preobr: binary_function <double, double, double>{
   double operator()(double x, double y) const{
      return x * x - y * y;}
};
int main(){
   const int m = 5;
   double a[m] = {5, 3, 2, 3, 1},
   b[m] = {1, 10, -3, 2, -4};
   transform(a, a + m, b, a, preobr());
   transform(b, b + m, b, negate<double>());
   int i;
   for (i = 0; i<m; i++) cout << a[i] << " ";
   cout << endl;
   for (i = 0; i<m; i++) cout << b[i] << " ";
   cout << endl;
   return 0;
}
```

Программа орындалуының нәтижесі:

```
24 -91 -5 5 -15
-1 -10 3 -2 4
```

unique, unique_copy

unique алгоритмі тізбектен қатар тұрған өзара тең екі элементті немесе **pred** бинарлы предикатының көмегімен тағайындалған критерийді қанағаттандыратын көршілес екі элементті өшіруді жүзеге асырады. Мұнда тізбектің өлшемі өзгертілмейді[1]. Алгоритм итераторды мәліметтердің жаңа логикалық соңына қайтарады.

```
template<class For> For unique(For first, For last);
template<class For, class BinPred>
   For unique(For first, For last, BinPred pred);
```

unique_copy алгоритмі дәл осы әрекеттерді тізбектің көшірмесімен орындайды:

```
template<class In, class Out>
   Out unique_copy(In first, In last, Out result);
template<class In, class Out, class BinPred>
   Out unique_copy(In first, In last,
      Out result, BinPred pred);
```

Сұрыптаумен байланысты алгоритмдер

Бұл категорияның алгоритмдері тізбектерді реттей отырып сұрыптайды, элементтерді іздеуді, тізбектерді біріктіруді, минимум мен максимумды табуды, лексикографиялық салыстыруды, орын ауыстыруларды және т.б. әрекеттерді жүзеге асырады.

14.3-кесте. Сұрыптаумен байланысты алгоритмдер

Алгоритм	Орындалатын функция
binary_search	Берілген мәнді іздеу
equal_range	Берілген мәні бар элементтер тізбегін табу
inplace_merge	Бір диапазондағы сұрыпталған тізбектерді біріктіру
lexicographical_compare	Екі тізбектің ішіндегі лексикографиялық түрдегі алғашқысы
lower_bound	Берілген мәннің алғашқы енгізілуін анықтау
max	Екі мәннің ішіндегі үлкені
max_element	Тізбектегі ең үлкен мән
merge	Сұрыпталған тізбектерді біріктіру
min	Екі мәннің кішісі
min_element	Тізбектегі ең кіші мән
next_permutation	Лексикографиялық тәртіппен келесі орналастыру

[1]Тізімдер үшін **list** әдісін қолданған жөн.

Алгоритм	Орындалатын функция
`nth_element`	**n**-ші элементті берілген орынға орналастыру
`partial_sort`	Ішінара бөлігін сұрыптау
`partial_sort_copy`	Көшіре отырып, ішінара бөлігін сұрыптау
`partition`	Шартты қанағаттандыратын элементтерді алға жылжыту
`prev_permutation`	Лексикографиялық тәртіппен алдыңғы орналастыру
`sort`	Сұрыптау
`stable_partition`	Шартты қанағаттандыратын элементтердің өзара орналасу реттілігін сақтай отырып, алға қарай жылжыту
`stable_sort`	Бірдей элементтер үшін олардың реттілігін сақтайтын сұрыптау
`upper_bound`	Берілген мәннен үлкен алғашқы элементті табу

Осы алгоритмдерді толығырақ қарастырайық. Алгоритмдердің әрқайсысы үшін екі форма бар: біреуі **<** операциясын, ал екіншісі қолданушы тағайындаған салыстыру функциясын қолданады. Алгоритмдердің басым бөлігіне кездейсоқ қол жеткізу итераторлары қажет екендігіне назар аударыңыз.

binary_search

`binary_search` алгоритмі **`first`** және **`last`** итераторлары арқылы берілген сұрыпталған тізбектегі **`value`** мәнін іздеуді орындайды. Ізделген мәннің табылғаны немес табылмағаны туралы ақпарат қана беріледі. Іздеудің екілік деп аталатын себебі – интервалды тізбекті түрде ортасынан екіге бөлу жолымен орындалады («шөлдегі арыстанды ұстау сияқты»).

```
template<class For, class T>
   bool binary_search(For first, For last, const T& value);
template<class For, class T, class Compare>
   bool binary_search(For first, For last,
      const T& value, Compare comp);
```

equal_range

`equal_range` алгоритмі элементтер тізбегінің шекараларын табуды жүзеге асырады, тізбектің кез келген жеріне берілген мәнді реттілікті бұлдірмей кірістіруге болады. Тізбек сұрыпталған болу керек. Функционалдық объектіні беру кезінде алгоритм **`k`** итераторының әрбір мәні үшін **`comp(*k, value) == false && comp(value, *k) == false`** шарты орындалатын аралықтың шекараларын табады.

```
template<class For, class T>
   pair<For, For> equal_range(For first, For last,
        const T& value);
template<class For, class T, class Compare>
   pair<For, For> equal_range(For first, For last,
        const T& value, Compare comp);
```

Мысалы, 2 4 5 5 7 9 12 18 тізбегі үшін **equal_range** алгоритмін **value =** 8 мәнімен шақыру нәтижесінде 9 және 9 элементтеріне нұсқайтын итераторлар жұбын, ал **value =** 5 мәнімен шақыру 5-ке тең элементтердің алғашқысын және 7 мәнін қайтарады.

inplace_merge

inplace_merge алгоритмі бір тізбектің сұрыпталған екі бөлігін біріктіруді жүзеге асырады. Бірінші бөліктің шеаралары алғашқы екі параметр көмегімен, ал екінші бөліктің басы үшінші параметр арқылы беріледі.

```
template<class Bi>
   void inplace_merge(Bi first, Bi middle, Bi last);
template<class Bi, class Compare>
   void inplace_merge(Bi first, Bi middle, Bi last,
                      Compare comp);
```

lexicographical_compare

lexicographical_compare алгоритмі **<** операциясын қолдану арқылы немесе **comp** функциясының көмегімен екі тізбекті әрбір элемент бойынша салыстыруды орындайды. Егер бірінші тізбек лексикографиялық тұрғыдан алғанда, екіншісінен кіші болса (яғни, бірінші тізбектің кезекті элементі екінші тізбектің соған сәйкес элементінен кіші болып шықты), **true** мәні, кері жағдайда **false** мәні қайтарылады. Егер тізбектердің ұзындықтары бір-біріне сәйкес келмесе, жеткіліксіз элементтер басқа тізбектің сәйкес элементтерінен кіші болып саналады.

```
template<class In1, class In2>
   bool lexicographical_compare
      (In1 first1, In1 last1, In2 first2, In2 last2);
template<class In1, class In2, class Compare>
   bool lexicographical_compare
      (In1 first1, In1 last1, In2 first2, In2 last2,
         Compare comp);
```

Мысалы:

```
#include <iostream>
#include <algorithm>
#include <functional>
```

```
using namespace std;
int main(){
   const int m = 5;
   double a[m] = {5, 3, 2, 3, 1},
   b[m] = {5, 3, 2, 3, 2},
   c[m] = {5, 3, 1, 3, 10};
   cout << lexicographical_compare(a,a+m,b,b+m); // 1
   cout << lexicographical_compare(a,a+m,c,c+m); // 0
   cout << lexicographical_compare(a,a+m,b,b+m,
                                   greater<int>()); // 0
   return 0;
}
```

lower_bound, upper_bound

lower_bound алгоритмі реттілікті бұзбай берілген мәнді алдына кірістіруге болатын сұрыпталған тізбектің элементтерінің алғашқысына, ал **upper_bound** соңғысына деген итераторды табады.

```
template<class For, class T>
   For lower_bound(For first, For last, const T& value);
template<class For, class T, class Compare>
   For lower_bound(For first, For last, const T& value,
      Compare comp);
template<class For, class T>
   For upper_bound(For first, For last, const T& value);
template<class For, class T, class Compare>
   For upper_bound(For first, For last,
      const T& value, Compare comp);
```

max, min

Салыстырудың өзіндік критериін немесе **<** операциясын қолдану арқылы **max** алгоритмі екі мәннің үлкенін, ал **min** – кішісін іздейді.

```
template<class T> const T& min(const T& a, const T& b);
template<class T, class Compare>
   const T& min(const T& a, const T& b, Compare comp);
template<class T> const T& max(const T& a, const T& b);
template<class T, class Compare>
   const T& max(const T& a, const T& b, Compare comp);
```

max_element, min_element

max_element алгоритмі итераторды тізбектің ең үлкен мәніне, ал **min_element** алгоритмі оны ең кіші мәніне қайтарады.

```
template<class For>
   For min_element(For first, For last);
template<class For, class Compare>
   For min_element(For first, For last, Compare comp);
template<class For>
   For max_element(For first, For last);
template<class For, class Compare>
   For max_element(For first, For last, Compare comp);
```

merge

merge алгоритмі сұрыпталған тізбектерді біріктіруді жүзеге асырады.

```
template<class In1, class In2, class Out>
   Out merge(In1 first1, In1 last1, In2 first2,
         In2 last2, Out result);
template<class In1, class In2, class Out, class Compare>
   Out merge(In1 first1, In1 last1, In2 first2,
         In2 last2, Out result, Compare comp);
```

Тізімдерді біріктірудің осы аттас әдісіне қарағанда, бастапқы тізбектердегі элементтер өшірілмейді. Кілттері тең болған жағдайда, бірінші тізбектің элементтері екінші тізбек элементтерінен бұрын орналасады (бұл жағдай, кілтпен қатар, ақпараттық бөлігі болатын мәліметтер құрылымдары үшін маңызды болып саналады).

Мысал:

```
#include <iostream>
#include <algorithm>
using namespace std;
int main(){
   const int m = 5;
   double a[m] = {3, 4, 8, 17, 20},
          b[m] = {5, 6, 8, 10, 35}, c[m * 2];
   int i;
   merge(a, a + m, b, b + m, c);
   for (i = 0; i < m * 2; i++)
      cout << c[i] << " ";    // 3 4 5 6 8 8 10 17 20 35
   cout << endl;
   return 0;
}
```

next_permutation, prev_permutation

Кез келген тізбектің элементтерін әртүрлі тәсілдермен орналастыруға болады. Ұзындығы **n** болып келетін тізбек үшін мұндай қайта орналастыру-

лар саны **n! (1*2*...*n)** болады. **next_permutation** алгоритмі кезекті қайта орналастыруды лексикографиялық тәртіппен, ал **prev_permutation** алгоритмі бұрынғы орналастыруды жүзеге асырады. Егер келесі орналастыру бар болатын болса, алгоритмдер **true** бульдік мәнін, кері жағдайда **false** мәнін қайтарады.

```
template<class Bi>
   bool next_permutation(Bi first, Bi last);
template<class Bi, class Compare>
   bool next_permutation(Bi first, Bi last, Compare comp);
template<class Bi>
   bool prev_permutation(Bi first, Bi last);
template<class Bi, class Compare>
   bool prev_permutation(Bi first, Bi last, Compare comp);
```

Мысал:

```
#include <iostream>
#include <algorithm>
using namespace std;
int main(){
   const int m = 3;
   int a[m]={1, 4, 2}, b[m];
   int i;
   copy(a, a + m, b);
   cout << " next_permutation(a, a + m):" << endl;
   while (next_permutation(a, a + m)){
      for (i = 0; i < m; i++) cout << a[i] << " ";
      cout << endl;}
   cout << " prev_permutation(b, b + m):" << endl;
   while (prev_permutation(b, b + m)){
      for (i = 0; i < m; i++) cout << b[i] << " ";
      cout << endl;}
   return 0;
}
```

Программа жұмысының нәтижесі:

```
next_permutation(a, a + m):
2 1 4
2 4 1
4 1 2
4 2 1
prev_permutation(b, b + m):
1 2 4
```

nth_element

Алгоритм жиымды ішінара сұрыптау әрекетін орындайды. Алгоритмнің орындалуынан кейін **nth** итераторы арқылы берілген элементтің мәні толық сұрыптаудан кейінгімен бірдей болады, яғни осы позициядан солға қарай орналасқан элементтер одан кіші, оңға қарай орналасқандары одан үлкен болады.

```
template<class Ran>
   void nth_element(Ran first, Ran nth, Ran last);
template<class Ran, class Compare>
   void nth_element(Ran first, Ran nth, Ran last,
      Compare comp);
```

partial_sort, partial_sort_copy

partial_sort алгоритмі де тізбекті ішінара сұрыптау әрекетін орындайды. Алгоритм орындалған соң **first** және **middle** аралығындағы элементтер толық сұрыптаудан кейінгідей тәртіппен орналасады.

partial_sort_copy алгоритмі тізбектің көшірмесімен дәл осындай әрекеттерді орындайды.

```
template<class Ran>
   void partial_sort(Ran first, Ran middle, Ran last);
template<class Ran, class Compare>
   void partial_sort(Ran first, Ran middle, Ran last,
      Compare comp);
template<class In, class Ran>
   Ran partial_sort_copy(In first, In last,
      Ran result_first, Ran result_last);
template<class In, class Ran, class Compare>
   Ran partial_sort_copy(In first, In last,
      Ran result_first, Ran result_last, Compare comp);
```

Ішінара сұрыптау біз үшін бірнеше ең кіші немесе ең үлкен мәндер ғана маңызды болған жағдайларда, уақытты үнемдейді.

```
#include <iostream>
#include <algorithm>
using namespace std;
int main(){
   const int m = 8;
   int a[m] = {3, 1, 2, 34, 8, 7, 20, 2};
   int i;
   partial_sort(a, a + 5, a + m, greater<int>());
   for (i = 0; i < m; i++) cout << a[i] << " ";
```

```
    cout << endl; // 34 20 8 7 3 1 2 2
    return 0;
}
```

partition, stable_partition

partition алгоритмі берілген шартты қанағаттандыратын элементтерді басқаларынан бұрын орналастырады. **stable_partition** алгоритмі де дәл осы әрекетті орындайды, бірақ ол элементтердің салыстырмалы реттілігін сақтайды. Шарт функцияның немесе функционалдық объектінің көмегімен беріледі.

```
template<class Bi, class Pred>
   Bi partition(Bi first, Bi last, Pred pred);
template<class Bi, class Pred>
   Bi stable_partition(Bi first, Bi last, Pred pred);
```

sort, stable_sort

sort алгоритмі тізбек үшін тиімді сұрыптауды $N \cdot \log_2 N$ шамасына пропорционал уақытта орындайды. Бірдей элементтердің орналасу реттілігін сақтау үшін **stable_sort** алгоритмін қолданған жөн. Оның жұмыс істеу уақыты $N \cdot (\log_2 N)^2$ шамасына пропорционал болады.

```
template<class Ran>
   void sort(Ran first, Ran last);
template<class Ran, class Compare>
   void sort(Ran first, Ran last, Compare comp);
template<class Ran>
   void stable_sort(Ran first, Ran last);
template<class Ran, class Compare>
   void stable_sort(Ran first, Ran last, Compare comp);
```

Бұл алгоритмге кездейсоқ қол жеткізу итераторлары қажет екеніне назар аударыңыз.

Жиындармен және пирамидалармен жұмыс істеу алгоритмдері

Бұл категорияның алгоритмдері жиындарды сұрыптауды және пирамидалармен орындалатын операцияларды жүзеге асырады. Бірінші жағдайда сұрыпталған тізбек жиын ретінде қарастырылады, ал бірігу және қиылысу операцияларының мағынасы дәл жиындар теориясындағымен бірдей болады. Келтірілген алгоритмдер кіріс тізбектерін өзгертпейді, ал шығыс тізбектері реттеліп орналасады.

Пирамида[1] деп барлық элементтері үшін келесі шарттар орындалатын тізбектерді айтады:

```
a[i] ≤ a[2*i + 1]
a[i] ≤ a[2*i + 2]
```

10 бүтін саннан тұратын пирамиданың мысалы: **23 20 21 17 19 18 15 12 10 14**.

Анықтамасында айтылып тұрағандай, пирамиданың ең үлкен элементі бірінші болып орналасқан, сондықтан пирамидаларды басымдылықтары бар тізімдерді (338 б. қараңыз) жүзеге асыру үшін қолданған ыңғайлы, себебі мұнда пирамидалық шарттарды сақтай отырып, алғашқы элементті шығарып алудың және жаңа элемент қосудың тиімді алгоритмдері бар.

Пирамидамен жұмыс істеу үшін кездейсоқ қол жеткізу итераторлары қажет.

Алгоритмдердің басқа категорияларындағыдай сияқты, мұнда да әрбір алгоритмнің екі формасы болады: бірі **<** операциясын, екіншісі қолданушы тағайындаған салыстыру фцнкциясын пайдаланады.

14.4-кесте. Жиындармен және пирамидалармен жұмыс істеу алгоритмдері

Алгоритм	**Орындалатын функция**
includes	Бір жиынды екіншісіне енгізу
set_intersection	Жиындардың сұрыпталған қиылысуын құру
set_difference	Екі тізбектің алғашқысына ғана тиісті болатын элементтердің сұрыпталған тізбегін құру
set_symmetric_difference	Екі тізбектің біреуіне ғана тиісті болатын элементтердің сұрыпталған тізбегін құру
set_union	Жиындардың сұрыпталған бірігуін құру
make_heap	Кездейсоқ қол жеткізуге болатын тізбекті пирамидаға түрлендіру
pop_heap	Пирамидадан элементті шығарып алу
push_heap	Пирамидаға элементті қосу
sort_heap	Пирамиданы сұрыптау

Осы алгоритмдерді толығырақ қарастырайық.

includes

includes алгоритмі бір тізбектің екіншісіне құрамына енуін тексереді. Мұнда **[first2, last2)** тізбегінің әрбір элементі **[first1, last1)** тізбегінің құрамына енген жағдайда ғана, нәтиже **true** мәніне тең болады[2].

[1]Пирамиданы көбінесе *үйінді* (ағылшынның heap сөзінің аудармасы) деп те атайды.

[2]Квадрат жақша сәйкес элементтің тізбекке тиісті екенін, ал дөңгелек жақша тиісті емес екенін (одан кейінгі элементті нұсқайды) білдіреді.

```
template<class In1, class In2>
   bool includes(In1 first1, In1 last1,
      In2 first2, In2 last2);
template<class In1, class In2, class Compare>
   bool includes (In1 first1, In1 last1, In2 first2,
      In2 last2, Compare comp);
```

set_intersection

set_intersection алгоритмі жиындардың сұрыпталған қиылысуын (яғни, бірінші жиынға да, екінші жиынға да тиісті болатын элементтерден ғана тұратын жиынды) құрады:

```
template<class In1, class In2, class Out>
   Out set_intersection (In1 first1, In1 last1,
      In2 first2, In2 last2, Out result);
template<class In1, class In2, class Out, class Compare>
   Out set_intersection (In1 first1, In1 last1,
      In2 first2, In2 last2, Out result, Compare comp);
```

set_difference, set_symmetric_difference

set_difference алгоритмі екі тізбектің алғашқысына ғана тиісті болатын элементтерді **Out** тізбегіне көшіруді жүзеге асырады:

```
template<class In1, class In2, class Out>
   Out set_difference(In1 first1, In1 last1,
      In2 first2, In2 last2, Out result);
template<class In1, class In2, class Out, class Compare>
   Out set_difference (In1 first1, In1 last1,
      In2 first2, In2 last2, Out result, Compare comp);
```

set_symmetric_difference алгоритмі екі тізбектің біреуіне ғана тиісті болатын элементтерді **Out** тізбегіне көшіруді жүзеге асырады:

```
template<class In1, class In2, class Out>
   Out set_symmetric_difference(In1 first1, In1 last1,
      In2 first2, In2 last2, Out result);
template<class In1, class In2, class Out, class Compare>
   Out set_symmetric_difference(In1 first1, In1 last1,
      In2 first2, In2 last2, Out result, Compare comp);
```

Нәтижелік тізбек бастапқы тізбектердің ешқайсысының ішкі тізбегі болмауы тиіс.

set_union

set_union алгоритмі жиындардың сұрыпталған бірігуін (яғни, екі жиынның да элементтерін қайталаусыз қамтитын жиынды) құрады:

```
template<class In1, class In2, class Out>
   Out set_union(In1 first1, In1 last1,
      In2 first2, In2 last2, Out result);
template<class In1, class In2, class Out, class Compare>
   Out set_union(In1 first1, In1 last1,
      In2 first2, In2 last2, Out result, Compare comp);
```

Жиындармен жұмыс істеу алгоритмдерін қолдану мысалы

```
#include <iostream>
#include <algorithm>
using namespace std;
void show(const char *s, const int *begin, const int *end){
   cout << s << " ";
   copy(begin, end, ostream_iterator<int>(cout, " "));
   cout << endl;
}
int main(){
   int a[4] = {2, 5, 7, 9}, b[3] = {1, 5, 9},
      Union[7], *p_union,
      isect[4], *p_isect,
      dif[3], *p_dif,
      symdif[7], *p_symdif;
   p_isect = set_intersection(a, a + 4, b, b + 3, isect);
   p_union = set_union(a, a + 4, b, b + 3, Union);
   p_dif = set_difference(a, a + 4, b, b + 3, dif);
   p_symdif = set_symmetric_difference(a, a + 4, b,
                                       b + 3, symdif);
   show("a:     ", a, a + 4);
   show("b:     ", b, b + 3);
   show("isect: ", isect, p_isect);
   show("Union: ", Union, p_union);
   show("dif:   ", dif, p_dif);
   show("symdif:", symdif, p_symdif);
   if (includes(a, a + 4, b, b + 3))
      cout << "a b-ны қамтиды.\n";
   else cout << "a b-ны қамтымайды.\n";
   if (includes(Union, p_union, b, b + 3))
      cout << "Union b-ны қамтиды.\n";
   else cout << "Union b-ны қамтымайды.\n";
   return 0;
}
```

Программаның жұмыс істеу нәтижесі:

```
a:       2 5 7 9
b:       1 5 9
isect:   5 9
Union:   1 2 5 7 9
dif:     2 7
symdif:  1 2 7
a b-ны қамтымайды.
Union b-ны қамтиды.
```

make_heap

make_heap алгоритмі кездейсоқ қол жеткізу мүмкіндігі бар тізбекті пирамидаға түрлендіреді:

```
template<class Ran>
   void make_heap(Ran first, Ran last);
template<class Ran, class Compare>
   void make_heap(Ran first, Ran last, Compare comp);
```

pop_heap

pop_heap алгоритмі тізбектің бірінші элементін өшіреді, содан кейін барып пирамидалылық шартын орындайды:

```
template<class Ran>
   void pop_heap(Ran first, Ran last);
template<class Ran, class Compare>
   void pop_heap(Ran first, Ran last, Compare comp);
```

Пирамидадан шығарылып алынатын элементтің мәнін алуды әдеттегі тәсілмен орындау талап етіледі. Мысалы, жоғарыда көрсетілген **a** жиымы үшін келесідей түрде жазуға болады:

```
x = *a; pop_heap(a, a + m);
```

push_heap

push_heap алгоритмі тізбекке соңғы элементті қосқаннан кейін оны пирамидаға түрлендіруді жүзеге асырады:

```
template<class Ran>
   void push_heap(Ran first, Ran last);
template<class Ran, class Compare>
   void push_heap(Ran first, Ran last, Compare comp);
```

push_heap алгоритмін шақыруға дейін элементті контейнер типіне сәйкес тәсілмен тізбекке қосу керектігіне көңіл аударыңыз, мысалы:

```
v.push_back(x); push_heap(v.begin(), v.end());
```

sort_heap

sort_heap алгоритмі пирамиданы өсуі бойынша сұрыпталған тізбеккке түрлендіреді:

```
template<class Ran>
   void sort_heap(Ran first, Ran last);
template<class Ran, class Compare>
   void sort_heap(Ran first, Ran last, Compare comp);
```

Бұл алгоритм пирамиданың қасиеттерін қолданатын болғандықтан, ол қарапайым сұрыптаудан гөрі жылдам жұмыс істейді. Сұрыптау кілттері бірдей элементтердің салыстырмалы орналасу реттілігін сақтамайды.

15-ТАРАУ

Сандық есептеулерге арналған құралдар

Жалпыланған сандық алгоритмдер

Сандық алгоритмдерді қолдану үшін **<numeric>** тақырыптық файлын қосу қажет.

15-кесте. Жалпыланған сандық алгоритмдер

Алгоритм	Орындалатын функция
accumulate	Жинақтау
inner_product	Скалярлық көбейту
partial_sum	Жинақтай отырып, қосындыны есептеу
adjacent_difference	Іргелес элементтер арасындағы айырманы есептеу

accumulate

accumulate алгоритмінің бірінші формасы **first** және **last** итераторларымен берілген тізбек элементтерінің қосындысын жинақтау үшін қолданылады. Қосындының бастапқы мәні (әдетте бұл 0) үшінші параметр арқылы беріледі. Бұл параметрдің типі нәтиженің типін анықтайды (функция есептелген қосындыны қайтарады):

```
template <class In, class T>
   T accumulate(In first, In last, T init);
```

accumulate алгоритмінің екінші формасы үшінші параметрге және тізбектің кезекті элементіне берілген операцияны қолдануға мүмкіндік береді:

```
template t<class In, class T, class BinOp>
   T accumulate(In first, In last, T init, BinOp binary_op);
```

Төменде көрсетілген мысалда жиым элементтерінің қосындысы, көбейтіндісі және элементтер квадраттарының қосындысы есептеледі:

```
#include <iostream>
#include <numeric>
#include <functional>
using namespace std;
int sumkv( int s, int x){ return s + x * x;}
int main(){
   const int m = 5;
   int a[m] = {3, 2, 5, 1, 6}, sum = 0, mul = 1, sum2 = 0;
```

```
    cout << accumulated, a + m, sum) << endl;
    cout << accumulate(a, a + m, mul, multiplies<int>())
         << endl;
    cout << accumulate(a, a + m, sum2, sumkv) << endl;
    return 0;
}
```

inner_product

inner_product алгоритмінің бірінші формасы екі тізбектің скалярлық көбейтіндісін есептеу үшін қолданылады (**a** және **b** тізбектерінің скалярлық көбейтіндісі $\sum a_i * b_i$ өрнегі болып табылады). Көбейтіндінің бастапқы мәні төртінші параметр арқылы беріледі. Осы параметрдің типі нәтиженің типін анықтайды (функция есептелген көбейтіндіні қайтарады):

```
template t<class In1, class In2, class T>
   T inner_product(In1 first1, In1 last1, In2 first2, T init);
```

inner_product алгоритмінің екінші формасы екі функцияның немесе функционалдық объектінің көмегімен берілген әрекеттерді екі тізбекпен орындау үшін қолданылады. Бірінші функционалдық объект қосу операциясының орнына, ал екіншісі көбейтудің орнына пайдаланылады:

```
template t<class In1, class In2, class T,
   class BinOp1, class BinOp2>
      T inner_product(In1 first1, In1 last1, In2 first2,
         T init, BinOp1 binary_op1, BinOp2 binary_op2);
```

Төмендегі шақыру **a** және **b** тізбектерінің сәйкес элементтерінің қосындыларының көбейтіндісін есептейді:

```
cout << inner_product(a, a + m, b, mul,
                      multiplies<int>(), plus<int>());
```

partial_sum

partial_sum алгоритмі элементтердің ішінара қосындыларынан тұратын тізбекті қалыптастырады. Мысалы, 3 1 2 3 5 сандарының тізбегі үшін нәтиже 3 4 6 9 14 болады, яғни нәтиженің әрбір элементі өзінің алдындағы элемент пен бастапқы тізбектегі ағымдағы элементтің қосындысына тең болады.

Қосындының орнына төртінші параметр арқылы басқа операцияны беруге болады. Нәтижені бастапқы тізбекке де орналастыруға болады, ол үшін үшінші параметр ретінде оның басына деген итератор көрсетіледі. Функция нәтижелік тізбектің соңынан кейінгі орналасатын элементке итераторды қайтарады.

```
template t<class In, class Out>
   Out partial_sum(In first, In last, Out result);
```

```
template t<class In, class Out, class BinOp>
   Out partial_sum(In first, In last, Out result,
      BinOp binary_op);
```

adjacent_difference

adjacent_difference алгоритмі іргелес элементтер арасындағы айырманы, яғни $d_i = a_i - a_{i-1}$ есептеуді жүзеге асырады. Айырманың орнына төртінші параметр арқылы басқа операцияны беруге болады. Функция нәтижелік тізбектің соңынан кейінгі орналасатын элементке итераторды қайтарады.

```
template t<class In, class Out>
   Out adjacent_difference(In first,In last,Out result);
template t<class In, class Out, class BinOp>
   Out adjacent_difference(In first, In last,
      Out result, BinOp binary_op);
```

Бұл алгоритм осының алдындағы алгоритмге қарамақарсы болып табылады, яғни бір тізбек үшін алдымен осы алгоритмдердің бірін, содан кейін екіншісін шақыру нәтижесінде бастапқы тізбекке қайтып ораламыз.

valarray класы

Сандар жиымдарымен нәтижелі жұмыс істеу үшін стандартты кітапханада **valarray** шаблондық класы анықталған. Осы класпен орындалатын операциялар олардың жоғары өнімділікті жүйелерде қолдануына негізделіп жүзеге асырылған. Кітапханада, сонымен қатар, **valarray** класының әртүрлі ішкі жиындарын алуға мүмкіндік беретін төрт қосымша класс сипатталған: **slice_array**, **gslice_array**, **mask_array** және **indirect_array**. Осы кластардың барлығы жиым элементінің типін білдіретін параметрі бар кластардың шаблондары ретінде жүзеге асырылған.

valarray класын және онымен байланысты құралдарды қолдану үшін программаға **<valarray>** тақырыптық файлын қосу қажет. Аталған кластар шаблондарымен қатар, мұнда жиым индекстерінің ішкі жиындарын тағайындайтын **slice** және **gslice** кластары, сонымен қатар осы шаблондармен және кластармен жұмыс істеуге арналған функциялардың тақырыптары да сипатталған.

slice_array шаблоны **valarray** элементтерінің жолын, бағанасын немесе басқа қалыпты (регулярлық) ішкі жиынын (мысалы, оның жұп элементтерін) бейнелейді. Осы үлгінің көмегімен **valarray** класын кездейсоқ өлшемді матрица ретінде сипаттауға болады.

gslice_array шаблоны жиымның ішкі жиынын қалыптастырудың айтарлықтай күрделі заңдарын тағайындауға, ал **mask_array** – биттік

масканың көмегімен кездейсоқ ішкі жиындарды тағайындауға мүмкіндік береді. **indirect_array** шаблоны жиым элементтерінен емес, олардың индекстерінен тұрады.

valarray класында бос жиым және бірдей немесе берілген мәндермен толтырылған жиым құруға мүмкіндік беретін **конструкторлар** анықталған:

```
template<class T> class valarray{
public:
   // Нөлдік өлшемді жиым:
   valarray();
   // n элементтен тұратын жиым (әрқайсысына
   // келісім бойынша конструктор қолданылады:
   explicit valarray(size_t n);
   // мәндері v болатын n элементтен тұратын жиым:
   valarray(const T& v, size_t n);
   // m массивынан алынған мәндері бар n элементтен
   // тұратын жиым:
   valarray(const T* m, size_t n);
   // Көшіру конструкторы
   valarray(const valarray&);
   // Қимадан алынған мәндерге ие болатын жиым:
   valarray(const slice_array<T>&);
   // Жалпыланған қимадан алынған мәндерге
   // ие болатын жиым:
   valarray(const gslice_array<T>&);
    // Ішкі жиыннан алынған мәндерге ие болатын жиым:
    valarray(const mask_array<T>&);
    // indirect_array класынан алынған мәндерге
    // ие болатын жиым:
    valarray(const indirect_array<T>&);
    ...
}
```

Жиымдарды құру мысалдары:

```
valarray <double> v1;       // Бос жиым
valarray <int> v2(100);     // 100 элементтен тұратын жиым,
// әр элемент келісім бойынша нөлге теңестірілген
valarray <int> v3(5, 100); // мәні 5-ке тең 100
                           // элементтен тұратын жиым
const int m = {4, 4, 3, 6, 2};
valarray <int> v4(m, 5);   // Келесі 5 элементтен
                           // тұратын жиым: 4,4,3,6,2
```

Жиымдарда **индекс бойынша қол жеткізу** және **меншіктеу** операциялары анықталған:

```
T operator[](size_t) const;
T& operator[](size_t);
// Жиымды көшіру:
valarray<T>& operator=(const valarray<T>&);
// Барлық элементтерге бірдей мәнді меншіктеу:
valarray<T>& operator=(const T&);
// valarray жиымына ішкі жиындарды меншіктеу:
valarray<T>& operator=(const slice_array<T>&);
valarray<T>& operator=(const gslice_array<T>&);
valarray<T>& operator=(const mask_array<T>&);
valarray<T>& operator=(const indirect_array<T>&);
```

valarray жиымының элементтерін нөмірлеу нөлден басталады. Диапазон шекараларынан шығып кету мүмкіндігі тексерілмейді.

Жоғарыда көрсетілгендерден басқа, жалпылама индекстеу немесе **valarray** жиымынан ішкі жиымдарды шығарып алу операциялары анықталған[1]:

```
valarray<T>     operator[](slice) const;
slice_array<T>  operator[](slice);
valarray<T>     operator[](const gslice&) const;
gslice_array<T> operator[](const gslice&);
valarray<T>     operator[](const valarray<bool>&) const;
mask_array<T>   operator[](const valarray<bool>&);
valarray<T>     operator[](const valarray<size_t>&) const;
indirect_array<T> operator[](const valarray<size_t>&);
```

const түйінді сөзі бар операциялардың формалары ішкі жиынды **valarray** типті жаңа жиым түрінде, ал **const** сөзі жоқ формалары сәйкес типтегі жиым түрінде қайтарады.

valarray жиымы үшін **+**, **-**, **~** және **!** унарлы операциялары, **+**, **-**, *****, **/**, **%**, **&**, **|**, **<<**, **>>** бинарлы операциялары, сонымен қатар екінші операнды сан да, жиым да бола алатын сәйкесінше күрделі меншіктеулер (**+=**, ***=** және т.с.с.) анықталған, мысалы:

```
valarray<T>& operator+= (const T&);
valarray<T>& operator+= (const valarray<T>&);
```

Бірінші жағдайда жиымның әрбір элементі берілген санға арттырылады, ал екінші жағдайда екі жиымның сәйкес элементтерін мүшелеп қосу орындалады. Операциялар элементтері үшін қолданылуы мүмкін болатын жиымдарға ғана пайдаланыла алады. Егер операцияға қатысатын жиымдардың өлшемдері сәйкес келмесе, онда нәтиже анықталмаған болып саналады. **<<** және **>>**

[1]Бұл операциялардың әрекеттеріне түсініктеме келесі бөлімде берілген.

операциялары әрбір элементті разряд бойынша жылжытуды (ығыстыруды) білдіреді (яғни элементтердің ығыстырылуы емес және енгізу/шығару да емес).

Мысалдар:

```
v4 += 1;  // Нәтижесі - 5,5,4,7,2 (v4 жоғарыда сипатталған)
v2 += v3; // i = 1, ..., 100 үшін v2[i] = v2[i] + v3[i]
v4 >>= 1; // Нәтижесі - 2,2,2,3,1
```

valarray класында келесідей әдістер анықталған:

```
size_t size() const;  // Жиымның өлшемі
T sum() const;        // Элементтердің қосындысы
T min() const;        // Ең кіші мән
T max() const;        // Ең үлкен мән
// i элементке логикалық ығысу (егер i>0,онда солға):
valarray<T> shift(int i) const;
// i элементке циклдік ығысу (егер i>0,онда солға):
valarray<T> cshift(int) const;
// Жиымның әрбір элементіне func функциясын қолдану:
valarray<T> apply(T func(T)) const;
valarray<T> apply(T func(const T&)) const;
// қайталап инициалдау (барлық элементтер келісім
// бойынша мәнге алмастырылады, вектор өлшемі - sz:
void resize(size t sz, T c = T());
```

shift және **cshift** ығыстыру функциялары және **apply** функциясын қолдану жаңа жиымды қайтарады, мұндайда бастапқы жиым өзгеріссіз қалады. Логикалық ығыстыру кезінде босаған мәндер берілген элементтер типі үшін келісім бойынша тағайындалған мәнмен толтырылады.

resize көмегімен қайта инициалдау бұрыннан бар жиымның өлшемін динамикалық түрде өзгертуге арналмаған. **valarray** сандармен жылдам жұмыс жасау үшін қолданылады деп саналады және оның өлшемі қолданылу барысында тұрақты болып қалады деп есептеледі.

Жиымдарға **abs**, **acos**, **asin**, **atan**, **atan2**, **cos**, **cosh**, **exp**, **log**, **log10**, **pow**, **sin**, **sinh**, **sqrt**, **tan**, **tanh** функцияларын қолдануға болады, мысалы:

```
v2 = cos(v3); // i = 1, ..., 100 үшін v2[i] = cos(v3[i])
```

valarray жиымының элементтері тізбек болып табылатындықтан, оған алдыңғы бөлімдерде қарастырылған барлық стандартты алгоритмдер мен итераторларды қолдануға болады. Алайда жиыммен қималар және ішкі жиындар арқылы жұмыс істеген әлдеқайда тиімді болып табылады, олар келесі бөлімде қарастырылады.

valarray үшін **<**, **>**, **==**, **!=** және т.б. логикалық операциялар анықталған, олар нәтижесінде **bool** типті жиым береді, оның әрбір элементі салысты-

рылатын жиымдардың сәйкес элементтер жұбына қолданылған операция нәтижесіне тең болады, мысалы:

```
template<class T> valarray<bool>
   operator== (const valarray<T>&, const valarray<T>&);
template<class T> valarray<bool>
   operator!= (const valarray<T>&, const valarray<T>&);
```

Егер операцияға қатысатын жиымдардың өлшемдері сәйкес келмесе, нәтиже анықталмаған болып саналады.

Көмекші кластар

slice (қима) класы жиым элементтері индекстерінің ішкі жиынын тағайындауға мүмкіндік береді:

```
class slice{
   public:
      slice();
      slice(size_t start, size_t size, size_t stride);
      size_t start() const; // бірінші элемент индексі
      size_t size() const;  // элементтер саны
      stride() const;       // қадам
};
```

Қадам – қиманың екі элементінің арасындағы қашықтық. Мысалы, **slice(3, 4, 2)** жиымның 3, 5, 7, 9 нөмірлі элементтерінен тұратын қиманы, ал **slice(1, 3, 4)** жиымның 1, 5, 9 нөмірлі элементтерінен тұратын қиманы береді. Осылайша, жолды немес бағананы тағайындау арқылы қималар көмегімен жиымды матрица ретінде бейнелеуге болады.

Енді осының алдындағы бөлімде сипатталған **valarray** жиымының ішкі жиындарын индекстеу операциясына мысал келтіруге болады: **a[slice(1, 5, 3)] = b** операторы **b** жиымын **a** жиымының қима арқылы берілген элементтеріне ғана меншіктеуді орындайды.

Жиым элементтерінің ішкі жиынымен ыңғайлы жұмыс істеу үшін **slice_array** класының шаблоны енгізілген. Бұл кластың тәуелсіз объектілері құрылмайды, оларға бағытталған барлық хабарласулар бастапқы жиымға қайта адрестеледі:

```
slice_array<double>& s1 = v2[slice(0, v.size() / 10, 10)];
// 10 X 10 матрицасының бірінші бағанасын 5-ке арттыру:
s1 += 5;
slice_array<double>& s2 = v2[slice(0, v.size() / 10, 1)];
// 10 X 10 матрицаның 1-жолының әрбір элементін 2 есе арттыру:
s2 *= 2;
```

slice_array шаблонында күрделі меншіктеу операциялары (**+=**, ***=**, **>=** және т.с.с.) анықталған, мысалы:

```
void operator*= (const valarray<T>&) const;
void operator/= (const valarray<T>&) const;
```

Олар **slice_array** сілтеме жасайтын қима арқылы берілген бастапқы жиым элементтеріне және аргументке берілген әрекеттерді қолданады. Сонымен қатар, скалярлық мәнді меншіктеу операциясы анықталған, ол қимаға енетін бастапқы жиымның барлық элементтеріне меншіктеуді орындайды.

Стандартты кітапхана матрицалармен жұмыс істеуге арналған құралдарды бермейді, бірақ қималар арқылы берілетін абстракциялардың көмегімен **valarray** жиымымен кездейсоқ өлшемді матрица тәрізді жұмыс істеуге болады.

Егер бір қима арқылы беруге болмайтын ішкі жиынмен жұмыс істеу керек болса, онда **gslice** – **жалпыланған қима** шаблоны қолданылады:

```
class gslice{
   public:
      gslice();
      gslice(size_t s, const valarray<size_t>& len,
         const valarray<size_t>& d);
      size_t start() const;    // бірінші элемент индексі
      valarray<size_t> size() const; // өлшемдегі
                                     // элементтер саны
      valarray<size_t> stride() const; // индекс[0],
                          // индекс[1],... үшін қадамдар
};
```

Жалпыланған қима бірнеше қиманы береді. Бастапқы элементтің индексі бірінші **s** параметрі арқылы, қималардың өлшемдері **len** жиымы арқылы, ал қадамдар **d** жиымы арқылы беріледі. Берілетін қималардың санын анықтайтын жиым өлшемдері бір-біріне сәйкес келуі тиіс.

Мысалы, егер

```
start = 0
len = {2, 3}
d = {4, 1}
```

параметрлерін берсек, онда

```
k = (0,1) x 4 + (0,1,2) x 1
```

тәрізді жиым индекстерінің тізбегін аламыз, олар жұмыс нәтижесінде компьютер жадында бағаналар бойынша орналасқан 3x4 матрицаның сол жақ жоғарғы бұрышынан алынған 2x3 матрицаны береді (ұзындығы 2 және қадамы 4 болатын бірінші қима жолдың екі элементін, ал ұзындығы 3 және қадамы 1 болатын екінші қима бағананың үш элементін сипаттайды):

Бастапқы матрица:

```
00 01 02
10 11 12
20 21 22
30 31 32
```

Жалпыланған қима:

```
00 01
10 11
20 21
```

Компьютер жадыднағы орналасуы:

```
00 10 20 30 01 11 21 31 02 12 22 32
```

Енді, мысалы, 2x3 ішкі матрицасының элементтерін нөлге теңестіру үшін келесі кодты жазуға болады:

```
valarray<double> v(1, 12); // Бірліктерден тұратын
                           // жиым құрамыз
int length[] = {2, 3};
int str[] = {4, 1};
valarray<int> len (length, 2);
valarray<int> d (str, 2);
// v[0], v[1], v[2], v[4], v[5], v[6] элементтерін
// нөлге теңестіреміз:
v[gslice(0, len, d)] = 0;
```

slice_array шаблоны сияқты, жалпыланған қимамен берілген жиым элементтерінің ішкі жиынымен жұмыс істеуіне арналған **gslice_array** класының шаблоны тағайындалған, ол дәл **slice_array** шаблонындағыдай элементтер жиынтығын қамтиды. **gslice_array** класының өзіндік объектілерін құруға тыйым салынған: оларға бағытталған барлық хабарласулар бастапқы жиымке қайта адрестеледі.

Егер индекстердің өзгеру заңын сипаттауға болмайтын жиым үшін оның элементтерінің кездейсоқ ішкі жиынын беру қажет болса, **valarray<bool>** болып табылатын **mask_array** логикалық жиымын қолданады. Осы жиымның **true** мәнін қабылдайтын элементі **valarray** класының сәйкес элементі ішкі жиымға қосылатынын білдіреді. **mask_array** үшін дәл **gslice_array** класындағы тәрізді өрістер мен әдістердің жиынтығы анықталған.

Бұл кластың объектілері осының алдындағы бөлімде сипатталған **valarray** жалпыланған индекстеу операцияларында қолдану нәтижесінде жасалады:

```
bool m[] = {true, true, false, true, false};
valarray<bool> mask (m, 5);
// v6 жиымының 0, 1 және 2 нөмірлі элементтеріне
// алдыңғы бөлімде сипатталған v4 жиымының сәйкес
// элементтері синусының мәні меншіктеледі:
valarray<int> v6 = sin(v4[mask]);
```

valarray элементтерін қайта реттеуге мүмкіндік беретін, кездейсоқ ішкі жиынды бөліп алудың басқа тәсілі **indirect_array** жанама жиымын қолдану болып табылады. Мұнда жиым индекстерінің тізбегі беріледі. Бұл

кластың объектісі индекс ретінде **valarray<size_t>** шамасын қолдану кезінде құрылады, мысалы:

```
int i[] = {3, 2, 4, 0, 1};
valarray<int> index(i, 5);
valarray<int> v7 = v4[index];
// v7[0] = v4[3], v7[1] = v4[2], v7[2] = v4[4],
// v7[3] = v4[0], v7[4] = v4[1]
```

Индекс ретінде қолданылатын жиым элементтерінің саны индекстелетін жиымдағы элементтер санынан артық болмауы тиіс.

Аталған **slice_array**, **gslice_array**, **indirect_array** және **mask_array** кластарын тікелей құруға және көшіруге болмайды. Олар қолданушылар үшін жасырын және **valarray** жиымының индексі ретінде сәйкесінше қималарды, жалпыланған қималарды, **valarray** логикалық жиымдарын және **size_t** типті жиымдарын қолдану нәтижесінде құрылады.

16-ТАРАУ

Стандартты кітапхананың басқа құралдары

Қарастырылған құралдардан басқа, C++ стандартты кітапханасында тілді сүйемелдеу, диагностика, оқшаулау, комплекстік сандармен жұмыс істеу құралдары, С стандартты кітапханасынан мұраланған 200-ден артық функциялар мен соларды сүйемелдеуге қажетті типтер және макростар бар.

Тілді сүйемелдеу құралдары программаның орындалуы кезінде болуы тиісті функциялар мен типтердің сипаттамаларын қамтиды. Олардың құрамына программаны іске қосу және аяқтау, **new** және **delete** операцияларын қолдау, құрамдас мәліметтер типтерін сипаттау, типтерді динамикалық түрде идентификациялау, ерекше жағдайларды өңдеу және орындау уақытының басқа да құралдары кіреді. Бұл құралдар **<cstddef>**, **<limits>**, **<climits>**, **<cfloat>**, **<cstdlib>**, **<new>**, **<typeinfo>**, **<exception>**, **<cstdarg>**, **<csetjmp>**, **<ctime>**, **<csignal>** тақырыптық файлдарында берілген.

Кітапханада (**<limits>** файлында) **numeric_limits** класының шаблоны анықталған, мұнда мәліметтер типінің жүзеге асырылуға тәуелді қасиеттері: диапазоны, дәлдігі, таңбасының болуы, т.с.с. анықталады. Мәліметтердің негізгі типтерінің әрқайсысы үшін **numeric_limits** шаблонының специализациясы анықталған. Дәл осы сипаттамалар С кітапханасынан мұраланған **<climits>** және **<cfloat>** тақырыптақ файлдарында сипатталған тұрақтыларда да анықталады, мысалы:

- **FLT_DIG** – **float** типті шамалардағы дұрыс мағыналы цифрлар саны;
- **FLT_MIN** – **float** типімен бейнелеуге болатын ең кіші нормаланған жылжымалы нүктелі сан;
- **UINT_MAX** – **unsigned int** типті шама қабылдай алатын ең үлкен мән (65 535 шамасынан кем болмауы тиіс).

Бұл тұрақтылар толығынан 5қосымшада сипатталған.

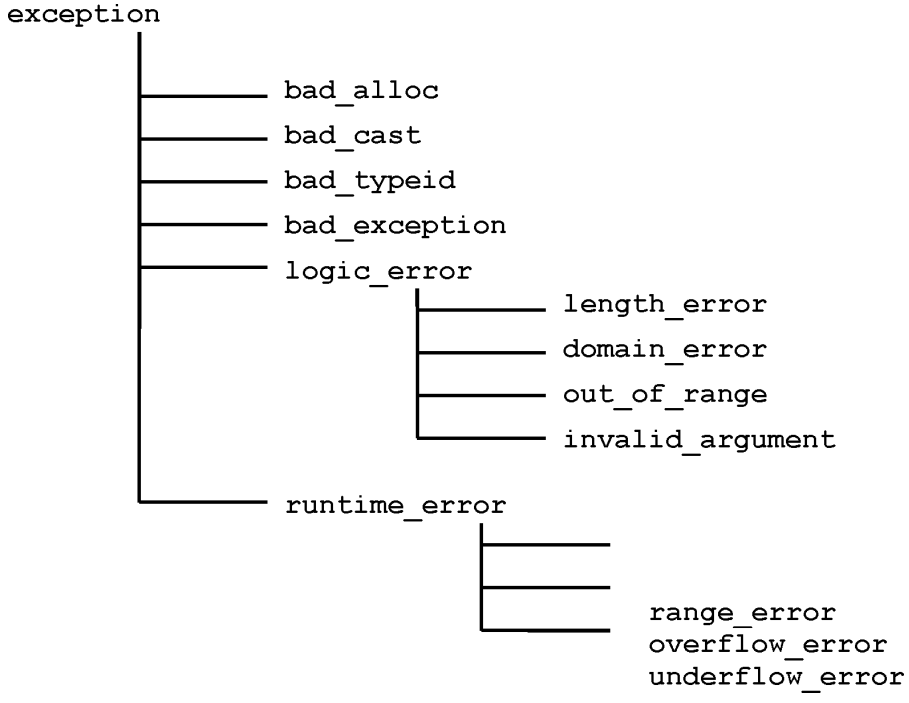
Жадыны бөлу кезіндегі қателерді өңдеу үшін кітапханада **bad_alloc** класы, **new_handler** қателерді өңдеушіфункциялар типі және **set_new_handler** жаңа өңдеушіні тағайындау функциясы сипатталған.

Жадыны бөлу қателіктерін өңдеудің механизмі келесідей: егер **new** операциясы динамикалық жадының қажетті көлемін бөле алмайтын болса, ол **new_handler** типті өңдеуші функцияны шақырады, бұл функция жадыны босатуға тырысуы тиіс немесе **bad_alloc** класының немесе одан туынды кластың объектісі түрінде аластаманы тудыруы керек, не болмаса программаны аяқтау үшін **abort** немесе **exit** функциясын шақыруы тиіс. **set_new_handler** функциясы стандартты өңдеушінің орнына өзіндік өңдеушіні орнату үшін қолданылады. Көбінесе ол **new** және **delete** операцияларын қайта жүктеу кезінде қолданылады.

Типтерді динамикалық түрде идентификациялауды қолдау үшін кітапханада **type_info** класы («Типтерді түрлендіру» бөлімінде, 249 б. қарастырылған), сондайақ **bad_cast** және **bad_typeid** ерекше жағдайлар кластары анықталған.

Барлық *стандартты ерекше жағдайлар* **<exception>** тақырыптық файлында сипатталған **exception** класының ұрпақтары болып табылады.

Диагностика құралдарының қатарына программаның жұмысы барысында туындайтын қателердің диагностикасын жүргізу үшін қолданылатын стандартты кластар мен функциялар жатады. Барлық қателер программаны іске қосудан бұрын табуға болатын *логикалық* (**logic_error**) қателер мен *орындалу уақытының қателері* (**runtime_error**) болып бөлінеді. Кітапханада ерекше жағдайлар кластарының қарапайым иерархиясы анықталған, ол программалаушы құратын ерекше жағдайларға негіз болуы тиіс.



16.1-сурет. Стандартты ерекше жағдайлар кластарының иерархиясы

Төменде осы кластар қысқаша сипатталған:

- **bad_alloc** – **new** көмегімен жадыны динамикалыық бөлу кезінде шығатын қателік;
- **bad_cast** – **dynamic_cast** операторын дұрыс қолданбау («Типтерді түрлендіру» бөлімін, 249 б. қараңыз);
- **bad_typeid** – **typeid** операциясы операнд типін анықтай алмайды (257 б.);

- **bad_exception** – ерекше жағдайларының спецификациясында осы класс көрсетілген функцияны шақыру кезінде күтпеген аластама орындалса, осы аластама туындайды;
- **length_error** – **size_t** типінің ең үлкен өлшемінен артық болатын объектіні құруға талпыныс кезінде шығады;
- **domain_error** – әрекетті орындау алдында ішкі шарттардың бұзылуы;
- **out_of_range** – функцияны оның параметрі қабылдай алмайтын мәндері арқылы шақыруға талпыныс;
- **invalid_argument** – функцияны дұрыс берілмеген параметрлер арқылы шақыруға талпыныс;
- **range_error** – орындалу нәтижесінің дұрыс болмауы;
- **overflow_error** – арифметикалық түрде толып кету (берілген мәліметтер типі үшін нәтиженің тым үлкен болатын жағдайы);
- **underflow_error** – дәреженің жоғалуы (берілген мәліметтер типі үшін нәтиженің тым кіші болуы).

Осы кластарды қолдану үшін **<stdexcept>** тақырыптық файлын іске қосу қажет.

Оқшаулау құралдарының қатарына әртүрлі мемлекеттер үшін ерекшеленетін категориялар бойынша реттелген ақпараттар кіреді, мысалы, оларға күнай мерзімі мен уақыттың бейнеленуі, ұлттық ақша бірлігінің форматы, сандар мен символдардың бейнеленуі жатады. Категориялар кластар арқылы берілген. Программалаушы оларды программаның әртүрлі мемлекеттердегі жұмыс істеуі кезінде әмбебап болуын қамтамасыз ету үшін қолдана алады. Оқшаулау құралдарының сипаттамалары **<locale>** және **<clocale>** тақырыптық файлдарында орналасқан.

Комплекстік сандармен жұмыс істеу үшін кітапхана **complex** класының шаблонын, **float**, **double** және **long double** типтері үшін оның специализацияларын және басқа да көптеген функцияларды қамтамасыз етеді. Осы құралдардың барлығына **<complex>** тақырыптық файлын іске қосқан кезде қол жеткізуге болады.

Кітапхана функциялары 6қосымшада, тұрақтылар, макростар және типтер 5-қосымшада, тақырыптық файлдар 4-қосымшада сипатталған.

III бөлімге арналған жаттығулар

Төменде көрсетілген тапсырмаларды жүзеге асыратын программалардың мысалдары практикумда [12] берілген.

Стандартты ағымдар

«Циклдік есептеу үрдістері» бөлімінде берілген (бірінші бөлімге арналған жаттығуларды қараңыз) тапсырмаларды стандартты ағымдардың, манипуляторлардың және форматтаушы әдістердің қолданылуымен орындау керек.

Файлдық ағымдар

«Функциялар және файлдар» бөлімінде берілген тапсырмаларды (бірінші бөлімге арналған жаттығуларды қараңыз) файлдық ағымдарды және қателерді өңдеу әдістерін қолдана отырып, орындау керек.

Ағымдар және қолданушылар анықтаған типтер

«Кластар» бөліміндегі программаларды (екінші бөлімге арналған жаттығуларды қараңыз) құрылған кластар үшін асыра жүктелген ағымға енгізу/шығару операцияларымен толықтырыңыз.

Тіркестер

«Тіркестермен және символдармен жұмыс істеуге арналған кітапхана функциялары» бөліміндегі тапсырмаларды (бірінші бөлімге арналған жаттығуларды қараңыз) **`string`** класын қолдану арқылы орындау қажет.

Контейнерлік кластар

«Бірөлшемді және екіөлшемді жиымдар», «Құрылымдар» және «Мәліметтердің динамикалық құрылымдары» бөлімдеріндегі тапсырмаларды (бірінші бөлімге арналған жаттығуларды қараңыз) кітапхананың сәйкес контейнерлік кластарын қолдану арқылы орындау қажет.

Алгоритмдер

«Бірөлшемді жиымдар» бөліміндегі тапсырмаларды (бірінші бөлімге арналған жаттығуларды қараңыз) кітапхананың контейнерлік кластарын және алгоритмдерін қолдану арқылы орындау керек.

valarray класы

«Бірөлшемді жиымдар» және «Екіөлшемді жиымдар» бөлімдеріндегі тапсырмаларды (бірінші бөлімге арналған жаттығуларды қараңыз) **`valarray`** класын және қималарды қолдану арқылы орындау керек.

Соңғы сөз

Бұл кітапта C++ тілінің негіздерін сипаттау мәселесіне үлкен көңіл бөлінген. Оларды жеткілікті деңгейде меңгеру программаны жазу үшін қажетті база болып табылады. Бірақ тілдің мүмкіндіктерінің жүзеге асырылу механизмдерін және олардың өзара әрекеттесу тәсілдерін білмей, тиімді, жеңіл оқылатын және кеңейтілетін программаны құру мүмкін емес. Сонымен қатар, Скотт Майерстің пайымдауынша, «C++ тілін дұрыс қолдана отырып жұмыс жасау адамды рахаттандыра алады». Ал тілдің қызметін толық деңгейде түсінбеу, керісінше, қателіктері көп және сүйемелдеуге келмейтін программалардың құрылуына алып келеді.

Тілдің ісәрекетін оның құрылуы кезінде қойылған мақсаттар анықтайды. C++ тілі үшін негізгі мақсаттар тиімділік, программаларды нақты тапсырмалардың кең ауқымды бөлігіне қолданылу мүмкіндігі, сонымен қатар C тілімен, дәстүрлі құралдармен және ортамен үйлесімділігі болып табылады.

Тілдің құрылу тарихы және оның қазіргі уақытқа дейін ие болған мүмкіндіктерінің енгізілу себептері Б. Страуструптың «C++ тілінің дизайны және эволюциясы» кітабында [18] толық сипатталған. Осы кітапты оқу барысында тілдің құралдары басқа жолмен емес, неге дәл осы жолмен жүзеге асырылғанын түсінуге болады. Нәтижелі программаларды сауатты жазуға арналған пайдалы практикалық кеңестер [17], [13] және [9] кітаптарында жазылған. Тіл және стандартты кітапхана туралы толық анықтамалық ақпарат алу үшін, басқасына қарағанда, ISO/IEC 14882:1998 стандартын пайдаланған жөн (оның соңғы нұсқасынан аздап өзгешеліктері бар бұрынғы нұсқаларын Интернеттен табуға болады).

Жақын арада «Питер» баспасынан практикумның [12] екінші бөлімі жарық көреді, оның құрамындағы мәліметтер ұсынылып отырған кітаптың екінші және үшінші бөлімдеріне сәйкес келеді, бірақ оның шеңберінен тыс шығады. Нақты жобалар бойынша жұмыс жасаған кез келген программалаушы операторлардың тек орындалу ережелері мен синтаксисін білу сапалы программа жазуға жеткіліксіз екендігін түсінеді. Практикумда қарастырылған сұрақтар – UML унификацияланған модельдеу тілінің негіздері, программалық инженерияның (software engineering) базалық концепциялары, олардың қатарындағы жобалау шаблондары, программаны түзетіп жөндеу технологиясы – программалауға жаңа келгендерді кәсіби программалау әлеміне жақындастырады.

Автор барлық оқырмандарға алдағы уақытта C++ тілін үйрену мен қолдануда сәттілік тілейді!

Қолданылған әдебиеттер

1. *Аммерааль Л.* STL для программистов на C++. – М.: ДМК, 2000. – 240 с.
2. *Бабэ Б.* Просто и ясно о Borland C++. – М.: БИНОМ, 1995. – 400 с.
3. *Бадд Т.* Объектно-ориентированное программирование в действии. – СПб.: Питер, 1997. – 464 с.
4. *Баженова И. Ю.* Visual C++ 6.0. Уроки программирования. – М.: Диалог- МИФИ, 1999. – 416 с.
5. *Дэвис С. C++* для «чайников». – К.: Диалектика, 1996. – 304 с.
6. *Брукс Ф.* Мифический человеко-месяц, или как создаются программные комплексы. – М.: «Символ-Плюс», 2000. – 304 с.
7. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами на C++. – М.: БИНОМ, 1998. – 560 с.
8. *Вирт Н.* Алгоритмы + структуры данных = программы. – М.: Мир, 1985. – 406 с.
9. *Голуб А. И.* C и C++. Правила программирования. – М.: БИНОМ, 1996. – 272 с.
10. *Карпов Б., Баранова Т.* C++: специальный справочник. – СПб.: Питер, 2000. – 480 с.
11. *Павловская Т. А., Щупак Ю. А.* C/C++. Структурное программирование: Практикум. – СПБ.: Питер, 2002. – 240 с.
12. *Павловская Т. А., Щупак Ю. А.* C/C++. Объектно-ориентированное программирование: Практикум. – СПБ.: Питер, 2004.
13. *Мейерс С.* Эффективное использование C++. – М.: ДМК, 2000. – 240 с.
14. *Подбельский В. В.* Язык Си++: Учебное пособие. – М.: Финансы и статистика, 1996. – 560 с.
15. *Пол И.* Объектно-ориентированное программирование с использованием C++. – К.: ДиаСофт, 1995. – 480 с.
16. *Романовская Л. М. и др.* Программирование в среде Си для ПЭВМ ЕС. – М.: Финансы и статистика, 1992. – 352 с.
17. *Страуструп Б.* Язык программирования C++. – СПб.: БИНОМ, 1999. – 991 с.
18. *Страуструп Б.* Дизайн и эволюция языка C++. – М.: ДМК, 2000. – 448 с.
19. *Шилдт Г.* Теория и практика C++. – СПб.: BHV, 1996. – 416 с.
20. Неформальное введение в C++ и Turbo Vision. – Галерея «Петрополь», 1992. – 381 с.

1-ҚОСЫМША

C++ тілінің операциялары

Операциялар басымдықтарының кемуі бойынша келтірілген. Әртүрлі басымдықтағы операциялар аралық сызықтармен ажыратылған.

Операция	Қысқаша сипаттамасы
Унарлы операциялар	
`::`	көріну аймағына қатынасу
`.`	таңдау
`->`	таңдау
`[ ]`	индекстеу
`( )`	функцияны шақыру
`<тип>( )`	құрастыру
`++`	постфикстік инкремент
`--`	постфикстік декремент
`typeid`	типті идентификациялау
`dynamic_cast`	орындалу кезеңінде тексеру арқылы типті түрлендіру
`static_cast`	компиляция кезеңінде тексеру арқылы типті түрлендіру
`reinterpret_cast`	типті тексерусіз түрлендіру
`const_cast`	типті тұрақтылық түрлендіру
`sizeof`	объектінің немесе типтің мөлшері
`--`	префикстік декремент
`++`	префикстік инкремент
`~`	разрядтар бойынша терістеу
`!`	логикалық терістеу
`-`	арифметикалық терістеу (унарлық минус)
`+`	унарлық плюс
`&`	адресті алу
`*`	адрессіздендіру
`new`	жады бөлу
`delete`	жадыны босату
`(<ТИП>)`	типті түрлендіру
`.*`	таңдау
`->*`	таңдау

Операция	Қысқаша сипаттамасы
Бинарлы және тернарлы операциялар	
`*`	көбейту
`/`	бөлу
`%`	бөлуден қалған қалдық
`+`	қосу
`-`	азайту
`<<`	солға ығыстыру
`>>`	оңға ығыстыру
`<`	кіші
`<=`	кіші немесе тең
`>`	үлкен
`>=`	үлкен немесе тең
`==`	тең
`!=`	тең емес
`&`	разрядтар бойынша конъюнкция (ЖӘНЕ)
`^`	разрядтары бойынша аластайтын НЕМЕСЕ
`|`	разрядтары бойынша дизъюнкция (НЕМЕСЕ)
`&&`	логикалық ЖӘНЕ
`||`	логикалық НЕМЕСЕ
`?:`	шартты операция (тернарлық)
`=`	меншіктеу
`*=`	көбейтіп барып меншіктеу
`/=`	бөліп барып меншіктеу
`%=`	бөлгендегі қалдықты тауып барып меншіктеу
`+=`	қосып барып меншіктеу
`-=`	азайтып барып меншіктеу
`<<=`	солға ығыстырып алып меншіктеу
`>>=`	оңға ығыстырып алып меншіктеу
`&=`	меншіктеуі бар разрядтары бойынша ЖӘНЕ
`|=`	меншіктеуі бар разрядтары бойынша НЕМЕСЕ
`^=`	меншіктеуі бар разрядтар бойынша аластайтын НЕМЕСЕ
`throw`	аластау
`.`	тізбектей отырып есептеу

2-ҚОСЫМША

printf топтамасының функциялары үшін формат спецификациялары

Спецификация	Түсініктемесі
c	аргумент жеке символ ретінде қарастырылады
d, i	аргумент ондық жүйеге түрлендіріледі
e, E	**float** немесе **double** типіндегі айнымалы ретінде қарастырылатын аргумент ондық санға [-]m.nnnnnne[+-]xx бейнесінде түрлендіріледі, мұнда ұзындығы n таңбадан тұратын тіркес көрсетілген дәлдікпен анықталады (келесі кестені қ.). Келісім бойынша алынатын дәлдік 6-ға тең
f	**float** немесе **double** типіндегі айнымалы ретінде қарастырылатын аргумент ондық санға **[-]mmm.nnnnn** бейнесінде түрлендіріледі, мұнда ұзындығы **n** таңбадан тұратын тіркес көрсетілген дәлдікпен анықталады (келесі кестені қ.). Келісім бойынша алынатын дәлдік 6-ға тең
g, G	**%e** немесе **%f** форматының қысқаша жазылатын түрі пайдаланылады; мағынасы жоқ нөлдер көрсетілмейді
o	аргумент таңбасыз сегіздік санға түрлендіріледі (алдыңғы нөл жазылмайды)
p	нұсқауышты он алтылық форматта шығару (бұл спецификация стандартқа кірмейді, бірақ ол жүзеге асырылған барлық нұсқаларда кездеседі)
s	аргумент тіркес болып табылады: тіркес символдары нөлдік символға жеткенше немесе дәлдік спецификациясында көрсетілген таңбалар саны біткенше шығарылады
u	аргумент таңбасыз ондық санға түрлендіріледі
x, X	аргумент таңбасыз он алтылық санға түрлендіріледі (алдыңғы 0x жазылмайды)
%	% символы шығарылады

Формат модификаторлары

Формат модификаторлары шығарылатын мәннің орналастырылу енін басқару үшін қолданылады. Модификаторлар бір немесе екі саннан тұрады, біріншісі – сан шығарылатын позициялардың ең кіші мәнін береді, екіншісі – сол позициялардың нешеуі санның бөлшегін (дәлдігін) көрсету үшін қолданылатынын білдіреді. Егер көрсетілген позициялар саны шығарылатын мәнді бейнелеуге жеткіліксіз болса, онда автоматты түрде мәнді толық шығаруға жеткілікті позициялар саны бөлінеді:

- **%-minC** немесе **%minC**;
- **%-min.precisionC** немесе **%min.precisionC**.

Мұндағы **C** – жоғарыда келтірілген кестеде көрсетілген формат спецификациясы, **min** – өрістің ең кіші мәнін беретін сан. Ондық санмен берілетін **precision** модификаторының мағынасы ол пайдаланатын төмендегі формат спецификацияларына тәуелді болады:

- тіркесті шығару кезінде (**%s** спецификациясы) **precision** шығарылатын символдардың максималды санын көрсетеді;
- нақты санды шығару кезінде (**%f** немесе **%e** спецификациялары) **precision** ондық нүктеден кейінгі шығарылатын цифрлар санын көрсетеді;
- бүтін санды шығару кезінде (**%d** немесе **%i** спецификациялары) **precision** шығарылатын цифрлардың минималды санын көрсетеді. Егер сан **precision** мәнінде көрсетілгеннен аз цифрлар санымен бейнеленсе, онда сан мәніне алдыңғы (бастапқы) нөлдер қосылып шығарылады.
- нақты санды шығару кезінде (**%d** немесе **%G** спецификациялары) **precision** шығарылатын мәні бар цифрлардың максималды санын көрсетеді.

Минус (**-**) символы шығарылатын мәннің сол жақ шетке тураланатынын білдіреді, егер қажет болса, босорын таңбалары сол мәннің оң жақ шетіне орналасады. Минус болмаса, шығарылатын мән оң жақ шетке тураланып, мәннің сол жақ шеті босорындармен толықтырылады.

Спецификация алдында **l** (ел) және **h префикстері** қолданылуы мүмкін, мысалы, **%lf**, **%hu.**

h префиксі **d**, **i**, **o**, **x** және **X** типтерінің бірімен қолданылса, ол аргумент типінің **short int** екенін білдіреді, ал **u** типімен бірге тұрса, аргумент типінің **short unsigned int** екенін көрсетеді.

Ал **l** префиксі **d, i, o, x** және **X** типтерінің бірімен қолданылса, ол аргумент типінің **long int** екенін білдіреді, **u** типімен бірге тұрса – **long unsigned int** екенін, ал **u** типі **e**, **E**, **f**, **g** және **G** типімен бірге қолданылса, аргумент типі **float** емес, **double** екенін білдіреді.

Мысал:

```
#include <stdio.h>
int main(){
   int int1 = 45, int2 = 13;
   float f = 3.621; double dbl =2.23;
   char ch = 'z', *str = "ramambahari";
   printf("int1 = %d| int2 = %3d| int2 = %-4d|\n", int1,
           int2, int2);
   printf("int1 = %X| int2 = %3x| int2 = %4o|\n", int1,
           int2, int2);
   printf("f = %f| f = %4.2f| f = %6.1f|\n", f, f, f);
   printf("f = %g| f = %e| f = %+E|\n", f, f, f);
   printf("dbl = %5.2lf| dbl = %e| dbl = %4.1G|\n", dbl,
           dbl, dbl);
   printf("ch = %c| ch = %3c|\n", ch, ch);
   printf("str = %14s|\nstr = %-14s|\nstr = %s|\n", str,
           str, str);
   return 0;
}
```

Программа жұмысының нәтижесі:

```
intl = 45| int2 =     13| int2 = 13  |
intl = 2D| int2 =      d| int2 = 15  |
f = 3.621000| f = 3.62  | f =   3.6  |
f = 3.621   | f =   3.621000e+000    | f = +3.621000E+000|
dbl = 2.23  | dbl = 2.230000e+000    | dbl = 2           |
ch = z|  ch = z  |
str =      ramambahari  |
str = ramambahari       |
str = ramambahari|
```

3-ҚОСЫМША

Типтердің арифметикалық түрленулері

Егер өрнектер құрамына кіретін операндтардың типтері әртүрлі болып келетін болса, онда типтерді түрлендіру орындалады. Төменде түрлендірулер тізбегі көрсетілген.

- **char, unsigned char** немесе **short** типінің кез келген операндтары **int** типіне келесі ережелерге сәйкес түрлендіріледі:
  - **char** нөлмен немесе **char** типі үшін алдын ала келісім бойынша алынған таңбамен кеңейтіледі;
  - **unsigned char** нөлмен кеңейтіледі;
  - **signed char** таңбамен кеңейтіледі;
  - **short**, **unsigned short** және **enum** түрлендірілу барысында өзгермейді.
  - Кейіннен кез келген екі операнд **int**, немесе **float**, **double** немесе **long double** болып өзгереді.
- Егер операндтардың біреуі **long double** типті болса, онда екіншісі де **long double** типіне түрлендіріледі.
- Егер операндтардың біреуі **double** типті болса, онда екіншісі де **double** типіне түрлендіріледі.
- Егер операндтардың біреуі **float** типті болса, онда екіншісі де **float** типіне түрлендіріледі.
- Басқаша айтқанда, егер операндтардың біреуі **unsigned long** типті болса, онда екіншісі де **unsigned long** типіне түрлендіріледі.
- Әйтпесе, егер операндтардың біреуі **long** типті болса, онда екіншісі **long** типіне түрлендіріледі.
- Әйтпесе, егер операндтардың біреуі **unsigned** типті болса, онда екіншісі **unsigned** типіне түрлендіріледі.
- Әйтпесе, екі операнд та **int** типті болуы керек.

Нәтиже типі де сол өрнек құрамындағы операндтар типтері сияқты болып келеді.

4-ҚОСЫМША

Стандартты кітапхананың тақырыптық файлдары

Енгізу/шығару

<cstdio>	С стилінде енгізу/шығару функциялары
<cstdlib>	Символдармен жұмыс істеу функциялары
<cwchar>	Көпбайтты символдар үшін С стилінде енгізу/шығару функциялары
<fstream>	Файлдық ағымдар
<iomanip>	Манипуляторлар
<ios>	Енгізу/шығару ағымдарының базалық кластары
<iosfwd>	Енгізу/шығару құралдарын алдын ала жариялау
<iostream>	Стандартты объектілер және енгізу/шығару ағымдарымен орындалатын операциялар
<istream>	Кіріс ағымдары
<ostream>	Шығыс ағымдары
<sstream>	Тіркестік ағымдар
<streambuf>	Ағымдарды буферлеу

Тіркестер

<cctype>	Символдарды жіктеу функциялары
<cwctype>	
<cwtype>	
<cstring>	С стилінде тіркестермен жұмыс істеу функциялары
<cstdlib>	Символдармен жұмыс істеу функциялары
<string>	Тіркестік класс

Контейнерлік кластар

<bitset>	Биттік жиын
<deque>	Екіжақты кезек
<list>	Қосбайланысты тізім
<map>	Сөздік, дубликаттары бар сөздік
<queue>	Кезек, басымдықтары бар кезек
<set>	Жиын, дубликаттары бар жиын
<stack>	Стек
<vector>	Бірөлшемді жиым

Алгоритмдер, итераторлар, утилиттер

<algorithm>	Алгоритмдер
<cstdlib>	С кітапханасынан сұрыптау және іздеу
<ctime>	С стиліндегі мерзім мен уақыт
<functional>	Функционалдық объектілер
<iterator>	Итераторлар
<memory>	Контейнерлер үшін жадыны бөлушілер
<utility>	Операторлар және жұптар

Диагностика, тілді сүйемелдеу, локализация

<cassert>	Assert макросы
<cerrno>	С стиліндегі қателерді өңдеу
<cfloat>	С стиліндегі шектелген мәндер макростары
<climits>	С стиліндегі шектелген мәндер макростары
<clocale>	С стиліндегі локализация
<csignal>	С стиліндегі сигналдарды өңдеу
<cstdarg>	Аргументтер саны айнымалы функцияларды сүйемелдеу
<cstddef>	С кітапханасы тілін сүйемелдеу
<cstdlib>	Программаны аяқтау
<ctime>	Жүйелік сағаттар
<exception>	Аластауларды сүйемелдеу
<limits>	Сандық шектеулер
<locale>	Локализация кластары
<new>	Динамикалық жадымен жұмыс істеу
<stdexcept>	Стандартты ерекше жағдайлар
<typeinfo>	Типтерді динамикалық идентификациялау

Сандық есептеулер

<cmath>	Математикалық функциялар
<complex>	Комплекстік сандар
<cstdlib>	С стиліндегі кездейсоқ сандар
<numeric>	Сандық операциялар
<valarray>	Сандық есептеулер үшін кластар

5-ҚОСЫМША

Тұрақтылар, макростар және стандартты кітапхананың мәліметтер типтері

Бұл қосымшада көрсетілген тұрақтылар, макростар және типтер, негізінен, С тілінен мұраланып келген стандартты кітапхананың функцияларында пайдаланылады.

assert	**<assert.h>**

```
void assert(int expression);
```

Егер **expression** өрнегінің мәні жалған (нөл) болса, **assert** макросы программаның орындалуын үзеді. Жұмыс аяқталғанда, қате туралы мынадай хабарлама шығарылады: **Assertion failed: expression, file <имя_файла>, line <номер_строки>.**

BUFSIZ	**<stdio.h>**

BUFSIZ тұрақтысы мәліметтер ағымы буферінің көлемін байтпен **fopen()**, **freopen()**, **setbuf()** функциялары арқылы беру үшін қолданылады. Мәліметтер ағымын ашу кезінде оған автоматты түрде көлемі **BUFSIZ** мәніне тең буфер қосылады. **BUFSIZ** буферінің ең кіші мәні – 255 байт.

CHAR_BIT	**<limits.h>**

CHAR_BIT тұрақтысы мәліметтердің ең қысқа типін (стандарт бойынша – **char** типі) битпен сақтайды. **CHAR_BIT** тұрақтысының ең кіші мәні – 8.

cin, cout, cerr, clog	**<iostream.h>**

Стандартты шеткері (перифериялық) құрылғылармен байланысқан ағым кластары объектілері.

Объект	**Мәліметтер ағымы**	**Класс**
cin	**stdin**	**istream**
cout	**stdout**	**ostream**
cerr	**stderr**	**ostream**
clog	**stderr** (буферленген)	**ostream**

CHAR_MAX, CHAR_MIN	**<limits.h>**

CHAR_MAX тұрақтысында **char** типі қабылдай алатын ең үлкен мән сақталады. Егер бір нақты нұсқада **signed char** типі сүйемелденетін болса, онда **CHAR_MAX** мәні **SCHAR_MAX** мәнімен бірдей болады, әйтпесе – **UCHAR_MAX** мәніндей болады.

CHAR_MIN тұрақтысында **char** типі қабылдай алатын ең кіші мән сақталады. Егер бір нақты нұсқада **signed char** типі сүйемелденетін болса, онда **CHAR_MIN** мәні **SCHAR_MIN** мәнімен бірдей болады, әйтпесе – **UCHAR_MIN** мәніндей болады.

clock_t	<time.h>

clock_t мәліметтер типі уақытты көрсетеді:

```
typedef long clock_t;
```

clock() функциясы осы типтегі мәнді қайтарады.

CLOCKS_PER_SEC	<time.h>

CLOCKS_PER_SEC тұрақтысында процессордың бір секундтағы циклдер саны сақталады. **clock()** функциясы процессор циклдерімен өрнектелген уақытты қайтарады. Оны секундтарға түрлендіріп көрсету үшін **CLOCKS_PER_SEC** тұрақтысына бөлу керек.

DBL...	<float.h>

(**FLT...** қара)

div_t	<float.h>

div_t құрылымы **div()** функциясының қайтаратын мәнінің типін анықтау үшін қолданылады. **quot** құрылымы өрісі бөліндіні, ал **rem** өрісі – бөлгеннен қалған қалдықты бейнелейді.

```
typedef struct{
   int quot;
   int rem;
}div_t;
```

EDOM	<errno.h>

EDOM макросы нөлден өзге бүтін тұрақтыны көрсетеді, ол аймақтың қатесін анықтау мүмкіндігін береді, яғни аргументтің мәні математикалық функция анықталған аймақтан шығып кеткен кезде туындайтын қатені көрсетеді.

EOF	<stdio.h>

EOF тұрақтысы файл біткенін білдіреді (теріс бүтін сан болып табылады).

ERANGE	<errno.h>

ERANGE макросы нөлден өзге бүтін тұрақтыны көрсетеді, ол диапазонның қатесін анықтау мүмкіндігін береді. Диапазонның қатесі функция нәтижесі **double** типті объект түрінде бейнелене алмайтын сәтте туындайды. Мұндайда **errno** айнымалысы (төменге қара) **ERANGE** мәнін қабылдайды.

errno	<errno.h>

Ауқымды бүтін сан типіндегі **errno** айнымалысы математикалық функцияларды пайдалану кезінде туындайтын қателерді өңдеу үшін қолданылады (анықталған соңғы қате жайлы қосымша ақпарат беретін қате нөмірін сақтай алады).

EXIT_FAILURE	<stdlib.h>

Қате жайлы хабарлама беру мақсатында **exit()** кітапханалық функциясын қолдану кезінде пайдалануға болатын қалып коды.

EXIT_SUCCESS	<stdlib.h>

Программаның сәтті аяқталғаны жайлы хабарлама беру мақсатында **exit()** кітапханалық функциясын қолдану кезінде пайдалануға болатын қалып коды.

FILE	<stdio.h>

FILE мәліметтер типі файлмен операциялар орындауға керекті ақпаратты сақтайды. Бұл типтің құрылымы өзінің нақты жүзеге асырылуына байланысты болып келеді, төменде оның мүмкін болатын бір сипаттамасы келтірілген.

```
typedef struct {
   int level;              // буфер қалпы жалаушасы
   unsigned flags;         // файл қалпы жалаушасы
   char fd;                // файл префиксі
   unsigned char hold;     // жіберілмеген символ
   int bsize;              // ішкі буфер көлемі
   unsigned char_FAR *buffer;  /* буфер ішіндегі
қатынастар үшін нұсқауыш мәні; буферлеу режиміне қарай
буфер басын, тіркес басын немесе буфер ішіндегі
нұсқауыштың ағымдағы мәнін береді  */
   unsigned char FAR *curp; /* буфер ішіндегі қатынастар
үшін нұсқауыштың ағымдағы мәні; программамен мәлімет
алмасу үшін буфердің ағымдағы позициясын береді */
   unsigned istemp;        // уақытша файлдың жалаушасы
   short token;            // файл нақтылығы маркері
} FILE;
```

filebuf	<fstream.h>

char мәліметтертипі үшін **basic_filebuf** класы шаблонының специализациясы.

```
typedef basic filebuf<char> filebuf;
```

FILENAME_MAX	<stdio.h>

FILENAME_MAX тұрақтысында файл атының ұзындығына шектеу қойылады.

float_round_style	<stdio.h>

float_round_style саналатын типі жылжымалы нүктелі сандарды (**numeric_limits** қара) қосу кезінде сандарды дөңгелектеудің әртүрлі тәсілдері үшін тұрақтыларды анықтайды .

```
enum float_round_style {
   round_indeterminate,
   round toward_zero,
   round to_nearest,
   round_toward_infinity,
   round_toward neg_infinity
};
```

FLT...	<float.h>

Келесі тұрақтылар **double (DBL_...)** және **long double(LDBL_...)** типтері үшін анықталған:

Тұрақты	Сипаттамасы
FLT_DIG	**float** типті объектідегі айқындалған ондық цифрлар саны
FLT_EPSILON	**float** типті айнымалының мүмкін болатын ең кіші мәні, ол мынадай болады: **1.0 + FLT_EPSILON != 1.0**
FLT_MANT_DIG	**float** типті сандар үшін мантиссадағы **FLT_RADIX** негізі бойынша цифрлар саны
FLT_MAX	**float** типті жылжымалы нүктелі санның ең үлкен мәні
FLT_MAX_10_EXP	10 санын **FLT_MAX_10_EXP** дәрежесіне шығарғандағы нәтиженің ең үлкен бүтін мәні **float** типті объект болып табылады
FLT_MAX_EXP	**FLT_RADIX** санын (**FLT_MAX_EXP-1**) дәрежесіне шығарғандағы нәтиженің ең үлкен бүтін мәні **float** типті объект болып табылады
FLT_MIN	**float** типті объект болып табылатын жылжымалы нүктелі ең кіші нормалданған сан
FLT_MIN_10_EXP	10 санын **FLT_MIN_10_EXP** дәрежесіне шығарғандағы нәтиженің ең кіші бүтін мәні **float** типті объект болып табылады
FLT_MIN_EXP	**FLT_RADIX** санын (**FLT_MIN_EXP-1**) дәрежесіне шығарғандағы нәтиженің ең кіші бүтін мәні **float** типті объект болып табылады

Келесі тұрақтылардың **double** және **long double** типтері үшін сәйкестігі жоқ.

Тұрақты	Сипаттамасы
FLT_RADIX	жылжымалы үтірлі сандар үшін экспонентаны бейнелеу негізі
FLT_ROUNDS	жылжымалы үтірлі сандармен операциялар орындау кезінде пайдаланылатын сандарды дөңгелектеу тәсілі

FOPEN_MAX	<stdio.h>

FOPEN_MAX тұрақтысында ашық ағымдардың ең үлкен саны беріледі. Ол 8-ден артық немесе 8-ге тең болуы тиіс, мұндайда әрбір программа автоматты түрде **stdin**, **stdout** және **stderr** стандартты мәліметтер ағымдарын ашады. Бестен артық ағымдарды пайдаланатын программалар программаға макростар арқылы жүйеден қажетті ағымдар берілетініне тексеру жүргізуі керек.

fpos_t	<stdio.h>

Файлдың ағымдағы позициясын сақтау үшін **fgetpos()** және **fsetpos()** функциялары арқылы **fpos_t** типі қолданылады:

```
typedef long fpos_t;
```

HUGE_VAL	<math.h>

HUGE_VAL макросында **double** типіндегі шама қабылдай алатын ең үлкен мән анықталады.

ifstream	<fstream.h>

char мәліметтер типі үшін **basic_ifstream** класы шаблонының специализациясы.

```
typedef basic_ifstream<char> ifstream;
```

INT_MAX, INT_MIN	<limits.h>

INT_MAX тұрақтысында **int** типіндегі шама қабылдай алатын ең үлкен мән сақталады. **INT_MAX** мәні 32 767-ден артық немесе соған тең болуы тиіс.

INT_MIN тұрақтысында **int** типіндегі шама қабылдай алатын ең кіші мән сақталады. **INT_MIN** мәні -32 767-ден кем немесе соған тең болуы тиіс.

_IOFBF, _IOLBF, _IONBF	<stdio.h>

Бұл тұрақтылар **setvbuf()** функциясында буферлеу режимін анықтау үшін қолданылады:

_IOFBF	0	толық буферлеу;
_IOLBF	1	мәтіндік файлды жолма-жол буферлеу;
_IONBF	2	буферлеуді болдырмау.

ios	<ios.h>

char мәліметтер типі үшін **basic_ios** класы шаблонының специализациясы.

```
typedef basic_ios<char> ios;
```

istream	<istream.h>

char мәліметтер типі үшін **basic_istream** класы шаблонының специализациясы.

```
typedef basic_istream<char> istream;
```

istringstream	<sstream.h>

char мәліметтер типі үшін **basic_istringstream** класы шаблонының специализациясы.

```
typedef basic_istringstream<char> istringstream;
```

L_tmpnam	<stdio.h>

L_tmpnam тұрақтысы файл аты ретінде **tmpnam()** функциясы беретін тіркестің ең кіші ұзындығын береді.

LC_...	<locale.h>

setlocale() және **localeconv()** функциялары локализация категорияларын таңдау үшін **LC_** префиксі бар тұрақтыларды пайдаланады.

Тұрақтылар	Сипаттамасы
LC_ALL	Барлық категориялар
LC_COLLATE	Мәтін мен сұрыптауды салыстыратын ақпарат
LC_CTYPE	Символдық жазбаларды өңдеуге арналған ақпарат
LC_MONETARY	Валюталық мәліметтерді қалыптастыруға арналған ақпарат
LC_NUMERIC	Ондық нүктені бейнелейтін территориялық-спецификалық ақпарат
LC_TIME	Мерзім және уақыт туралы территориялық-спецификалық ақпарат

lconv	<locale.h>

lconv құрылымы территориялық-спецификалық ақпаратты сақтайды:

```
typedef struct lconv {
   char *currency_symbol;     // Валюта символы
   char *decimal_point;       // Ондық нүктеге арналған
                              // символ
   char * frac_digits;        // Валюта үшін үтірден
                              // кейінгі разряд
   char grouping;             // Цифрларды топтау
   char *int_curr_symbol;     // ISO валюталық символы
   char *mon_decimal_point;   // Валютаның ондық нүктеге
                              //арналған символы
   char mon_grouping;         // Валюталық мәліметтердің
                              //цифрларын топтау
   char *mon_thousand_sep;    // Валютадағы мыңдықтарды
                              // ажыратқыштар
   char n_cs_precedes;        // Валютадағы – белгісінің
                              // позициясы
   char n_sep_by_space;       // – белгісі мен теріс
            // валюталық шама арасындағы босорындар
   char n_sign_posn;          // Теріс валюталық шама үшін
            // символдың разряды мен форматы
```

```
    char *negative_sign;      // Теріс валюталық шама үшін
                              // белгілер
    char p_cs_precedes;   // Валюталық символ позициясы
    char p_sep_by_space;      // Валюталық символ мен
              // валюталық шама арасындағы босорындар
    char p_sign_posn;         /* Оң валюталық шама үшін +
         белгісінің позициясы және  форматталуы */
    char *positive_sign;      // Оң валюта үшін белгілер
    char *thousands_sep;      // Мыңдықтарды ажыратқыш
};
```

LDBL_...	<float.h>

(**FLT_...** қара)

ldiv_t	<stdlib.h>

div_t құрылымы **ldiv()** функциясы қайтаратын мәннің типін анықтау үшін қолданылады. **quot** құрылымының өрісі бөліндіні, ал **rem** өрісі – бөлгенде қалған қалдықты сақтайды.

```
typedef struct {
   int quot;
   int rem;
} div_t;
```

LONG_MAX, LONG_MIN	<limits.h>

LONG_MAX тұрақтысында **long** типті шама қабылдай алатын ең үлкен мән сақталады. **LONG_MAX** мәні кем дегенде 2 147 483 647 болуы тиіс.

LONG_MIN тұрақтысында **long** типті шама қабылдай алатын ең кіші мән сақталады. **LONG_MIN** мәні -2 147 483 647-ден артық болмауы тиіс.

MB_CUR_MAX	<stdlib.h>

MB_CUR_MAX макросында пайдаланылып отырған символдар жиынының бір символындағы байттар санының ең үлкен мәні беріледі.

MB_LEN_MAX	<limits.h>

Бұл тұрақтыда символдары бір байттан артық болып келетін барлық қосалқы символдар жиындарының бір символындағы байттар санының ең үлкен мәні сақталады.

mbstate_t	<wchar.h>

Символдық тіркестерді өңдейтін әртүрлі функциялар пайдаланатын типті анықтау.

new_handler	<new.h>

new және **new[]** операцияларын пайдалану кезінде қолданылатын, қателерді өңдеуге арналған функция типі

```
typedef void (*new handler)();
```

NULL	<stddef.h>, <string.h> және т.с.с.

NULL макросы нөлге нұсқауышты анықтайды. ANSI-стандартына сәйкес **NULL** 0 немесе 0L болып анықтала алады.

offsetof	<stddef.h>

offsetof макросы құрылымдағы элементтердің ығысуын байтпен қайтарады. Құрылым сілтемелерді немесе нұсқауыштарды сақтай алмайды

```
size_t offsetof(struct_bez, element_bez);
```

ofstream	<fstream.h>

char мәліметтер типі үшін **basic_ofstream** класы шаблонының специализациясы.

```
typedef basic_ofstream<char> ofstream;
```

ostream	<ostream.h>

char мәліметтер типі үшін **basic_ostream** класы шаблонының специализациясы.

```
typedef basic_ostream<char> ostream;
```

ostringstream	<sstream.h>

char мәліметтер типі үшін **basic_ostringstream** класы шаблонының специализациясы.

```
typedef basic_ostringstream<char> ostringstream;
```

ptrdiff_t	<stddef.h>

ptrdiff_t мәліметтер типі екі нұсқауыштың айырмасын сақтауға арналған таңбалы бүтін сандық тип болып табылады.

```
typedef int ptrdiff_t;
```

RAND_MAX	<stdlib.h>

rand() функциясы қайтаратын жалған кездейсоқ сандардың мүмкін болатын ең үлкен мәнін сақтайды.

SCHAR_MAX, SCHAR_MIN	<limits.h>

SCHAR_MAX тұрақтысында **signed char** типті шама қабылдай алатын ең үлкен мән сақталады. Ол 127-ден аз болмауы тиіс.

SCHAR_MIN тұрақтысында **signed char** типті шама қабылдай алатын ең кіші мән сақталады. Ол -127-ден артық болмауы тиіс.

SEEK_CUR, SEEK_END, SEEK_SET	<stdio.h>

Тұрақтылар **fseek()** функциясының аргументі ретінде пайдаланылып, файлдың ағымдағы позициялық көрсеткішін жылжыту үшін бастапқы нүктені тағайындайды.

SEEK_SET 0 Жылжыту файл басынан бастап орындалады;
SEEK_CUR 1 Жылжыту файлдан жазу/оқу кезінде ағымдағы позициялық көрсеткіштен бастап орындалады;
SEEK_END 2 Жылжыту файл соңынан бастап орындалады.

SHRT_MAX, SHRT_MIN	<limits.h>

SHRT_MAX тұрақтысында **short int** типті шама қабылдай алатын ең үлкен мән сақталады; ол 32 767-ден аз болмауы тиіс.

SHRT_ MIN тұрақтысында **short int** типті шама қабылдай алатын ең кіші мән сақталады; ол -32 767-ден артық болмауы тиіс.

SIG...	<signal.h>

Сигналдардың төменде көрсетілген типтері **raise()** функциясы арқылы қайтарылады да, **signal()** функциясы көмегімен өңделеді.

Сигнал	Мәні
SIG_DFL	Келісім бойынша аластауларды өңдеуіш-программаға сілтеме
SIG_ERR	Қате туындағанда, **signal()** функциясының қайтаратын мәні
SIG_IGN	Алынған сигналды елемеу
SIGABRT	Программаның кенеттен (апатты түрде) аяқталуы
SIGFPE	Нақты және бүтін сандардағы арифметикалық операциялар қатесі
SIGILL	Орындауға болмайтын машиналық нұсқау
SIGINT	Пернетақта арқылы жұмысты үзу
SIGSEGV	Компьютер жадын қорғау шарттарын бұзу
SIGTERM	Жұмысты аяқтауға сұраныс

sig_atomic_t	<signal.h>

sig_atomic_t мәліметтер типі программаны асинхрондық үзу кезінде онымен қатынасу қажет болған кездегі ақпаратты сақтау үшін қолданылады.

```
typedef sig_atomic_t;
```

size_t	<stdio.h>

size_t типі операциясының нәтижесін бейнелеу үшін пайдаланылатын таңбасыз бүтін сандық тип болып табылады. Бұл тип **unsigned long** типінен ұзынырақ бола алмайды.

```
typedef unsigned size_t;
```

stdin, stdout, stderr	<stdio.h>

stdin, **stdout** және **stderr** ағымдар атаулары, олар С кітапханасынан мұраланған функциялар арқылы құрылғылармен қатынас құру үшін пайдаланылатын атаулар болып табылады.

stderr	Қателер шығарылатын стандартты құрылғы. Келісім бойынша дисплей
stdin	Стандартты енгізу құрылғысы. Келісім бойынша пернетақта
stdout	Стандартты шығару құрылғысы. Келісім бойынша дисплей

streambuf	<streambuf.h>

char мәліметтер типі үшін **basic_streambuf** класы шаблонының специализациясы.

```
typedef basic_streambuf<char> streambuf;
```

streamoff, streampos	<ios.h>

Ағымдардағы үстіміздегі көрсеткіш позициясын орналастыруға арналған мәліметтер типі.

```
typedef OFF_T streamoff;
typedef POS_T streampos;
```

streamsize	<ios.h>

Жіберілген символдар санын немесе жұмыс істейтін буферлер санын сақтауға арналған мәліметтер типі.

```
typedef INT_T streamsize;
```

string	<string.h>

char мәліметтер типі үшін **basic_string** класы шаблонының специализациясы.

```
typedef basic_string<char> string;
```

stringbuf	<sstream.h>

char мәліметтер типі үшін **basic_stringbuf** класы шаблонының специализациясы.

```
typedef basic_stringbuf<char> stringbuf;
```

terminate_handler	<exception.h>

Программа жұмысын апаттық жағдайда аяқтағанда, **set_ terminate()** функциясы арқылы пайдаланылатын қателерді өңдейтін функция типі.

```
typedef void (*terminate handler)();
```

time_t	<time.h>

time_t типі уақытты сақтауға арналған. Ол **time()** функциясы арқылы қайтарылатын және **difftime()** функциясымен өңделетін календарлық уақытты сақтайды.

```
typedef long time_t;
```

TMP_MAX	<stdio.h>

TMP_MAX тұрақтысында файлдың әртүрлі уақытша аттарының ең үлкен сандары сақталады, олар программа жұмысының бір сеансы кезінде **tmpnam()** функциясы арқылы құрыла алады. **TMP_MAX** мәні 25-тен төмен бола алмайды.

struct tm	<time.h>

tm құрылымы **locatime()** және **gmtime()** функциялары пайдаланатын календарлық уақыттың барлық компоненттерін сақтайды.

```
struct tm {
   int tm_sec;  // минут басынан есептелетін секундтар
                   (0 ... 59)
   int tm_min;   // сағат басынан есептелетін минуттар
                   (0 ... 59)
   int tm_hour; // тун ортасына дейінгі сағаттар
                   (0 ... 23)
   int tm_mday; // ай күндері (0 ... 31)
   int tm_mon;  // жыл басынан есептелетін айлар
                   (0 ... 11)
   int tm_year; // 1900-ден кейінгі жыл
   int tm_yday; // жыл басынан есептелетін күн
                   (0 ... 365)
   int tm_isdst; // жазғы уақыт жалаушасы
};
```

UCHAR_MAX	<limits.h>

UCHAR_MAX тұрақтысында **unsigned char** типті шама қабылдай алатын ең үлкен мән сақталады. Оның мәні 127-ден аз болмауы тиіс.

UINT_MAX	<limits.h>

UINT_MAX тұрақтысында **unsigned int** типті шама қабылдай алатын ең үлкен мән сақталады. Оның мәні 65 535-тен аз болмауы тиіс.

ULONG_MAX	<limits.h>

ULONG_MAX тұрақтысында **unsigned long int** типті шама қабылдай алатын ең үлкен мән сақталады. Оның мәні 4 294 967 295-тен аз болмауы тиіс.

USHRT_MAX	<limits.h>

USHRT_MAX тұрақтысында **unsigned short int** типті шама қабылдай алатын ең үлкен мән сақталады. Оның мәні 65 535-тен аз болмауы тиіс.

unexpected_handler	<exception.h>

Аластауларды өңдейтін функция типі. Ол **set_unexpected()** функциясы арқылы функция ерекше жағдайлары тізімінде қарастырылмаған ерекше жағдайларды ұстау үшін пайдаланылады.

```
typedef void (*unexpected_handler) ();
```

va_arg, va_end, va_list	<stdarg.h>

Бұл шамалар аргументтер саны айнымалы функциялармен жұмыс істеу үшін қолданылады. Олар **<stdarg.h>** тақырыптық файлында сипатталған. **va_list** типі кезектегі аргументке нұсқауышты сақтауға арналған. **va_start** макросы сол нұсқауышты инициалдайды. **va_arg** макросы кезектегі аргумент мәнін қайтарады, оны әрбір шақыру **va_list** макросында сақталатын нұсқауыштың жылжып отыруын қамтамасыз етеді. Аргументтерді қарастырып болғаннан кейін, бірақ аргументтер саны айнымалы функциядан шыққанға дейін **va_end** макросын пайдалану керек.

```
typedef void_FAR *va_list;
```

WCHAR_MAX, WCHAR_MIN	<wchar.h>

WCHAR_MAX тұрақтысында **wchar_t** типті айнымалы қабылдай алатын ең үлкен мән сақталады.

WCHAR_MIN тұрақтысында **wchar_t** типті айнымалы қабылдай алатын ең кіші мән сақталады.

wtrans_t	<wctype.h>

Кеңейтілген жиын символдарын түрлендіру үшін қолданылатын қосалқы тип.

wctype_t	<wctype.h>

wctype_t мәліметтер құрылымында символдарды түрлендіруге және жіктеуге арналған ақпарат сақталады.

```
struct wctype_t {
   long tmin;        // Символдың ең кіші (минимал) коды
   long tmax;        // Символдың ең үлкен (максимал) коды
   unsigned char *index;  // Жіктеу индексі
   unsigned int *type;    // Типі
   long cmin;        // Түрлендіру үшін минимал код
   long cmax;        // Түрлендіру үшін максимал код
   long  *code;      // Түлендіру коды
};
```

WEOF	<wctype.h>

WEOF тұрақтысы файлдың біткенін хабарлайды (көпбайтты символдар үшін **EOF** синонимі).

werr, win, wlog, wout	<iostream.h>

Стандартты шеткері (перифериялық) құрылғылармен байланысқан кеңейтілген символдар мәліметтерінің ағымдары үшін ағымдық кластар объектілері.

Объект	Мәліметтер ағымы	Класс
win	stdin	wistream
wout	stdout	wostream
werr	stderr	wostream
wlog	stderr (буферленген)	wostream

wfilebuf	<fstream.h>

wchar_t мәліметтер типі үшін **basic_filebuf** класы шаблонының специализациясы.

```
typedef basic_filebuf <wchar_t> wfilebuf;
```

wifstream	<fstream.h>

wchar_t мәліметтер типі үшін **basic_ifstream** класы шаблонының специализациясы.

```
typedef basic_ifstream<wchar_t> wifstream;
```

wint_t	<wctype.h>

Кеңейтілген жиын символдарын сақтауға арналған бүтінсандық тип.

wios	<ios.h>

wchar_t мәліметтер типі үшін **basic_ios** класы шаблонының специализациясы.

```
typedef basic_ios<wchar_t> wios;
```

wistream	<istream.h>

wchar_t мәліметтер типі үшін **basic_istream** класы шаблонының специализациясы.

```
typedef basic_istream<wchar_t> wistream;
```

wistringstream	<sstream.h>

wchar_t мәліметтер типі үшін **basic_istringstream** класы шаблонының специализациясы.

```
typedef basic_istringstream<wchar_t> istringstream;
```

wofstream	<fstream.h>

wchar_t мәліметтер типі үшін **basic_ofstream** класы шаблонының специализациясы.

```
typedef basic_ofstream<wchar_t> wofstream;
```

wostream	<ostream.h>

wchar_t мәліметтер типі үшін **basic_ostream** класы шаблонының специализациясы.

```
typedef basic_ostream<wchar_t> wostream;
```

wostringstream	<sstream.h>

wchar_t мәліметтер типі үшін **basic_ostringstream** класы шаблонының специализациясы.

```
typedef basic_ostringstream<wchar_t> wostringstream;
```

wstreambuf	<streambuf.h>

wchar_t мәліметтер типі үшін **basic_streambuf** класы шаблонының специализациясы.

```
typedef basic_streambuf<wchar_t> wstreambuf;
```

wstreamoff, wstreampos	<ios.h>

Кеңейтілген жиын символдары мәліметтерінің ағымдарында үстіміздегі көрсеткішті орналастыруға арналған мәліметтер типі.

```
typedef OFF_T wstreamoff;
typedef POS_T wstreampos;
```

wstring	<string.h>

wchar_t мәліметтер типі үшін **basic_string** класы шаблонының специализациясы.

```
typedef basic_string<wchar_t> wstring;
```

wstringbuf	<sstream.h>

wchar_t мәліметтер типі үшін **basic_stringbuf** класы шаблонының специализациясы.

```
typedef basic_stringbuf<wchar_t> wstringbuf;
```

6-ҚОСЫМША

Стандартты кітапхана функциялары

Қосымша екі бөлімнен тұрады. Бірінші бөлімде функциялар өздері сипатталған тақырыптық файлдар бойынша топталған, екінші бөлімде функциялардың алфавиттік тізімі солардың сипатталуымен бірге берілген.

Тақырыптық файлдар құрамы

<ctype.h> (<cctype>) тақырыптық файлы — символдарды жіктеу және түрлендіру функциялары

isalnum	Символдың әріп немесе цифр екендігін тексереді
isalpha	Символдың әріп екендігін тексереді
iscntrl	Символдың басқарушы символ екендігін тексереді
isdigit	Символдың цифр екендігін тексереді
isgraph	Символдың көрінетіндігін тексереді
islower	Символдың төменгі регистр әрпі екендігін тексереді
isprint	Символдың басып шығарылатын символ екендігін тексереді
ispunct	Символдың пунктуация символы екендігін тексереді
isspace	Символдың шектеуіш символ екендігін тексереді
isupper	Символдың жоғарғы регистр әрпі екендігін тексереді
iswalnum	Символдың әріп немесе цифр екендігін тексереді
iswalpha	Символдың әріп екендігін тексереді
iswcntrl	Символдың басқарушы символ екендігін тексереді
iswctype	Көпбайтты символды тексереді
iswdigit	Символдың цифр екендігін тексереді
iswgraph	Символдың көрінетіндігін тексереді
iswlower	Символдың төменгі регистр әрпі екендігін тексереді
iswprint	Символдың басып шығарылатын символ екендігін тексереді
iswpunct	Символдың пунктуация символы екендігін тексереді
iswspace	Символдың шектеуіш символ екендігін тексереді
iswupper	Символдың жоғарғы регистр әрпі екендігін тексереді
iswxdigit	Символдың оналтылық санның символы екендігін тексереді
isxdigit	Символдың оналтылық санның символы екендігін тексереді
tolower	Төменгі регистрдегі символды қайтарады
toupper	Жоғарғы регистрдегі символды қайтарады
towlower	Төменгі регистрдегі символды қайтарады
towupper	Жоғарғы регистрдегі символды қайтарады

<locale.h> (<clocale>) тақырыптық файлы — локализация функциялары

localeconv	Жергілікті (локальді) ортаның орнату сипаттамасын қайтарады
setlocale	Жаңа локализацияны орнатады

<math.h> (<cmath>) тақырыптық файлы — математикалық функциялар

acos	Аргументтің арккосинусын қайтарады
asin	Аргументтің арксинусын қайтарады
atan	Аргументтің арктангенсін қайтарады
atan2	Аргументтер қатынасының арктангенсін қайтарады
ceil	Жоғарыға дөңгелектейді
cos	Косинусты есептейді
cosh	Гиперболалық косинусты есептейді
exp	Е санының дәрежесін қайтарады
fabs	Санның модулін қайтарады
floor	Төменге дөңгелектейді
fmod	х-ті у-ке бөлгендегі қалдықты қайтарады
frexp	Санның мантиссасы мен экспоненциалдық бөлігін ажыратады
ldexp	Мантисса мен дәреже көрсеткішін санға түрлендіреді
log	Натуралдық логарифмді есептейді
log10	Негізі 10 болып келетін логарифмді есептейді
modf	Санды бүтін және бөлшек бөліктерге бөледі
pow	Санды дәрежелейді
sin	Синусты есептейді
sinh	Гиперболалық синусты есептейді
sqrt	Квадрат түбірді есептейді
tan	Аргументтің тангенсін қайтарады
tanh	Аргументтің гиперболалық тангенсін қайтарады

<signal.h> (<csignal>) тақырыптық файлы – сигналдарды өңдеу функциялары

raise	Программаның орындалуын үзеді
signal	Сигналдарды өңдеуішті тіркейді

<stdio.h> (<cstdio>) тақырыптық файлы – С стилінде енгізу/шығару функциялары

clearerr	Ағыммен жұмыс істеу кезінде қателер жалаушасын тазартады
fclose	Енгізу/шығару ағымын жабады
feof	Файл соңына жеткендігін тексереді
ferror	Ағыммен жұмыс істеу кезінде қате кодын қайтарады
fflush	Буферден мәліметтер жазады
fgetc	Ағымнан символ оқиды

fgetpos	Файлдың ағымдағы позициясын қайтарады
fgets	Ағымнан n символды тіркеске оқиды
fgetwc	Ағымнан символ оқиды
fgetws	Ағымнан n символды тіркеске оқиды
fopen	Енгізу/шығару ағымын ашады
fprintf	Ағымға мәліметтер жазады
fputc	Ағымға символ жазады
fputs	Ағымға символдар тіркесін жазады
fputwc	Ағымға символ жазады
fputws	Ағымға символдар тіркесін жазады
fread	Енгізу ағымынан мәлімет оқиды
freopen	Енгізу/шығару ағымын ашады
fscanf	Файлдан форматталған мәліметтерді шығарады
fseek	Файлдың позициясын ағымдағы орнынан жылжытады
fsetpos	Файлдың ағымдағы позициясын оның басына қатысты жылжытады
ftell	Файлдың ағымдағы позициясын қайтарады
fwprintf	Ағымға мәліметтер жазады
fwrite	Берілген буферден ағымға мәліметтер жазады
fwscanf	Ағымнан мәліметтер оқып, оларды берілген адрестерге жазады
getc	Ағымнан символ оқиды
getchar	Стандартты енгізу құрылғысынан символ оқиды
gets	Пернетақтадан символдарды жаңа жол символы кездескенше оқиды
getwc	Ағымнан символ оқиды
getwchar	Пернетақтадан кезектегі символды қайтарады
perror	«s: error хабарлама» түріндегі тіркесті шығарады
printf	Белгілі бір форматтағы параметрлер тіркесін шығарады
putc	Ағымға символ жазады
putchar	Стандартты шығару құрылғысына символ шығарады
puts	Стандартты шығару құрылғысына тіркес шығарады
putwc	Ағымға символ жазады
putwchar	Стандартты шығару құрылғысына символ шығарады
remove	Файлды жояды
rename	Файлдың атын өзгертеді
rewind	Ағыммен жұмыс істеу кезінде қате жалаушаларын тазартып, файл басына көшеді
scanf	Белгілі бір форматтағы параметрлер тіркесін енгізеді
setbuf	Енгізу/шығару ағымын буферлеуді орнатады
setvbuf	Ағымдағы локализация негізінде тіркестерді түрлендіреді

sprintf	Белгілі бір форматтағы параметрлер тіркесін шығарады
sscanf	Тіркестен мәлімет енгізеді
swprintf	Белгілі бір форматтағы параметрлер тіркесін шығарады
swscanf	Тіркестен мәлімет енгізеді
tmpfile	Уақытша файлға екілік енгізу/шығару ағымын ашады
tmpnam	Файлдың бірегей атын жасайды
ungetc	Ағымға символ қайтарады
ungetwc	Ағымға символ қайтарады
vfprintf	Форматталған шығару тіркесін ағымға жібереді
vfwprintf	Форматталған шығару тіркесін ағымға жібереді
vprintf	Форматталған шығару тіркесін стандартты шығару ағымына жібереді
vsprintf	Белгілі бір форматтағы параметрлер тіркесін шығарады
vswprintf	Белгілі бір форматтағы параметрлер тіркесін шығарады
vwprintf	Форматталған шығару тіркесін стандартты шығару ағымына жібереді
wprintf	Белгілі бір форматтағы параметрлер тіркесін шығарады
wscanf	Белгілі бір форматтағы параметрлер тіркесін енгізеді

<stdlib.h> (<cstdlib>) тақырыптық файлы – С стиліндегі әртүрлі функциялар

abort	Программаның орындалуын үзеді
abs	Санның модулін қайтарады
atexit	Программа жұмысын аяқтау кезінде шақырылатын функцияны тіркейді
atof	Тіркесті нақты санға түрлендіреді
atoi	Тіркесті бүтін санға түрлендіреді
atol	Тіркесті ұзын бүтін санға түрлендіреді
bsearch	Сұрыпталған жиымнан элемент іздейді
calloc	Жады блогын бөледі
div	Қалдықпен бөледі
exit	Программаның орындалуын үзеді
free	Жады блогын босатады
getenv	Орта айнымалысы мәнін қайтарады
labs	Санның модулін қайтарады
ldiv	Қалдықпен бөледі
malloc	Жады блогын бөледі
mblen	Көпбайтты символдар мөлшерін анықтайды
mbstowcs	Көпбайтты символдар тіркесін **wchar_t** жиымына түрлендіреді
mbtowc	Көпбайтты символдарды **wchar_t** символдарына түрлендіреді
qsort	Берілген жиымды сұрыптайды

rand	Кездейсоқ сандарды береді
realloc	Бұрын бөлінген жады блогы мөлшерін өзгертеді
srand	Бастапқы жалған кездейсоқ санды орнатады
strtod	Тіркесті санға түрлендіреді
strtol	Санау жүйесін есепке ала отырып, тіркесті санға түрлендіреді
strtoul	Санау жүйесін есепке ала отырып, тіркесті санға түрлендіреді
system	Тіркесті ОЖ командалық процессорына береді
wcstod	Тіркесті санға түрлендіреді
wcstol	Санау жүйесін есепке ала отырып, тіркесті санға түрлендіреді
wcstombs	**wchar_t** символдарын көпбайтты символдарға түрлендіреді
wcstoul	Санау жүйесін есепке ала отырып, тіркесті санға түрлендіреді
wctomb	**wchar_t** символдарын көпбайтты символдарға түрлендіреді

<string.h> (<cstring>) тақырыптық файлы – С стиліндегі тіркестермен жұмыс істейтін функциялар

memchr	Символдың жады блогына алғашқы кіруін іздейді
memcmp	Жады блоктарын салыстырады
memcpy	Жады блогын көшіреді
memmove	Жады блогын жылжытады
memset	Жады блогын символмен толтырады
strcat	Тіркестерді қосады
strchr	Тіркестен символ іздейді
strcmp	Тіркестерді салыстырады
strcoll	Орнатылған локализацияны есепке ала отырып, тіркестерді салыстырады
strcpy	Бір тіркесті екінші тіркеске көшіреді
strcspn	Бір тіркестегі бір символды екінші тіркестен іздейді
strerror	Нұсқауышты қатенің сипатталуы бар тіркеске қайтарады
strlen	Тіркестің ұзындығын қайтарады
strncat	**n** символы бар бір тіркесті екінші тіркеске қосады
strncmp	**n** символы бар бір тіркесті екінші тіркеспен салыстырады
strncpy	Бір тіркестің алғашқы n символын екінші тіркеске көшіреді
strpbrk	Бір тіркес ішіндегі бір символды екінші тіркестен іздейді
strrchr	Тіркестен символ іздейді
Strspn	Бір тіркесте бар, ал екіншісінде жоқ символды іздейді
strstr	Тіркестегі ішкі тіркесті іздейді
strtok	Тіркестегі лексеманы ерекшелейді
strxfrm	Ағымдағы локализация негізінде тіркестерді түрлендіреді
wcscat	Тіркестерді қосады
wcschr	Тіркестен символ іздейді

wcscmp	Тіркестерді салыстырады
wcscoll	Орнатылған локализацияны есепке ала отырып, тіркестерді салыстырады
wcscpy	Бір тіркесті екінші тіркеске көшіреді
wcscspn	Бір тіркестегі бір символды екінші тіркестен іздейді
wcslen	Тіркестің ұзындығын қайтарады
wcsncat	n символы бар бір тіркесті екінші тіркеске қосады
wcsncmp	n символы бар бір тіркесті екінші тіркеспен салыстырады
wcsncpy	Бір тіркестің алғашқы n символын екінші тіркеске көшіреді
wcspbrk	Бір тіркес ішіндегі бір символды екінші тіркестен іздейді
wcsrchr	Тіркестен символ іздейді
wcsspn	Бір тіркесте бар, ал екіншісінде жоқ символды іздейді
wcsstr	Тіркестен ішкі тіркес іздейді
wcstok	Тіркестегі лексемаларды ерекшелейді
wcstrxfrm	Ағымдағы локализацияны есепке ала отырып, тіркестерді түрлендіреді
wmemcpy	Жады блогын көшіреді
wmemmove	Жады блогын жылжытады
wmemset	Жады блогын символмен толтырады

<time.h> (<ctime>) тақырыптық файлы – С стиліндегі мерзіммен (датамен) және уақытпен жұмыс істейтін функциялар

asctime	Дата/уақытты тіркеске түрлендіреді
clock	Программаның атқарылу уақытын қайтарады
ctime	Уақытты тіркеске түрлендіреді
difftime	Уақыттар айырмасын қайтарады
gmtime	**time_t** негізінде **tm** құрылымын инициалдайды
localtime	**time_t** негізінде **tm** құрылымын инициалдайды
mktime	Апта күні және жыл күні өрістерін толтырады
strftime	**fmt** форматындағы уақытты **tm** форматына түрлендіреді
time	Ағымдағы дата/уақытты **time_t** түрінде қайтарады
wcsftime	**fmt** форматындағы уақытты **tm** форматына түрлендіреді

<wchar.h> (<cwchar>) тақырыптық файлы — С стиліндегі көпбайтты символдармен жұмыс істейтін функциялар

btowc	Символды ұзын бүтін санға түрлендіреді
fwide	Ағым түрін анықтайды
wmemchr	Символдың жады блогына алғашқы кіруін іздейді
wmemcmp	Жады блоктарын салыстырады

Функциялардың алфавиттік тізімі

Төмендегі функциялар пайдаланатын тұрақтылар, типтер және құрылымдар 5-қосымшада сипатталған.

abort	Программа орындалуын үзеді	<stdlib.h>

```
void abort(void);
```

Бұл функция «abnormal program termination» хабарламасын шығарып, программаның жұмысын үзеді. Мұндайда буферлер жүктелмейді және ағымдар жабылмайды. Негізінде **exit** сөзін қолданған дұрыс. Функция шақыру программасына басқаруды қайтармайды, ол, келісім бойынша 3-ке тең аяқтау программасының кодын қайтарады.

abs	Санның модулін қайтарады	<stdlib.h>

```
int abs (int num)
```

Функция **num** санының абсолюттік мәнін қайтарады.

acos	Аргументтің арккосинусын қайтарады	<math.h>

```
double acos(double x)
```

Функция **x** аргументінің арккосинусы мәнін радианмен қайтарады.

asctime	Дата/уақытты тіркеске түрлендіреді	<math.h>

```
char *asctime(const struct tm *ctm);
```

Функция **tm** типті құрылымнан дата мен уақыттың символдық бейнесі болып табылатын тіркесті қайтарады.

asin	Аргументтің арксинусын қайтарады	<math.h>

```
double asin(double x);
```

Функция **x** аргументінің арксинусы мәнін радианмен қайтарады.

atan	Аргументтің арктангенсін қайтарады	<math.h>

```
double atan(double x);
```

Функция **x** аргументінің арктангенсі мәнін радианмен қайтарады.

atan2	Аргументтер қатынасының арктангенсін қайтарады	<math.h>

```
double atan2(double y,double x);
```

Функция **y/x** параметрлері қатынасының арктангенсі мәнін радианмен қайтарады.

atexit	Программа жұмысының аяқталуы кезінде шақырылатын функияны тіркейді	<stdlib.h>

```
void atexit(void (*func)(void));
```

Бұл функция программаның қалыпты аяқталуында шақырылатын **func** функиясын тіркейді. Оның параметрлері және қайтарылатын мәні болмауы керек. **atexit**-тің қайта шақырылуы функияның стегін қалыптастырады.

atof	Тіркесті нақты санға түрлендіреді	**<stdlib.h>**

```
double atof(const char *str);
```

Функция құрамында бүтін және нақты сандардың символдық бейнесі бар тіркесті соған сәйкес екі еселенген дәлдіктегі нақты санға түрлендіреді. Санның соңы ретінде санға жатқызыла алмайтын бірінші символ есептеледі. Егер түрлендіру жүзеге асырылмаса, 0.0 мәні қайтарылады.

atoi	Тіркесті бүтін санға түрлендіреді	**<stdlib.h>**

```
int atoi(const char *str);
```

Функция құрамында бүтін санның символдық бейнесі бар тіркесті соған сәйкес бүтін санға түрлендіреді. Санның соңы ретінде санға жатқызыла алмайтын бірінші символ есептеледі. Егер түрлендіру жүзеге асырылмаса, 0 мәні қайтарылады.

atol	Тіркесті ұзын бүтін санға түрлендіреді	**<stdlib.h>**

```
long atol(const char *str);
```

Функция құрамында бүтін санның символдық бейнесі бар тіркесті соған сәйкес ұзын бүтін санға түрлендіреді. Санның соңы ретінде санға жатқызыла алмайтын бірінші символ есептеледі. Егер түрлендіру жүзеге асырылмаса, 0L мәні қайтарылады.

bsearch	Сұрыпталған жиымнан элемент іздейді	**<stdlib.h>**

```
void bsearch(const void *key, const void *buf, size_t num,
   size_t size, int e(*compare)(const void *, const void *));
```

Бұл функция **buf** реттелген жиымындағы **key** нұсқап тұрған элементпен сәйкес келетін элементті дихотомикалық түрде іздейді де, нұсқауышты табылған элементке қайтарады. Мұндағы **num – buf** жиымындағы элементтер саны, **size** – жиым элементінің байтпен берілген көлемі.

compare() функциясы жиымның екі элементін салыстырып, -1, 0 немесе 1 деген үш мәннің біреуін қайтаруы керек. Бұл мәндер сәйкесінше бірінші элементтің екінші элементпен салыстырғанда кіші, тең немесе үлкен болуына байланысты қалыптасады. **compare** функциясының екі параметрі де **void*** ретінде анықталғанына қарамастан, функцияның нақты параметрлері болып элементтерге нұсқауыштар есептеледі. Функция оларды атаусыздандыру алдында (адрес бойынша мәндерін алу арқылы) параметрлеріне сәйкес типке келтіруі тиіс.

Егер **bsearch** функциясы ізделінген элементті тапса, ол нұсқауышты соған қайтарады, ал кері жағдайда **NULL** мәні қайтарылады.

btowc	Символды ұзын бүтін санға түрлендіреді	<wchar.h>

```
wint_t btowc(int c);
```

Егер **c EOF**-қа тең болса, онда функция **WEOF** мәнін қайтарады, әйтпесе **c** символын **mbrtows** функциясы тәрізді көпбайттық формаға түрлендіреді. Егер түрлендіру жүзеге аспаса, **WEOF** мәнін қайтарады.

calloc	Жады блогын бөледі	<stdlib.h>

```
void *calloc(size_t num, size_t size);
```

Функция жады блогын (әрқайсысы **size** байттан **num** элемент) бөледі де, нұсқауышты соған қайтарады. Алынған жиымның әрбір элементі нөлдік мәнмен инициалданады (**malloc** функциясына қарағанда). Егер жады блогын ерекшелеу іске аспаса, **NULL** қайтарылады.

ceil	Жоғарыға дөңгелектейді	<math.h>

```
double ceil(double x);
```

Функция **x** нақты мәнін одан үлкен ең жақын бүтін санға дейін дөңгелектейді де, оны нақты сан түрінде қайтарады.

clearerr	Ағыммен жұмыс істеу кезінде қателер жалаушасын тазартады	<stdio.h>

```
void clearerr(FILE *f)
```

Функция **f** ағымы үшін қателер жалаушасын тазартады (**clearerr**, **fseek**, **fsetpos** немесе **rewind** функцияларының бірі шақырылмайынша, қателер жалаушалары түсірілмейді).

clock	Программаның атқарылу уақытын қайтарады	<time.h>

```
clock_t clock(void);
```

Функция программаның атқарылу уақытын **clock_t** типті мән түрінде қайтарады (бұл мән **time.h** файлында анықталған **CLOCKS_PER_SEC** тұрақтысына бөлу арқылы секундқа өзгертілуі мүмкін).

cos	Косинусты есептейді	<math.h>

```
double cos(double x);
```

Функция **x** радианға тең бұрыштың косинусының мәнін қайтарады.

cosh	Гиперболалық косинусты есептейді	<math.h>

```
double cosh(double x);
```

Функция **x** радианға тең бұрыштың гиперболалық косинусының мәнін қайтарады. Егер функцияның мәні көзделген диапазоннан тысқары шығатын болса, онда функция **HUGEVAL** мәнін қайтарады, ал **errno** ауқымды (глобалды) айнымалысы **ERANGE** мәніне ие болады.

ctime	Уақытты тіркеске түрлендіреді	<time.h>

```
char *ctime(time_t *t);
```

Функция **time_t** типті параметрдің дата мен уақытының символдық бейнесін көрсететін тіркесті қайтарады.

difftime	Уақыттар айырмасын қайтарады	<time.h>

```
double difftime(time_t t2,time_t t1);
```

Функция уақыттар айырмасын (**t2-t1**) секундпен қайтарады.

div	Қалдықпен бөледі	<stdlib.h>

```
div_t div(int numerator, denominator);
```

Функция **numerator** бүтінін **denominator** бүтініне бөледі де, нәтижесін бөлінді мен қалдықтан тұратын құрылым түрінде қайтарады:

```
struct div_t{
   int quot; // бөлінді
   int rem;  // қалдық
};
```

exit	Программаның орындалуын үзеді	<stdlib.h>

```
void exit(int exit_code);
```

Функция барлық буферлерді керекті файлдарға жазып, ағымдарды жауып, **atexit** көмегімен тіркелген функцияларды тіркелуге кері бағытта шақыра отырып, программаның қалыпты аяқталуын жүзеге асырады.

exp	Е санының дәрежесін қайтарады	<math.h>

```
double exp(double x);
```

Функция **e** санын көрсетілген дәрежеге шығару нәтижесін қайтарады.

fabs	Санның модулін қайтарады	<math.h>

```
double fabs(double num);
```

Функция **num** санының абсолюттік мәнін қайтарады.

fclose	Енгізу/шығару ағымын жабады	<stdio.h>

```
int fclose(FILE *f);
```

Функция **f** енгізу/шығару ағымын жабады.

feof	Файл соңына жеткендігін тексереді	<stdio.h>

```
int feof(FILE *f);
```

Егер файл соңына жетсек, функция **EOF** мәнін немесе 0-ден өзгеше мәнді қайтарады, кері жағдайда 0 мәнін қайтарады.

ferror	Ағыммен жұмыс істеу кезінде қате кодын қайтарады	<stdio.h>

```
int ferror(FILE *f);
```

Функция қате кодын білдіретін бүтін мән қайтарады; 0 – қатенің жоқ болғаны.

fflush	Буферден мәліметтер жазады	<stdio.h>

```
int fflush(FILE *f);
```

Функция физикалық құрылғыға жазу үшін мәліметтерді бірден жөнелту арқылы шығару буферін тазалайды. Сәтті аяқталған жағдайда 0 мәнін қайтарады, кері жағдайда **EOF** мәні қайтарылады.

fgetc	Ағымнан символ оқиды	<stdio.h>

```
int fgetc(FILE *f);
```

Функция **f** ағымынан **int** формасындағы кезекті символды қайтарады. Егер символ оқылмайтын болса, онда **EOF** мәні қайтарылады.

fgetpos	Файлдың ағымдағы позициясын қайтарады	<stdio.h>

```
int fgetpos(FILE *f, fpos_t *pos);
```

Функция **f** ағымымен байланысты файлдағы ағымдық позицияны қайтарады да, **pos** адресі бойынша мәнді көшіреді. Бұл мәнді кейінірек **fsetpos** функциясы пайдалана алады. Қайтарылатын мән **fpos_t** типінде болады.

fgets	Ағымнан n символды тіркеске оқиды	<stdio.h>

```
char *fgets(char *s, int n. FILE *f);
```

Функция **f** ағымынан **s** тіркесіне **n-1** байттан аспайтын мәлімет оқиды, мұнда ол жаңа жол символын немесе файл соңы белгісін кездестірсе, оқуды тоқтатады. Оқу кезінде жаңа жол символы алынып тасталмайды, ол тек жол соңына орналастырылады. Оқылған тіркес тек оның соңын шектеу символымен (‘\0’) толықтырылады. Қате табылған жағдайда немесе файлдың соңына жеткенде, **NULL** мәні, ал кері жағдайда – **s** тіркесіне нұсқауыш қайтарылады.

fgetwc	Ағымнан символ оқиды	<stdio.h>

```
wint_t fgetwc(FILE *f);
```

Функция **f** ағымынан **wint_t** формасындағы кезекті символды қайтарады. Егер символ оқылмаса, **WEOF** мәні қайтарылады. Ол көпбайтты символдарға арналған **fgetc** функциясының баламасы болып табылады.

fgetws	Ағымнан n символды тіркеске оқиды	<stdio.h>

```
wchar_t *fgetws(wchar_t *s, int n, FILE *f);
```

Функция **f** ағымынан **s** тіркесіне **n-1** байттан аспайтын мәлімет оқиды, мұнда ол жаңа жол символын немесе файл соңы белгісін кездестірсе, оқуды тоқтатады. Оқу кезінде жаңа жол символы алынып тасталмайды, ол тек жол соңына орналастырылады. Оқылған тіркес тек оның соңын шектеу символы-

мен (‘\0’) толықтырылады. Қате табылған жағдайда немесе файлдың соңына жеткенде, **NULL** мәні, ал кері жағдайда – **s** тіркесіне нұсқауыш қайтарылады. Ол көпбайтты символдарға арналған **fgets** функциясының баламасы болып табылады.

floor	Төменге дөңгелектейді	**<math.h>**

```
double floor(double x);
```

Функция нақты **x** мәнін ең жақын төменгі бүтінге дөңгелектейді де, оны нақты сан түрінде қайтарады.

fmod	x-ті y-ке бөлгендегі қалдықты қайтарады	**<math.h>**

```
double fmod (double x, double y);
```

Функция **x**-ті **y**-ке бөлгендегі қалдықты қайтарады. **%** операциясы тәрізді, тек нақты сандармен жұмыс істейді.

fopen	Енгізу/шығару ағымын ашады	**<stdio.h>**

```
FILE *fopen(const char *fname, const char *mode);
```

Функция **mode** тіркесімен көрсетілген режимде жұмыс істеу үшін **fname** атты файлды ашады. Файл сәтті ашылған жағдайда ағымға нұсқауыш қайтарылады (осылайша алынған ағым файлмен байланысады), кері жағдайда **NULL** мәні қайтарылады.

fprintf	Ағымға мәліметтер жазады	**<stdio.h>**

```
int fprintf(FILE *f, const char *fmt,…);
```

Функция **f** ағымына тізімдері көпнүктемен (...) берілген айнымалыларды **fmt** тіркесінде көрсетілген форматта жазады. Жазылған символдар санын қайтарады.

fputc	Ағымға символ жазады	**<stdio.h>**

```
int fputc(int ch, FILE *f);
```

Функция **f** ағымына **ch** символын жазады. Қате болған жағдайда **EOF** мәнін, әйтпесе – жазылған символды қайтарады.

fputs	Ағымға символдар тіркесін жазады	**<stdio.h>**

```
int fputs(const char *s, FILE *f);
```

Функция **f** ағымына **s** символдар тіркесін жазады. Тіркес соңы символы файлға жазылмайды. Қате болған жағдайда **EOF** мәнін, әйтпесе – теріс емес сан мәнін қайтарады.

fputwc	Ағымға символ жазады	**<stdio.h>**

```
w_int fputwc(w_int ch, FILE *f);
```

Функция **f** ағымына **ch** символын жазады. Қате болған жағдайда **EOF** мәнін, әйтпесе – жазылған символды қайтарады. Ол көпбайтты символдарға арналған **fputc** функциясының баламасы болып табылады.

fputws	Ағымға символдар тіркесін жазады	<stdio.h>

```
int fputws(const wchar_t *s, FILE *f);
```

Функция **f** ағымына **s** символдар тіркесін жазады. Тіркес соңы символы файлға жазылмайды. Қате болған жағдайда **WEOF** мәнін, әйтпесе – теріс емес сан мәнін қайтарады. Ол көпбайтты символдарға арналған **fputs** функциясының баламасы болып табылады.

free	Жады блогын босатады	<stdio.h>

```
void free(void *pointer);
```

Функция бұрынырақ **malloc**, **calloc** немесе **realloc** функцияларының бірінің көмегімен алынған жады блогын босатады.

fread	Енгізу ағымынан мәлімет оқиды	<stdio.h>

```
size_t fread(void *buffer,size_t size,size_t count,
                                          FILE *stream);
```

Функция **stream** ағымынан **buffer** нұсқауышы арқылы берілген аймаққа **size** байттарынан тұратын **count** элементтерін оқиды. Егер мәлімет оқылып жатқан кезде қате шықса немесе файлдың соңы кездессе, онда бұл функция оқылған элементтердің ішінен **count**-тан кіші болып келетіндерінің санын қайтарады.

freopen	Енгізу/шығару ағымын ашады	<stdio.h>

```
FILE *freopen(const char *fname, const char *mode, FILE *f);
```

Бұл функция **fopen** функциясы тәрізді жұмыс істейді, бірақ **f** ағымы бұрыннан ашық болса, оны алдын ала жабады.

frexp	Санның мантиссасы мен экспоненциалдық бөлігін ажыратады	<stdio.h>

```
double frexp(double x, int *expptr);
```

Функция **x** санының мантиссасы мен экспоненциалдық бөлігін ажыратады. Мантисса мәнін қайтарады да, экспоненциалдық бөлігін **expptr** адресі бойынша көшіреді.

fscanf	Файлдан форматталған мәліметтерді шығарады	<stdio.h>

```
int fscanf(FILE *f, const char *fmt [,par1, par2, ...]);
```

Функция **f** файлынан **fmt** тіркесімен анықталған форматтағы **par1**, **par2** және т.б. параметрлер тіркесін енгізеді. Мән меншіктелген айнымалылар санын қайтарады.

fseek	Файлдың позициясын ағымдағы орнынан жылжытады	<stdio.h>

```
int fseek(FILE *f, long off, int org);
```

Функция **f** ағымымен байланысқан файлдағы ағымдағы позицияны **off** позициясына сәйкес жылжытады; **off** позициясы **org** мәнінен бастап есептеледі, ол **<stdio.h>** файлында анықталған үш тұрақтының біреуіне тең болуы керек, олар:

SEEK_CUR – нұсқауыштың ағымдағы позициясынан есептеледі;

SEEK_END – файлдың соңынан есептеледі;

SEEK_SET – файлдың басынан есептеледі.

fsetpos	Файлдың ағымдағы позициясын оның басына қатысты жылжытады	**<stdio.h>**

```
int fsetpos(FILE *f, const fpos_t *pos);
```

Функция **f** ағымымен байланысқан файлдың ағымдағы позициясын **fgetpos** функциясы арқылы алдын ала алынған ***pos** позициясына жылжытады.

ftell	Файлдың ағымдағы позициясын қайтарады	**<stdio.h>**

```
long int ftell(FILE *f);
```

Функция **f** ағымымен байланысқан файлдың ағымдағы позициясын ұзын бүтін сан ретінде қайтарады.

fwide	Ағым түрін анықтайды	**<wchar.h>**

```
int fwide(FILE *stream, int mode);
```

Функция **mode** режиміне байланысты ағымның түрін анықтайды – байтқа бағытталған ағым (**mode < 0**) немесе көпбайтты символдардан тұратын ағым (**mode > 0**). ағымның ағымдағы түрін қайтарады.

fwprintf	Ағымға мәліметтер жазады	**<stdio.h>**

```
int fwprintf(FILE *f, const wchar_t *fmt, ...);
```

Функция **f** ағымына **fmt** тіркесінде көрсетілген форматта тізімі көпнүктемен берілген айнымалыларды жазады. Жазылған символдардың санын қайтарады. Көпбайтты символдарға арналған **fprintf** функциясының баламасы болып табылады.

fwrite	Берілген буферден ағымға мәліметтер жазады	**<stdio.h>**

```
size_t fwrite(const void *p,size_t size,
                                    size_t n, FILE *f);
```

Функция **f** ағымына **p** нұсқауышымен берілген буферден ұзындығы **size** байттан тұратын **n** элементті жазады. Жазылған элементтер санын қайтарады.

fwscanf	Ағымнан мәліметтер оқып, оларды берілген адрестерге жазады	<stdio.h>

```
int fwscanf (FILE *f, const wchar_t *fmt, ...);
```

Функция **f** ағымынан **fmt** тіркесінде көрсетілген форматқа сәйкес адрестер тізімі көпнүктемен берілген айнымалыларды оқиды. Оқылған элементтер санын қайтарады, ал олар сәтсіз оқылған жағдайда **WEOF** мәнін қайтарады. Көпбайтты символдарға арналған **fscanf** функциясының баламасы болып табылады.

getc	Ағымнан символ оқиды	<stdio.h>

```
int getc(FILE *f);
```

Функция **f** ағымынан **int** формасындағы кезекті символды қайтарады. Егер символ оқылмайтын болса, онда **EOF** мәнін қайтарады.

getchar	Стандартты енгізу құрылғысынан символ оқиды	<stdio.h>

```
int getchar(void);
```

Функция стандартты енгізу құрылғысынан **int** формасындағы кезекті символды қайтарады. Егер символ оқылмайтын болса, онда **EOF** мәнін қайтарады.

getenv	Орта айнымалысы мәнін қайтарады	<stdio.h>

```
int getenv(const char *var_name);
```

Функция көрсетілген орта айнымалысының сипаттамасы бар тіркеске нұсқауыш қайтарады.

gets	Пернетақтадан символдарды жаңа жол символы кездескенше оқиды	<stdio.h>

```
char *gets (char *s);
```

Функция пернетақтадан символдарды жаңа жол символы кездескенше оқиды да, оларды **s** тіркесіне орналастырады (жаңа жол символының өзі тіркеске енгізілмейді). **s** тіркесіне нұсқауыш қайтарады.

getwc	Ағымнан символ оқиды	<stdio.h>

```
wint_t getws(FILE *f);
```

Функция **f** ағымынан **wint_t** формасындағы кезекті символды қайтарады. Егер символды оқу мүмкін болмаса, **WEOF** мәні қайтарылады. Көпбайтты символдарға арналған **getc** функциясының баламасы болып табылады.

getwchar	Пернетақтадан кезектегі символды қайтарады	<stdio.h>

```
wint_t getwchar(void);
```

Функция стандартты енгізу ағымынан (әдетте пернетақтадан) кезекті символды қайтарады. Көпбайтты символдарға арналған **getchar** функциясының баламасы болып табылады.

gmtime	**time_t** негізінде **tm** құрылымын инициалдайды	**<time.h>**

```
const struct tm *gmtime(const time_t *t);
```

Функция **t** ортасынан дата/уақыт негізінде инициалданған **tm** типті уақытша құрылымға нұсқауыш қайтарады. Уақыт Гринвич бойынша орнатылады.

isalnum	Символдың әріп немесе цифр екендігін тексереді	**<ctype.h>**

```
int isalnum(int ch);
```

Функция **ch** параметрінің кіші байтын белгілеп алады да, егер **ch** символы әріп немесе цифр болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады.

isalpha	Символдың әріп екендігін тексереді	**<ctype.h>**

```
int isalpha(int ch);
```

Функция **ch** параметрінің кіші байтын белгілеп алады да, егер **ch** символы әріп болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады.

iscntrl	Символдың басқарушы символ екендігін тексереді	**<ctype.h>**

```
int iscntrl(int ch);
```

Функция **ch** параметрінің кіші байтын белгілеп алады да, егер **ch** символы басқарушы символ (**line feed**, **del**, табуляция және т.б., көбісі 0x01–0x1F диапазонында орналасқан символдар (ASCII кодтары үшін)) болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады.

isdigit	Символдың цифр екендігін тексереді	**<ctype.h>**

```
int isdigit (int ch);
```

Функция **ch** параметрінің кіші байтын белгілеп алады да, егер **ch** символы цифр болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады.

isgraph	Символдың көрінетіндігін тексереді	**<ctype.h>**

```
int isgraph(int ch);
```

Функция **ch** параметрінің кіші байтын белгілеп алады да, егер **ch** символы экранда көрінетін (яғни ол босорын, табуляция, т.с.с. символдар емес) болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады.

islower	Символдың төменгі регистр әрпі екендігін тексереді	<ctype.h>

```
int islower (int ch);
```

Функция **ch** параметрінің кіші байтын белгілеп алады да, егер **ch** символы төменгі регистр әрпі болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады.

isprint	Символдың басып шығарылатын символ екендігін тексереді	<ctype.h>

```
int isprint(int ch);
```

Функция **ch** параметрінің кіші байтын белгілеп алады да, егер **ch** символы басып шығарылатын символ (**isgraph+**босорын) болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады.

ispunct	Символдың пунктуация символы екендігін тексереді	<ctype.h>

```
int ispunct(int ch);
```

Функция **ch** параметрінің кіші байтын белгілеп алады да, егер **ch** символы пунктуация символы (яғни басылып шығарылатын символ, ол әріп, цифр, босорын емес) болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады.

isspace	Символдың шектеуіш символ екендігін тексереді	<ctype.h>

```
int isspace(int ch);
```

Функция **ch** параметрінің кіші байтын белгілеп алады да, егер **ch** символы бос орын немесе табуляция, немесе жаңа жол, немесе жаңа бет символы (формат ауысуының символы) болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады.

isupper	Символдың жоғарғы регистр әрпі екендігін тексереді	<ctype.h>

```
int isupper(int ch);
```

Функция **ch** параметрінің кіші байтын белгілеп алады да, егер **ch** символы жоғарғы регистр әрпі болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады.

iswalnum	Символдың әріп немесе цифр екендігін тексереді	<ctype.h>

```
int iswalnum(wint_t ch);
```

Функция **ch** символы әріп немесе цифр болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады. Көпбайтты символдарға арналған **isalnum** функциясының баламасы болып табылады.

iswalpha	Символдың әріп екендігін тексереді	<ctype.h>

```
int iswalpha(wint_t ch);
```

Функция **ch** символы әріп болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады. Көпбайтты символдарға арналған **isalpha** функциясының баламасы болып табылады.

iswcntrl	Символдың басқарушы символ екендігін тексереді	<ctype.h>

```
int iswcntrl(wint_t ch);
```

Функция **ch** символы басқарушы символ (**line feed**, **del**, табуляция және т.б., көбісінің ASCII кодтары 0x01–0xlF диапазонында орналасқан басқарушы символдар үшін) болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады. Көпбайтты символдарға арналған **iscntrl** функциясының баламасы болып табылады.

iswctype	Көпбайтты символды тексереді	<ctype.h>

```
int iswctype(wint_t c, wctype_t desc);
```

Функция егер символдың **desc** қасиеті болса, нөлге тең емес мән қайтарады, ал кері жағдайда қайтарылатын мән нөлге тең болады.

iswdigit	Символдың цифр екендігін тексереді	<ctype.h>

```
int iswdigit(wint_t ch);
```

Функция **ch** символы цифр болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады. Көпбайтты символдарға арналған **isdigit** функциясының баламасы болып табылады.

iswgraph	Символдың көрінетіндігін тексереді	<ctype.h>

```
int iswgraph(wint_t ch);
```

Функция **ch** символы көрінетін символ (яғни ол босорын, табуляция және т.б. символдары болмаса) болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады. Көпбайтты символдарға арналған **isgraph** функциясының баламасы болып табылады.

iswlower	Символдың төменгі регистр әрпі екендігін тексереді	<ctype.h>

```
int iswlower(wint_t ch);
```

Функция **ch** символы төменгі регистрдің әрпі болған жағдайда **true** мәнін қайтарады, ал кері жағдайда **false** болады. Көпбайтты символдарға арналған **islower** функциясының баламасы болып табылады.

iswprint	Символдың басып шығарылатын символ екендігін тексереді	<ctype.h>

```
int iswprint(wint_t ch);
```

Функция **ch** символы басып шығарылатын символ (**iswgraph +** бос орын) болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады. Көпбайтты символдарға арналған **iswprint** функциясының баламасы болып табылады.

iswpunct	Символдың пунктуация символы екендігін тексереді	**<ctype.h>**

```
int iswpunct(wint_t ch);
```

Функция **ch** символы пунктуация символы (яғни басылып шығарылатын символ, бірақ әріп те, сан да, босорын да болмайтын) болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады. Көпбайтты символдарға арналған **ispunct** функциясының баламасы болып табылады.

iswspace	Символдың шектеуіш символ екендігін тексереді	**<ctype.h>**

```
int iswspace(wint_t ch);
```

Функция **ch** символы босорын немесе табуляция, жаңа жол немесе жаңа бет символы (форматты ауыстыру символы) болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады. Көпбайтты символдарға арналған **isspace** функциясының баламасы болып табылады.

iswupper	Символдың жоғарғы регистр әрпі екендігін тексереді	**<ctype.h>**

```
int iswupper(wint_t ch);
```

Функция **ch** символы жоғарғы регистрдің әрпі болған жағдайда **true** мәнін қайтарады, ал кері жағдайда **false** болады. Көпбайтты символдарға арналған **isupper** функциясының баламасы болып табылады.

iswxdigit	Символдың оналтылық санның символы екендігін тексереді	**<ctype.h>**

```
int iswxdigit(wint_t ch);
```

Функция **ch** символы оналтылық санның символы (цифрлар және де төменгі немесе жоғарғы регистрлердегі A-дан F-қа дейінгі әріптер) болған жағдайда, **true** мәнін қайтарады, ал кері жағдайда **false** болады. Көпбайтты символдарға арналған **isxdigit** функциясының баламасы болып табылады.

isxdigit	Символдың оналтылық санның символы екендігін тексереді	**<ctype.h>**

```
int isxdigit(int ch);
```

Функция **ch** параметрінің кіші байтын белгілеп алып, егер **ch** символы оналтылық санның символы (цифрлар және төменгі немесе жоғарғы регистрлердегі A-дан F-қа дейінгі әріптер) болса, **true** мәнін қайтарады, ал кері жағдайда **false** болады.

labs	Санның модулін қайтарады	<stdlib.h>

```
long int labs(long int num);
```

Функция **num** санының абсолюттік мәнін қайтарады.

ldexp	Мантисса мен дәреже көрсеткішін санға түрлендіреді	<math.h>

```
double ldexp(double x, int exp);
```

Функция **x** мантиссасын және **exp** дәреже көрсеткішін қабылдайды да, мантиссаның 2-нің дәреже көрсеткішіне дәрежеленген санға көбейтіндісіне тең болатын санды қайтарады. **frexp** функциясына кері функция болып табылады.

ldiv	Қалдықпен бөледі	<stdlib.h>

```
ldiv_t ldiv(long numerator, long denominator);
```

Функция ұзын бүтін типтегі **numerator** санын **denominator** ұзын бүтін санына бөледі де, бөлінді мен қалдықтан құралған құрылымды нәтиже ретінде қайтарады.

localeconv	Жергілікті (локальді) ортаның орнату сипаттамасын қайтарады	<locale.h>

```
struct lconv *localeconv(void);
```

Функция жергілікті ортаның ағымдағы орнатуларының айқын сипаттамасы бар құрылымды қайтарады, яғни бұл елге тәуелді болатын ақпарат.

localtime	**time_t** негізінде **tm** құрылымын инициалдайды	<time.h>

```
struct tm localtime(time_t *t);
```

Функция **t** ортасынан алынған дата/уақыт негізінде инициалданған **tm** типті уақытша құрылымға нұсқауыш қайтарады. Жергілікті уақыт орнатылады (уақыт Гринвич бойынша орнатылған **gmtime** функциясына қарағанда өзгеше).

log	Натуралдық логарифмді есептейді	<math.h>

```
double log(double x);
```

Функция **x**-тің натуралдық логарифмінің мәнін қайтарады.

log10	Негізі 10 болып келетін логарифмді есептейді	<math.h>

```
double log(double x);
```

Функция 10-дық негіз бойынша **x**-тің натурал логарифмінің мәнін қайтарады.

malloc	Жады блогын бөледі	<stdlib.h>

```
void *malloc(size_t size);
```

Функция көлемі **size** байт жады блогын бөледі де, оған нұсқауышты қайтарады. Сәтсіз жағдайда **NULL** мәні қайтарылатын болады. **malloc** орнына **new** операциясын қолдану тиімді болып саналады.

mblen	Көпбайтты символдар мөлшерін анықтайды	<stdlib.h>

```
int mblen(const char *s, size_t n);
```

Функция, егер **s NULL** болмаса, **s** символының көлемін қайтарады.

mbstowcs	Көпбайтты символдар тіркесін **wchar_t** жиымына түрлендіреді	<stdlib.h>

```
size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);
```

Функция көпбайтты символдардың тіркесін **wchar_t**-дан жиымға түрлендіреді. Егер рұқсат етілмеген символ кездессе, функция -1-ді қайтарады. Кері жағдайда, функция түрлендірілген символдардың санын қайтарады.

mbtowc	Көпбайтты символдарды **wchar_t** символдарына түрлендіреді	<stdlib.h>

```
int mbtowc(wchar_t *pwc, const char *s, size_t n);
```

Функция көпбайтты символды **wchart** символдарына түрлендіреді. Егер **s** көпбайтты емес символға нұсқап тұрса, функция -1-ді қайтарады. Егер **s NULL**-ға нұсқаса, қайтарымды мән 0-ге тең болады. Кері жағдайда, функция символдың ұзындығын байтпен қайтарады.

memchr	Символдың жады блогына алғашқы кіруін іздейді	<string.h>

```
void *memchr(const void *p, int ch, size_t n);
```

Функция нұсқауышты байттың алғашқы кірісіне қайтарады, ол **p** жады блогында, ұзындығы **n** болып келген **ch** аргументінің кіші байтымен сипатталған.

memcmp	Жады блоктарын салыстырады	<string.h>

```
int memcmp(const void *p1, const void *p2, size_t n);
```

Функция компьютер жадындағы екі блокты салыстырып, нөлден кіші, нөлге тең немесе нөлден үлкен үш түрлі мән қайтара алады – олар **stremp** функцияның қайтаратын кодтарына ұқсас болып келеді.

memcpy	Жады блогын көшіреді	<string.h>

```
int *memcpy(void *dest, const void *src, size_t n);
```

Функция ұзындығы **n** байт болып келетін жады блогын **src** адресінен **dest** адресіне көшіреді.

memmove	Жады блогын жылжытады	<string.h>

```
int *memmove(void *dest, const void *src, size_t n);
```

Функция **memcpy** функциясы тәрізді, бірақ **src** және **dest** адрестері қабаттасып та кете береді.

memset	Жады блогын символмен толтырады	<string.h>

```
int *memset(const void *p,int ch,size_t n);
```

Функция **ch** төменгі байтынан алынған символмен блокты толтырады.

mktime	Апта күні және жыл күні өрістерін толтырады	<time.h>

```
struct tm mktime(time_t *t);
```

Функция **t** ортасында бар ақпарат негізінде **tm** құрылымының апта күні және жыл күні өрістерін толтырады.

modf	Санды бүтін және бөлшек бөліктерге бөледі	<math.h>

```
double modf(double x, double *intptr);
```

Функция **x**-ті бүтін және бөлшек бөлікке бөледі, бірақ бөлшек бөлігін қайтарып, ал бүтін бөлігін **intptr** нұсқауышы анықтайтын адреске орналастырады.

perror	«s: error хабарлама» түріндегі тіркесті шығарады	<stdio.h>

```
void perror(const char *s);
```

Функция «**s:** error хабарлама» түріндегі тіркесті баспаға шығарады, ондағы хабарлама **errno** ауқымды айнымалы мәні негізінде анықталады. Егер **s** мәні **NULL** болса, онда тек қате жайлы хабарлама басылып шығады. Нәтиже әдетте монитор болып белгіленген қателердің **stderr** стандартты ағымында шығарылады.

pow	Санды дәрежелейді	<math.h>

```
double pow(double x, double y);
```

Функция **x**-тің **y** дәрежеге шығарылған мәнін есептейді.

printf	Белгілі бір форматтағы параметрлер тіркесін шығарады	<stdio.h>

```
int printf (const char *fmt, ...);
```

Функция стандартты нәтижелік құрылғыға тізімде көпнүктемен белгіленіп көрсетілген айнымалылардың мәндерін **format** тіркесімен анықталған форматта шығарады. Форматтың спецификациялары 2-қосымшада көрсетілген.

putc	Ағымға символ жазады	<stdio.h>

```
int putc(int ch, FILE *f);
```

Функция **ch** символын **f** ағымына жазады. Қате кеткен жағдайда – **EOF** мәнін, әйтпесе жазылған символды қайтарады. Толығымен **fputc** функциясы тәрізді жұмыс істейді.

putchar	Стандартты шығару құрылғысына символ шығарады	**<stdio.h>**

```
int putchar(int ch);
```

Функция **ch** символын стандартты шығару құрылғысына шығарады. Егер мәлімет дұрыс шығарылса, **ch** мәні, әйтпесе – **EOF** мәні қайтарылады.

puts	Стандартты шығару құрылғысына тіркес шығарады	**<stdio.h>**

```
int puts(char *s);
```

Функция **s** тіркесін соңына жаңа жол символын енгізе отырып, стандартты шығару құрылғысына шығарады. Егер мәлімет дұрыс шығарылса, теріс емес мән, әйтпесе – **EOF** мәні қайтарылады.

putwc	Ағымға символ жазады	**<stdio.h>**

```
wint_t putws(wint_t ch, FILE *f);
```

Функция **ch** символын **f** ағымына жазады. Егер қате кетсе, **WEOF** мәні, әйтпесе – жазылған символ қайтарылады. Бұл көпбайтты символдарға арналған **putc** функциясының баламасы болып табылады.

putwchar	Стандартты шығару құрылғысына символ шығарады	**<stdio.h>**

```
wint_t putwchar(wint_t ch);
```

Функция **ch** символын стандартты шығару құрылғысына шығарады. Егер мәлімет дұрыс шығарылса, **ch** мәні, әйтпесе – **WEOF** мәні қайтарылады. Бұл көпбайтты символдарға арналған **putchar** функциясының баламасы болып табылады.

qsort	Берілген жиымды сұрыптайды	**<stdlib.h>**

```
void qsort(void *buf, size_t num, size_t size,
   int (compare)(const void *, const void *));
```

Функция кез келген көлемдегі және типтегі жиымдарды сұрыптайды. Сұрыпталатын жиым: **buf** – жиымға нұсқауыш, **num** – элементтер саны, **size** - әрбір элемент мөлшері. Compare аргументі – **qsort** функциясы жіберген екі мәнді салыстыратын және соларға сәйкес кодтар қайтаратын функция адресі; кодтар мәні төмендегідей: нөлден кіші (егер бірінші аргумент екіншісінен кіші болса), нөл (егер аргументтер тең болса) немесе нөлден артық (егер бірінші аргумент екіншісінен үлкен болса); **qsort** функциясына берілетін аргументтер **void*** түрінде анықталған, сондықтан олар функция ішінде керекті типке түрлендірілуі тиіс.

raise	Программаның орындалуын үзеді	<signal.h>

```
int raise(int signal);
```

Функция жұмыс кезінде болған қателер сипатын анықтап, ол туралы программаға сигнал жібереді. Бұл функция ескірген, тек С тілімен үйлесімділікті сақтау үшін қалдырылған. С++ тілінде бұл мақсатта ерекше жағдайлар механизмі жұмыс істейді. Егер жұмыс сәтті аяқталса, функция нөл мәнін, әйтпесе – нөлге тең емес мәнді қайтарады.

rand	Кездейсоқ сандарды береді	<stdlib.h>

```
int rand(void);
```

Функция 0 және **RAND_MAX** арасынан кезекті кездейсоқ санды қайтарады.

realloc	Бұрын бөлінген жады блогы мөлшерін өзгертеді	<stdlib.h>

```
void *realloc(void *memblock, size_t size);
```

Функция жады блогын (бұдан бұрын **malloc**, **calloc** немесе **realloc** функциялары арқылы бөлінген) алады, қажет болса, оны жаңа көлемге сәйкес келетіндей етіп өзгертуге болады. Мұндайда бұл орында бұрын сақталған мәліметтер жоғалмайды. **malloc** функциясына қатысты көптеген ережелер бұл функцияға да қатысты болып табылады.

Функция қайтаратын мән жады блогының жаңа орнына нұсқауыш болып табылады. Ол ескі мәннен, егер **realloc** функциясы жады блогын өзгертіп жіберсе ғана, басқаша болады. Егер жады блогын үлкейте алмасақ, онда қайтарылатын мән **NULL** болады, мұндайда **memlock** мәні программадан қол жетімді болады.

remove	Файлды жояды	<stdio.h>

```
int remove(const char *filename);
```

Функция бұрыннан бар файлды жояды. Егер жұмыс сәтті аяқталса, функция нөл мәнін, әйтпесе – нөлге тең емес мәнді қайтарады.

rename	Файлдың атын өзгертеді	<stdio.h>

```
int rename(const char *oldname,const char *newname);
```

Функция бұрыннан бар файлдың немесе буманың атын өзгереді. Егер жұмыс сәтті аяқталса, функция нөл мәнін, әйтпесе – нөлге тең емес мәнді қайтарады.

rewind	Ағыммен жұмыс істеу кезінде қате жалаушаларын тазартып, файл басына көшеді	<stdio.h>

```
void rewind(FILE *f);
```

Функция **f** ағымындағы қателер жалаушасын тазартады да, ағымдағы позицияны файл басына орнатады.

scanf	Белгілі бір форматтағы параметрлер тіркесін енгізеді	<stdio.h>

```
int scanf(const char *fmt [, par1, par2,…]);
```

Функция **par1**, **par2** және т.б. параметрлер тіркесін стандартты енгізу құрылғысынан **fmt** тіркесімен анықталған форматта енгізеді (әдетте пернетақтадан). Функция мән меншіктелген айнымалылар санын қайтарады.

setbuf	Енгізу/шығару ағымын буферлеуді орнатады	<stdio.h>

```
setbuf void setbuf(FILE *f, char *p);
```

Функция **p** көрсеткішімен берілген енгізу/шығару буферін орнатады. Буфердің көлемі **BUFSIZ** ұзындығындай болуы тиіс. Егер **p** мәні **null** болса, онда буферлеу жүргізілмейді.

setlocale	Жаңа локализацияны орнатады	<stdio.h>

```
char *setlocale(int type, const char *locale);
```

Функция программаның жаңа локализациясын орнатады.

setvbuf	Енгізу/шығару ағымын буферлеуді орнатады	<stdio.h>

```
void setvbuf(FILE *f, char *p, int mode, size_t size);
```

Функция **p** көрсеткішімен берілген енгізу/шығару буферін орнатады. **mode** параметрі режимді, ал **size** – буфер көлемін көрсетеді. Жұмыс сәтті аяқталғанда, функция 0-дік мән, ал керісінше, қате жағдайда нөлге тең емес мән қайтарады. **mode** параметрі келесі мәндердің бірін қабылдайтын бүтін сан болып табылады: **_IOFBF** (толық буферлеу), **_IONBF** (буферлеуге тыйым салу) және **_IOLBF** (нәтиже үшін — қатар бойынша буферлеу, яғни буферге жаңа жол символын жазу кезінде оның босатылуы).

signal	Сигналдарды өңдеуішті тіркейді	<stdio.h>

```
void (*signal(int signum, void (*func)(int)))(int);
```

Функция сигналдарды өңдеуішті тіркейді. C++ ерекше жағдайлар өңдеуіштері қателерді басқарудың тиімді механизмін қамтамасыз етуіне қарамастан, **signal** және **raise** функциялары бұларды C кітапханасымен үйлестіру үшін енгізілген.

sin	Синусты есептейді	<math.h>

```
double sin(double x);
```

Функция **x** радианға тең бұрыш синусының мәнін қайтарады.

sinh	Гиперболалық синусты есептейді	<math.h>

```
double sinh(double x);
```

Функция **x** радианға тең бұрыштың гиперболалық синусы мәнін қайтарады. Егер функция мәні мүмкін болатын диапазоннан шығып кетсе, онда функция **HUGEVAL** мәнін қайтарады да, **errno** ауқымды айнымалысы **ERANGE** мәнін алады.

sprintf	Белгілі бір форматтағы параметрлер тіркесін шығарады	<stdio.h>

```
int sprintf(double x);
```

Функция **x** радианға тең бұрыштың гиперболалық синусы мәнін қайтарады. Егер функция мәні мүмкін болатын диапазоннан шығып кетсе, онда функция **HUGEVAL** мәнін қайтарып, **errno** ауқымды айнымалысы **ERANGE** мәнін алады.

sqrt	Квадрат түбірді есептейді	<math.h>

```
double sin(double x);
```

Функция **x** санының квадрат түбірінің мәнін қайтарады.

srand	Бастапқы жалған кездейсоқ санды орнатады	<stdlib.h>

```
void srand(unsigned int seed);
```

Функция жалған кездейсоқ сандар тізбегінің бастапқы мәнін орнатады.

sscanf	Тіркестен мәлімет енгізеді	<stdlib.h>

```
int sscanf(const char *buf,
           const char *format [,parl, par2, ...]);
```

Функция **scanf** функциясына ұқсас, бірақ ол мәліметтерді пернетақтадан емес, оған бірінші параметр арқылы жеткізілген символдар тіркесінен енгізеді. **buf** аргументі – мәндер енгізілетін символдар тіркесі, **format** – формат тіркесі, осыған сәйкес мәліметтер түрлендіріледі, ал мұндағы көпнүкте енгізілетін мәндердің адрестеріне сәйкес міндетті емес аргументтердің бар екенін көрсетеді.

strcat	Тіркестерді қосады	<string.h>

```
char *strcat(char *s1, char *s2);
```

Функция **s1**-ге **s2**-ні қосады да, **s1** мәнін қайтарады. Нәтижелік тіркес соңына нөл-символ қосылады.

strchr	Тіркестен символ іздейді	<string.h>

```
char * strchr(char *s, int ch);
```

Функция **ch** символының **s** тіркесіндегі алғашқы енгізілуін анықтап,

соған нұсқауышты қайтарады, ал егер ондай нұсқа болмаса, онда **NULL** мәні қайтарылады.

strcmp	Тіркестерді салыстырады	**<string.h>**

```
char *strcmp(char *s1, char *s2);
```

Функция тіркестерді салыстырады да, теріс мән (егер **s1** **s2**-ден кіші болса), нөлге тең мән (егер **s1** **s2**-ге тең болса) немесе оң мән (егер **s1** **s2**-ден артық болса) қайтарады.

strcoll	Орнатылған локализацияны есепке ала отырып, тіркестерді салыстырады	**<string.h>**

```
char *strcoll(char *s1, char *s2);
```

Функция **strcmp** тәрізді тіркестерді салыстырады, бірақ локализация орнатуларын есепке алып отырады (**setlocale** функциясын қара).

strcpy	Бір тіркесті екінші тіркеске көшіреді	**<string.h>**

```
char *strcy(char *s1, char *s2);
```

Функция **s2** тіркесін **s1**-ге көшіреді де, **s1**-ді қайтарады.

strcspn	Бір тіркестегі бір символды екінші тіркестен іздейді	**<string.h>**

```
size_t strcspn(char *s1, char *s2);
```

Функция **s2** тіркесінің кез келген символының **s1** тіркесінде орналасу индексінің мәнін қайтарады.

strerror	Нұсқауышты қатенің сипатталуы бар тіркеске қайтарады	**<string.h>**

```
char *strerror(size_t n);
```

Функция **n** нөмірлі қатенің сипатталуы бар тіркеске нұсқауышты қайтарады.

strftime	**fmt** форматындағы уақытты **tm** форматына түрлендіреді	**<time.h>**

```
struct tm strftime(char*s,size_t size,fmt, const struct tm *ctm);
```

Функция **fmt** форматы негізінде дата мен уақыты бар форматталған тіркесті қайтарады. Функция мәні **tm** типіне сәйкес **time_t** типінде болады.

strlen	Тіркестің ұзындығын қайтарады	**<string.h>**

```
size_t strlen(char *s);
```

Функция тіркес ұзындығын (тіркестің аяқталу символын есепке алмай) қайтарады.

strncat	**n** символы бар бір тіркесті екінші тіркеске қосады	**<string.h>**

```
char *strncat(char *s1, char *s2, size_t n);
```

Функция **s1** тіркесіндегі ұзындығы **n** символдан аспайтын тіркесті **s2** тіркесіне қосып, **s1**-ді қайтарады. **s2** тіркесінің бірінші символы **s1** тіркесін аяқтайтын нөл-символдың орнына жазылады. Егер **s2** тіркесінің ұзындығы **n**-нан кіші болса, онда **s2** тіркесінің барлық символдары **s1**-ге жазылады. **s1** тіркесіне нөл-символ қосылып жазылады. Егер тіркестер бір-бірімен қабаттасып кететін болса, онда оның тәртібі анықталмайды.

strncmp	**n** символы бар бір тіркесті екінші тіркеспен салыстырады	**<string.h>**

```
int strncmp(char *s1, char *s2, size_t n);
```

Функция бірінші тіркес пен екінші тіркестің алғашқы **n** символдарын салыстырады да, теріс (егер **s1** **s2**-ден кіші болса), нөл (егер **s1** **s2**-ге тең болса) немесе оң (егер **s1** **s2**-ден үлкен болса) мәндер қайтарады.

strncpy	Бір тіркестің алғашқы n символын екінші тіркеске көшіреді	**<string.h>**

```
char *strncpy(char *s1, char *s2, size_t n);
```

Функция **s2** тіркесінен **s1** тіркесіне **n** символдан аспайтын ішкі тіркесті көшіреді де, **s1**-ді қайтарады. Егер бастапқы тіркестің ұзындығы **n**-ге тең немесе одан артық болса, **s1** тіркесінің соңына нөл-символ қосылмайды. Кері жағдайда тіркес **n** символына дейін нөл-символдармен толықтырылады. Егер тіркестер бір-бірімен қабаттасып кететін болса, онда оның тәртібі анықталмаған болып саналады.

strpbrk	Бір тіркес ішіндегі символды екінші тіркестен іздейді	**<string.h>**

```
char *strpbrk(char *s1, char *s2);
```

Функция **s2** тіркесінің кез келген символы **s1** тіркесінен бастапқы рет кездесетін символға нұсқауышты қайтарады, егер ондай символ болмаса, **NULL** мәні қайтарылады.

strrchr	Тіркестен символ іздейді	**<string.h>**

```
char *strrchr(char *s, int ch);
```

Функция **s** тіркесінің оң жақ шетінен бастап есептегенде, **ch** символының алғашқы рет кездесетін символына нұсқауышты қайтарады, егер ондай символ болмаса, **NULL** мәні қайтарылады.

strspn	Бір тіркесте бар, ал екіншісінде жоқ символды іздейді	**<string.h>**

```
size_t strspn(char *s1, char *s2);
```

Функция **s2** тіркесінде кездеспейтін **s1** тіркесіндегі алғашқы символдың индексін қайтарады.

strstr	Тіркестегі ішкі тіркесті іздейді	<string.h>

```
char *strstr(char *s1, char *s2);
```

Функция **s1** тіркесінен **s2** ішкі тіркесінің бастапқы кездесетін нұсқасын іздеу ісін жүргізеді. Іздеу сәтті аяқтаса, **s1** тіркесіндегі **s2** ішкі тіркесі басталатын элементке нұсқауышты қайтарады, егер ондай элемент болмаса, **NULL** мәні қайтарылады.

strtod	Тіркесті санға түрлендіреді	<stdlib.h>

```
double strtod(const char *str, char **end);
```

Функция символдар тіркесін сандық мәнге түрлендіріп, сол мәнді қайтарады. Шамадан тыс толып кеткен жағдайда **+/-HUGE_VAL** мәнін қайтарады. Түрлендіруді жүзеге асыру мүмкін болмаса немесе дәреженің жойылып кетуі туындаса, 0 мәні қайтарылады. Соңғы екі жағдайда **errno ERANGE** мәнін қабылдайды. **end** түрлендіру аяқталған символға нұсқауышты қайтарады.

strtok	Тіркестегі лексеманы ерекшелейді	<string.h>

```
char *strtok(char *s1, char *s2);
```

Функция **s2** жиынындағы кез келген бір символмен бөлінген **s1** тіркесінің келесі лексемасын қайтарады.

strtol	Санау жүйесін есепке ала отырып, тіркесті санға түрлендіреді	<stdlib.h>

```
double strtol(const char *str, char **end, int radix);
```

Функция символдар тіркесін көрсетілген **radix** санау жүйесін есепке ала отырып сандық мәнге түрлендіреді де, алынған санды қайтарады. Функция мүмкін болатын бастапқы босорындарды есепке алмай, түрлендіруді сан ретінде бейнелене алмайтын бірінші символда аяқтайды. **end** параметрі **char*** типті нұсқауыштың адресі болып табылады; бұл нұсқауышта түрлендірілмеген алғашқы символдың адресі сақталады.

Функция шамадан тыс толып кету кезінде **LONG_MAX** немесе **LONG_MIN** мәнін қайтарады. Түрлендіруді орындау мүмкін болмаған жағдайда, 0 мәнін қайтарады. Соңғы екі жағдайда **errno ERANGE** мәнін қабылдайды.

strtoul	Санау жүйесін есепке ала отырып, тіркесті санға түрлендіреді	<stdlib.h>

```
double strtoul(const char *str, char **end, int radix);
```

Функция **strtol** функциясы тәрізді, бірақ ол таңбасыз ұзын бүтін санмен жұмыс істейді. Функция толып кету кезінде **ULONG_MAX** мәнін қайтарады.

strxfrm	Ағымдағы локализация негізінде тіркестерді түрлендіреді	<string.h>

```
size_t strxfrm(char *s1, char *s2, size_t n);
```

Функция **s2**-ден алынған тіркесті түрлендіреді де, оны ағымдағы локализация негізінде **s1**-ге орналастырады. Мұнда **n** символдан аспайтын тіркес түрлендіріледі.

swprintf	Белгілі бір форматтағы параметрлер тіркесін шығарады	<stdio.h>

```
int swprintf(wchar_t *buffer,
                const wchar_t format[,argument,...]);
```

Функция **format** жолымен анықталған форматтағы көпнүктемен белгіленген тізімдегі көрсетілген айнымалылардың мәндерін **buffer** тіркесіне шығарады. Көпбайтты символдарға арналған **sprintf** функциясының баламасы болып табылады.

swscanf	Тіркестен мәлімет енгізеді	<stdio.h>

```
int swscanf(const wchar_t *buf, const wchar_t *format,...);
```

Функция **scanf** функциясы тәрізді мәліметтер енгізеді, бірақ ол пернетақтадан емес, оған бірінші параметр ретінде берілген символдар тіркесінен енгізеді. **buf** аргументі – мәндер енгізілетін символдар тіркесі, **format** – мәліметтердің түрлендірілуін жүзеге асыратын формат тіркесі, ал көпнүкте енгізілетін мәндердің адрестеріне сәйкес міндетті емес аргументтердің бар екенін көрсетеді. Көпбайтты символдарға арналған **sprintf** функциясының баламасы болып табылады.

system	Тіркесті ОЖ командалық процессорына береді	<stdlib.h>

```
int system(const char *command);
```

Функция сөз тіркесін операциялық жүйенің командалық процессорына жібереді. Бұл басқа программаны немесе утилитті орындауға мүмкіндік береді. Қайтарылатын мән жүзеге асырылуға тәуелді болады, бірақ көбінесе 0 мәні оның сәтті аяқталуын көрсетеді.

tan	Аргументтің арктангенсін қайтарады	<math.h>

```
double tan(double x);
```

Функция **x** аргументінің тангенсі мәнін қайтарады.

tanh	Аргументтің гиперболалық тангенсін қайтарады	<math.h>

```
double tan(double x);
```

Функция **x** аргументінің гиперболалық тангенсі мәнін қайтарады.

time	Ағымдағы дата/уақытты **time_t** түрінде қайтарады	**<math.h>**

```
time_t time(time_t *t);
```

Функция **time_t** түрінде (1970 жылдың 1 қаңтарының түні ортасынан бері өткен секундтар саны) ағымдағы дата/уақытты қайтарады. Егер ***t** мәні **NULL**-ге тең болмаса, онда бұл адреске де мән енгізіледі.

tmpfile	Уақытша файлға екілік енгізу/шығару ағымын ашады	**<stdio.h>**

```
FILE *tmpfile(void);
```

Функция екілік енгізу/шығару режимінде уақытша файл ашады да, ағымға нұсқауыш қайтарады.

tmpnam	Файлдың бірегей атын жасайды	**<stdio.h>**

```
char *tmpnam(char *s);
```

Функция файлдың бірегей атын құрастырады, ол уақытша файлдың аты ретінде сәтті түрде қолданылуы мүмкін. Әртүрлі тіркестер **TMP_MAX** рет генерацияланады. **TMP_MAX** **<stdio.h>** файлында анықталады, ол 65535-ке тең болады. **s** параметрі 0 немесе саны **L_tmpnam** символдан аз болмайтын жиымға нұсқауыш болып табылады. **L_tmpnam** **<stdio.h>** файлында анықталған. Егер **s** нөлге тең болса, онда **tmpnam** ішкі статикалық объектіде генерацияланған уақытша файлдың атын қалдырыды да, нұсқауышты сол объектіге қайтарады. Егер **s** нөлге тең болмаса, онда **tmpnam** өз нәтижесін ұзындығы **L_tmpnam** символдан аспайтын, көрсетілген жиымға орналастырады да, **s**-ті объектіге қайтарады.

tolower	Төменгі регистрдегі символды қайтарады	**<stype.h>**

```
int tolower(int ch);
```

Функция **ch** параметрін алады да, оны төменгі регистрде қайтарады. **ch** параметрінде тек кіші байт пайдаланылады.

toupper	Жоғарғы регистрдегі символды қайтарады	**<stype.h>**

```
int toupper(int ch);
```

Функция **ch** параметрін алады да, оны жоғарғы регистрде қайтарады. **ch** параметрінде тек кіші байт пайдаланылады.

towlower	Төменгі регистрдегі символды қайтарады	**<stype.h>**

```
int towlower(wint_t ch);
```

Функция **ch** параметрін алады да, оны төменгі регистрде қайтарады. Көпбайтты символдарға арналған **tolower** функциясының баламасы ретінде қарастырылады.

towupper	Жоғарғы регистрдегі символды қайтарады	**<stype.h>**

```
int towupper(wint_t ch);
```

Функция **ch** параметрін алады да, оны жоғарғы регистрде қайтарады. Көпбайтты символдарға арналған **toupper** функциясының баламасы ретінде қарастырылады.

ungetc	Ағымға символ қайтарады	**<stdio.h>**

```
int ungetc(int ch, FILE *f);
```

Функция **ch** символын қайтадан **f** кіріс ағымына қайтарады. Қате болған жағдайда, **EOF** мәнін, әйтпесе **ch** мәнін қайтарады. Тек бір ғана символдың қайтарылуына кепілдік беріледі.

ungetwc	Ағымға символ қайтарады	**<stdio.h>**

```
wint_t ungetwc(wint_t ch, FILE *f);
```

Функция **ch** символын қайтадан **f** кіріс ағымына қайтарады. Қате болған жағдайда, **WEOF** мәнін, әйтпесе **ch** мәнін қайтарады. Тек бір ғана символдың қайтарылуына кепілдік беріледі. Көпбайтты символдарға арналған **ungetc** функциясының баламасы болып табылады.

vfprintf	Форматталған шығару тіркесін ағымға жібереді	**<stdio.h>**

```
int vfprintf(FILE *stream, char *format, va_list arglist);
```

Функция **fprintf** функциясы тәрізді, бірақ ол тізімнің өзіне емес, тек аргументтер тізіміне нұсқауышқа қол жеткізе алады.

vfwprintf	Форматталған шығару тіркесін ағымға жібереді	**<stdio.h>**

```
int vfwprintf(FILE *stream, wchar_t *format,va_list arglist);
```

Функция **fprintf** функциясы тәрізді, бірақ ол тізімнің өзіне емес, тек аргументтер тізіміне нұсқауышқа қол жеткізе алады және де көпбайтты символдармен жұмыс істейді.

vprintf	Форматталған шығару тіркесін стандартты шығару ағымына жібереді	**<stdio.h>**

```
int vprintf(consr char *format,va_list arglist);
```

Функция **printf** функциясы тәрізді, бірақ ол тізімнің өзіне емес, тек аргументтер тізіміне нұсқауышқа қол жеткізе алады.

vsprintf	Белгілі бір форматтағы параметрлер тіркесін шығарады	**<stdio.h>**

```
int vsprintf(char *string,const char *format,
                                        va_list arglist);
```

Функция **sprintf** функциясы тәрізді, бірақ ол тізімнің өзіне емес, тек аргументтер тізіміне нұсқауышқа қол жеткізе алады.

vswprintf	Белгілі бір форматтағы параметрлер тіркесін шығарады	<stdio.h>

```
int vswprintf(wchar_t *string,const wchar_t *format,
                                        va_list arglist);
```

Функция **sprintf** функциясы тәрізді, бірақ ол тізімнің өзіне емес, тек аргументтер тізіміне нұсқауышқа қол жеткізе алады және де де көпбайтты символдармен жұмыс істейді.

vwprintf	Форматталған шығару тіркесін стандартты шығару ағымына жібереді	<stdio.h>

```
int vwprintf(const wchar_t *format,va_list arglist);
```

Функция **printf** функциясы тәрізді, бірақ ол тізімнің өзіне емес, тек аргументтер тізіміне нұсқауышқа қол жеткізе алады және де де көпбайтты символдармен жұмыс істейді.

wcscat	Тіркестерді қосады	<string.h>

```
wchar_t * wcscat(wchar_t *s1, wchar_t *s2);
```

Функция **s1**-ге **s2**-ні қосады да, **s1**-ді қайтарады. Көпбайтты символдарға арналған **strcat** функциясының баламасы болып табылады.

wcschr	Тіркестен символ іздейді	<string.h>

```
wchar_t * wcschr(wchar_t *s, int ch);
```

Функция **s** тіркесіндегі **ch** символының алғашқы кіруіне нұсқауышты қайтарады, ал егер ондай мүмкіндік болмаса, онда **NULL** қайтарылады. Көпбайтты символдарға арналған **strchr** функциясының баламасы болып табылады.

wcscmp	Тіркестерді салыстырады	<string.h>

```
int *wcscmp(wchar_t *s1, wchar_t *s2);
```

Функция тіркестерді салыстырады да, теріс (егер **s1 s2**-ден кіші болса), нөл (егер **s1 s2**-ге тең болса) және оң (егер **s1 s2**-ден үлкен болса) мәндер қайтарады.

wcscoll	Орнатылған локализацияны есепке ала отырып, тіркестерді салыстырады	<string.h>

```
wchar_t *wcscoll(wchar_t *s1, wchar_t *s2);
```

Функция **strcmp** тәрізді тіркестерді салыстырады, бірақ локализация орнатуларын есепке алады (**setlocale** функциясын қара). Көпбайтты символдарға арналған **strcoll** функциясының баламасы болып табылады.

wcscpy	Бір тіркесті екінші тіркеске көшіреді	<string.h>

```
wchar_t *wcscpy(wchar_t *s1, wchar_t *s2);
```

Функция **s2** тіркесін **s1**-ге көшіреді де, **s1** мәнін қайтарады. Көпбайтты символдарға арналған **strcpy** функциясының баламасы болып табылады.

wcscspn	Бір тіркестегі бір символды екінші тіркестен іздейді	**<string.h>**

```
size_t wcscspn(wchar_t *s1, char *s2);
```

Функция **s1** тіркесіндегі **s2**-ден алынған кез келген символдың индексі мәнін қайтарады. Көпбайтты символдарға арналған **strcspn** функциясының баламасы болып табылады.

wcsftime	**fmt** форматындағы уақытты **tm** форматына түрлендіреді	**<time.h>**

```
struct tm wcsftime(wchar_t *s,size_t size,fmt,
                                    const struct tm *ctm);
```

Функция **fmt** форматы негізінде құрамында дата мен уақыты бар форматталған тіркесті қайтарады. Функция мәні **tm** типіне сәйкес **time_t** типінде болады. Көпбайтты символдарға арналған **strftime** функциясының баламасы болып табылады.

wcslen	Тіркестің ұзындығын қайтарады	**<string.h>**

```
size_t wcslen(wchar_t *s);
```

Функция тіркес ұзындығын (тіркестің аяқталу символын есепке алмай) қайтарады. Көпбайтты символдарға арналған **strlen** функциясының баламасы болып табылады.

wcsncat	**n** символы бар бір тіркесті екінші тіркеске қосады	**<string.h>**

```
wchar_t *wcsncat(wchar_t *s1, wchar_t *s2, size_t n);
```

Функция **n** символдан аспайтын тіркесті **s2**-ден **s1**-ге қосады да, **s1**-ді қайтарады. Көпбайтты символдарға арналған **strncat** функциясының баламасы болып табылады.

wcsncmp	**n** символы бар бір тіркесті екінші тіркеспен салыстырады	**<string.h>**

```
int wcsncmp(wchar_t *s1, wchar_t *s2, size_t n);
```

Функция бірінші тіркес пен екінші тіркестің алғашқы **n** символдарын салыстырады да, теріс (егер **s1 s2**-ден кіші болса), нөлдік (егер **s1 s2**-ге тең болса) немесе оң (егер **s1 s2**-ден үлкен болса) мәндерді қайтарады. Көпбайтты символдарға арналған **strncmp** функциясының баламасы болып табылады.

wcsncpy	Бір тіркестің алғашқы **n** символын екінші тіркеске көшіреді	<string.h>

```
wchar_t * wcsncpy(wchar_t *s1, wchar_t *s2, size_t n);
```

Функция **s2** тіркесін **s1**-ге көшіреді де, **s1** мәнін қайтарады. Көпбайтты символдарға арналған **strncpy** функциясының баламасы болып табылады.

wcspbrk	Бір тіркес ішіндегі бір символды екінші тіркестен іздейді	<string.h>

```
wchar_t *wcspbrk(wchar_t *sl, wchar_t *s2);
```

Функция **s2**-ден **s1** тіркесіне кез келген символдың бірінші кірістірілуі болып табылатын символға нұсқауышты қайтарады, егер ол болмаса, **NULL** қайтарылады. Көпбайтты символдарға арналған **strpbrk** функциясының баламасы болып табылады.

wcsrchr	Тіркестен символ іздейді	<string.h>

```
char *wcsrchr(wchar_t *s,wchar_t ch);
```

Функция нұсқауышты **s** тіркесінің оң жағыман есептегенде, **ch** символының бірінші кірістірілуіне қайтарады, егер ол жоқ болса, онда **NULL** қайтарылады. Көпбайтты символдарға арналған **strrchr** функциясының баламасы болып табылады.

wcsspn	Бір тіркесте бар, ал екіншісінде жоқ символды іздейді	<string.h>

```
size_t wcsspn(wchar_t *s1, wchar_t *s2);
```

Функция **s2** тіркесінде жоқ символдың **s1** тіркесіндегі алғашқы кездесетін индексі мәнін қайтарады. Көпбайтты символдарға арналған **strspn** функциясының баламасы болып табылады.

wcsstr	Тіркестен ішкі тіркес іздейді	<string.h>

```
wchar_t *wcsstr(wchar_t *s1, wchar_t *s2);
```

Функция **s2** ішкі тіркесінің **s1** тіркесіндегі алғашқы кездесетін нұсқасын іздеуді орындайды. Іздеу сәтті аяқталған жағдайда, ол нұсқауышты **s2** ішкі тіркесі басталатын **s1** элементіне қайтарады, ал кері жағдайда, **NULL** мәні қайтарылады. Көпбайтты символдарға арналған **strstr** функциясының баламасы болып табылады.

wcstod	Тіркесті санға түрлендіреді	<stdlib.h>

```
double wcstod(const wchar_t *str, wchar_t **end);
```

Функция символдар тіркесін сандық мәнге түрлендіреді де, сол мәнді қайтарады. Көпбайтты символдарға арналған **strtod** функциясының баламасы болып табылады.

wcstok	Тіркесті санға түрлендіреді	<string.h>

```
wchar_t *wcstok(wchar_t *s1, wchar_t *s2);
```

Функция **s2** жиыны символдарының кез келгенімен бөлінген **s1** тіркесінің келесі лексемасын қайтарады. Көпбайтты символдарға арналған **strtok** функциясының баламасы болып табылады.

wcstol	Санау жүйесін есепке ала отырып, тіркесті санға түрлендіреді	<stdlib.h>

```
double wcstol(const wchar_t *str, wchar_t **end,
                                          int radix);
```

Функция **radix** санау жүйесін есепке алып символ тіркесін сандық мәнге түрлендіреді де, алынған санды қайтарады. Функция мүмкін болатын бастапқы босорындарды есепке алмай, сан түрінде бейнелене алмайтын бірінші символда түрлендіруді аяқтайды. **end** параметрі **char*** типті нұсқауыштың адресі болып табылады; бұл нұсқауышта бірінші түрлендірілмеген символдың адресі сақталады. Көпбайтты символдарға арналған **strtol** функциясының баламасы болып табылады.

wcstombs	**wchar_t** символдарын көпбайтты символдарға түрлендіреді	<stdlib.h>

```
size_t wcstombs(char *s, const wchar_t *pwcs,size_t n);
```

Функция **wchar_t**-ны көпбайтты символға түрлендіреді. Егер **s** көпбайтты емес символға нұсқаса, онда функция -1 мәнін қайтарады, кері жағдайда функция түрлендірілген байттар санын қайтарады.

wcstoul	Санау жүйесін есепке ала отырып, тіркесті санға түрлендіреді	<stdlib.h>

```
double wcstoul(const wchar_t *str, wchar_t **end,
                                          int radix);
```

Функция **strtol** функциясы тәрізді, бірақ таңбасыз ұзын бүтін санмен жұмыс істейді. Көпбайтты символдарға арналған **strtoul** функциясының баламасы болып табылады.

wcstrxfrm	Ағымдағы локализацияны есепке ала отырып, тіркестерді түрлендіреді	<string.h>

```
size_t wcstrxfrm(wchar_t *s1, wchar_t *s2,size_t n);
```

Функция ағымдағы локализация негізінде **s2**-ден алынған тіркесті түрлендіріп, оны **s1** тіркесіне орналастырады. **n** символдан аспайтын тіркес түрлендіріледі. Көпбайтты символдарға арналған **strxfrm** функциясының баламасы болып табылады.

wctomb	**wchar_t** символдарын көпбайтты символдарға түрлендіреді	<stdlib.h>

```
int wctomb(char *s, wchar_t wc);
```

Функция **wchar_t**-ны көпбайтты символға түрлендіреді. Егер **s** нұсқауышы **NULL** болмаса, онда функция нөлдік емес мән қайтарады, ал егер көпбайтты символ аймаққа тәуелді сәйкестікте болса, онда нөл мәнін қайтарады. Егер **NULL** нұсқауышы болса, **wctomb** -1 мәнін қайтарады.

wmemchr	Символдың жады блогына алғашқы кіруін іздейді	<string.h>

```
wchar_t *wmemchr(wchar_t *p, wchar_t c, size_t n);
```

Функция ұзындығы **n** байт болып келетін **p** жады блогындағы **c** аргументінің алғашқы кіруіне нұсқауыш қайтарады. Көпбайтты символдарға арналған **memchr** функциясының баламасы болып табылады.

wmemcmp	Жады блоктарын салыстырады	<wchar.h>

```
wchar_t *wmemcmp(wchar_t *s1, const wchar_t s2, size_t n);
```

Функция екі жады блоктарын салыстырады да, **strcmp** функциясының қайтару коды тәрізді – нөлден кіші, нөлге тең немесе нөлден артық мәндер қайтарады. Көпбайтты символдарға арналған **memcmp** функциясының баламасы болып табылады.

wmemcpy	Жады блогын көшіреді	<string.h>

```
wchar_t *wmemcpy(wchar_t *s1, const wchar_t s2, size_t n);
```

Функция ұзындығы **n** байт болып келетін жады блогын **s2** адресінен **s1** адресіне көшіреді. Көпбайтты символдарға арналған **memcpy** функциясының баламасы болып табылады.

wmemmove	Жады блогын жылжытады	<string.h>

```
wchar_t *wmemmove(wchar_t *s1, const wchar_t s2,size_t n);
```

Функция **wmemcpy** функциясы тәрізді, бірақ **s1** және **s2** блоктары қабаттасып келе береді. Көпбайтты символдарға арналған **memmove** функциясының баламасы болып табылады.

wmemset	Жады блогын символмен толтырады	<string.h>

```
wchar_t *wmemset(wchar_t *s, wchar_t c, size_t n);
```

Функция **c** символымен жады блогын толтырады. Көпбайтты символдарға арналған **memset** функциясының баламасы болып табылады.

wprintf	Белгілі бір форматтағы параметрлер тіркесін шығарады	**<stdio.h>**

int wprintf(const wchar_t *fmt, ...);

Функция стандартты нәтижелік құрылғыға тізімде көпнүктемен белгіленіп көрсетілген айнымалылардың мәндерін **format** тіркесімен анықталған форматта шығарады. Көпбайтты символдарға арналған **printf** функциясының баламасы болып табылады.

wscanf	Белгілі бір форматтағы параметрлер тіркесін енгізеді	**<stdio.h>**

int wscanf(const wchar_t *fmt[,par1, par2, …]);

Функция **par1**, **par2** және т.б. параметрлер тіркесін стандартты енгізу құрылғысынан **fmt** тіркесімен анықталған форматта енгізеді (әдетте пернетақтадан). Мәндері меншіктелген айнымалылар санын қайтарады. Көпбайтты символдарға арналған **scanf** функциясының баламасы болып табылады.

7-ҚОСЫМША

ASCII кодтарының кестесі

0-127 кодтары бар символдар

код	символ	код	символ	код	символ	код	символ
0	nul	32		64	@	96	`
1	soh	33	!	65	A	97	a
2	stx	34	«	66	B	98	b
3	etx	35	#	67	C	99	c
4	eot	36	$	68	D	100	d
5	enq	37	%	69	E	101	e
6	ack	38	&	70	F	102	f
7	bel	39	'	71	G	103	g
8	bs	40	(	72	H	104	h
9	ht	41	)	73	I	105	i
10	lf	42	*	74	J	106	j
11	vt	43	+	75	K	107	k
12	ff	44	,	76	L	108	l
13	cr	45	-	77	M	109	m
14	soh	46	.	78	N	110	n
15	si	47	/	79	O	111	o
16	dle	48	0	80	P	112	P
17	dc1	49	1	81	Q	113	q
18	dc2	50	2	82	R	114	r
19	dc3	51	3	83	S	115	s
20	dc4	52	4	84	T	116	t
21	nak	53	5	85	U	117	u
22	syn	54	6	86	V	118	v
23	etb	55	7	87	W	119	w
24	can	56	8	88	X	120	x
25	em	57	9	89	Y	121	y
26	sub	58	:	90	Z	122	z
27	esc	59	;	91	[	123	{
28	fs	60	<	92	\	124	|
29	gs	61	=	93	]	125	}
30	rs	62	>	94	^	126	~
31	us	63	?	95	_	127	del

128-255 кодтары бар символдар (1251 кодтық кестесі – Windows)

код	символ	код	символ	код	символ	код	символ
128	Ђ	160		192	А	224	а
129	Ѓ	161	Ў	193	Б	225	б
130	‚	162	ў	194	В	226	в
131	ѓ	163	J	195	Г	227	г
132	„	164	¤	196	Д	228	д
133	…	165	Ґ	197	Е	229	е
134	†	166	¦	198	Ж	230	ж
135	‡	167	§	199	З	231	з
136	€	168	Ё	200	И	232	и
137	‰	169	©	201	Й	233	й
138	Љ	170	Є	202	К	234	к
139	‹	171	«	203	Л	235	л
140	Њ	172	¬	204	М	236	м
141	Ќ	173	-	205	Н	237	н
142	Ћ	174	®	206	О	238	о
143	Џ	175	Ї	207	П	239	п
144	ђ	176	°	208	Р	240	р
145	‘	177	±	209	С	241	с
146	’	178	I	210	Т	242	т
147	“	179	і	211	У	243	у
148	”	180	ґ	212	Ф	244	ф
149	•	181	µ	213	Х	245	х
150	–	182	¶	214	Ц	246	ц
151	—	183	·	215	Ч	247	ч
152	□	184	ё	216	Ш	248	ш
153	™	185	№	217	Щ	249	щ
154	љ	186	є	218	Ъ	250	ъ
155	›	187	»	219	Ы	251	ы
156	њ	188	j	220	Ь	252	ь
157	ќ	189	S	221	Э	253	э
158	ћ	190	s	222	Ю	254	ю
159	џ	191	ї	223	Я	255	я

128-255 кодтары бар символдар (866 кодтық кестесі – MS DOS)

код	символ	код	символ	код	сим-вол	код	символ
128	А	160	а	192	└	224	р
129	Б	161	б	193	┴	225	с
130	В	162	в	194	┬	226	т
131	Г	163	г	195	├	227	у
132	Д	164	д	196	─	228	ф
133	Е	165	е	197	┼	229	х
134	Ж	166	ж	198	╞	230	ц
135	З	167	з	199	╟	231	ч
136	И	168	и	200	╚	232	ш
137	Й	169	й	201	╔	233	щ
138	К	170	к	202	╩	234	ъ
139	Л	171	л	203	╦	235	ы
140	М	172	м	204	╠	236	ь
141	Н	173	н	205	═	237	э
142	О	174	о	206	╬	238	ю
143	П	175	п	207	╧	239	я
144	Р	176	░	208	╨	240	Ё
145	С	177	▒	209	╤	241	ё
146	Т	178	▓	210	╥	242	Є
147	У	179	│	211	╙	243	є
148	Ф	180	┤	212	╘	244	Ї
149	Х	181	╡	213	╒	245	ї
150	Ц	182	╢	214	╓	246	Ў
151	Ч	183	╖	215	╫	247	ў
152	Ш	184	╕	216	╪	248	°
153	Щ	185	╣	217	┘	249	∙
154	Ъ	186	║	218	┌	250	·
155	Ы	187	╗	219	█	251	√
156	Ь	188	╝	220	▄	252	№
157	Э	189	╜	221	▌	253	¤
158	Ю	190	╛	222	▐	254	■
159	Я	191	┐	223	▀	255	

8-ҚОСЫМША

Қолданылған терминдердің орысша (ағылшынша)-қазақша сөздігі

Абстрактный класс – *Abstract class*	Абстрактылы класс
Аварийное завершение – *Abnormal termination*	Апаттық (қатемен) аяқталу
Альтернативный способ – *Alternative method*	Баламалы тәсіл
База данных – *Database*	Мәліметтер базасы
Бинарное дерево – *Binary tree*	Бинарлы бұтақтар
Битовые поля – *Bit fields*	Биттік өрістер
Вектор – *Vector*	Векторлар
Ветвление – *Branching*	Тармақталу
Внутреннее представление данных – *Internal representation*	Мәліметтердің ішкі бейнесі
Время жизни переменной – *Lifetime*	Айнымалының өмірлік кезеңі
Встроенный метод – *Inline method*	Құрамдас (кіріктірілген) әдістер
Встроенная функция – *Inline function*	Құрамдас (кіріктірілген) функциялар
Выражение – *Expression*	Өрнек
Данные – *Data*	Мәліметтер
Двоичное дерево – *Binary tree*	Екілік бұтақтар
Двунаправленный итератор – *Bidirectional iterator*	Екібағытты итератор
Двусторонняя очередь – *Deque*	Екіжақты кезектер
Диалоговый режим – *Interactive mode, dialog mode*	Сұхбаттасу режимі
Диапазон – *Range*	Ауқым; диапазон
Дружественная функция – *Friend function*	Достас функциялар
«Заглушка» программы – *Stub*	Программа «жапсырмасы»
Заголовок функции – *Function header*	Функция тақырыбы
Заголовочный файл – *Header file*	Тақырыптық файл
Зарезервированное слово – *Keyword*	Резервтегі сөздер
Знак операции – *Operator*	Операциялар таңбалары (белгілері)
Идентификатор – *Identifier*	Идентификатор, атау, аты
Извлечение данных – *Data extraction*	Мәліметтерді шығарып алу
Имя переменной – *Variable name*	Айнымалы аты, атауы
Инкапсуляция – *Encapsulation*	Инкапсуляция; жасыру

Инстанцирование шаблона – *Template instantiation*	Үлгіні инстанциялау
Информация – *Information*	Ақпарат
Исключение – *Exception*	Ерекше жағдайлар, айрықша жағдайлар Аластамалар
Итератор (действительный или недействительный) – *Iterator (valid or invalid)*	Жарамды немесе жарамсыз итератор
Категории функции	Функция санаттары (категориялары)
Класс – *Class*	Кластар, топтар
Класс подмешивания – *Mix-in class*	Араластыру кластары
Ключевое слово – *Keyword*	Түйінді (өзекті) сөздер
Ключ доступа – *Access specifier*	Қол жеткізу (қатынасу) кілттері
Комментарий – *Comment*	Түсініктеме, комментарий
Компилятор – *Compiler*	Компилятор, түрлендіргіш
Компоновщик – *Linker*	Жинақтауыш
Конечное количество величин	Шамалардың ақырлы (шекті) саны
Константа – *Literal, constant*	Тұрақтылар
Константа символьная – *Character literal*	Символдық тұрақтылар
Константа строковая – *String literal*	Тіркестік тұрақтылар
Конструктор копирования – *Copy constructor*	Көшіру (көшірме) конструкторы
Конструктор по умолчанию – *Default constructor*	Келісім бойынша құрылған конструктор
Куча – *Heap*	Үйінді
Мантисса и порядок числа – *Fractional and exponent parts*	*Санның мантиссасы* мен *дәрежесі*
Массив – *Array*	Жиым
Метка оператора – *Label*	Оператор белгілері
Метод вложения	Қабаттастыру әдісі
Многомерный массив – *Multidimentional array*	Көпөлшемді жиым
Множественное наследование – *Multiple inheritance*	Көптік мұралау
Множество – *Set*	Жиындар Жиын
Множество битовое – *Bitset*	Биттік жиындар
Множество с дубликатами – *Multiset*	Дубликатты (көшірмелі) жиындар
Наименование данных	Мәліметтер атаулары
Наследование – *Inheritance*	Мұралау
Наследование множественное – *Multiple inheritance*	Көптік мұралау
Недоступные ячейки памяти – «Мусор» – *Garbage*	Қол жетімсіз жады ұялары – «Қоқыс»

Нисходящее проектирование – *Top-down design*	Төмен қарай жобалау
Область видимости – *Visibility*	Көріну аймағы
Область действия – *Scope*	Әрекет ету аймағы
Обработка исключительных ситуаций – *Exception handling*	Ерекше жағдайларды өңдеу
Обратный итератор – *Reverse iterator*	Кері итератор
Обход дерева – *Tree traverse*	Бұтақты қарап шығу
Объединение – *Union*	Біріктірмелер
Объектно-ориентированное программирование – *Object-oriented programming*	Объектіге бағытталған программалау
Объектный модуль – *Object module*	Объектілік (нысандық) модуль
Объявление переменной – *Variable declaration*	Айнымалыны жариялау
Оператор исполняемый – *Executable statement*	Атқарылатын (орындалатын) оператор
Оператор описания – *Declaration statement*	Атқарылмайтын оператор
Операция разадресации – *Indirection operator*	Адрессіздендіру операциясы
Операция сглаживания – *Smoothing operation*	Тегістеу операциясы
Отрицатель – *Negator*	Терістеуіштер
Очередь – *Queue*	Кезектер
Очередь с приоритетами – *Priority queue*	Басымдықтары бар кезектер (басымдықты кезектер)
Параметризованный класс– *Class template*	Параметрленген класс
Перегрузка операций – *Operator overloading*	Операцияларды асыра жүктеу
Перегрузка функций – *Function overloading*	Функцияларды асыра жүктеу
Переименование типов – *Typedef*	Типтердің аттарын өзгерту
Переключатель – *Switch*	Ауыстырғыш (switch операторы)
Перехват исключений – *Exception handling*	Ерекше жағдайларды айқындап атқару
Перечисление – *Enumeration*	Тізбелер
Повышающее преобразование	Жоғарылататын түрлендіру
Понижающее преобразование	Төмендететін түрлендіру
Поразрядное исключающее ИЛИ (арифметическое ИЛИ) – *Bitwise exclusive OR*	Разрядтар бойынша аластамалы НЕМЕСЕ (арифметикалық НЕМЕСЕ)
Последовательность – *Sequence*	Тізбек
Поток – *Stream*	Ағым

Пошаговая детализация программы	Программаны біртіндеп байыптау
Преобразование перекрестное – *Cross cast*	Қиылысқан (айқасқан) түрлендіру
Преобразование типов – *Type conversion, cast*	Типтерді түрлендіру
Принципы программирования – *Programming principles*	Программалау қағидалары
Приоритет операции – *Operator priority*	Операциялар басымдықтары
Программирование – *Programming*	Программалау (бағдарламалау)
Программирование восходящее – *Bottom-up programming*	Төменнен жоғары қарай программалау
Программирование нисходящее – *Top-down programming*	Жоғарыдан төмен қарай программалау
Программное обеспечение – *Software*	Программалық жабдықтама; Программамен қамтамасыз ету
Программы отладка – *Debug*	Программаны жөндеп түзету
Программы фрагмент	Программа үзіндісі (фрагменті)
Производный класс – *Derived class*	Туынды класс
Пространство имен – *Namespace*	Атаулар кеңістіктері
Процесс программирования – *Programming process*	Программалау үрдісі (процесі)
Прямой итератор – *Forward iterator*	Тура итератор
Разветвление – *Selection*	Тармақталу
Размер типа – *Size of type*	Тип мөлшері (көлемі, ені)
Размерность жиыма – *Array dimension*	*Жиым* өлшемі
Разрешающая способность дисплея – *Screen resolution, display resolution*	Дисплейдің шешу қабілеті
Разрешение перегрузки – *Overload resolution*	Асыра жүктеудің рұқсатнамасы
Разыменование – *Indirection*	Атаусыздандыру
Раскручивание стека – *Stack*	Стекті тарқату
Сбалансированное дерево – Balanced tree	Теңдестірілген бұтақтар
Связывание позднее – *Late binding*	Кейінгі байланыстыру
Связывание раннее – *Early binding*	Ерте байланыстыру
Связыватель – *Binder*	Байланыстырғыштар
Седловая точка – *Saddle point*	Қайқы нүкте; ершік нүктесі
Следование – *Sequencing*	Тізбектелу
Словарь – *Map*	Сөздіктер
Словарь с дубликатами – *Multimap*	Дубликатты (көшірмелі) сөздіктер
Собственные типы данных	Өзіндік мәліметтер типтері
Составной оператор – *Block statement*	Құрама оператор

Составной тип данных – *Compound data type*	Мәліметтердің құрамалы типтері
Состояние параметра	Параметрдің қалып-күйі (қалпы, күйі)
Специальный преобразователь (адаптер) – *Adaptor*	Арнайы түрлендіргіштер (адаптерлер)
Спецификатор доступа – *Access specifier*	Қол жеткізу (қатынасу) спецификаторы
Список – *List*	Тізімдер
Срез – *Slice*	Қима
Средства программирования – *Software Tools*	Программалау құралдары
Ссылка – *Reference*	Сілтеме
Стек вызовов – *Call stack*	Шақырулар стегі
Стек – *Stack*	Стектер
Структурное программирование – *Structured programming*	Құрылымдық программалау
Структурный подход – *Structured approach*	Құрылымдық көзқарас (тәсіл)
Структура – *Structure*	Құрылымдар
Сцепка списков – *Splice*	Тізімдердің тіркесуі
Таблицы или матрицы столбец – *Column*	Матрица немесе кесте бағанасы
Таблицы или матрицы строка – *Row*	Матрица немесе кесте жолы
Таблицы колонка – *Column*	Кесте бағаны
Текущая дата – *Current date*	Үстіміздегі (ағымдағы) мерзім не дата
Текущий символ	Ағымдағы (үстіміздегі) символ
Тело оператора – *Statement body*	Оператор тұлғасы (ішкі операторлары)
Термин – *Term*	Атаунамалар; терминдер
Тип возвращаемого значения – *Return type*	Қайтарылатын мәннің типі
Тип привязываемого значения	Түйістірілетін мәннің типі
Угловые скобки – *Angle brackets*	Бұрыштық жақшалар
Указатель на ссылку – *Pointer to reference*	Сілтемелерге нұсқауыштар
Указатель – *Pointer*	Нұсқауыш
Унаследованные функции из языка С	С тілінен мұраланған функциялар
Форматированный ввод/вывод – *Formatted input/output*	Форматталған енгізу/шығару
Функции вызов – *Function call*	Функцияны шақыру
Функции объявление – *Function declaration*	Функцияны жариялау
Функции определение – *Function definition*	Функцияны анықтау

Функция с переменным числом параметров – *Function with variable number of parameters*	Параметрлерінің саны айнымалы функциялар
Хаос алгоритмов – *Algorithmic chaos*	Алгоритмдер аласапыраны (хаосы)
Цикл – *Iteration*	Цикл; қайталау
Число вещественное – *Floating value*	Нақты сандар
Число с плавающей точкой – *Floating point value*	Жылжымалы нүктелі сандар
Число с фиксированной точкой – *Fixed point value*	Бекітілген нүктелі сандар
Число целое – *Integer value*	Бүтін сандар
Число конечное – *Finite number*	Ақырлы (шекті) сан
Число ограниченное – *Bounded number*	Шектеулі сан
Шаблон функции – *Function template*	Функция үлгілері (шаблондары)
Электронная почта – *e-mail*	Электрондық пошта

Алфавиттік нұсқауыш

#%&

#define, 100
#elif, 101
#else, 101
#endif, 101
#if, 100
#ifdef, 100
#ifndef, 100
#include, 99
#undef, 102
__cplusplus, 102
__DATE__, 103
__FILE__, 103
__LINE__, 103
__TIME__, 103
_IOFBF, 429
_IOLBF, 429
_IONBF, 429

A

abort, 241, 247, 445
abs, 98, 445
accumulate, алгоритм, 401
acos, 98, 445
adjacent_difference, алгоритм, 403
adjacent_find, алгоритм, 374
adjustfield, ios, 289
advance, 359
any, bitset, 352
app, ios, 301
append, string, 310
apply, valarray, 406
asctime, 445
asin, 98, 445
assert, 425
assign, string, 310
assign, vector, 324
at, string, 309
at, vector, 324
at, контейнерлер, 321
atan, 98, 445
atan2, 98, 445
ate, ios, 301
atexit, 445
atof, 97, 446
atoi, 97, 446
atol, 97, 446
auto, 23

B

back, list, 331
back, queue, 337
back, vector, 325
back_insert_iterator, 361
back_inserter, 361
bad, ios, 299
bad_alloc, 248, 411, 412
bad_cast, 251, 412
bad_exception, 412
bad_typeid, 257, 412
badbit, ios, 298
basefield, ios, 289
before, type_info, 257
begin, итератор, 320
binary, ios, 302
binary_function, 364
binary_negate, 366
binary_search, алгоритм, 389
bind1st, байланыстырғыштар, 366
bind2nd, байланыстырғыштар, 366
binder1st, 367
binder2nd, 367
bitset, 350
bool, 17
bsearch, 446
btowc, 447
BUFSIZ, 425

C

C_str, string, 312
calloc, 447
capacity, string, 317
capacity, vector, 325
catch, 242, 325

ceil, 98, 447
cerr, 285, 425
char, 17
CHAR_BIT, 425
CHAR_MAX, 425
CHAR_MIN, 425
cin, 285, 425
class, 192
clear, ios, 299
clear, list, 332
clear, map, 345
clear, string, 312
clear, vector, 326
clearerr, 447
clock, 447
clock_t, 426
CLOCKS_PER_SEC, 426
clog, 285, 425
close, fstream, 303
compare, string, 316
complex, 413
const, 22, 50
const_cast, 250
const_iterator begin, контейнерлер, 320
const_iterator end, контейнерлер, 320
const_iterator, контейнерлер, 319
const_reference, контейнерлер, 319
const_reverse_iterator rbegin, контейнерлер, 320
const_reverse_iterator rend, контейнерлер, 320
const_reverse_iterator, контейнерлер, 319
copy, string, 313
copy, алгоритм, 380
copy_backward, алгоритм, 380
cos, 98, 447
cosh, 98, 447
count, bitset, 352
count, map, 345
count, set, 348
count, алгоритм, 375
count_if, алгоритм, 367, 375
cout, 285, 425
cshift, valarray, 406
ctime, 448

D

data, string, 312
DBL, 426
dec, ios, 288
dec, манипулятор, 290
deque, 320, 329
difftime, 448
distance, 359
div, 448
div_t, 426, 433
divides, функционалды объект, 364
domain_error, 412
double, 17
dynamic_cast, 250

E

EDOM, 426
empty, proirity_queue, 339
empty, queue, 337
empty, stack, 336
empty, string, 317
empty, контейнерлер, 320
end, итератор, 320
endl, манипулятор, 291
ends, манипулятор, 291
enum, 66
eof, 321
EOF, 426
eof, ios, 299
eofbit, ios, 298
equal, алгоритм, 376
equal_range, map, 347
equal_range, set, 349
equal_range, алгоритм, 389
equal_to, предикат, 365
ERANGE, 426
erase, list, 331
erase, map, 345
erase, set, 348
erase, string, 311
erase, vector, 326
erase, контейнерлер, 321
errno, 97, 427
exception, 248, 412
exit, 448

EXIT_FAILURE, 427
EXIT_SUCCESS, 427
exp, 98, 448
explicit, 323
extern, 24, 75, 105

F

fabs, 98, 448
fail, ios, 299
failbit, ios, 298
false, 17
fclose, 95, 448
feof, 95, 448
ferror, 95, 449
fflush, 449
fgetc, 94, 449
fgetpos, 94, 449
fgets, 95, 449
fgetwc, 449
fgetws, 449
FILE, 93, 427
filebuf, 427
FILENAME_MAX, 427
fill, ios, 289
fill, алгоритм, 381
fill_n, алгоритм, 381
find, map, 345
find, set, 348
find, string, 314
find, алгоритм, 376
find_end, алгоритм, 377
find_first_not_of, string, 314
find_first_of, string, 314
find_first_of, алгоритм, 377
find_if, алгоритм, 376
find_last_not_of, string, 315
find_last_of, string, 314
fixed, ios, 288
flags, ios, 288
flip, bitset, 351
flip, vector_bool, 328
float, 17
float_round_style, 428
floatfield, ios, 289
floor, 98, 450
FLT, 428
FLT_DIG, 411, 428
FLT_EPSILON, 428
FLT_MANT_DIG, 428
FLT_MAX, 428
FLT_MAX_EXP, 428
FLT_MIN, 411, 428
FLT_MIN_10_EXP, 428
FLT_MIN_EXP, 428
FLT_RADIX, 428
FLT_ROUNDS, 428
FLT_MAX_10_EXP, 428
flush, ostream, 294
flush, манипулятор, 291
fmod, 98, 450
fopen, 93, 450
FOPEN_MAX, 429
for_each, алгоритм, 378
fpos_t, 429
fprintf, 95, 450
fputc, 95, 450
fputs, 95, 450
fputwc, 450
fputws, 451
fread, 94, 451
free, 52, 451
freopen, 451
frexp, 451
friend, 202
front, list, 331
front, queue, 337
front, vector, 325
front_insert_iterator, 361
front_inserter, 361
fscanf, 95, 451
fseek, 94, 451
fsetpos, 94, 452
fstream, 301
ftell, 94, 452
fwide, 452
fwprintf, 452
fwrite, 94, 452
fwscanf, 453

G

gcount, istream, 292
generate, алгоритм, 381

generate_n, алгоритм, 381
get, istream, 292
getc, 94, 453
getchar, 94, 453
getenv, 453
getline, istream, 293
gets, 94, 453
getwc, 453
getwchar, 453
gmtime, 454
good, ios, 299
goodbit, ios, 298
greater, предикат, 365
greater_equal, предикат, 365
gslice, 408
gslice_array, 403, 409

H

hardfail, ios, 298
hex, ios, 288
hex, манипулятор, 291
HUGE_VAL, 429

I

ifstream, 301, 429
ignore, istream, 292
in, ios, 301
includes, алгоритм, 396
indirect_array, 403б 409
inline, 75
inner_product, алгоритм, 402
inplace_merge, алгоритм, 390
insert, list, 331
insert, map, 345
insert, set, 348
insert, string, 311
insert, vector, 326
insert, контейнерлер, 321
insert_iterator, 361
inserter, 361
int, 16
INT_MAX, 429
INT_MIN, 429
internal, ios, 288
invalid_argument, 412
ios, 283, 429
  x_flags, 287
isalfa, 98
isalnum, 98, 454
isalpha, 454
iscntrl, 98, 454
isdigit, 98, 454
isgraph, 98, 454
islower, 98, 455
isprint, 98, 455
ispunct, 98, 455
isspace, 98, 455
istream, 283, 429
  әдістер, 292
istream_iterator, 361
istringstream, 303, 430
isupper, 98, 455
iswalnum, 455
iswalpha, 456
iswcntrl, 456
iswctype, 456
iswdigit, 456
iswgraph, 456
iswlower, 456
iswprint, 456
iswpunct, 457
iswspace, 457
iswupper, 457
iswxdigit, 457
isxdigit, 98, 457
iter_swap, алгоритм, 382
iterator, 321
iterator begin, контейнерлер, 320
iterator end, контейнерлер, 320
iterator, контейнерлер, 319
iterator_traits, 358

K

key_compare, контейнерлер, 319
key_type, контейнерлер, 319

L

L_tmpnam, 430
labs, 458
LC_ALL, 430

LC_COLLATE, 430
LC_CTYPE, 430
LC_MONETARY, 430
LC_NUMERIC, 430
LC_TIME, 430
lconv, 430
LDBL, 431
ldexp, 458
ldiv, 458
ldiv_t, 431
left, ios, 287
length, string, 317
length_error, 412
less, предикат, 365
less_equal, предикат, 365
lexicographical_compare, алгоритм, 390
list, 320, 331
localeconv, 458
localtime, 458
log, 98, 458
log10, 98, 458
logic_error, 412
logical_and, предикат, 365
logical_not, предикат, 365
logical_or, предикат, 365
long, 16
long double, 17
LONG_MAX, 431
LONG_MIN, 431
lower_bound, map, 345
lower_bound, set, 348
lower_bound, алгоритм, 391
L-мән, 27

M

main, 20, 91
make_heap, алгоритм, 399
make_pair, pair, 342
malloc, 54, 249, 459
map, 341, 342, 352
mask_array, 403, 409
max, valarray, 406
max, алгоритм, 391
max_element, алгоритм, 391
max_size, string, 317
max_size, контейнерлер, 320
MB_CUR_MAX, 431
MB_LEN_MAX, 431
mblen, 459
mbstate_t, 431
mbstowcs, 459
mbtowc, 459
mem_fun, 372
mem_fun_ref, 372
memchr, 459
memcmp, 459
memcpy, 459
memmove, 460
memset, 460
merge, list, 334
merge, алгоритм, 392
min, valarray, 406
min, алгоритм, 391
min_element, алгоритм, 391
minus, функционалды объект, 364
mismatch, алгоритм, 378
mktime, 460
modf, 99, 460
modulus, функционалды объект, 364
multimap, 347
multiplies, функционалды объект, 364
multiset, 349

N

name, type_info, 257
namespace, 103
negate, функционалды объект, 364
new, 51, 125, 199
new_handler, 411, 432
next_permutation, алгоритм, 392
nocreate, ios, 302
none, bitset, 352
noreplace, ios, 302
not_equal_to, предикат, 365
not1, терістеуіш, 366
not2, терістеуіш, 366
nth_element, алгоритм, 394
NULL, 54, 432
numeric_limits, 411

O

oct, ios, 288
oct, манипулятор, 290
offsetof, 432
ofstream, 301, 432
open, fstream, 301, 302
open_mode, ios, 301
operator, 205
ostream, 283, 432
ostream_iterator, 362
ostringstream, 303, 432
out, ios, 301
out_of_range, 309, 325, 412
overflow_error, 412

P

pair, 341
partial_sort, алгоритм, 394
paqueue, 338
pop, queue, 3rtial_sort_copy, алгоритм, 394
partial_sum, алгоритм, 402
partition, алгоритм, 395
peek, istream, 293
perror, 460
plus, функционалды объект, 364
pointer_to_binary_function, 367
pointer_to_unary_function, 367
pop, stack, 336
pop_back, list, 331
pop_back, vector, 326
pop_back, контейнерлер, 321
pop_front, deque, 330
pop_front, list, 331
pop_front, контейнерлер, 321
pop_heap, алгоритм, 399
pow, 98, 460
precision, ios, 289
prev_permutation, алгоритм, 392
printf, 20, 84, 95, 249, 460
private, 192, 216
proirity_queue, 338
protected, 216
ptr_fun, 371
ptrdiff_t, 359, 432
public, 192, 216, 226
push, proirity_queue, 339
push, queue, 337
push, stack, 336
push_back, list, 331
push_back, vector, 326
push_back, контейнерлер, 321
push_front, deque, 331
push_front, list, 331
push_front, контейнерлер, 321
push_heap, алгоритм, 399
put, ostream, 294
putback, istream, 293
putc, 95, 460
putchar, 95, 461
puts, 95, 461
putwc, 461
putwchar, 461

Q

qsort, 461
queue, 337

R

raise, 462
rand, 462
RAND_MAX, 432
random_shuffle, алгоритм, 382
range_error, 412
rbegin, итератор, 320
rdstate, ios, 299
read, istream, 293
readsome, 293
realloc, 462
reference, контейнерлер, 319
register, 24
reinterpret_cast, 256
remove, 462
remove, list, 333
remove, алгоритм, 383
remove_copy, алгоритм, 383
remove_copy_if, алгоритм, 383
remove_if, list, 333
remove_if, алгоритм, 383
rename, 462
rend, итератор, 320

replace, string, 312
replace, алгоритм, 385
replace_copy, алгоритм, 385
replace_copy_if, алгоритм, 385
replace_if, алгоритм, 385
reserve, vector, 325
reset, bitset, 351
resetiosflags, манипулятор, 291
resize, list, 332
resize, valarray, 406
resize, vector, 326
return, 78
reverse, list, 334
reverse, алгоритм, 386
reverse_copy, алгоритм, 386
reverse_iterator, 361
reverse_iterator rbegin, контейнерлер, 320
reverse_iterator rend, контейнерлер, 320
reverse_iterator, контейнерлер, 319
rewind, 462
rfind, string, 314
right, ios, 287
rotate, алгоритм, 386
rotate_copy, алгоритм, 386
RTTI, 251, 257
runtime_error, 412

S

scanf, 20, 95, 463
SCHAR_MAX, 432
SCHAR_MIN, 432
scientific, ios, 288
search, алгоритм, 378
search_n, алгоритм, 378
SEEK_CUR, 433
SEEK_END, 433
SEEK_SET, 433
seekg, istream, 293
seekg, ostream, 294
set, 348, 354
set, bitset, 351
set_difference, алгоритм, 397
set_intersection, алгоритм, 397
set_new_handler, 411
set_symmetric_difference, алгоритм, 397
set_terminate, 246
set_unexpected, 246
set_union, алгоритм, 397
setbase, манипулятор, 291
setbuf, 94, 463
setf, ios, 289
setfill, манипулятор, 291
setiosflags, манипулятор, 291
setlocale, 463
setprecision, манипулятор, 291
setvbuf, 94, 463
setw, манипулятор, 291
shift, valarray, 406
short, 16
showbase, ios, 288
showpoint, ios, 288
showpos, ios, 288
SHRT_MAX, 433
SHRT_MIN, 433
SIG_DFL, 433
SIG_ERR, 433
SIG_IGN, 433
SIGABRT, 433
SIGFPE, 433
SIGILL, 433
SIGINT, 433
signal, 463
siigned, 16
SIGSEGV, 433
SIGTERM, 433
sin, 98, 463
sinh, 98, 464
size, bitset, 352
size, proirity queue, 339
size, queue, 337
size, stack, 336
size, string, 317
size, valarray, 405
size, контейнерлер, 320
size_t, 208, 433
size type, контейнерлер, 319
sizeof, 249
skipws, ios, 287
slice, 407
slice_array, 403, 408
sort, list, 333

sort, алгоритм, 395
sort_heap, алгоритм, 400
splice, list, 332
sprintf, 95, 464
sqrt, 98, 464
srand, 464
sscanf, 95, 464
stable_partition, алгоритм, 395
stable_sort, алгоритм, 395
stack, 336
state, ios, 298
static, 24, 75, 200
static cast, 2555
std, 107, 282
stdaux, 94
stderr, 94, 434
stdin, 94, 434
stdio, ios, 288
stdout, 94, 434
stdprn, 94
STL, 318
STL алгоритмдері, 373
str, stringstream, 304
strcat, 97, 464
strchr, 97, 464
strcmp, 97, 465
strcoll, 465
strcpy, 97, 465
strcspn, 465
streambuf, 283, 434
streamoff, 434
streamsize, 434
strerror, 465
strftime, 465
string, 307, 434
  конструкторлар, 308
stringbuf, 434
stringstream, 303
strlen, 97, 465
strncat, 97, 465
strncmp, 97, 466
strncpy, 97, 466
strpbrk, 97, 466
strrchr, 97, 466
strspn, 466
strstr, 97, 467
strtod, 467
strtok, 467
strtol, 467
strtoul, 467
struct, 67, 226
strxfrm, 468
substr, string, 312
sum, valarray, 406
swap, list, 331
swap, map, 347
swap, set, 348
swap, string, 312
swap, vector, 326
swap, алгоритм, 382
swap_ranges, алгоритм, 382
swprintf, 468
swscanf, 468
sync_with_stdio, ios_base, 298
system, 468

T

tan, 98, 468
tanh, 98, 468
tellg, istream, 293
tellg, ostream, 294
template, 232
terminate, 241, 246
terminate_handler, 434
test, bitset, 352
this, 195
throw, 240, 245
tie, basic_ios, 298
time, 469
time_t, 435
tm, 435
TMP_MAX, 435
tmpfile, 469
tmpnam, 469
to_string, bitset, 352
to_ulong, bitset, 352
tolower, 98, 469
top, proirity_queue, 339
top, stack, 336
toupper, 98, 469
towlower, 469

towupper, 470
transform, алгоритм, 386
true, 17
trunc, ios, 301
try, 240, 325
type_info, 257
  before, 257
  name, 257
typedef, 65, 238

U

UCHAR_MAX, 435
UINT_MAX, 411, 435
ULONG_MAX, 435
unary_function, 363
unary_negate, 366
underflow_error, 412
unexpected, 246
unexpected_handler, 436
unget, istream, 293
ungetc, 470
ungetwc, 470
union, 70, 226
unique, list, 334
unique, алгоритм, 388
unique_copy, алгоритм, 388
unitbuf, ios, 288
unsetf, ios, 289
unsigned, 16
upper_bound, map, 345
upper_bound, set, 348
upper_bound, алгоритм, 391
uppercase, ios, 288
USHRT_MAX, 436

V

va_arg, 436
va_list, 436
va_start, 436
valarray, 403
value_type, контейнерлер, 319
vector, 320, 322
vector_bool, 328
vfprintf, 470
vfwprintf, 470
virtual, 222
void, 19
vprintf, 470
vptr, 223
vsprintf, 470
vswprintf, 471
vtbl, 223
vwprintf, 471

W

WCHAR_MAX, 436
wchar_t, 17
wcscat, 471
wcschr, 471
wcscmp, 471
wcscoll, 471
wcscpy, 471
wcscspn, 472
wcsftime, 472
wcslen, 472
wcsncat, 472
wcsncmp, 472
wcsncpy, 473
wcspbrk, 473
wcsrchr, 473
wcsspn, 473
wcsstr, 473
wcstod, 473
wcstok, 474
wcstol, 474
wcstombs, 474
wcstoul, 474
wcstrxfrm, 474
wctomb, 475
wctype_t, 436
WEOF, 437
werr, 437
wfilebuf, 437
whar_t, 284
width, ios, 289
wifstream, 437
win, 437
wint_t, 437
wios, 437
wistream, 284, 437
wistringstream, 437

wlog, 437
wmemchr, 475
wmemcmp, 475
wmemcpy, 475
wmemmove, 475
wmemset, 475
wofstream, 284, 438
wostream, 284, 438
wostringstream, 438
wout, 437
wprintf, 476
write, ostream, 294
ws, манипулятор, 291
wscanf, 476
wstreambuf, 438
wstreamoff, 438
wstring, 438
wstringbuf, 438
wtrans_t, 436

X

x_fill, ios, 289
x_flags, 288
x_precision, ios, 289
x_width, ios, 289

А

абстрактылы класс, 224
ағымға қосу, 283
ағымдар, 283
  форматтау, 287
ағымдық итераторлар, 361
ағымнан шығарып алу, 283
адаптер
  әдіс а.-і, 371
  итератор а.-і, 359
  контейнер а.-і, 318, 336
  функцияға нұсқауыш а.-і, 367
  функция а.-і, 364
аймақ
  әрекет ету а.-ы, 23, 104
  көріну а.-ы, 23
айнымалы, 22
айнымалыны анықтау, 25
аластама, 240
  функция а.-лары, 245
аластамалы (ерекше) жағдай, 240
аластаманы генерациялау, 241
алфавит, 7
ассоциативті контейнерлер, 319, 341
асыра жүктеу
  операцияларды, 205
    << және >> о.-н, 305
  функцияларды, 86
атау берілген аймақ, 103, 106
атаулар кеңістігі, 104

Б

базалық конструкциялар, 33
базалық кластар, 216
байланыстырғыштар, 366
бақылаудағы блок, 240
басқарушы тізбек, 13
басымдылықты кезек, 339
белгі, 47
бинарлы бұтақ, 132
биттік жиын, 350
биттік өрістер, 70
блок, 7
бос оператор, 36
босорын символдары, 10
бұтақты қарап шығу, 136
буфер, 94, 283
біріктірмелер, 226

В

вектор, 321
виртуалды әдістер, 221
виртуалды әдістер кестесі, 222

Д

декремент, 27
деструктор, 204
  мұралау, 221
диагностика құралдары, 412
динамикалық айнымалылар, 50
динамикалық мәлімет құрылымдары, 122
  д.м.қ. элементі, 123
динамикалық жиым, 59
достас

кластар, 203
функциялар, 202
дубликатты жиын, 349
дубликатты сөздіктер, 347

Е

екібағытты итератор, 357
екіжақты кезек, 321, 329
ерте байланыстыру, 222

Ж

жады класы, 23
жалаушалар, ағымдар, 287
жалпыланған қима, 408
жариялау
айнымалыны ж., 25
функцияны ж., 74
жинақтауыш, 8
жиым, 57
индексі, 58
өлшемі, 57
жиын, 348
жоғарылататын түрлендіру, 251

И

идентификатор, 10
инициализатор, 22
инициализаторлар (мәндер) тізімі, 199
инициалдау, 22
инкапсуляция, 190
инкремент, 27
итератор, 319б 356
кірістіру и.-ы, 361
кіріс ағым и.-ы, 362
шығыс ағым и.-ы, 362

К

кезек, 130, 337
басымдылықты к., 338
кездейсоқ қатынасу итераторы, 357
кейінгі байланыстыру, 223
кері итератор, 359
класс, 192
әдістері, 192
мұралау, 221
өрістері, 192
шаблонының
әдістері, 223
специализациясы, 238
экземпляры (данасы), 194
элементтері, 192
кластар шаблондары, 228
компилятор, 8
конструктор, 194, 196
келісім бойынша к., 196
көшіру к.-ы, 199
мұралау, 220
түрлендіру к.-ы, 200
контейнерлік кластар, 228, 318
көпөлшемді жиым, 60
көптік мұралау, 225
кілт, 124, 341
қатынасу к.-і, 216
кіріс итератор, 357

Қ

қарапайым мұралау, 217
қима, 407
қиылысқан түрлендіру, 251
құрамдас функция, 75
құрама оператор, 7
құрылымдар, 226
құрылымдық программалау, 5, 6

Л

лексема, 7
локалды (жергілікті) класс, 19
локализация (оқшаулау) құралдары, 413

М

манипуляторлар, 290
қарапайым, 290
параметрленген, 291
мәліметтер типі, 15
bool, 17
char, 17
double, 17
float, 17
int, 16

void, 19
wchar_t, 17
модуль, 20
мұралау, 190
көптік, 225
қарапайым, 217

Н

нөлдік символ, 14
нұсқауыш, 49
ағымдық, 94
класс әдісіне, 213
класс өрісіне, 214
шығыс, 286

О

объект, 194
оқиғалық-басқарылатын модель, 189
операторлар, 7
break, 47
continue, 48
do while, 43
for, 44
goto, 46
if, 36
namespace, 106
return, 48
switch, 39
using, 107
using namespace, 107
while, 42
операциялар, 25
. (нүкте), 69, 195
.->, 69, 195
const cast, 250
delete, 52
асыра жүктеу, 208
dynamic_cast, 250
new, 52
асыра жүктеу, 208
reinterpret_cast, 256
sizeof, 28
static_cast, 255
typeid, 257
ағымнан шығарып алу, 285
адрессіздендіру *, 54
адресті алу &, 51
бөлу /, 28
бөлуден қалған қалдық %, 28
декремент --, 27
индекстеу, 212
инкремент ++, 27
көріну аймағына қатынасу, 194
қатынас о.-ы, 29
логикалық &&, ||, 30
логикалық терістеу !, 28
меншіктеу
+=, -=, *=, /=, %=, 30
асыра жүктеу, 207
разрядтар бойынша терістеу, 28
разрядтар бойынша &, |, ^, 29
типті келтіру, 54, 249
асыра жүктеу, 211
таңдау . (нүкте), 69, 195
таңдау ->, 69, 195
унарлы минус -, 28
функцияны шақыру (), 212
шартты (?:), 31
ығыстыру, 29

Ө

өмірлік кезең, 23
өрнек, 7, 22, 31

П

парадигма, 189
параметр беру
адрес бойынша, 79
мән бойынша, 79
сілтеме бойынша, 80
параметрлік полиморфизм, 239
пирамида, 395
полиморфизм, 190
полиморфты функциялар, 225
полиморфты объект, 225
предикат, 363
препроцессор, 8, 99
препроцессор директивасы, 99
программа, 8

Р

рекурсивті функция, 85

С

сапа критерийлері (программаның), 5
сөздік, 342
стандартты
  ағымдар, 283
  аластамалар, 248
статикалық
  әдістер, 201
  өрістер, 201
стек, 129, 336
стекті тарқату, 241
сызықтық тізім, 124
сілтеме, 56

Т

таза виртуалды әдіс, 223
тақырыптық файл, 8, 99
таңдау әдісі, 58
тармақталу, 33
терістеуіштер, 366
типтелген тұрақты, 22
типті түрлендіру, 33
төмен қарай
  жобалау, 119
  тестілеу, 121
төмендететін түрлендіру, 251
түйінді сөздер, 11
түсініктеме, 14
тура итератор, 357
туынды кластар, 216
тұрақты, 12
  класс шаблондарының т.-ы, 236
тұрақтылық итератор, 357
тұрақтылық
  әдіс, 195
  объект, 195
тізбек, 356
тізбектелу, 33
тізбектік контейнерлер, 318
тізім, 228, 321, 331
тізім түйіні, 228
тілді сүйемелдеу құралдары, 411
тіркес, 63
тіркестік литерал, 13

Ф

файлдық ағымдар, 300
формат
  модификаторлары, 419
  спецификациялары, 419
форматтаушы әдістер, ағымдар, 287
функция
  прототипі, 74
  сигнатурасы, 74
  тақырыбы, 74
  типі, 75
  шаблонының специализациясы, 90
функциялар, 74
  main, 91
  анықтамасы, 74
  асыра жүктелуі, 86
  жариялануы, 74
  құрамдас, 75
  рекурсивті, 85
  шаблондары, 88
функция-операциялар, 205
функционалдық класс, 211
функциональдық объект, 363

Х

хабарлама, 189

Ц

цикл, 33

Ш

шаблондардың стандартты
  кітапханасы, 318
шығыс итератор, 357

МАЗМҰНЫ

Алғы сөз 3

І БӨЛІМ. Құрылымдық программалау 5

1-тарау. C++ тілінің базалық құралдары 7
Тілдің құрамы 7
C++ тілінің алфавиті 10
Идентификаторлар 10
Түйінді сөздер 11
Операциялар таңбалары 11
Тұрақтылар 12
Түсініктемелер 14
C++ тіліндегі мәліметтер типтері 15
Мәліметтер типтерінің тұжырымдамасы 15
Мәліметтердің негізгі типтері 15
Программа құрылымы 19
Айнымалылар және өрнектер 22
Айнымалылар 22
Операциялар 25
Өрнектер 32
Құрылымдық программалаудың базалық конструкциялары 33
«Өрнек» операторы 35
Тармақталу операторлары 36
Цикл операторлары 40
Басқаруды беру операторлары 46
Нұсқауыштар мен жиымдар 48
Нұсқауыштар 48
Сілтемелер 56
Жиымдар 57
Қолданушылар анықтайтын мәліметтер типтері 65
Типтердің атын ауыстыру (typedef) 65
Тізбелер (enum) 66
Құрылымдар (struct) 67
Біріктірмелер (union) 70

2-тарау. Модульдік программалау 73
Функциялар 74
Функцияларды жариялау және анықтау 74
Ауқымды айнымалылар 78
Қайтарылатын мән 78
Функцияның параметрлері 79
Рекурсивті функциялар 85
Функцияларды асыра жүктеу 86
Функциялардың шаблондары 88
main() функциясы 91
Стандартты кітапхана функциялары 92
Препроцессор директивалары 99
#include директивасы 99
#define директивасы 100
Шартты компиляция директивалары 100
#undef директивасы 102
Идентификаторлардың әрекет ету аймақтары 103
Сыртқы жарияланымдар 104
Атау берілген аймақтар 106

3-тарау. Программаларды құру технологиясы 109
Программаны кодтау және құжаттау....109
Программаны жобалау және тесттен өткізу....117
Мәліметтердің динамикалық құрылымдары....122
Сызықтық тізімдер....124
Стектер....129
Кезектер....130
Бинарлы бұтақтар....132
Динамикалық құрылымдарды жиым көмегімен жүзеге асыру....136

I бөлімге арналған жаттығулар 138
Циклдік есептеу процестері....138
Бірөлшемді жиымдар....148
Екіөлшемді жиымдар....151
Бірөлшемді және екіөлшемді жиымдар....155
Құрылымдар....157
Нұсқауыштар....163
Қарапайым функциялар....164
Функциялар және файлдар....164
Тіркестермен және символдармен жұмыс істеуге арналған кітапхана функциялары....173
Функциялар шаблондары....178
Модульдік программалау....178
Мәліметтердің динамикалық құрылымдары....179

II БӨЛІМ. Объектіге бағытталған программалау....187

4-тарау. Кластар 192
Класты сипаттау....192
Объектілерді сипаттау....194
this нұсқауышы....195
Конструкторлар....196
Көшіру конструкторы....199
Кластың статикалық элементтері....200
Статикалық өрістер....201
Статикалық әдістер....201
Достас функциялар және кластар....202
Достас функциялар....202
Достас класс....203
Деструкторлар....204
Операциялардың асыра жүктелуі....205
Унарлы операциялардың асыра жүктелуі....206
Бинарлы операциялардың асыра жүктелуі....207
Меншіктеу операциясының асыра жүктелуі....207
new және delete операцияларының асыра жүктелуі....208
Типті келтіру операциясының асыра жүктелуі....211
Функцияны шақыру операциясының асыра жүктелуі....211
Индекстеу операциясының асыра жүктелуі....212
Класс элементтеріне нұсқауыштар....213
Класс құрамы бойынша ұсыныстар....215

5-тарау. Мұралау 216
Қол жеткізу кілттері....216
Қарапайым мұралау....217
Виртуалды әдістер....221
Кейінгі байланыстыру механизмі....223
Абстрактылы кластар....224

Көптік мұралау 225
Құрылымдар мен біріктірмелердің кластардан айырмашылығы 226

6-тарау. Класс шаблондары 228
Кластардың шаблондарын құру 228
Класс шаблондарын қолдану 236
Класс шаблондарының специализациясы 238
Шаблондардың артықшылықтары мен кемшіліктері 239

7-тарау. Аластамалы жағдайларды өңдеу 240
Аластамаларды өңдеудің жалпы механизмі 240
Аластамалардың синтаксисі 241
Аластамаларды айқындап атқару 243
Функцияның аластамалар тізімі 245
Конструкторлар мен деструкторлардағы аластамалар 247
Аластамалардың иерархиялары 248

8-тарау. Типтерді түрлендіру 249
С стиліндегі типтерді келтіру операциясы 249
const_cast операциясы 250
dynamic_cast операциясы 250
Жоғарылататын түрлендіру 251
Төмендететін түрлендіру 251
Сілтемелерді түрлендіру 253
Қиылысқан түрлендіру 254
static_cast операциясы 255
reinterpret_cast операциясы 256
Типті динамикалық түрде анықтау 257

9-тарау. Программалау бойынша ұсыныстар 259

II бөлімге арналған жаттығулар 266
Кластар 266
Мұралау 270
Класс шаблондары 278

III БӨЛІМ. Стандартты кітапхана 281

10 тарау. Ағымдық кластар 283
Стандартты ағымдар 285
Мәліметтерді форматтау 287
Жалаушалар және форматтаушы әдістер 287
Манипуляторлар 290
Ағымдармен мәлімет алмасу әдістері 292
Ағымдар қателіктері 298
Файлдық ағымдар 300
Тіркестік ағымдар 303
Қолданушы анықтаған ағымдар мен типтер 305

11 тарау. Тіркестер 307
Конструкторлар және тіркестерді меншіктеу 308
Операциялар 309
Функциялар 309
Меншіктеу және тіркес бөліктерін қосу 310
Тіркестерді түрлендіру 311
Ішкі тіркестерді іздеу 313
Тіркестердің бөліктерін салыстыру 316
Тіркестердің сипаттамаларын алу 317

12 тарау. Контейнерлік кластар 318
Тізбекті контейнерлер 320
Векторлар (vector) 323
Екіжақты кезектер (deque) 329
Тізімдер (list) 331
Стектер (stack) 336
Кезектер (queue) 337
Басымдылықтары бар кезектер (priority_queue) 338
Ассоциативті контейнерлер 341
Сөздіктер (map) 342
Дубликатты сөздіктер (multimap) 347
Жиындар (set) 348
Дубликатты жиындар (multiset) 349
Биттік жиындар (bitset) 350
Контейнерлерді қолдану мысалы 352

13 тарау. Итераторлар және функционалдық объектілер 356
Итераторлар 356
Кері итераторлар 359
Кірістіру итераторлары 361
Ағымдық итераторлар 361
Функционалдық объектілер 363
Арифметикалық функционалдық объектілер 364
Предикаттар 365
Терістеуіштер 366
Байланыстырғыштар 366
Функцияларға нұсқауыштар адаптерлері 367
Әдістер адаптерлері 371

14 тарау. Алгоритмдер 373
Тізбектерді модификацияламайтын операциялар 374
adjacent_find 374
count, count_if 375
equal 376
find, find_if 376
find_first_of 377
find_end 377
for_each 378
mismatch 378
search, search_n 378
Тізбектермен орындалатын модификациялаушы операциялар 379
copy, copy_backward 380
fill, fill_n 381
generate, generate_n 381
iter_swap, swap, swap_ranges 382
random_shuffle 382
remove, remove_if, remove_copy, remove_copy_if 383
replace, replace_if, replace_copy, replace_copy_if 385
reverse, reverse_copy 386
rotate, rotate_copy 386
transform 386
unique, unique_copy 388
Сұрыптаумен байланысты алгоритмдер 388
binary_search 389
equal_range 389
inplace_merge 390

lexicographical_compare....390
lower_bound, upper_bound....391
max, min....391
max_element, min_element....391
merge....392
next_permutation, prev_permutation....392
nth_element....394
partial_sort, partial_sort_copy....394
partition, stable_partition....395
sort, stable_sort....395
Жиындармен және пирамидалармен жұмыс істеу алгоритмдері....395
includes....396
set_intersection....397
set_difference, set_symmetric_difference....397
set_union....397
Жиындармен жұмыс істеу алгоритмдерін қолдану мысалы....398
make_heap....399
pop_heap....399
push_heap....399
sort_heap....400

15 тарау. Сандық есептеулерге арналған құралдар....401
Жалпыланған сандық алгоритмдер....401
accumulate....401
inner_product....402
partial_sum....402
adjacent_difference....403
valarray класы....403
Көмекші кластар....407

16 тарау. Стандартты кітапхананың басқа құралдары....411

III бөлімге арналған жаттығулар....414

Соңғы сөз....415

Қолданылған әдебиеттер....416

1-ҚОСЫМША. C++ тілінің операциялары....417

2-ҚОСЫМША. printf топтамасының функциялары үшін формат спецификациялары....419

3-ҚОСЫМША. Типтердің арифметикалық түрленулері....422

4-ҚОСЫМША. Стандартты кітапхананың тақырыптық файлдары....423

5-ҚОСЫМША. Тұрақтылар, макростар және стандартты кітапхананың мәліметтер типтері....425

6-ҚОСЫМША. Стандартты кітапхана функциялары....439
Тақырыптық файлдар құрамы....439
Функциялардың алфавиттік тізімі....445

7-ҚОСЫМША. ASCII кодтарының кестесі....471

8-ҚОСЫМША. Қолданылған терминдердің орысша (ағылшынша)-қазақша сөздігі....474

Алфавиттік нұсқауыш....478

Т. А. Павловская

C/C++
ЖОҒАРҒЫ ДЕҢГЕЙЛІ ТІЛДЕ
ПРОГРАММАЛАУ

Басуға **.**.2012 қол қойылды. Қаріп түрі «Times».
Офсеттік басылыс. Пішіні 70х100/16. Баспа табағы 31,5.
Таралымы **** дана. Тапсырыс №1140.

Тапсырыс берушінің дайын файлдарынан басылып шықты.

ЖШС РПБК «Дәуір», 050009,
Алматы қаласы, Гагарин д-лы, 93а.
E-mail: rpik-dauir81@mail.ru