ЖОҒАРЫ БІЛІМ

М. Мұқашева

Программалау

/C++ Builder 6/

ФОЛИАНТ

ЖОҒАРЫ  БІЛІМ

М. Мұқашева

Программалау

/C++ Builder 6/

Қазақстан Республикасы Білім және ғылым министрлігі жоғары оқу орындарының студенттеріне оқулық ретінде ұсынған

FOLIANT
БАСПАСЫ
Астана-2013

УДК 004. 43(075.8)
ББК 32.973 – 018.1я73
М 84

Пікір жазғандар:
Нәрішбаев А. – *техника ғылымдарының докторы, профессор;*
Адамов Ә. – *техника ғылымдарының докторы, профессор;*
Сауханова Ж. – *физика-математика ғылымдарының кандидаты, доцент*

Мұқашева М.
М 84 **Программалау /C++Builder 6/:** *Оқулық.* – Астана: Фолиант, 2013. – 456 б.

ISBN 978-601-292-597-5

Оқулықта C++ тілінде программалаудың негізгі ұғымдары, объектіге бағдарланған программалаудың қағидалары бойынша кластың жарияланууы, инкапсуляциялық, полиморфтық және қабылдаушылық қасиеттері жан-жақты көрсетіледі. Визуалды компоненттерді пайдаланып Windows қосымшаларын, қосымшаның анықтамалық жүйесін құру және ерекше жағдайларды өңдеу қарастырылады. Оқулықта жаттығулар, программалардың мәтіндеріне түсініктемелер беріледі.

Оқулық жоғары оқу орындарының «Математикалық және компьютерлік модельдеу», «Ақпараттық жүйелер», «Есептеу техникасы» және т.б. жаратылыстану-техникалық бағыттағы мамандықтарының студенттеріне арналған.

УДК 004. 43(075.8)
ББК 32.973 – 018.1я73

ISBN 978-601-292-597-5

Мазмұны

I БӨЛІМ. C++ BUILDER 6 ОРТАСЫНДА ПРОГРАММАЛАУ НЕГІЗДЕРІ

КІРІСПЕ

RAD (Rapid Application Development) технологиялар қатарына жататын C++ Builder 6 программалау ортасын C++ тілінде программалар жазу үшін қолданады. Бұл программалау ортасы Borland Software компаниясының өнімі болып саналады. C++ Builder 6 ортасының кейінгі нұсқаларын (C++ Builder 2007 бастап, соңғы 2010 жылы шығарылған C++ Builder XE нұсқасын қоса алғанда) CodeGear компаниясы шығарып жүр.

C++ Builder 6 программалау ортасы жоғары оқу орындары студенттеріне C++-те программалауды үйретуде тиімді және қолжетімді программалық құралдардың бірі болып есептеледі.

Оқулықтың құрылымдық программалауға арналған бірінші бөлімінде C++ программалау тілінің негізгі конструкциялары мен стандарт типтері және функцияларынан басқа, қолданбалы программаларды жазуда қажет болатын құрылымдық типтер де толығымен қарастырылды.

Объектіге бағдарланған программалаудың негіздері болып есептелетін инкапсуляция, полиморфизм және қабылдаушылық қағидаларының программалауда жүзеге асырылулары, оқулықтың екінші бөлімінде консолдық қосымшаларды пайдалану арқылы өздерінің бастапқы пайда болған күйлерінде берілді. Оқулықтың объектілік программалауға арналған осы екінші бөлімінде жолдар, ағындар мен файлдар туралы жан-жақты мәліметтер берілді.

Оқулықтың он екінші тарауында программадағы қателердің түрлері және оларды ерекше жағдайлар түрінде өңдеудің жолдары көрсетілді.

Оқулық программалауды студенттерге үйрету мақсатында жазылғандықтан, тақырыптар бойынша көптеген жаттығулар (программалар), бақылау сұрақтары мен өз бетінше орындауға арналған тапсырмалар қарастырылды. Келтірілген жаттығуларға сәйкес программалық кодтарға мүмкіндігінше түсініктемелер беріліп отырды. Оқулықта

келтірілген программалық кодтар стандарт C++ тілінде жазылды, сондықтан оларды стандарт C++-ті қолдайтын компиляторларда орындай беруге болады.

Жоғары оқу орындарының «математикалық және компьютерлік модельдеу», «ақпараттық жүйелер» мамандықтарындағы, «C++ тілінде объектіге бағдарланған программалау», «Программалау технологиясы» міндетті пәндері үшін жазылған бұл оқулықты «информатика», «есептеу техникасы», «автоматтандыру және басқару» және т.б. мамандықтардың программалау саласына қатысты пәндерін оқытуда альтернатив оқулық ретінде қолдануға болады.

I БӨЛІМ. C++ BUILDER 6 ОРТАСЫНДА ПРОГРАММАЛАУ НЕГІЗДЕРІ

- C++Builder 6 ортасының құрылымы
- C++Builder 6 консолдық қосымшалары
- C++ тілінің алфавиті және синтаксисі
- Деректердің типтері
- Функция
- Динамикалық құрылымдар

1-тарау. C++ BUILDER 6 ОРТАСЫ

1.1. C++Builder 6 ортасының құрылымы

1.1.1. C++ Builder 6 ортасының негізгі терезелері

C++Builder 6 программа құрудың кіріктірілген ортасы (Integrated Development Environment – IDE) қосымшаның интерфейсін жобалау, программалық кодын жазу, тексеру, жөндеу, орындау және т.б. көптеген әрекеттерді орындай алады. C++Builder 6 ортасының құрамына программалық код редакторы, компилятор, саймандар панелі, графиктік редактор, деректер қорымен жұмыс жасайтын саймандар және т.б. біріктірілген.

C++Builder 6 ортасын іске қосу үшін (компьютерде орнатылған болуы керек) *Пуск/Программы/Borland C++Builder 6/C++Builder 6* командалары орындалады. C++Builder 6 программалау ортасы іске қосылғаннан кейінгі терезелердің орналасуы келесі түрде болады (1.1-сурет):

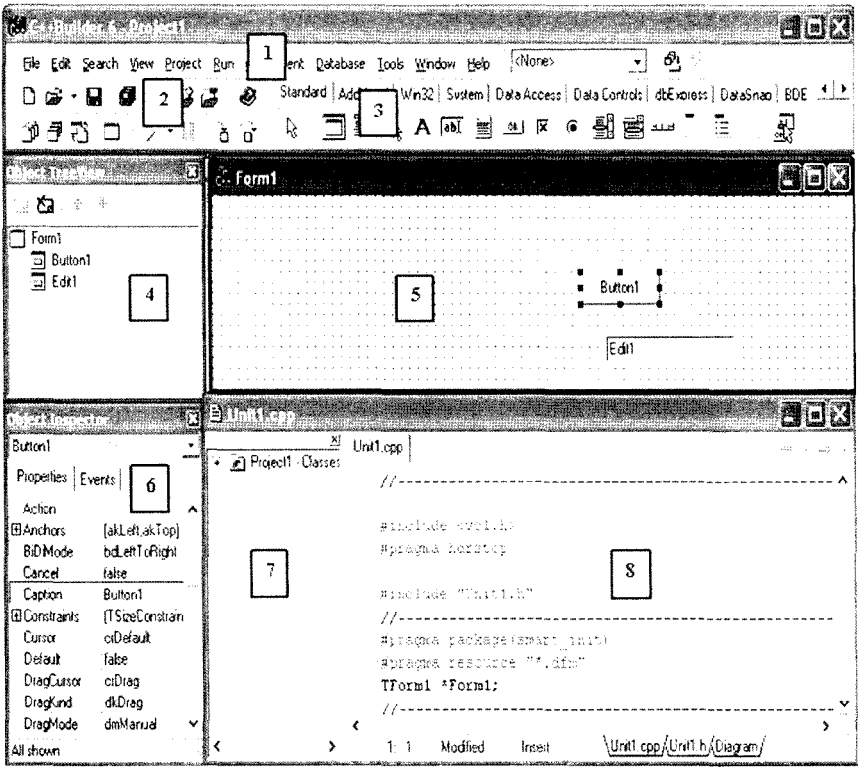
1. Негізгі мәзір терезесі;
2. C++Builder ортасын жылдам басқару батырмаларының терезесі;
3. Компоненттер палитрасының терезесі (Component Palette);
4. Объектілердің бағынышты орналасу терезесі (Object TreeView);
5. Форма терезесі (Form1);
6. Объектілер инспекторының терезесі (Object Inspector);

7. Кластарды қарау терезесі (Class Explorer);
8. Код редакторының терезесі (Unit1.cpp).

Негізгі мәзірде C++Builder программалау ортасында қосымша құрудың барлық кезеңін қамтитын және басқаратын командалар жинақталған.

C++Builder ортасын жылдам басқару терезесінде негізгі мәзірдің жиі қолданылатын командаларына сәйкес келетін жылдам басқару батырмалары, мысалы, Run (Орындау), View Unit (Модульдер тізімін қарау), View Form (Формалар тізімін қарау), Add to Project (Жобаға файлды қосу) және т.б. орналасады.

Компоненттер палитрасының терезесіндегі компоненттер палитрасында әдетте визуалдық кітапхана компоненттері (Visual Component



1.1-сурет. *C++Builder программалау ортасының интерфейсі*

Library – VCL) орналасады (9.1. қараңыз). Компоненттер палитрасының құрамы, жоғарысында орналасқан Standard, Additional, Win32, System және т.б. жапсырмаларға байланысты өзгеріп отырады. Бұл жапсырмалардың өздері де құрылатын қосымшаның түріне байланысты өзгеріп отырады, мысалы, кросс-платформалық қосымшаларды құрғанда (New->CLX Application), бұл терезеге кросс-платформалық кітапхана компоненттері (Component Library for Cross-Platform – CLX) орналасқан жапсырмалар қосылады.

Объектілердің бағынышты орналасу терезесінде қосымшаға қатысатын барлық объектілер бұтақ түрінде бағыну тәртібі (иерархия) бойынша орналасады. Бұл орналасуды C++ Builder ортасы автоматты түрде өзі жүзеге асырады.

Форма терезесінде қосымшаның терезелік интерфейсінің негізі құрылады, яғни қосымша интерфейсін құру осы форма терезесінен басталады. Қосымшаны құрғанда жаңа форманың терезесі автоматты түрде өзі пайда болады.

Объектілер инспекторының терезесі қосымшаға қатысатын объектілердің қасиеттерін (Properties) және оқиғаларын (Events) өзгерту үшін қолданылады. Қосымша объектілерімен жасалатын жұмыстардың көбі осы терезеде орындалады. Бұл терезеде сол мезетте ғана таңдалып тұрған объектінің, мысалы, Button1 (1.1-сурет) компонентінің қасиеттерімен, оқиғаларымен жұмыс жасау мүмкін болады.

Кластарды қарау терезесінде программаға қатысатын барлық кластар мен функциялар туралы мәліметтер беріледі.

Код редакторының терезесі кәдімгі мәтіндік редактордың қызметін атқарады, мұнда C++ тіліндегі программалар мәтіндері жазылады және жөнделетін болады.

1.1.2. Негізгі мәзір және оның командалары

Мәзірдің **File** бөліміндегі командалар көмегімен жаңа жобалар түрлерін таңдау, жобаны құру, сақтау, ашу-жабу, атын өзгерту және т.б. көптеген әрекеттерді орындауға болады (1-қосымша).

Негізгі мәзірдің **Edit** бөлімінде орналасқан командалар негізінен алмасу буферімен жұмыс жасауға арналған. Мысалы, код редакторының терезесінде белгіленген мәтін фрагментін немесе форма бетінде

белгіленген объектіні буферге көшіруге, басқа орынға апарып қоюға болады.

Search бөлімінің командалары код редакторының терезесінен немесе файлдан берілген мәтінді іздеу, мәтін фрагменттерін алмастыру секілді қызметтерді орындай алады.

Негізгі мәзірдің **View** бөлімінде жиі қолданылатын Project Manager, Object Inspector, Object TreeView, ClassExplorer, Units, Forms, Toolbars сияқты терезелерді экранға қоятын немесе экраннан алып тастайтын командалар орналасқан.

Project бөлімі жобамен жұмыс жасайтын командаларды біріктіреді. Бұл командалар көмегімен жоба құрамына файлдарды қосуды немесе керісінше алып тастауды, жоба туралы мәліметтерді қарауды, жобаның опцияларын тағайындауды орындауға болады.

Run бөліміндегі командалар қосымшаны компиляциялау және орындау немесе программаны бірден орындау, программаны әрбір жолы бойынша қадамдап орындау, орындалып тұрған программаны уақытша тоқтату және программаның орындалуын аяқтап, уақытша жадыны босату секілді әрекеттерді жүзеге асырады.

Негізгі мәзірдің **Component** бөлімінде жаңа компонентті Component Expert арқылы іздеуді ұйымдастыратын, жаңа компонентті немесе жаңа пакетті орнататын, компоненттер палитрасының конфигурациясын тағайындайтын сұхбат терезесін шақыратын және т.б. командалар берілген.

Database бөлімінің командалары деректер қорымен жұмыс жасайтын SQL Explore, Form Wizard, SQL Monitor программаларын шақыруға арналған.

Tools бөліміндегі командалар арнайы сұхбат терезелерінде программистің өз қажетіне қарай орта конфигурациясын, мысалы, компоненттер палитрасын, форма терезесінің дизайнын, код редакторының конфигурациясын және т.б. өзгертуіне мүмкіндік бере алады. Сондай-ақ, осы терезеде компиляция жасаудың опцияларын тағайындайтын Debugger Options сұхбат терезесін, Database Desktop программасын, Microsoft Visual C++ жобаларының файлдарын C++Builder жобаларының файлдарына айналдыру, қосымша үшін ресурс файлдарын, таңбаларды (icon) және т.б. құратын және өзгертетін Image Editor редакторын шақыру командалары орналасқан.

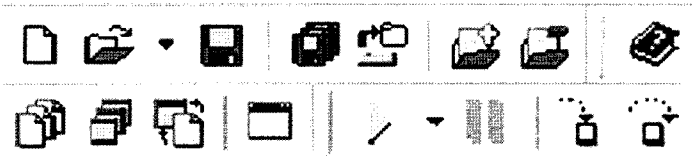
Негізгі мәзірдің **Windows** бөлімінде сол жобадағы ашық тұрған терезелердің тізімдері орналасады, тізімнен таңдау арқылы қалаған терезеге ауысуға болады.

Help бөлімінің командалары C++ Builder ортасының анықтама жүйесін шақыру, арнаулы web-сайттардан ақпараттар алу және т.б. әрекеттерді орындайды.

Негізгі мәзірдің бөлімдеріндегі кейбір командалардың сол жақ тұсында жылдам басқару батырмасының таңбасы, ал оң жағында жылдам шақыру пернесі орналасады, мысалы, File -> Save All (Shift+Ctrl+S) командасы.

1.1.3. C++Builder ортасын жылдам басқару батырмалары

C++Builder ортасын жылдам басқару батырмаларының терезесінде (1.2-сурет) жоба құру барысында жиі қолданылатын негізгі мәзір бөлімдерінің командалары орналасады. Бұл командалар батырмалар түрінде беріледі және кейбіреулері жылдам шақыру пернелерінің комбинациясы түрінде де анықталған.



1.2-сурет. *C++Builder ортасын жылдам басқару батырмалары*

Бұл терезедегі батырмаларға курсор нұсқауышын жақындатқанда, оның қандай командаға сәйкестігін көрсететін жылжымалы мәтінді (всплывающая подсказка) көруге болады. Бұл жылдам басқару батырмалары келесі қызметтерді атқарады:

New –	New Items жаңа жобаны ашу сұхбат терезесін шақырады (1-қосымша).
Open –	Жобаға қатысатын файлдарды сұхбат терезесінде ашуды жүзеге асырады.

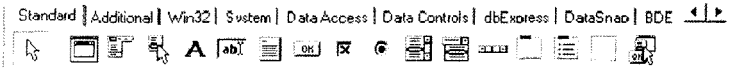
Save (Ctrl+S) –	Ағымдағы жобаны сақтау, егер жоба бірінші рет сақталатын болса, онда жоба файлы мен модуль файлының аттары сұралады.
Save All (Shift+Ctrl+S) –	Ағымдағы жобаның жоба файлын және модуль файлын қатарынан немесе бәрін сақтау.
Open Project (Ctrl+F11) –	Сұхбат терезесінде тек қана *.bpr немесе * .bpk кеңеймелі жоба файлдарын ашуды орындайды.
Add file to project (Shift+F11) –	Add to Project сұхбат терезесінде таңдалған файлды жоба құрамына кіргізуді орындайды.
Remove file from project –	Remove from Project сұхбат терезесінен таңдалған файлды жоба құрамынан алып тастайды.
View Unit (Ctrl+F12) –	View Unit сұхбат терезесін экранға шығару, бұл терезеде жобадағы барлық модульдердің тізімі шығады.
View Form (Shift+F12) –	View Form сұхбат терезесін экранға шығару, бұл терезеде жобадағы барлық формалардың тізімі шығады.
Toggle Form/Unit (F12)	Форма немесе модуль терезелерін алдыға қарай шығаруды орындайды.
New Form	Жаңа форма терезесін ашады.
Run (F9)	Программаны бірден орындауға жібереді.
Pause	Орындалып тұрған программаны уақытша тоқтата тұрады.
Trace Into (F7)	Программаны қадамдап орындау, мұнда подпрограммалардың әрбір жолы жеке, бір қадам ретінде орындалады.
Step Over (F8)	Программаны әрбір жолы бойынша қадамдап орындау (F8 пернесі басылып отырады), мұнда бір подпрограмма бір қадам ретінде орындалады.

1.1.4. Компоненттер палитрасы

C++Builder ортасында құрылатын қосымшаның интерфейсін жобалау үшін көптеген визуалды компоненттер қолданылады. Бұл компоненттердің пиктограммалары компоненттер палитрасында орналасқан парақтарға кіргенде пайда болады. Компоненттер палитрасы екі жолдан тұрады, бірінші жолында парақтардың атаулары, мысалы, Standard, Additional, Win32 және т.б. орналасса, ал екінші жолында әрбір параққа мағынасы мен қызметіне қарай топтастырылған компоненттерге сәйкес пиктограммалар тізімі шығып тұрады.

Әдетте C++Builder ортасы іске қосылғанда Standard парағы ашық тұрады (1.3-сурет), яғни төмендегі пиктограммаларға сәйкес компоненттер осы Standard парағында жинақталған, егер Additional парағына шертілсе, онда осы парақта жинақталған компоненттер пиктограммаларын алуға болады (1.4-сурет). Компоненттер палитрасында орналасқан компоненттер VCL (Visual Component Library) кітапханасына тиесілі болып есептеледі.

1.3-сурет. *Компоненттер палитрасының Standard парағының компоненттері*



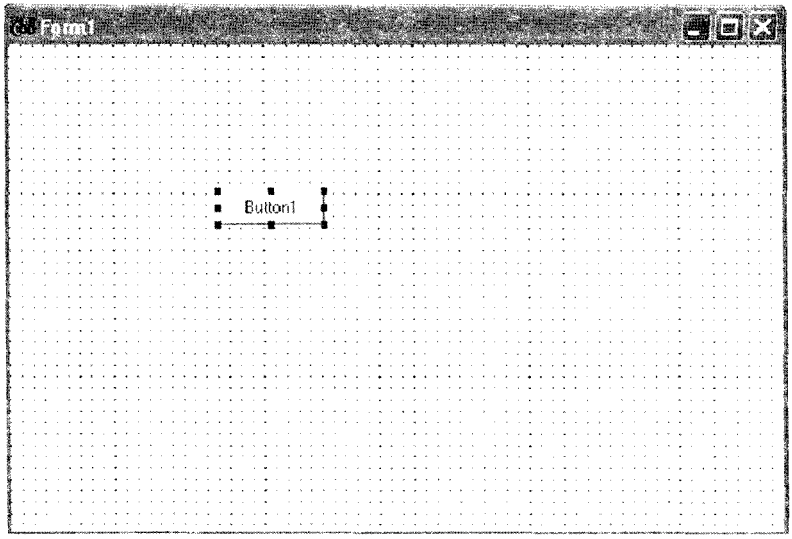
1.4-сурет. *Компоненттер палитрасының Additional парағының компоненттері*

Компоненттер палитрасындағы жолдарда барлық парақтар немесе барлық пиктограммалар бірдей көрініп тұрмайды, оларды көру үшін сол жолдардағы оңға немесе солға бағытталып тұрған нұсқағыштарды пайдаланады. Компоненттер палитрасынан таңдалған қажет компонентті формаға орналастыру үшін алдымен сол компоненттің пиктограммасына шертіледі, сонан соң барып тышқан нұсқағышын формадағы қажет орынға әкеліп тағы да шертеді, сонда формада сәйкесінше компоненттің номерленген көшірмесі пайда болады, мысалы,

келесі пиктограммаға сәйкес компонент формада Button1 түрінде (1.5-сурет) орналасады. Қажет компонентті формаға орналастырудың тағы бір жолы – оның пиктограммасына екі рет шерту, сол кезде компоненттің көшірмесі форманың дәл ортасында пайда болады.

Компоненттер палитрасы парақтарында орналасқан компоненттер өте көп және олардың қай парақта орналасқанын, пиктограммаларын есте ұстау қиынға соғады, мұндай жағдайда компонентті іздеу үшін жылжымалы мәтінді пайдалану ыңғайлы болады, ол үшін тышқанның нұсқағышын қажет пиктограммаға апарады, сол кезде пиктограмманың тұсында жылжымалы мәтін түрінде оның аты көрінетін болады.

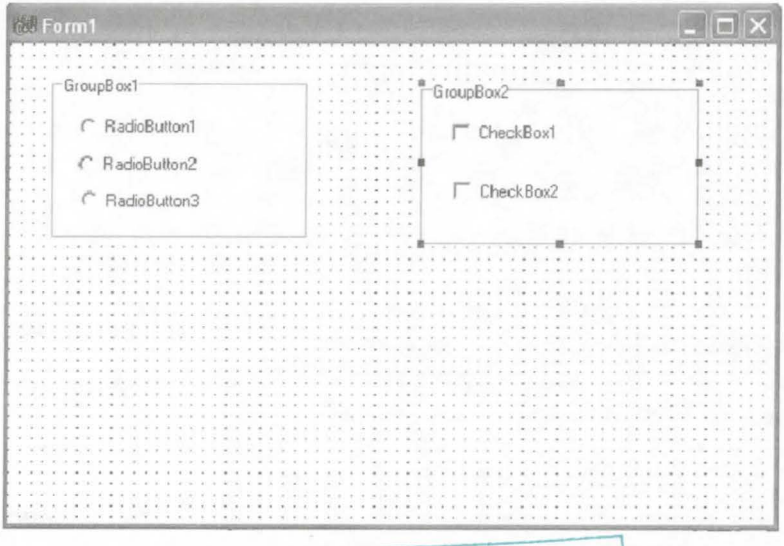
Сондай-ақ, компоненттер палитрасының контекстік мәзіріндегі (компоненттер палитрасында тышқанның оң жақ батырмасын шерту арқылы шақырылады) Tabs бөлімінде барлық парақтардың атаулары алфавит бойынша орналасады, сол жерден қажет парақты жылдам табуға болады. Компонент туралы толық мәлімет алу үшін оның пиктограммасын белгілеп, F1 пернесін басса жеткілікті, C++Builder ортасының анықтамалығы іске қосылады, мұндағы мәтіндерде компонент атауының алдына «T» әрпі қосылады, бұл кластардың атына қосылатын префикс болып табылады.



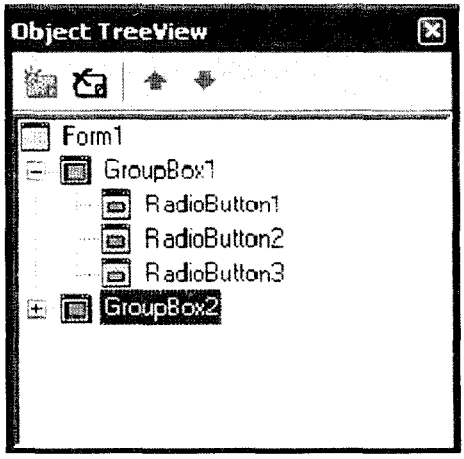
1.5-сурет. *Формаға Button компонентін орналастыру*

1.1.5. Объектілердің бағынышты орналасу терезесі (Object TreeView)

Қосымшалар құру барысында кейбір компоненттерді басқаларының ішіне орналастыру қажеттілігі туады, яғни мұнда сыртқы компонент контейнердің ролін атқарады. Осы компоненттердің немесе объектілердің қалай орналасқанын, бір-біріне қалай бағынатынын Object TreeView терезесінде қарауға болады, егер компонентке басқа бір компоненттер бағынатын болса, оның атының тұсында «+» белгісі болады, егер оған шертілетін болса, ол әрі қарай ашылып, бағынышты компоненттердің орналасуын көрсететін болады, мысалы, контейнер компоненттің ролін атқаратын GroupBox1 компоненті үш RadioButton1, RadioButton2, RadioButton3 компоненттерін қамтитын болса (1.6-сурет), онда олардың Object TreeView терезесіндегі орналасуы 1.7-суреттегідей болады. Формада орналасқан барлық компоненттер үшін форманың өзі ең сыртқы контейнер компонент болып табылады. Object TreeView терезесінің бұл қызметін, әсіресе деректер қорымен жұмыс жасайтын қосымшалар құруда тиімді пайдалануға болады.



1.6-сурет. *Формада компоненттердің өзара орналасуы*



1.7-сурет. *Компоненттердің өзара бағынышты орналасуы*

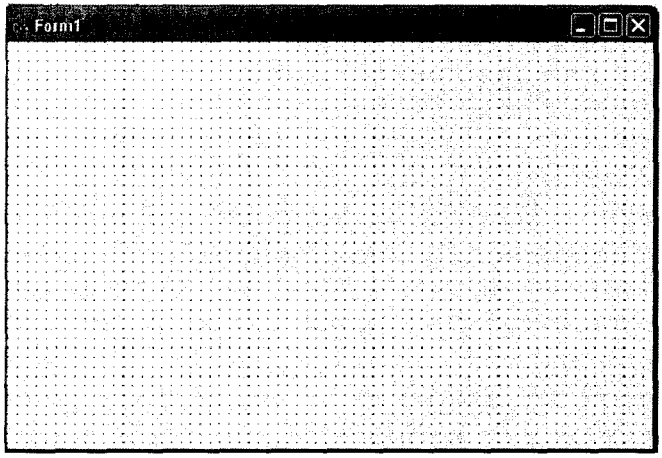
1.1.6. Форма терезесі

C++ Builder ортасында құрылатын қосымшаның орындалуы кезінде экранға шығатын визуалдық бөлімінің негізін Form компоненті құрайды. C++ Builder ортасы жүктелгенде, құрылатын жобаға (Project1) сәйкес форма терезесі (Form1) өзі автоматты түрде пайда болады (1.8-сурет).

Форма терезесі – бұл Windows қосымшаларына тән тақырыптық зонасы, жүйелік мәзір батырмасы, басқару батырмалары және таза жұмыстық облысы бар, өлшемі өзгеріп отыратын терезе. Жобаға тағы да форма терезелерін қосу үшін File → New → Form командасы орындалады.

Қосымшада бірнеше форма болған жағдайда олардың кез келген біреуі (әдетте бұл Form1) негізгі немесе бас форманың ролін атқарады және ол программа орындалған кезде экранға бірінші болып шығады, ал негізгі форманың ролін кейін тұрған Form2, Form3, т.б. біреуі атқару керек болған жағдайда жоба программасының мәтініндегі формалардың құрылу тәртібі өзгертіледі.

Форма терезесімен жұмыс жасау үшін оны TForm класының өкілі болып табылатын Form компоненті ретінде қарастырады.



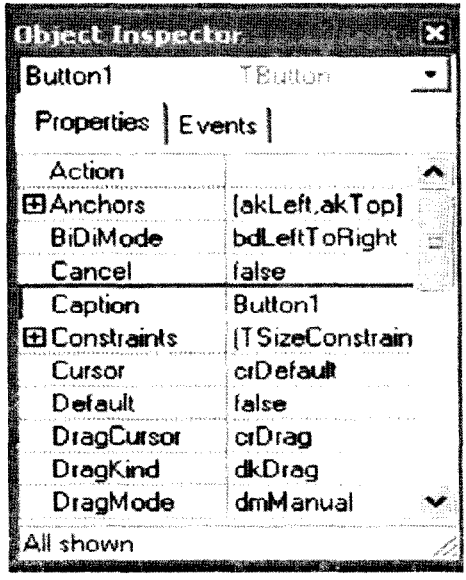
1.8-сурет. *Форма терезесі*

1.1.7. Объектілер инспекторының терезесі (Object Inspector)

Қосымшада пайдаланылатын объектілердің немесе компоненттердің қасиеттерін өзгерту екі түрлі жолмен – объектілер инспекторының (Object Inspector) терезесінде немесе программалау жолымен жасалады.

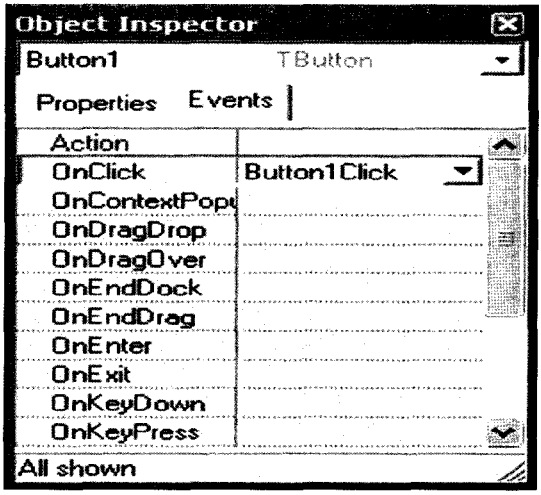
Мұндағы бірінші тәсіл өте қолайлы болып келеді. Объектілер инспекторы терезесінің бірінші жолында формадан сол мезетте таңдалған компоненттің аты, мысалы, Button1, оның жататын класы, мысалы, TButton1 шығып тұрады. Осы жолдағы ▼ белгісіне шерткенде, Object Inspector терезесінде көруге болатын объектілер тізімі шығады (1.9-сурет). Егер де формадан бірде-бір элемент таңдалмаған болса, онда бұл бірінші жолда форманың өзінің аты Form1 деп шығып тұрады. Object Inspector терезесінің келесі, екінші жолында екі жапсырма Properties (қасиеттер) және *Events* (оқиғалар) орналасқан.

Мұндағы Properties парағының сол жақ бағанында компоненттің (бірінші жолда аты шығып тұрған) қасиеттері, ал оң жақ бағанында сол қасиеттерге сәйкес мәндері шығып тұрады, мысалы, Button1 компонентінің Caption қасиетінің мәні «Button1» деген мәтін болып тұр, қажет болса қолданушы мәтінді сол жерден өзгерте алады.

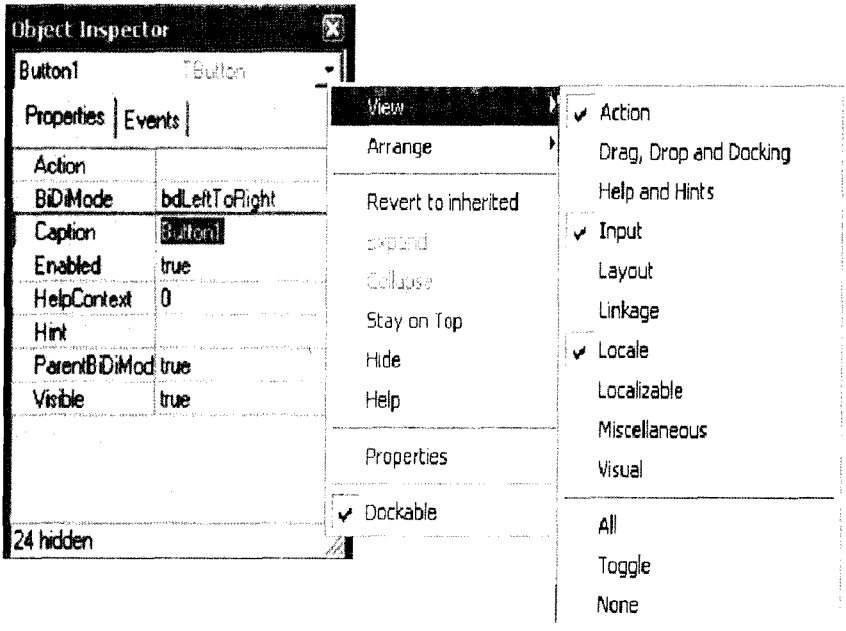


1.9-сурет. *Object Inspector терезесі*

Қасиеттердің тұсындағы «+» таңбасы оның әрі қарай тағы да ашылатынын білдіреді. *Events* парағының сол жақ бағанында компонент оқиғаларының аттары, ал оң жақ бағанында сол оқиғаға сәйкес функцияның аты орналасады, мысалы, Button1 компонентінің OnClick оқиғасына Button1Click функциясы тағайындалған (1.10-сурет), яғни Button1 батырмасына шерткенде, сол функцияда жазылған кодқа сәйкес әрекет орындалатын болады. Оқиғаға сәйкес келетін функциялар бірнеше болғанда, олардың тізімін ▼ белгісіне шерту арқылы қарап, тізімнен таңдауға болады, егер бірде-бір функция болмаса тізім бос болады. Егер оқиғаның тұсындағы функцияның атына шертілетін болса, онда курсор, код редакторының терезесіндегі сол функцияның денесіне барып түседі, ал функция болмаған жағдайда бос орынға шертілетін болса, сол оқиғаға сәйкес функция шаблоны ашылады. Object Inspector терезесінің контекстік мәзіріндегі (тышқанның оң жақ батырмасын шерткенде шығады) View бөлімінде компоненттердің қасиеттерін біріктіретін категориялар берілген, олардың тұсындағы индикаторды пайдаланып бұл категорияларға сәйкес қасиеттерді терезеден алып тастауға немесе қайтадан қоюға болады, мысалы, 1.11-суретте Action,



1.10-сурет. *Object Inspector терезесі*



1.11-сурет. *Object Inspector терезесіне өзгеріс жасау*

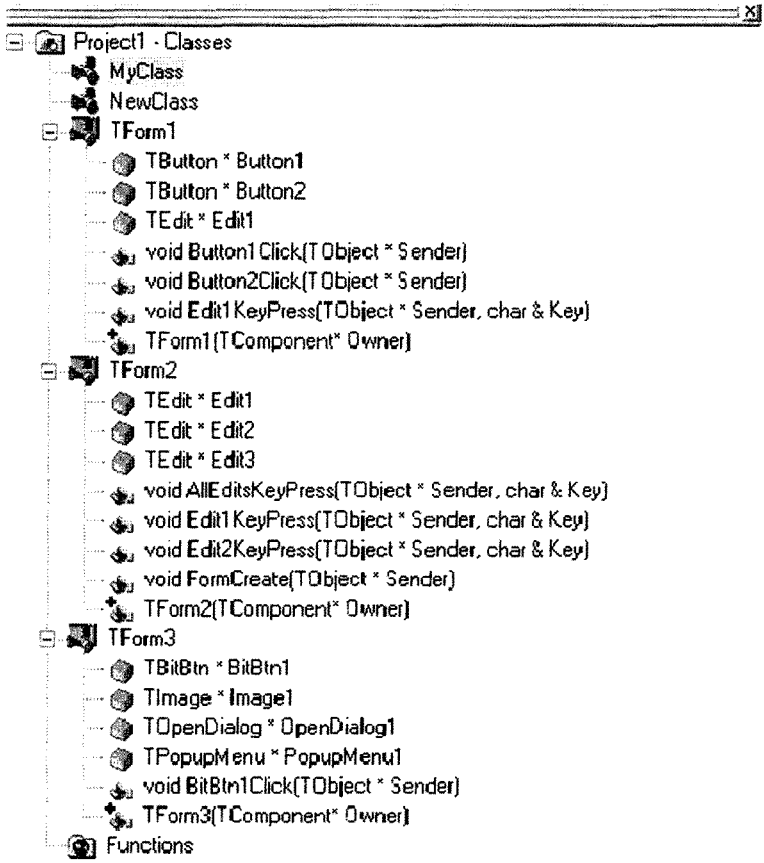
Input Locale категорияларына жататын қасиеттер ғана қалдырылған, ал терезенің төменгі жағында 24 қасиеттің жасырынып, көрінбей тұрғаны хабарланады (24 hidden).

Қасиеттердің бәрін қалпына келтіру үшін View ->All командалары орындалады. View ->Toggle командасы таңдалған категорияның қасиеттерін көрінбейтін қылады немесе керісінше таңдалмаған категорияның қасиеттерін терезеде қалдыратын болады. Контекстік мәзірдің Arrange ->byCategory командасы қасиеттер мен оқиғаларды категориялары бойынша топтауды орындаса, Arrange->byName оларды терезеде алфавит бойынша орналастырады.

1.1.8. Кластарды қарау терезесі (Class Explorer)

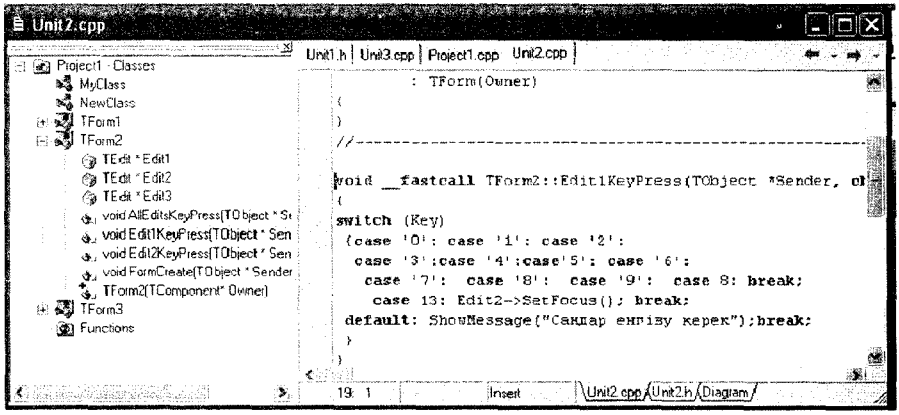
Class Explorer терезесі код редакторының терезесінде орналасады (1.1-суреттегі 7-номер). Бұл терезеде жобаны құру барысында қосылып отыратын кластар және функциялар тіркелін отырады. Бұл терезе көбінесе жобада көптеген формалар, көптеген модульдер (юниттер) болған жағдайда жобадағы кез келген кластардың сипаттамаларын немесе функцияларды тез табуға көмектеседі. Мысалы, жобаға (Project1) кіретін кластардың барлығы: MyClass, NewClass – қолданушының өзі құрған кластары, TForm1, TButton1, TForm2, т.б. стандарт кластар және барлық функциялар Class Explorer терезесінде көрініп тұрады (1.12-сурет).

Функцияларға байланысты ескерілетін бір нәрсе, мұнда тек Unit*.h- негізгі файлда немесе немесе Unit*.cpp- орындалатын файлда прототиптері (тақырыптары) анықталған функциялар ғана Class Explorer терезесінде көрініп тұрады, ал функцияның Unit* cpp файлда прототипсіз жай коды ғана берілетін болса, онда ол терезеде де болмайды. Терезедегі кез келген класқа екі рет шерткенде, код редакторының терезесінде сәйкес файлдағы (Unit*.h) сол кластың сипаттамасын көруге болады, ал егер функция тақырыбына шертілсе, онда курсор Unit*.cpp файлдағы функцияның кодына түседі (1.13-сурет). Бұл Class Explorer терезесін, терезедегі ☒ белгісіне шертіп, жауып тастауға болады, ал қажет болған жағдайда View->Class Explorer командаларын орындау арқылы қайтадан ашуға да болады.



1.12-сурет. *Class Explorer терезесіндегі жоба кластары мен функциялары*

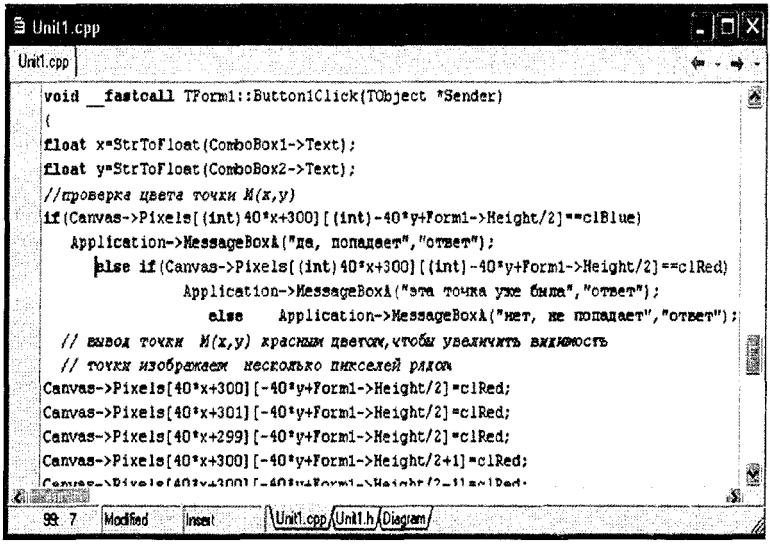
Class Explorer терезесіндегі кластың контекстік мәзіріндегі Go to Declaration командасын орындағанда да, жоғарыда жазылғандай, екі рет шерткендегі сияқты сол кластың негізгі файлдағы сипаттамасы табылады, ал мұны функция үшін орындаса, онда негізгі файлдағы функция прототипі табылады. Сондай-ақ, терезедегі функциялар үшін ғана Go to Implementation командасы орындалады, бұл команда орындалатын файлдағы функцияның орындалатын бөлігін, яғни кодын тауып береді. Контекстік мәзірдің Class Hierarchy командасы басқа терезеде кластардың бағынышты орналасуын ынғайлы түрде көрсетеді.



1.13-сурет. *Class Explorer терезесінде таңдалған функция кодының код редакторы терезесіндегі мәтінін табу*

1.1.9. Код редакторы

C++Builder программалау ортасындағы негізгі терезелердің бірі – бұл код редакторының терезесі. Бұл терезеде негізінен қосымшаның коды жазылады. Терезедегі кодты немесе программа мәтінін жазуда бірнеше стильдер сақталынады, мысалы, C++-тегі «void», «class», «public», «int», «float» және т.б. қызметші сөздер қара қалың бояумен жазылып тұрады, яғни мұндай қызметші сөздер дұрыс жазылса, онда олар бірден автоматты түрде қалың қара шрифтке өзгереді, сол сияқты «#» белгісінен басталатын директивалар жасыл түспен болектеніп тұрады, ал көк түспен түсіндірме-комментарийлер ажыратылады (1.14-сурет). Код редакторы да кәдімгі мәтіндік редактор сияқты бірнеше парактардан тұра алады. Әдетте, жоба ашылғанда код редакторының терезесінде негізгі формаға сәйкес модульдің (мысалы, Form1 болса, оған сәйкес Unit1.cpp) файлындағы мәтін шығып тұратын парақ ашық тұрады, ашық тұрған парақтағы файлдың аты терезенің жоғарысындағы немесе төменгі жағындағы жапсырмада көрініп тұрады (1.14-сурет). Бұл парақтан басқа терезенің төменгі жағында тағы да, екі парақтың Unit1.h және Diagram парақтарының жапсырмалары көрініп тұрады. Unit1.h жапсырмасына шертілсе, код редакторының терезесінде осы файлда сақталған мәтін парағы ашылады. Diagram



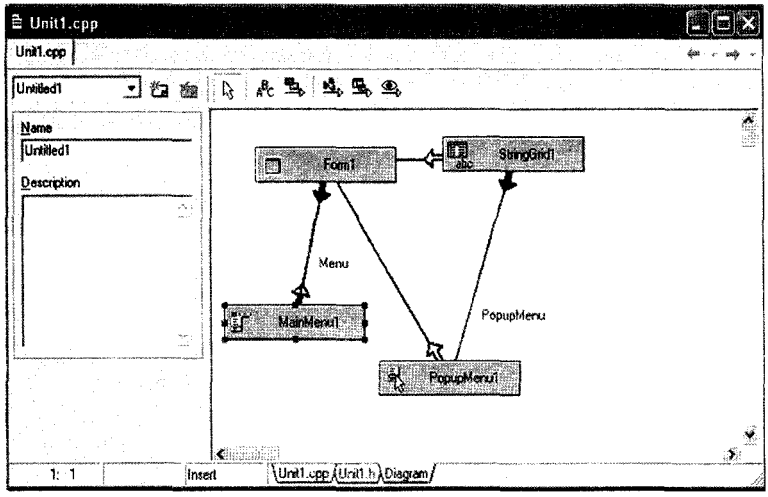
1.14-сурет. *Код редакторының терезесі*

парағында қосымша формаларындағы компоненттердің өзара байланысын көрнекі түрде қарауға болады (1.15-сурет).

Ол үшін Object TreeView терезесіндегі компоненттерді диаграмма терезесіне тышқанмен сүйретіп апарады, сол кезде байланыстырушы сызықтар өздері пайда болады, бұл әсіресе деректер қорымен жұмыс жасайтын элементтер арасындағы байланыстарды қарауда тиімді болып табылады.

Код редакторы терезесінің төменгі жағындағы (1.14-сурет) қалыпкүй қатарындағы «:» белгісі арқылы ажыратылған сандар, курсор тұрған жолдың және позицияның номерлерін көрсетеді, «Modified» – терезедегі мәтіннің өзгертілгенін білдірсе, «Insert» редакциялау режимін көрсетеді.

Код редакторының терезесінде C++Builder ортасының анықтама файлдарын пайдалану мүмкіндігі де қарастырылған. Ол үшін код редакторының терезесіндегі программа мәтініндегі сөздерге (бұған түсіндірмелер немесе комментарийлер кірмейді) курсорды қойып, F1 пернесі басылатын болса, онда сол сөзге қатысты C++Builder анықтамалығы бөлімдерінің тізімі шығады, тізімнен таңдап, қалаған анықтамалық бөлімді қарауға болады.



1.15-сурет. *Код редакторының Diagram парағы*

1.2. C++Builder 6 жобалары

1.2.1. C++Builder-дегі қосымша түрлері. Қосымша жобасын құру

C++Builder ортасында қосымшалар құруда негізінен екі ұстаным: форма графиктік конструкторын және код редакторын пайдалану қолданылады. Осы екі ұстанымды пайдалана отырып, C++Builder ортасында түрлі қызметтерді орындай алатын, өзіндік ерекшеліктері бар түрліше қосымшалар құруға болады. C++Builder ортасында құрылатын жобаның түрі осы қосымшаға байланысты анықталады. Олардың кейбіреулері төменде берілген:

– Windows қосымшалары: консолдық және терезелік қосымшалар;

– Динамикалық түрде қосылатын (DLL,Dynamic Link Libraries) кітапханалар;

– Қолданушының компоненттері және пакеттері;

– Кроссплатформалы (CLX, Component Library for Cross Platform) қосымшалар;

– COM (Component Object Model) және ActiveX басқарушы элементтер;

– Клиент/сервер архитектуралы қосымшалар және т.б.

Бұл тізім C++Builder ортасы жетілдірілген сайын жаңа мүмкіндіктермен толықтырылып отырады.

Бұл қосымшалардың барлығы да белгілі бір жобалардың негізінде құрылады. «Жоба» деп әдетте программаның C++-тегі коды жазылатын, форманың сипаттамалары, жобаның параметрлері сақталатын және т.б. көптеген қосымшаға қажет файлдардың жиынтығын айтады.

Құрылатын қосымшаның түріне байланысты жоба құрамына кіретін файлдардың саны да өзгеріп отыруы мүмкін, бірақ кейбір жекелеген файлдар барлық қосымшаларда да міндетті түрде қатысатын болады.

C++Builder программалау ортасында құрылатын Windows қосымшалары терезелік немесе консолдық қосымшалар – бірнеше файлдарды біріктіретін бір жобаны (мысалы, Project1) құрып және оны компиляциялау нәтижесінде алынатын, орындалатын *.exe кеңеймелі файл болып табылады.

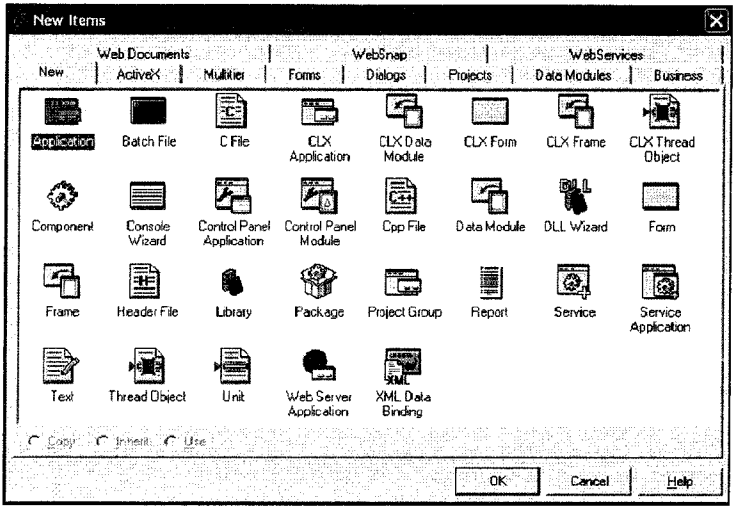
C++Builder ортасында Windows қосымшасының жаңа жобасын құру үшін негізгі мәзірден File→ New → Application командалары орындалады. Мұндай жолмен құрылған қосымшаларда кәдімгі Windows қосымшаларындағы секілді терезелер, басқарушы батырмалар, мәзірлер және т.б. элементтер болады.

Модульдік программалаудың басты жетістіктерінің бірі – қосымшаға динамикалық жолмен қосылатын DLL (Dynamic Link Libraries) модульдерді немесе кітапханаларды пайдалану C++Builder-де қарастырылған. DLL кітапханаларды пайдалану өте кең тараған, мысалы, Windows операциялық жүйесі, C++Builder ортасының өздері де көптеген DLL кітапханаларды пайдаланатыны белгілі. DLL кітапханалар орындалатын файл түріндегі кодтар болып табылады, бірақ олар өздігінен орындала алмайды, яғни олардың жадыға жүктелуін және орындалуын қамтамасыз ететін басқа файлдар болады, мысалы, бұл қосымшаның орындалатын *.exe файлы болуы мүмкін. DLL кітапханаларды пайдаланудағы басты себеп, бұл қосымшаның орындалуы кезіндегі жадыны үнемдеу болып табылады, яғни бұл кітапханалардың файлдары, егер олар қосымшаның орындалуына қажет болса, сонда ғана жадыға

жүктеледі, ал қажет болмаған жағдайда жадыны босатады. C++Builder ортасында DLL кітапхана үшін жаңа жоба құрудың тиімді жолы DLL Wizard (DLL шебері) құралын пайдалану, оны шақыру үшін негізгі мәзірдің File→New→Other командаларын орындағанда шығатын «New Items» (1.15-сурет) терезесінің New парағындағы DLL Wizard таңбасына шертіледі.

C++Builder құрамында көптеген компоненттер бар екені белгілі, бірақ, соған қарамастан, кейбір жағдайларда қолданушының өзінің талаптарын қанағаттандыратын компонентті құруын қажет ететін жағдайлар да кездеседі. Бұл жаңа компоненттер бұрыннан бар компоненттер кластарын түпкі класс ретінде пайдаланып жасалатын тума кластар болып табылады, яғни сол түпкі кластардың қасиеттерінің, әдістерінің және оқиғаларының кодтарына өзгерістер жасау арқылы тума кластар түріндегі жаңа компоненттер алынады. Жаңа компонент үшін бұрыннан бар түпкі кластарды пайдалану бұл жобаны құру уақытын жылдамдатады және қателердің барынша аз болуына септігін тигізеді. Ол жаңа компоненттерді де басқа компоненттер сияқты C++ Builder ортасының компоненттер палитрасына орналастырып қосымшалар құрғанда пайдалана беруге болады. Бұл жаңа компоненттерді құру – қолданушыдан объектіге бағдарланған программалаудың негізгі қағидаларының бірі қабылдаушылық туралы және де кластардың қасиеттерінің, әдістерінің және оқиғаларының қалай құрылатындығы туралы білімдерді қажет ететін күрделі процесс. Қолданушының компоненттерін құру үшін жаңа жобаны ашу негізгі мәзірдің File→ New → Other командаларын орындағанда шығатын «New Items» (1.16-сурет) терезесінің New парағындағы Component таңбасын таңдаудан басталады.

Пакеттер (Packages) – бұл визуалдық компоненттер мен басқа да объектілердің, функциялардың және т.б. кітапханаларын біріктіре отырып қамтитын, қосымшаға динамикалық түрде қосылатын DLL модульдің немесе кітапхананың арнайы бір түрі. Көп жағдайларда орындалатын файлдардың бірі емес бірнешеуі болады және де оларда бірдей визуалдық компоненттердің, функциялардың және т.б. кітапханалары пайдаланылады, демек сол кітапханаларды әрбір орындалатын файлға жеке-жеке қосқаннан, бір пакетті бәрінің ортақ



1.16-сурет. *New Items терезесінде жаңа жоба түрін таңдау*

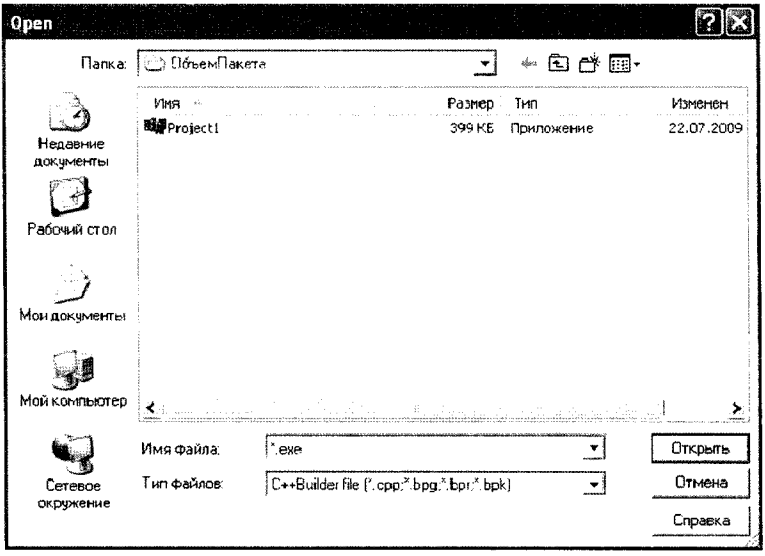
пайдаланғаны әлдеқайда тиімді болады, яғни бұл жобадағы кодтардың артық қайталанбауын қамтамасыз етеді. Пакеттердің үш түрі болады:

– «жобалау кезеңінің» (designtime-only, только времени проектирования) пакеттері;

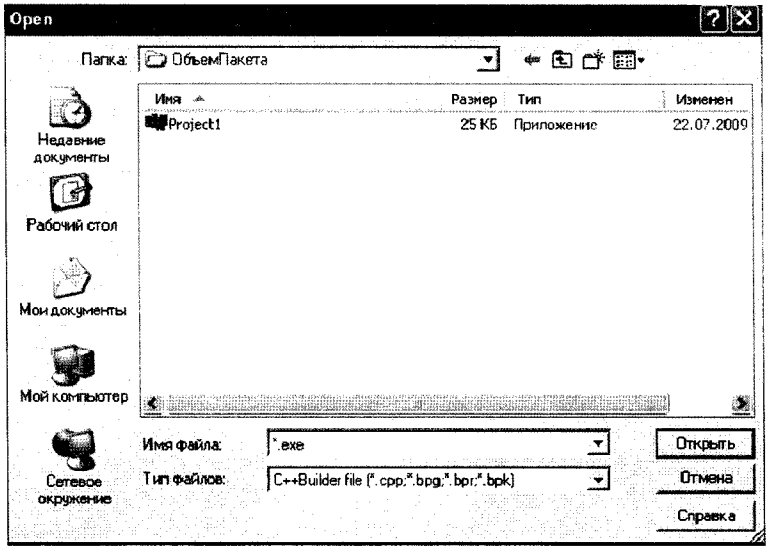
– «орындалу кезеңінің» (runtime-only, только времени выполнения) пакеттері;

– жобалау-орындалу кезеңдерінің (designtime-runtime) пакеттері.

Олардың бір-бірінен айырмашылығы мынада: «жобалау кезеңінің» пакеттері C++ Builder ортасының өзінде ғана қолданылады, мысалы, палитраға компоненттерді орнатуда және т.б., бұл пакеттер қосымшаның орындалуы кезінде қолданылмайды; ал «орындалу кезеңінің» пакеттері, аты айтып тұрғандай, қосымшаның орындалуы кезінде қолданылады. Мысалы, C++ Builder-дің визуалдық компоненттер кітапханасы осындай пакеттерде орналасқан. Бұл орындалу кезеңінің пакеттерінде программалық кодтардың көп бөлігі сақталады, сондықтан оларды қолдану қосымшаның орындалатын *.exe файлының көлемін едәуір азайтады. Егер «орындалу кезеңінің» пакеттері қолданылмаса, ең қарапайым деген қосымшаның жеке орындалатын (автономно) файлының көлемі 399 Кбайтты (1.17, а-сурет) құрайды, ал орындалу кезеңінің пакеттері колданылса, оның көлемі 25 Кбайтқа (1.17, б-сурет) дейін азаяды.



1.17-сурет. *а) «Орындалу кезеңінің» пакеттері қолданылмаған *.exe файл*



1.17-сурет. *б) «Орындалу кезеңінің» пакеттері қолданылған *.exe файл*

Егер де қосымшада орындалу кезеңінің пакеттері қолданылатын болса, онда қосымшаны пайдаланушының компьютерінде міндетті түрде сол орындалу кезеңінің пакеттері орнатылған болуы тиіс, онсыз қосымша толық жұмыс жасай алмайтын болады. Қосымшаны пайдаланушының компьютерінде сол, орындалу кезеңінің пакеттері бір-ақ рет орнатылса жеткілікті, сонан кейін оны пайдаланатын көлемі шағын ғана орындалатын файлдарды желімен жібере салуға болады.

Пакет құрудың жаңа жобасы негізгі мәзірдің File→New→Other командаларын орындағанда шығатын «New Items» (1.15-сурет) терезесінің New парағындағы Package танбасына шертуден басталады. Пайда болған терезеде пакет құрамына енгізу, шығару, пакет файлдарын құру және т.б. орындалады. Ал қосымшаны пакетпен байланыстыру қосымшаны жобалау кезеңінің басында, жобаны компиляциялаудан бұрын анықталады, ол үшін негізгі мәзірден Project→Options командасы орындалады да, пайда болған терезеін Package парағындағы Build with Runtime Packages опциясы іске қосылуы керек.

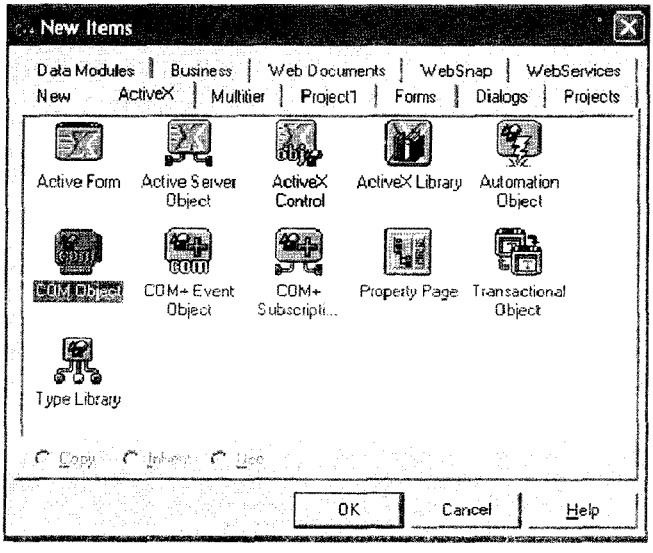
Кроссплатформалы (CLX-қосымшалар) қосымшалардың жаңа жобасын құру негізгі мәзірден File→New→CLX Application командаларын орындау арқылы жасалады. Бұл команда орындалғанда компоненттер палитрасының терезесінде өзгерістер болғанын, мысалы, жаңадан Common Controls парағының пайда болғанын көруге болады. Жалпы C++ Builder негізінен Windows жүйесінде жұмыс жасайтын программалар құруға арналған, бірақ C++ Builder ортасында CLX компоненттерді қолданып құрылған кез келген программаны Linux операциялық жүйесінің Kylix ортасын пайдаланып компиляция жасалса, онда Linux операциялық жүйесінде жұмыс жасайтын қосымшаны алуға болады. Сондықтан мұндай қосымшаларды кроссплатформалы немесе платформааралық қосымшалар (межплатформенные приложении) деп атайды.

COM (Component Object Model) немесе компоненттік объектілер моделі бұл бинарлық программалық компоненттерді (немесе программалық коды екілік форматта жазылған компоненттерді) біріктірудің объектіге бағдарланған технологиясы болып табылады. Бұл технологияның артықшылығы бинарлық деңгейдегі стандартты сақтай отырып кез келген программалау тілінде, кез келген ортада жасалған компоненттерді кез келген программада пайдалана беруге болады. Мы-

салы, COM базасында жасалған Windows қосымшасы Paint графиктік редакторын Word мәтіндік редакторына кіріктіруге болады, мұны көру үшін Вставка→Объект→Точечный рисунок (Insert→Object→Bitmap Image) командаларын орындаса жеткілікті. Пайда болған графиктік редактор терезесінде салынған сурет Word құжатына кіріктіріліп, яғни оның элементі болып қала береді. Мұны Excel электрондық кестелерінде де жасауға болады, яғни бұл COM технологияны қолдайтын қосымшалар компоненттерін бір-біріне ауыстырып қолдана беруге болады дегенді білдіреді. Қазіргі кең тараған сервер және клиент архитектуралары, ActiveX басқарушы элементтері, объектілерді кіріктіру мен байланыстыру (OLE, Object Linking and Embedding) және автоматтандыру – барлығы да осы COM технологияға жатады. C++ Builder қолдайтын COM технологияларға келесілер жатады:

- ActiveX басқарушы элементтері (ActiveX Control) – бұл COM объектілерді беретін компиляциядан өткен программалық компоненттер. COM объектіні құру File→New→Other командаларын орындағанда шығатын «New Items» (1.17-сурет) терезесінің ActiveX парағындағы ActiveX Control таңбасына шертуден басталады. Қосымшаны құру кезінде бұл объектілердің қасиеттерін өзгерту және оларды қосымшаға кіріктіру мүмкіндіктері қарастырылған;
- ActiveX объектілер кітапханасы (ActiveX Library) – бұл бір немесе бірнеше COM объектілерден тұратын сервер (DLL). Оны құру үшін «New Items» (1.18-сурет) терезесінің ActiveX парағындағы ActiveX Library таңбасына шертеді;
- Қасиеттер парағы (Property Page) – қосымшаны құру кезінде ActiveX басқарушы элементтердің қасиеттерін өзгертуді орындайтын диалогтік терезе;
- Актив формалар (Active Form) – Web-браузерлермен жұмыс жасау үшін алдын ала конфигурацияланған қарапайым ActiveX басқарушы элементтің бір түрі;
- Сервердің актив объектілері (Active Server Object) – құрылған қосымшаны актив сервер парағына түрлендіру үшін қолданылатын COM объект, мұны қолданғанда, HTML-парақ генерацияланады;
- Автоматтандыру контроллері (COM Object) – тысқары қосымшадағы автоматтандыру объектілерінің мүмкіндіктерін пайдаланатын және көрсететін клиенттік қосымша;

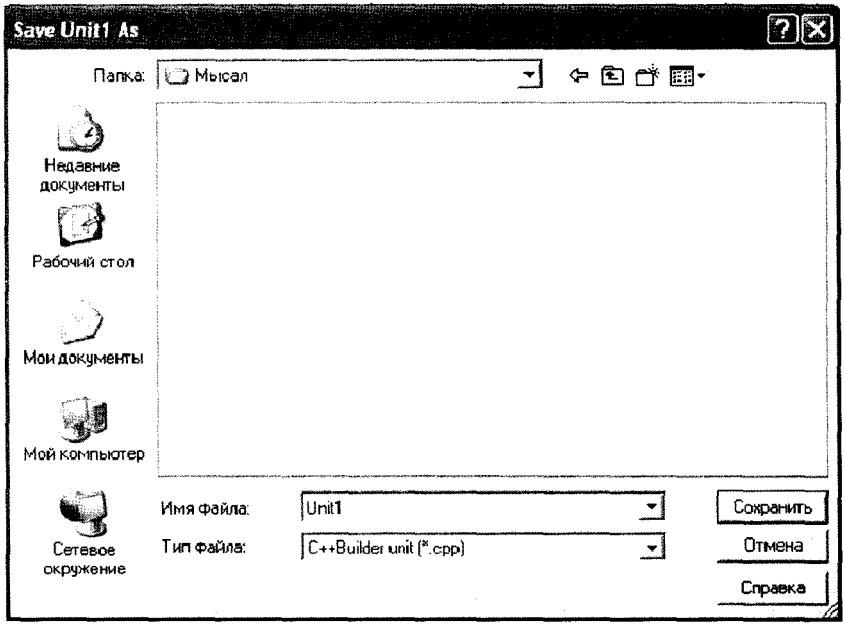
– Автоматтандыру объектісі (Automation Object) – автоматтандыру сервері деп аталатын қосымшаларда болатын COM объектісі. Оның қызметі басқа қосымшаларды программалық жолмен басқарып отыру болып табылады;
– COM+ оқиғалар объектісі (COM+ Event Object) – тіркелген клиенттерге арналған сервер оқиғаларын басқару мақсатында қолданылатын COM+ объектісі. Ол қосымшаның публикация-жазылу (публикации-подписки) режимінде жұмыс жасауына мүмкіндік жасайды;
– COM+ жазылулар объектісі (COM+ Subscription Object) – бұл да COM+ объектісі, ол публикация-қосымшалардан хабарламалар алу үшін клиенттерді тіркеуді ұйымдастырады;
– Транзакция объектісі (Transactional Object) – көп клиенттерге қызмет көрсетуде қолданылатын COM+ объектісі. Қажет болған жағдайда іске қосылуды, транзакцияларды және қауіпсіздік жүйесін қамтамасыз ете алады;
– Типтер кітапханасы (Type Library) бұл қолданушы интерфейстерін, диспетчер интерфейстерін (немесе диспинтерфейстерін), CoClass-тарды, псевдонимдерді, жазбаларды, модульдерді және т.б. анықтау үшін қолданылады.



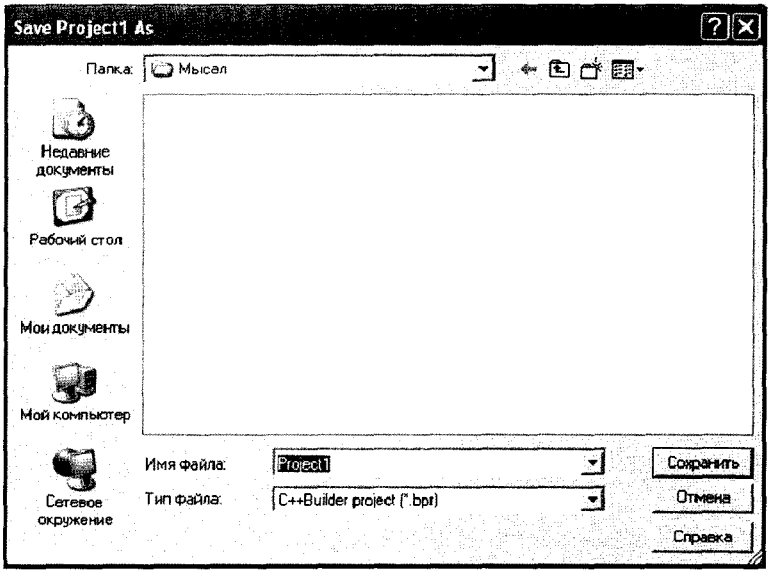
1.18-сурет. *C++ Builder қолдайтын COM технологияларды таңдау терезесі*

1.2.2. Жоба файлдары және оларды сақтау

Жалпы іс жүзінде жұмыс жасағанда жобаны бірден сақтап қойған дұрыс болады. Жобаға кіретін барлық файлдар бір бумада сақталуы тиіс, сондықтан әрбір жобаны сақтамас бұрын сол жобаның өзіне ғана арналған жаңа бума ашылғаны тиімді болады. Жобаны толығымен сақтау үшін негізгі мәзірден File → Save All командасы орындалады. Бұл команда орындалғаннан кейін алдымен модуль файлын (Unit1.cpp) сақтау ұсынылады (1.19, а-сурет), мұндағы «Имя файла» тұсындағы терезеге жаңа ат беруге болады немесе сол күйінде Unit1 деп қалдыра беруге де болады, келесі қадамда жоба файлын (Project1.bpr) сақтау сұралады (1.19, б-сурет), мұнда да файлға жаңа ат беруге болады немесе сол атты қалдыра беруге де болады, бірақ бұл файлдарды бір бумада сақтауды (1.19-суреттерде файлдар «Мысал» деп аталатын бумаға сақталып жатыр) ұмытпау керек. Файлдарға жаңа ат берілген жағдайда ол атаулардың жобаға қатысты мағыналы атаулар болғаны дұрыс.



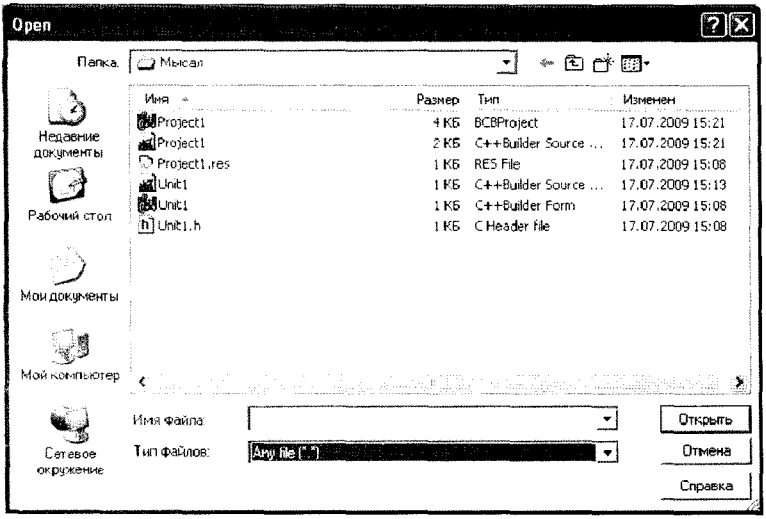
1.19-сурет. *а) Жаңа құрылған жобаның Unit1.cpp файлын сақтау*



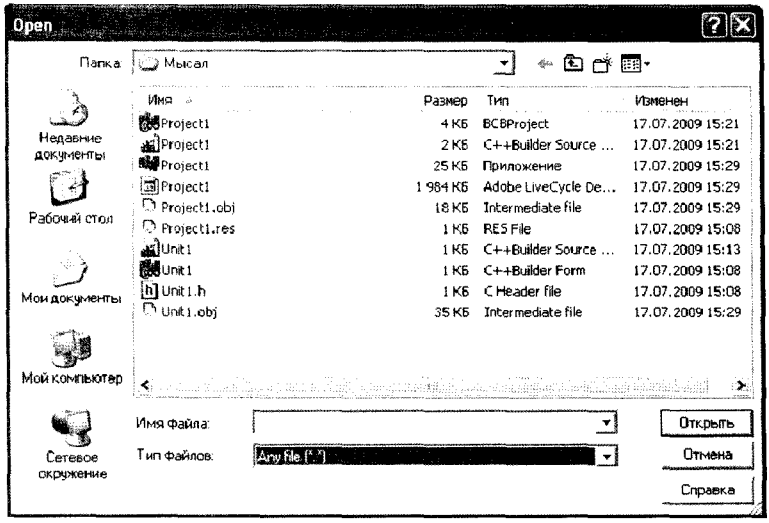
1.19-сурет. *б) Жаңа құрылған жобаның Project1.bpr файлын сақтау*

Бұл қолданушыға сақтауды ұсынған файлдардан басқа тағы да сол жобаға қажет көптеген файлдарды C++Builder ортасы өзі автоматты түрде құрып, сақтап отырады. Бұл файлдарды құрылу ретіне қарай шартты түрде C++Builder ортасында құрылатын файлдар және жобаны компиляциялағаннан кейін компилятор құратын файлдар деп екі топқа біріктіруге болады (1.1, 1.2-кестелер). Бұл файлдар тізімін салыстырып қарау үшін жаңа жобаны компиляциялаудан бұрын сақтап, сонан соң негізгі мәзірден File→ Open командасын орындап, шыққан Open сұхбат терезесіндегі «Тип файла» тұсына барлық файлдарды (Any file (*.*)) көрсету сұралады (1.20, а-сурет).

Бұл тізімде жобаға қатысты орта құрған компиляциялауға дейінгі файлдар орналасқан. Әрі қарай жобаны компиляциялаудан немесе орындауға жібергеннен кейін (бірнеше жолдары бар, мысалы, Run → Run командасын орындау, F9 пернесін басу және т.б.) тура жоғарыдағыдай жоба сақталған бума ашылып, барлық файлдардың аттары көрсетілген болса, онда жоба файлдарының тізіміне тағы басқа компилятор құрған жаңа файлдар қосылғанын көруге болады (1.20, б-сурет).



1.20-сурет. *а) Компиляциялауға дейінгі жоба файлдары*



1.20-сурет. *б) Компиляциялаудан кейінгі жоба файлдары*

Жоба құрамына кіретін әрбір файлдың өзінің атқаратын қызметтері болады. Жобаны құру кезіндегі файлдардың қызметтері келесі кестеде берілген (1.1, 1.2-кестелер).

1.1-кесте

C++ Builder жобасының файлдары

№	Файл кеңеймесі	Қызметі мен қысқаша сипаттамасы
1	Жобаның бас файлы (*.cpp)	Жобаның коды сақталатын негізгі файл, бұл файл қосымшаны жасайтын және оны орындайтын WinMain негізгі функция үшін құрылады, әдеттегі аты – Project1.cpp. Негізгі мәзірдің Project-> View Source командасын орындағанда, жоба бас файлының мәтіні код редакторының терезесіне шығады.
2	Жоба файлы (*.bpr)	Бұл – қандай файлдардың компиляцияланып жоба құрамына кіретінін тағайындайтын жоба опциялары, параметрлері сақталатын XML форматтағы файл.
3	Ресурстар файлы (*.res)	Жоба ресурстары, мысалы, пиктограммалар сақталатын файл, оны Image Editor редакторында өзгертуге болады.
4	Модуль файлы (*.cpp)	Жобадағы әрбір формаға сәйкес модульдің орындаушы файлы (файл реализации модуля) болады, әдеттегі аты Unit1.cpp болады. Бұл мәтіндік файлда программаның коды сақталады, мәтіні код редакторының терезесінде көрініп тұрады. Жоба құрамында формаға тәуелсіз қолданушы өзі құрған жеке модуль файлдары да болады.
5	Модульдің тақырыптық файлы (*.h)	Әрбір формаға сәйкес модульдің орындаушы файлынан басқа тағы да сол форманың және ондағы басқа компоненттердің класс ретіндегі сипаттамалары сақталатын модульдің тақырыптық файлы (заголовочный файл модуля) болады. Модульдің тақырыптық файлын код редакторының терезесіне шығару үшін код редакторының терезесіндегі контекстік мәзірден Open Source/Header File командасын орындаса жеткілікті. Мұндай файлды қолданушының өзіне де құруға болады.

6	Форма файлы (* .dfm)	Бұл форма және ондағы компоненттердің касиеттерінің мәндері, параметрлері сияқты мәліметтер сақталатын файл, бұл мәтіндік файл ретінде код редакторының терезесінде қарауға болады. Ол үшін форма терезесінің контекстік мәзірінен View as Text командасы орындалады. Екілік форматтағы * .dfm файл кәдімгі форманың өзі және ондағы компоненттер болып табылады.
7	Компонент бас файлы (*.hpp)	Мұндай кеңеймелі файлдар жаңадан компоненттер құрылғанда (New->Other->Component) пайда болады. Әтетте жоба құрғанда мұндай кеңеймелі файлдарды Include->VCL бумасында орналасқан C++Builder компоненттерінің кітапханасынан алып жобаға қосу жиі кездеседі.
8	Жобалар тобының файлы (*.bpg)	Жобалар тобының файлы (файл группы проектов) – бұл жобалар тобын құрғанда пайда болатын мәтіндік файл. Бұл өзі қамтитын бір немесе бірнеше жобалар туралы ақпараттарды, сипаттауларды сақтай алады. Файлдағы мәтінді қарау үшін View->Project Manager терезесінен қажет жобалар тобын белгілеп (мысалы, Project Group1), оның контекстік мәзірінен View Project Group source командасы орындалады.
9	Пакеттер файлдары (*.bpl , *.bpk)	Бұл екілік форматтағы файлдар пакеттер құрғанда (New->Other->Package) пайда болады. Мұндағы *.bpl – пакет-жобаның (Borland Package Library) өзінің файлы, ал пакетті компиляциялау мен жинақтауды анықтайтын параметрлер немесе опциялар *. bpk файлда болады.

10	Жоба жұмыс столының файлы (*.dsk)	Бұл мәтіндік файлда жобамен жасалған соңғы сеанстағы әрекеттер сақталып қалады, мысалы, ашық тұрған терезелер, олардың орындары және өлшемдері, т.б. Жоба жұмыс столының файлы құрылуы үшін негізгі мәзірдің Tools->Environment Options командасын орындап, ашылған терезедегі Autosave Options / Project desktop параметрі іске қосылуы тиіс.
11	Резерв көшірме файлы *.~bp, *.~df, *.~cp, *.~h	Бұлар сәйкесінше жоба, форма, модуль және тақырыптық файлдардың резервтік көшірме файлдары болып табылады. Жобамен жұмыс жасау барысында жобаның бір жерлерін әбден бұлдіріп, қайтадан дұрыстау мүмкін болмаған жағдайларда осы резерв файлдардың кеңеймелерін өзгертіп, бастапқы бүлінбеген варианттағы файлдарды алуға болады.

1.2-кесте

Компиляциядан кейінгі құрылатын файлдар

№	**Файл кеңеймесі**	**Қызметі мен қысқаша сипаттамасы**
1	Орындалатын файл (*.exe)	Қосымшаның орындалатын файлы (исполняемый файл приложения) өз алдына жеке орындалатын файл (DLL, OCX және т.б. пакеттердің кітапханаларын қолданылмаған жағдайда).
2	Модульдің объектілі файлы (*.obj)	Модульдің объектілі файлы (объектный файл модуля) – бұл модуль файлын (*.cpp) компиляциялау нәтижесінде құрылып және ақырғы орындалатын файлға жинақталатын файл.
3	Динамикалық түрде қосылатын кітапхана (*.dll)	Бұл файл құрылатын жоба DLL пакеті түріндегі жоба болса ғана пайда болады.
4	Символдар кестесінің файлы (*.tds)	Қосымшаны компиляциялау процесінде компилятор (отладчик) қолданатын екілік форматтағы файл.

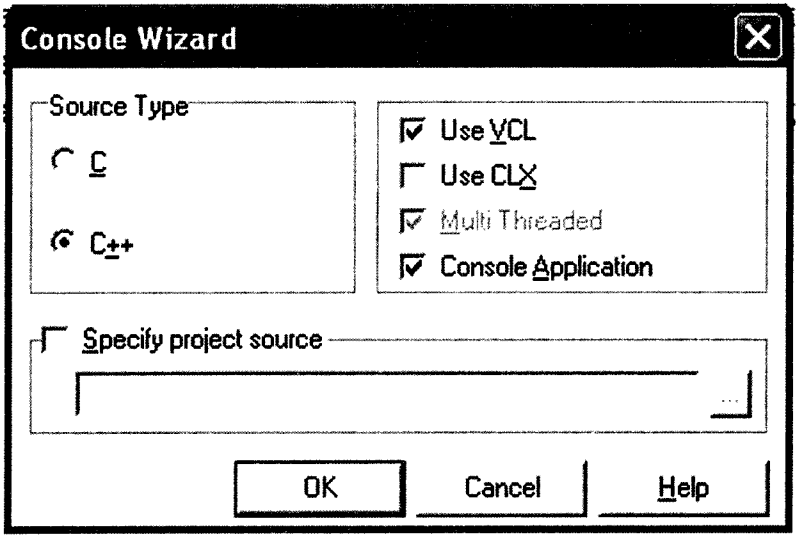
5	Таңдалған файлдарды жинақтау (*.il?)	Кеңеймелері il-дан басталатын файлдар (*.ilc, *.ild, *.ilf, *.ils – файлы выборочной компоновки) жобадағы тек қана соңғы өзгерістер жасалған файлдарды ғана қайта жинақтайды.
6	Анықтама файлы (*.hlp)	Қосымша пайдаланатын Windows жүйесінің стандарт анықтамалық файлы.

Ескерту. Кеңеймелері: *.cpp, *.h, *.dfm, *.bpr, *.res. болатын файлдар маңызды болып есептеледі, себебі олар барлық жобаларға қатысады.

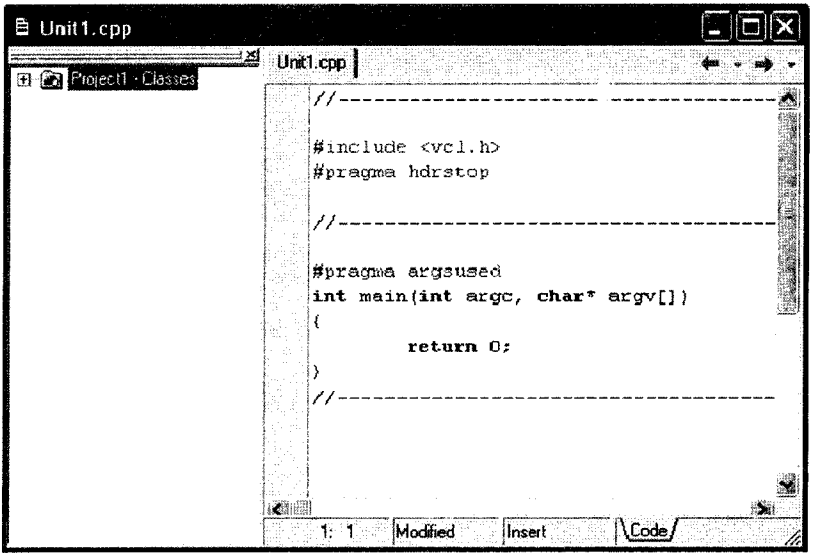
1.3. Консолдық қосымшаның жобасын құру

Консолдық жобаларға сәйкес қосымшалар да Windows қосымшалары болып есептеледі. Бірақ бұл қосымшалар орындалғанда пайда болатын экран DOS жүйесіндегі сияқты мәтіндік режимде жұмыс жасайтын болады. Программалауда қосымшаның интерфейсіне қатаң талаптар қойылмаған жағдайда немесе интерфейс өте қарапайым болған жағдайда консолдық қосымшаларды қолдану тиімді болады. Консолдық жобаны құру терезелік қосымшалардың жобаларын құруға қарағанда өзгешелеу болып келеді, себебі мұнда форма болмайды. Консолдық жобаны құру үшін негізгі мәзірден File → New → Other командалары орындалады да, пайда болған «New Items» терезесінен Console Wizard таңбасы таңдалынады (1.16-сурет). Пайда болған Console Wizard терезесінде болашақ жобаның параметрлері тағайындалып, OK батырмасына шертіледі (1.21-сурет).

Бұл терезедегі Source Type бөлімінде, қосымша кодының қай тілде жазылатыны таңдалынады, егер код C++ тілінде жазылатын болса, суреттегідей (1.21-сурет) C++ таңдалынады. Сол сияқты таңдалынған Use VCL бөлімі қосымша кодына автоматты түрде қосылатын кітапхананың типін көрсетеді. OK батырмасы шертілгеннен кейін келесі қадамда программа мәтіні жазылатын код редакторының терезесі ашылады (1.22-сурет). Код редакторы терезесіндегі бірінші жол #include <vcl.h> – VCL кітапханасы файлдарындағы мәтіндердің программа кодына кірістірілуін қамтамасыз ететін болады, ал егер



1.21-сурет. *Консолдық жобаның параметрлерін тағайындау*



1.22-сурет. *Консолдық жобадағы программа мәтіні жазылатын код редакторының терезесі*

бастапқыда Use CLX бөлімі таңдалса, оған сәйкес директива #include <clx.h> болып келеді.

Келесі жолдағы #pragma hdrstop – директивасы компиляторға тақырыптық файлдарды компиляциялау ретін хабарлайды, яғни оған дейінгі орналасқан алдын ала компиляцияланған тақырыптық файлдар қайтадан компиляцияланбайды, ал одан кейін орналасқан тақырыптық файлдар жазылған программалық кодпен бірге компиляцияланып отыратын болады.

Суретте көрініп тұрғандай, консолдық жобадағы код редакторының терезесінде C++ программалау тіліндегі орындалатын функцияның, яғни main() функциясының шаблоны шығып тұрады.

Сондай-ақ, C++Builder программалау ортасында құрылатын консолдық қосымша жобасына препроцессорға арналған командалардың vcl.h тақырыптық файлының қатысуы міндетті болып саналады, сондықтан кез келген құрылатын консолдық қосымшаға #include <vcl.h> командасын C++Builder программалау ортасының өзі автоматты түрде қосып отырады.

Консолдық қосымша жобасын орындап көру үшін негізгі мәзірден Run командасы орындалады немесе жылдам орындауға жіберу үшін F9 пернесін басады. Програма орындалуының нәтижесі DOS жүйесіндегі сияқты мәтіндік режимде жұмыс жасайтын қара экран болып табылады, программаға мәліметтерді енгізу және шыққан нәтижелерді қарау – барлығы осы экранда жүзеге асырылатын болады. Бұл туралы толық мәліметтер 2.2. және 2.3-тақырыптарда қарастырылады.

Бақылау сұрақтары

1. Object Inspector терезесі қандай қызмет атқарады?
2. Программаның коды немесе программа мәтіндері жазылатын терезені қалай атайды?
3. Жобаның құрамына қандай файлдар кіреді?
4. Жоба параметрлері сақталатын файлдың кеңеймесі қандай?
5. *.exe кеңеймелі файлда не сақталады?
6. Жоба файлындағы программа мәтінін қалай қарайды?
7. DLL Wizard құралының қызметі қандай?

8. Пакеттер (Packages) не үшін қолданылады?

9. Пакеттердің қандай түрлері бар?

10. Программаның коды қандай файлда сақталады?

11. Бір жобада құрылған файлды, екінші жобаға апарып қосатын команданың аты қандай?

12. Жобаның құрамындағы қажет болмай қалған файлды жоятын команда қалай аталады?

13. Жоба параметрлерін тағайындайтын терезе қалай аталады?

14. Кроссплатформалы қосымшалардың артықшылықтары неде?

15. COM технологиялардың артықшылықтары қандай?

16. ActiveX, OLE элементтермен жұмыс жасау қандай технологияға негізделген?

17. Консолдық қосымшаның негізгі орындалатын функциясы қалай аталады?

18. Программа мәтініндегі #pragma hdrstop – директивасының қызметі қандай?

19. Код редакторының Diagram парағының қызметі қандай?

Тапсырмалар

1. C++Builder ортасында терезелік қосымшаның жобасын құрып, орындап көріңіз.

2. Өз жобаңыздың құрамына кіретін файлдарды, олардың атын, қызметін көрсететін кесте жасаңыз.

3. C++Builder ортасының негізгі терезесіндегі (C++Builder Project1) бас мәзірдің опцияларындағы жиі қолданылатын командаларын және олардың қызметін жазып алыңыз.

4. C++Builder ортасында консолдық қосымшаның жобасын құрып, орындап көріңіз.

2-тарау. C++ ТІЛІНДЕ ПРОГРАММАЛАУ НЕГІЗДЕРІ

2.1. C++ тілінің алфавиті және синтаксисі

C++ программалау тілінде латын алфавитінің бас (**A**-дан **Z**-ке дейін) және кіші (**a**-дан **z**-ке дейін) әріптері және астын сызу белгісі (_) қолданылады.

Программа жазуда **0** және **1-9** дейінгі араб цифрлары қолданылады, он алтылық санау жүйесіндегі әрбір цифр 0-ден бастап 15-ке дейінгі мән қабылдайды және олардың алғашқы оны 0-9 дейінгі мәндерді қабылдайды да, ал қалғандары латын әріптерімен (**A, B, C, D, E, F** немесе **a, b, c, d, e, f**) белгіленеді.

Программа жазуда арнайы символдар: + (қосу), – (азайту), * (көбейту), / (бөлу), < (кіші), > (үлкен), [ ] (тік жақшалар), { } (фигуралы жақшалар), **( )** (жай жақшалар), **.** (нүкте), **,** (үтір), **:** (қос нүкте), **;** (нүктелі үтір), ‘ ‘ (апостроф), % (модуль бойынша бөлу белгісі), ? (сұрақ белгісі), # (диез немесе фунт белгісі) және т.б. символдар, сондай-ақ жазғанда міндетті түрде бірге жазылатын, яғни араларына бос орын қоюға болмайтын құрама символдар: == (тең), != (тең емес), <= (кіші немесе тең), >= (үлкен немесе тең), && (логикалық “және”), ||(логикалық “немесе”) және т.б. қолданылады.

C++ программалау тіліндегі программаны жазуда сақталуы тиіс негізгі синтаксистік ережелерге төменде берілгендерді жатқызуға болады:

– программаның мәтінін жазғанда бас әріптерді немесе кіші әріптерді пайдаланудың айырмашылығы бар, мысалы, atay мен ATAY екі түрлі нәрсені немесе атауды білдіреді;

– *идентификаторлар,* бұл – белгілі бір шаманың, мәліметтің, деректің аты, аталуы деген мағынаны білдіреді. Мысалы, тұрақтының, айнымалының, белгінің, объектінің, типтің, функцияның аттары идентификаторлар болып табылады. Қазақ, орыс алфавитінің әріптері, қызметші сөздер және бос орын идентификатор бола алмайды және оның құрамына да кіре алмайды, ал латын әріптері, сандар, астын сызу белгісі идентификатор құрамына кіре алады және идентификатор міндетті түрде әріптен немесе астын сызу белгісінен (_) басталуы керек. Мысалы: esep_1, Lab_12, _misal. Идентификатор ұзындығына

шектеу қойылмайды, бірақ программаны жазуда тым ұзын атауларды қолданудың тиімсіз екені белгілі;

– программада кездесетін *түсіндірме мәтіндер* (комментарийлер) көлбеу жақшаның (немесе слэш белгісі, /) ішіне, екі жағынан жұлдызша белгісіне (*) алынып жазылады, мысалы: /* *түсіндірмелер осылай жазылады* */. Түсіндірме мәтіндерді программаның кез келген жерінде жаза береді және ол бірнеше жолдардан тұруы мүмкін, егер программада /* белгісі кездессе, онда осы белгі кездескен жерден бастап оның жабылатын бөлігі */ белгісі кездескенге дейінгі аралықтағы барлық мәтіндер түсіндірме болып табылады. Программадағы түсіндірме мәтіндер программа мәтінінен басқа түспен және басқа шрифтпен ажыратылып тұрады. Егер түсіндірме мәтін бір-ақ жолға сыятын болса, онда оны // (екі слэш) арқылы да жаза беруге болады;

– программа мәтінінің әрбір мағыналы сөйлемі (;) нүктелі үтір белгісімен аяқталады, мысалы, **int** i, j;

– программада кездесетін барлық типтер, тұрақтылар, айнымалылар, функциялар және т.б. ең алғашқы пайдаланылуға дейін сипатталынуы немесе жарияланылуы тиіс. Оларды сипаттау программаның кез келген жерінде жүргізіле береді;

– программаның фигуралы жақшаларға { } алынып жазылған бөліктерін *блоктар* деп атайды. Тілдің синтаксисі бойынша бір блок бір құрама оператор сияқты қабылданады.

Программа жазуда жоғарыда аталған негізгі синтаксистік ережелерден басқа программаның оқылуын жеңілдету үшін программалық блоктарды немесе жолдарды, жаңа жолдан, бірінің астына бірін табуляция жасай отырып, жазу стилін сақтаған дұрыс болады. Мысалы:

unsigned int x=7; **if**(x!=0) **for** (**int i**=0; i<=x; i++) {**float** s=s+i; cout<<s;};

түрінде жазылған программалық код қате болып есептелмейді, яғни программада осылай да жазуға болады, бірақ мұны программалық кодтың келесі түрдегі жазылуымен:

```
unsigned int x=7;
if(x!=0)
    for (int i=0; i<=x; i++)
        {
          float s=s+i; cout<<s;
        } ;
```

салыстыратын болса, бұл екінші түрдегі жазылудың әлдекайда түсінікті және оқуға жеңіл екенін байқауға болады.

2.2. C++ тіліндегі программа құрылымы

C++ тіліндегі кез келген программа бұл программаға қатысатын барлық айнымалылардың, типтердің, тұрақтылардың, функциялардың жарияланулырынан және функциялардың өздерінен тұрады. Бұл функциялардың арасында міндетті түрде бір негізгі функция болады, ол WIN32-дегі консолдық қосымшалар үшін main(), ал Windows қосымшалары үшін WinMain() функциялары болады. Программа іске қосылғаннан кейін ең бірінші болып осы функциялар орындалады.

C++ тіліндегі программа құрамында директивалар, деректердің түрліше типтері, тұрақтылар, айнымалылар, амалдар, операторлар, функциялар және т.б. болады.

Директивалар. Программалау тілдерінде программа мәтінін машиналық кодқа аударатын компиляторларға арналған командалар болады, бұл командаларды директивалар деп атайды. C++ программалау тілінде директивалардың аты # (диез немесе фунт) белгісінен бастап жазылады. Программадағы include директивасының жазылуы (немесе синтаксисі:

#include <файлдың аты >

Мысалы, #include < stdio.h > мұның мағынасы, бұл include директивасы компиляторға stdio.h тақырыптық файлының мәтінін, жазылып жатырған программа мәтініне кірістіруге нұсқау береді.

Деректердің типтері. Программаға қатысатын барлық шамалар белгілі бір типтерге жатады, мысалы, сандық шамалар үшін бүтін және нақты типтер анықталған, сондай-ақ жекелеген символдармен жұмыс жасауға арналған символдық тип, мәтінмен жұмыс жасайтын жолдық тип, бульдік тип және т.б. қолданылады. Бұл қарапайым типтерден басқа осы типтер негізінде құрылатын массив, жазба деп аталатын және т.б. көптеген құрылымдық типтер де болады.

Тұрақтылар. Тұрақтыларды мәні белгілі, алдын ала анықталған идентификатор ретінде қарастыруға болады. С++ программалау тілінде тұрақтының мәндері ретінде бүтін сандар, нақты сандар, он алтылық сандар, логикалық тұрақтылар, жол түріндегі символдар тізбегі, символдардың өздері және т.б. қолданылады.

const float pi = 3.14159265

Айнымалылар. Программаның орындалуы барысында мәндері өзгеріп отыратын шамаларды айнымалылар деп атайды. Программада міндетті түрде айнымалылардың атауы, типі көрсетіледі. Айнымалылар түрлі типтерге жатады.

Амалдар. С++-тегі амалдарды арнайы құрылған функциялар (встоенные функции) деп қарастыруға болады. Амалдар көбінесе екі операндасы бар бинарлық болып келеді, одан басқа унарлық амалдар да бар. С++ программалау тілінде әдеттегі стандарт қосу, азайту, көбейту және бөлу амалдарынан басқа тағы да көптеген амалдар қолданылады.

Операторлар. Программадағы әрекеттердің орындалу тәртібін операторлар анықтайды. С++-тегі операторларды, мысалы, шартты түрде басқару операторлары, цикл операторлары деп бөлуге болады. Операторлардың жазылуының қабылданған синтаксисі болады.

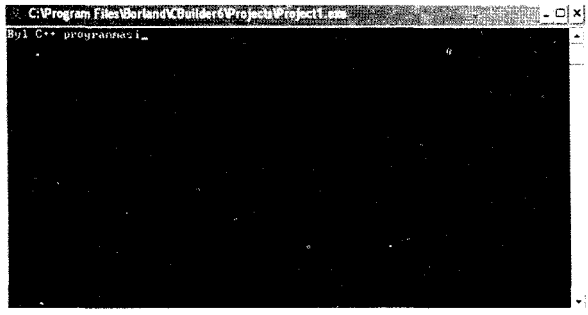
Функциялар. Функциялар – бұл программаның кез келген жерінде шақырып алып, пайдалана беруге болатындай өз алдына жеке программалық блоктар. Бұл программалық блоктар белгілі бір әрекеттерді орындап, міндетті түрде белгілі бір типке жататын нәтижелер беруі керек. Программалар көптеген функциялардан құралады.

2.1-жаттығу. Ең қарапайым программаға мысал қарастырайық, ол үшін С++ Builder ортасында қарапайым консолдық қосымша құрылуы тиіс. Ол консолдық қосымшаны құру үшін С++Builder терезесінің негізгі мәзірінен File→ New → Other командалары орындалады да, пайда болған «New Items» терезесінен Console Wizard таңбасына шертіледі, сонда код редакторының терезесінде программа шаблоны пайда болады. Сол шаблонға келесі мәтін кірістіріліп жазылады:

```
#include <vcl.h>
#pragma hdrstop
#include <iostream.h>
#include <conio.h>
//---------------------------------------------
#pragma argsused
int main()
{
cout<< "Byl C++ programmasi";
 getch();
 // return 0;
}
```

Бұл программаны орындауға жіберу үшін F9 пернесін баса салуға болады, сонда нәтиже келесі терезеге шығады (2.1-сурет):

Программадағы бастапқы жолдарда # (диез немесе фунт) белгісінен басталатын директивалар орналасқан, олардың әрқайсысының атқаратын қызметтері бар, болашақта C++Builder ортасындағы консолдық қосымшалардың программаларын жазғанда, бұл директивалар барлық программаларда да қолданылатын болады. Мұндағы #include <vcl.h>, #pragma hdrstop, #pragma argsused директивалары программа шаблондарымен бірге автоматты түрде өздері пайда болады. Ал #include <iostream.h> және #include <conio.h> директиваларын қолданушы өзі жазады, қажет болса тағы да басқа директиваларды, мысалы, элементар математикалық функциялардың сипаттамалары сақталған math.h файлын да директива түрінде қосуға болады.



2.1-сурет. *Консолдық қосымшаның орындалу терезесі*

Мұндағы iostream.h тақырыптық файлында, кейіннен программа мәтінінде пайдаланылатын cout объектісінің және "<<" амалының сипаттамалары бар, ал бұл сипаттамаларсыз компилятор cout және "<<" белгілерін тани алмайды. Сол сияқты программадағы getch(); функциясының сипаттамасы conio.h файлында сақталады, демек iostream.h, conio.h файлдарын программаға директивалар түрінде қосудың неге қажет болғаны түсінікті.

Әрі қарай талданатын болса, программаны басқару main() функциясына беріледі. Егер программада main() функциясы болмаса, онда қате жөнінде хабарлама беріледі. Берілген программадағы бүтін **int** типті main() негізгі функция болады. Функцияның құрамына бірінші жол кірмейді, яғни функцияның алдында берілген **int** сөзі функцияның бүтін типті қайтаратындығын көрсетеді. Функцияның денесі { } фигуралық жақшаның ішінде жазылған. Бұл жақшалар міндетті түрде берілуі керек. Функцияның денесінде бір оператор – **cout** және бір функция – **getch();** орналасқан, ал *return* операторы программа жұмысына қатыспайды, себебі ол түсіндірме сияқты, яғни // белгісімен беріліп тұр. Программаны орындау барысында ең бірінші орындалатын оператор, main() функциясының бірінші операторы болады. Оператор C++ тілінің құрылымдық бірлігі болып есептеледі. Оператор «нүктелі үтір» белгісімен аяқталады. Мұндағы бірінші орындалатын **cout** операторының қызметі тырнақша ішіндегі «Byl C++ programmasi» мәтінді "<<" ағымға қосу амалын пайдаланып экранға шығарады.

Келесі орындалатын getch(); функциясының қызметі – пернетақтадан басылған кез келген символды қабылдау болып есептеледі, яғни бұл жерде нәтижелер шығарылатын «қара экранның» бір рет қана көрініп, жоғалып кетпей, кез келген бір перне басылғанша тұра тұруын қаматамасыз етеді.

2.3. Препроцессордың директивалары

Компилятордың құрамдас бөлігі болып табылатын препроцессордың қызметі – программаны компиляция жасамас бұрын алдын ала өңдеуге мүмкіндік жасау. Бұл алдын ала өңдеу кезеңінде, компиляция

жасалатын файлға (программаға) басқа файлдарды қосу, символдық тұрақтыларды және макростарды анықтау, программалық кодты шартты түрде компиляция жасау және препроцессор директиваларын шартты түрде орындау режимдерін тағайындау сияқты әрекеттер жүргізіледі.

Программа мәтініндегі # (диез немесе фунт) белгісінен бастап жазылатын кез келген жолды, компилятор, препроцессордың директивасы деп қабылдайды. Тілдің синтаксисі бойынша препроцессор директивасынан кейін «;» (нүктелі үтір) белгісі қойылмайды.

#include <файлдың аты > директивасының қызметі аты көрсетілген файлды тауып алып, сол директива тұрған жерге кірістіру болып табылады. Бұл директиваның жазылуының үш түрі кездеседі:

```
#include <файлдың аты >
#include "файлдың аты"
#include макрос идентификаторы
```

мұндағы алғашқы екеуінің айырмашылығы, бұл аты көрсетілген файлды іздеуді ұйымдастыруда, егер файлдың аты < > белгілеріне алынып жазылса, онда файлды іздеу реті алдын ала берілген кірістірілетін стандарт кітапханалар каталогтары (include directories) бойынша жүргізіледі. Егер де файлдың аты " " (тырнақша) белгілеріне алынып жазылса, онда бұл файлды іздеу реті басқаша болады, яғни бірінші сол #include директивасы қосылып тұрған программаның файлы сақталған каталогтан ізделеді; екінші, осы программаға #include директивасы арқылы қосылған басқа да файлдардың каталогтарынан карастырылады; үшінші ретте ағымдағы каталогтан іздейді; төртінші ретте компилятордың /I опциясында көрсетілген каталогтар тексеріледі; соңғы болып INCLUDE айнымалысындағы каталогтар қаралады. Файлдарды іздеудің мұндай жолдары препроцессор директиваларындағы файл маршруттары берілмеген жағдайда ғана мүмкін болатыны түсінікті.

Мысалы, келесі препроцессор директивалары:

```
#include <vcl.h>
#include <math.h>
```

директиваларда аттары көрсетілген vcl.h және math.h файлдарын, кірістірілетін файлдардың стандарт кітапханалар каталогтарынан іздейді, ал келесі

```
#include "Unit1.h"
```

директивада аты көрсетілген Unit1.h файлын сол программа сақталған каталогтан іздейтін болады.

include макрос идентификаторы түріндегі жазылу егер макрос алдын ала анықталған жағдайда ғана қолданылады, мысалы:

```
#define myFile  "D:\Misal5\myFe.h"
#include myFile
```

директивалары программаға көрсетілген D:\Misal5 каталогындағы myFe.h файлын қосатын болады.

#define препроцессордың директивасы программада символдық тұрақтыларды және параметрлі макростарды құру үшін қолданылады. Символдық тұрақтыларды параметрі жоқ макростар деп те атайды. Символдық тұрақты деп белгілі бір мәтінді алмастыратын идентификаторды айтады, программадағы жазылуы:

```
#define идентификатор_атауы   алмастырушы мәтін
```

Мысалы, программа мәтініндегі

```
#define SimvTyrakti  "Menin atim Koga"
```

директивасымен анықталған SimvTyrakti идентификаторы программаның кез келген жерінде қолданылады да, "Menin atim Koga" деген мәтінді беріп тұрады. Бұл жерде ескерілетін нәрсе символдық тұрақтыға жазылған мәтін тура сол күйінде өзгермей беріледі, айталық, әдетте π санын беру үшін қолданылатын

```
#define PI  3.14159
```

директивасы

#define PI =3.1415 түрінде жазылатын болса, онда PI-ді алмастыратын мәтін де тура солай «=3.14159» болып кетеді.

#define директивасымен анықталатын параметрлі макростар, аты айтып тұрғандай, белгілі бір параметрге тәуелді алмастырушы мәтіндерді беру үшін қолданылады, жазылуы:

```
#define макрос_идентификаторы (параметр)  алмастырушы мәтін
```

Мысалы, дөңгелектің ауданын есептеу үшін қолданылатын dongAud(r) параметрлі макросын программада директива түрінде анықтау келесі түрде болады:

```
#define PI 3.14159
#define dongAud(r) (PI*r*r)
```

Әрі қарай программаның кез келген жеріндегі dongAud(r) мәтіні r параметрдің берілген мәніне сәйкес дөңгелектің ауданын есептейтін болады.

Егер параметрлі макростың аргументтері немесе параметрлері жай айнымалылар емес өрнектер болатын болса, онда оларды міндетті түрде жақшаға алып жазады, мысалы:

```
#define dongAud(r1+r2)  (PI*(r1+r2)*(r1+r2))
```

Параметрлі макростар қызметі жағынан функциялар сияқты жұмыс жасайды. Программада макростарды қолданудың тиімді жағы да, тиімсіз жағы да бар, сондықтан оны қажет жерлерде ғана пайдалана білу керек.

#undef директивасы программада #define директивасымен анықталған символдық тұрақтылар мен параметрлі макростардың қызметін тоқтатады. Жазылуы:

```
#undef  идентификатор_аты
```

Мысалы,

```
#define Tyrakti  33  //Tyrakti-нің мәні 33-ке тең
 . . .
#undef  Tyrakti  //идентификатор жойылды, оны пайдалануға болмайды
. . .
```

```
#define Tyrakti 77   // Идентификатордың аты басқа
                     // тұрақты үшін берілді
. . .
```

#undef директивасымен жойылған символдық тұрақтылар мен параметрлі макростарды программада әрі қарай пайдалануға болмайды.

#if, #endif, #ifdef, #ifndef, #else, #elif директивалары компиляторға шартты түрде компиляциялауды хабарлау үшін қолданылады. #if, #endif директивалары шарт операторлары сияқты жұмыс жасайды, жазылуы:

```
#if шарт
    программа кодының фрагменті
#endif
```

мұндағы шарт бүтін санды өрнек болып табылады, егер оның мәні нольден өзгеше болса, онда шарт «ақиқат» болады да, программа фрагменті компиляцияланады, ал «жалған» болса, фрагмент компиляцияланбайды. Кейбір жағдайларда шарт орнындағы өрнектің орнына

```
defined идентификатор_аты
```

конструкциясын қолдануға да болады, бірақ мұнда идентификатор алдын ала #define директивасымен анықталған болуы керек, мысалы:

```
#define MyText boladi
#if defined MyText
    программа кодының фрагменті
#endif
```

Мұнда MyText идентификаторы алдын ала #define директивасымен анықталғандықтан, defined конструкциясының мәні 1-ге тең болып, код фрагменті компиляцияланады, ал егер идентификатор анықталмаған болса, онда defined 0-ге тең болады да, фрагмент компиляцияланбайды.

Тура осы сияқты келесі

```
! defined идентификатор_аты
```

конструкциясын да қолдануға болады, мұнда керісінше идентификатор анықталған болса, defined мәні 0-ге тең болады, ал анықталмаса, defined 1-ге тең болады.

#if defined және #if ! defined түріндегі директивалардың орнына олардың сәйкесінше қысқаша түрлері #ifdef және #ifndef қолданыла береді.

Әдетте #define директивасын символдық тұрақтылар мен макростар тағайындау үшін ғана емес, сондай-ақ программа мәтініне қосылатын файлдардың қайталанып қосылып кетпеуін қадағалау үшін қолданылады. Мысалы, бұл C++Builder-де *.h кеңеймелі тақырыптық файлдар (заголовочные файлы) мәтінінде келесі түрде

```
#ifndef Unit1H
#define Unit1H
   тақырыптық файлдың коды
#endif
```

болады. Бұл жазылу #ifndef және #endif директиваларының арасындағы кодтың бір рет қана қосылуын қамтамасыз етеді. Мысалы, жобаға Unit1.h файлы бірінші рет қосылғанда, алдыңғы екі директива орындалады: алғашқы тұрған #ifndef Unit1H ең бірінші рет орындалғанда, Unit1H идентификаторы әлі анықталмағандықтан, ifndef (немесе #if ! defined) мәні 1-ге тең болады да, келесі жол #define Unit1H орындалады, яғни Unit1H анықталады, сонан соң файл мәтіні компиляциялланады, ал кейіннен тағы да #include директивасы арқылы Unit1.h файлын қосу қайталанатын болса, онда манағы Unit1H идентификаторы анықталғандықтан, ifndef мәні 0-ге тең болады да, әрі қарайғы директивалар орындалмайды, яғни қайтадан компиляция болмайды.

2.2-жаттығу. Консолдық қосымшада #include, #define директиваларының қолданылулары қарастырылсын, консолдық қосымшаны құру үшін C++Builder терезесінің негізгі мәзірінен File→ New → Other командалары орындалады да, пайда болған «New Items» терезесінен Console Wizard таңбасына шертіледі, сонда код редакторының терезесінде пайда болған шаблонға келесі мәтін жазылады:

```
#include <vcl.h>
#include <iostream.h>
#include <conio.h>

#define PI 3.14159
#define dongAud(r) (PI*r*r)
#define SimvTyrakti "Menin atim Koga"

int main()
{
 float r;
   cin>>r;
   cout<<"dongAud = "<<dongAud(r)<<endl;
   cout<< "SimvTyrakti mani = "<<SimvTyrakti;
   getch();
 }
```

Программа орындалуының нәтижесі:

```
5
dongAud = 78.5397
SimvTyrakti mani = Menin atim Koga
```

Егер программа кодын талдайтын болса, мұнда #include директивасы арқылы программа кодына vcl.h, iostream.h, conio.h файлдарының мәтіндері қосылатын болады, мысалы, cin, cout операторларының сипаттамалары iostream.h файлында, ал getch () функциясының сипаттамасы conio.h файлында сақталады. Сондай-ақ, программа кодында #define директивасы арқылы PI тұрақтысы, dongAud(r) параметрлі макростары анықталып және олар әрі қарай пайдаланылған.

2.4. Деректердің негізгі типтері

Программада *бүтін сандарды* пайдалану үшін негізінен **int** қызметші сөзінің көмегімен берілетін бүтін типтер қолданылады. Бүтін

типтің қабылдайтын мәндері жүйе платформасына тәуелді болады, 16-биттік платформада -32768 бен 32767, ал 32 биттік платформада 2147483648 бен 2147483647 аралығындағы мәндерді қабылдай алады және сәйкесінше жадыдан 2 немесе 4 байт орын алады.

Нақты сандар программада **float** және **double** (қосарланған дәлдікпен берілген нақты сандар) қызметші сөздерінің көмегімен жазылады. Олар программада "-" немесе "+" таңбаларымен немесе таңбасыз ондық нүкте арқылы, немесе экспоненциалдық бөліктің (e) көмегімен жазылады. Мысалы: -1.75e+7 немесе бұл $-1.75*10^7$ дегенді білдіреді. **float** типі жадыдан 4 байт орын алады да, сәйкесінше $3.4*10^{-38}$ мен $3.4*10^{+3}$ және $-3.4*10^{-38}$ мен $-3.4*10^{+38}$ аралығындағы мәндерді қабылдай алады. **double** типі 8 байт көлемінде орын алады да, $1.7*10^{-308}$ мен $1.7*10^{+308}$ және $-1.7*10^{-308}$ мен $-1.7*10^{+308}$ аралығындағы мәндерді қабылдай алады (2.1-кесте).

Символдық типтің мәніне апострофқа алынған кез келген символ жатады. Мысалы: 'A', '['. Символдық тип **char** қызметші сөзінің көмегімен беріледі.

Логикалық типтің мәні бұл «жалған» немесе «ақиқат» деген сөздердің біреуі болып табылады, яғни жалған – **«false»,** ақиқат – «**true**». Логикалық типті көрсету үшін программада **bool** қызметші сөзі қолданылады, жадыдан алатын орны 1 байт көлемінде.

2.1-кесте

C++-тегі деректердің қарапайым типтері

Типтің аты	Төменгі шекарасы	Жоғарғы шекарасы	Дәлдігі	Ұзындығы (байт)
bool	false(немесе 0)	true (немесе 1)	Жоқ	1
char	-128	127	Жоқ	1
short	-32 768	32 767	Жоқ	2
int	-2 147 483 648	2 147 483 647	Жоқ	4
long	-2 147 483 648	2 147 483 647	Жоқ	4
float	$-3,4*10^{-38}$	$3,4*10^{38}$	7	4
double	$-1,7*10^{-308}$	$1,7*10^{308}$	15	8

С++ программалау тілінде деректердің **int, char, double** типтерін тиімді пайдалану үшін, типтердің берілу дәлдігін және қабылдайтын мәндерінің диапазондарын өзгерту мүмкіндігі қарастырылған. Мұны жүзеге асыруда модификаторлар деп аталатын **signed** (таңбамен берілу), **unsigned** (таңбасыз берілу), **short** (қысқа аралықта берілу), **long** (үлкен аралықта берілу) қызметші сөздері қолданылады. Мысалы, бүтін сандардың берілуінен таңба алынып тасталса, онда жоғарғы шекарасы екі есе үлкейген деректер типін алуға болады (2.2-кесте).

2.2-кесте

***unsigned** модификаторын қолдану*

Типтің аты	Төменгі шекарасы	Жоғарғы шекарасы	Ұзындығы (байт
unsigned char	0	255	1
unsigned short	0	65535	2
unsigned int	0	4 294 967 295	4
unsigned long	0	4 294 967 295	4

Сондай-ақ, С++ тіліндегі **void** деп аталатын тип, бұл қандай да болмасын бір функцияның орындалғаннан кейін мәнді нәтиже бере алмайтындығын көрсету үшін қолданылады, яғни функция процедура сияқты қызмет атқарады (Деректердің типтері 3-тарауда толық қарастырылады).

2.4.1. Типтерді келтіру

С++ тілінде деректер типтерін қолданудағы бір ерекшелік – бұл арифметикалық өрнектерде әр түрлі типтегі операндаларды немесе сандық шамаларды қолданғанда, деректер типтерінің өздігінен автоматты түрде өзгеріп кетуі. Бұл түрлендіру егер өрнек құрамындағы операндалар әр түрлі типке жататын болса, онда өрнектегі төменгі дәлдіктегі тип жоғарғы дәлдіктегі типке өзгеріп кетеді, яғни екі операнда да жоғары дәлдіктегі типке түрленеді және өрнектің нәтижесі де сондай типті шама болып шығады. Мысалы, өрнектегі операндалардың біреуі бүтін (), екіншісі нақты типке жататын болса, онда шығатын

нәтиже де нақты типке жатады. Бұл арифметикалық амалдар үшін ғана осылай болады, ал меншіктеу амалында (=) басқашалау болады.

Меншіктеу амалында (=) тең боладының сол жағындағы тип түріне қарамастан, «жоғары» болып есептеледі де, оң жақтағы мән дәлдігіне немесе приоритетіне қарамастан, сол тең боладының сол жағындағы мәннің типіне түрленеді. Мысалы, меншіктеу амалының сол жағындағы тип бүтін, ал оң жағындағы өрнектегі операндалар нақты болса, онда нәтиже тіпті де қате болып шығады.

2.3-жаттығу. Келесі программалар типтерді келтіруді демонстрациялайды.

а) Консолдық қосымшадағы келесі программа фрагменті орындалғанда,

```
#include <vcl.h>
 #include <iostream.h>
 #include <conio.h>
int main()
{   int a=5;
    float b=3.57;
    a=b+a;
    cout<<a;
   getch(); }
```

нәтижесінде 8.57 шығуы керек, бірақ нәтижеде *a*-ның типі бүтін болғандықтан, 8 шығады.

b) Ал келесі программа орындалғанда,

```
#include <vcl.h>
 #include <iostream.h>
 #include <conio.h>
int main()
{    double  a=300,b=600;
   short  c=a*b;
     cout<<c;
    getch(); }
```

тіпті де басқаша мән -16608 алынады, себебі мұндағы нәтиже *c*-ның типі **short** және оның ең жоғарғы шекарасы 32 767, сондықтан да нәтиже дұрыс болмайды.

Программадағы мұндай жағдайлардан құтылу үшін типтерді келтіру операциясы қолданылады, синтаксисі:

(типтің _аты) өрнек

c) Мысалы, келесі программаның

```
#include <vcl.h>
 #include <iostream.h>
 #include <conio.h>
int main()
{  float c;
   float a=8.16;
   float b=2.4;
    cout<<"a="<<a<<endl;
    cout<<"b="<<b<<endl;
     c= a/b;              // 1-жағдай
        cout<<"a/b="<<c<<endl;
     c= (int)a/b;         // 2-жағдай
        cout<<"(int)a/b="<<c<<endl;
     c= int (a/b);        // 3-жағдай
        cout<<"int (a/b)="<<c<<endl;
     c= a/(int)b;         // 3-жағдай
        cout<<"a/(int)b="<<c<<endl;
   getch(); }
```

орындалуының нәтижесі келесі түрде болады:

```
a=8.16
b=2.4
a/b=3.4
(int)a/b=3.33333
int (a/b)=3
a/(int)b=4.08
```

Бұл программа орындалғанда, с-нәтиженің мәні төрт түрлі жағдайда есептеледі:

1-жағдайда *a, b* және *c* шамаларының типтері бірдей **float** болады, сондықтан a=8.16, b=2.4 мәндері сол күйінде өзгеріссіз қалады да, c=8.16/2.4 орындалып, нәтижеде a/b =3.4 шығады;

2-жағдайда есептелетін өрнектегі *a*-ның типі **int** болып түрленеді, ал *b* және *c* шамаларының типтері **float** болып қала береді, сондықтан a=8, b=2.4 мәндері алынады, әрі қарай c=8/2.4 орындалып, нәтижеде (int) a/b =3.33333 шығады;

3-жағдайда алдымен амалдар приоритеті бойынша жақша ішіндегі өрнек (a/b)-ның мәні есептеледі де, 3.4 алынады, сонан соң барып оның алдындағы **int** бүтін типі тұрғандықтан, шыққан 3.4 шама бүтін типке келтіріледі де, нәтижеде int (a/b)=3 шығады;

4-жағдайда есептелетін өрнектегі *b*-ның типі **int** болып түрленеді, ал *a* және *c* шамаларының типтері **float** болып қала береді, сондықтан a=8.16, b=2 мәндері алынады, әрі қарай c=8.16/2 орындалып, нәтижеде a/(int) b =4.08 шығады.

2.5. Тұрақтылар

Тұрақтылар мәні – алдын ала белгілі және программаның орындалуы барысында өзгермейтін шамалар. С++ программалау тілінде тұрақтының мәндері ретінде бүтін сандар, нақты сандар, логикалық тұрақтылар, жол түріндегі символдар тізбегі, символдардың өздері және т.б. қолданылады.

Бүтін тұрақтылар ондық, сегіздік және он алтылық жүйелерде берілуі мүмкін. Мысалы,

3,7,777 – ондық жүйедегі бүтін типті тұрақтылар;

075, 032, 017 – сегіздік жүйедегі бүтін типті тұрақтылар, олар нольден басталады, ал қалған цифрлары 0-7 аралығында болады, 8-өзі кірмейді;

0X7D9, 0xff – он алтылық жүйедегі бүтін типті тұрақтылар. С++ программалау тілінде он алтылық жүйедегі сандар 0 цифрынан және X немесе x символдарынан басталады да, әрі қарай он алтылық жүйедегі мәні жазылады, мысалы, 255_{10}=$0xff_{16}$ немесе 2009_{10} = $0X7D9_{16}$.

Нақты типке жататын тұрақтылар жылжымалы үтірмен немесе экспоненциалдық түрде беріледі, мысалы:

4.778, 0.25, 147.5 – жылжымалы үтірмен берілген нақты тұрақтылар;

-1.75e+7 (-1.75*10^7 дегенді білдіреді), 7E+17 – экспоненциалдық түрде берілген нақты тұрақтылар.

Символдық тұрақтының мәні апострофқа алынған кез келген жалғыз символ болып табылады. Мысалы:

'A', '1', '[' – символдық тұрақтылар. Символдық тұрақты болу үшін апострофтың ішіндегі символ жалғыз ғана болуы керек. Символдық тұрақтылар программада **char** , **signed char** , **unsigned char** типтерінің бірі ретінде сақталынады.

Жолдық тұрақтыларды тырнақшаға алынған мәтіндер құрайды, мысалы:

"DALA", "Astana-ylken kala" – жолдық тұрақтылар.

Программада тұрақтының мәнін белгілі бір идентификаторға бекітіп пайдалану жиі кездеседі. Мысалы,

```
const float PI = 3.14159265;
```

Тұрақтылардың мұндай түрі программада бір шаманы қайта-қайта жазып жатпас үшін пайдаланылады. Программада осылайша анықталатын тұрақтылардың типтері әдетте **int** болып қабылданады, сондықтан егер тұрақты басқа типке жататын болса, онда оның типі міндетті түрде көрсетілуі керек. Мысалы,

const PI = 3.14159265 түрінде жазылатын болса, онда PI -дің мәні тек 3-ке ғана тең болады.

2.6. Айнымалылар

2.6.1. Айнымалыларды жариялау

Айнымалыларды программаның орындалуы барысында мәндері өзгеріп отыратын идентификаторлар деп қарастыруға болады. Программаға қатысатын мәні өзгеріп отыратын барлық шамалар айнымалылар ретінде жарияланады. C++ тілінде айнымалыларды жариялау

немесе сипаттау программаның кез келген қажет болған жерлерінде жүргізіле береді.

Синтаксисі:

деректің_типі айнымалының_аты;
деректің_типі айнымалының_аты = бастапқы мәні;
деректің_типі айн1_аты = баст_мәні, айн2_аты= баст_ мәні;

Мысалы:

```
short  i, j ;
char  s='55', k='77';
unsigned  long int  maxSize, minSize, fixSize=255;
```

2.6.2. Айнымалылар және жады кластары

С++ тілінде айнымалылардың программа жұмысына қатысуын басқару мүмкіндігі қарастырылған. Кейбір айнымалылар программада қажет болған жағдайда ғана құрылып, ол қажет болмаған кезде жойылып кетеді, кейбіреулері бірнеше рет құрылып, бірнеше рет жойылады, ал кейбір айнымалылар программа жұмысына бастан-аяқ қатысатындай болуы мүмкін.

Айнымалылардың программаға қатысу уақытының ұзақтығына байланысты олардың жадыдағы орын алу уақыты анықталады. Айнымалының жадыдан орын алу уақытын сипаттау үшін *жады кластары* немесе *жады түрлері* деп те айтуға болатын арнайы ұғым қолданылады. С++ тілінде айнымалылар үшін төрт түрлі жады кластарын анықтайтын спецификаторлар қолданылады: **auto, register, extern** және **static**. Бұл спецификаторлар айнымалыны жариялағанда бірге көрсетіледі, синтаксисі:

спецификатор_аты айнымалы_типі айнымалы_аты;

Мысалы,

```
auto float x,y;
register  int i=1;
extern  int a;
static int b=5;
```

Осы аталған спецификаторлар сол айнымалылардың программа жұмысына қатысу мүмкіндігін тағайындайтын болады, яғни осы спецификаторлар көмегімен айнымалының программадағы көріну немесе әсер ету аймағы және қай жады класына жататындығы анықталады.

Айнымалының әсер ету аймағы (немесе көріну аймағы) деп программаның сол айнымалыны (ол жай айнымалы немесе тұрақты немесе функция немесе т.б. идентификаторлар болуы мүмкін) пайдалануға болатын ғана фрагментін немесе бөлігін айтады. Бұл кейбір айнымалыларды программаның тек белгілі бір рұқсат етілген бөлігінде ғана пайдалануға болады дегенді білдіреді. Мұндай айнымалыларды әдетте *жергілікті (локалдық) айнымалылар* деп атайды. Жергілікті айнымалылар { }- фигуралы жақшаларға алынған кез келген блоктар ішінде жарияланады және сол блоктың ішінде ғана көрінетін болады, яғни сыртқа шықпайды. Сондай-ақ кейбір айнымалыларды программаның кез келген жерінде шақырып пайдалана беруге болады, яғни олар программаның кез келген жерінен көрініп тұратын айнымалылар, оларды *кең ауқымды (глобалдық) айнымалылар* деп атайды. Кең ауқымды айнымалылар блоктан тыс, оның сыртында жарияланады. Мысалы,

```
. . .
int max;  // кең ауқымды айнымалыны жариялау
. . .
int main()
{
   int a=4,b=3; // жергілікті айнымалыларды жариялау
    . . .
}
```

Айнымалылардың жергілікті және кең ауқымды болып бөлінуі жады кластарының екі түрге бөлінуімен байланысты: жергілікті уақытпен берілген *автоматты жады кластары* (автоматический класс памяти с локальным временем жизни) және *кең ауқымды уақытпен берілген статикалық жады кластары* (статический класс памяти с глобальным временем жизни).

Жергілікті уақытпен берілген автоматты жады кластары айнымалыларын жариялағанда, **auto** немесе **register** спецификаторларының

біреуі қолданылады. Әдетте (по умолчанию) барлық программалардағы блоктың ішінде сипатталған барлық айнымалылар жергілікті уақытпен берілген автоматты жады кластарының айнымалылары болып қабылданған, сондықтан олардың алдына **auto** спецификаторын жазбай-ақ кете беруге болады. Бұл жергілікті уақытпен берілген айнымалылар блок орындалған кезде ғана пайда болады, ал блок біткеннен кейін олар жадыдан жойылып кетеді.

Егер компилятор айнымалыны оперативті жадыда емес, жоғары жылдамдықтағы регистрлердің бірінде сақтауы керек болса, онда жергілікті уақытпен берілген айнымалы идентификаторының алдын **register** спецификаторы жазылады. Бірақ соңғы уақыттағы жетілдірілген компиляторлар жергілікті уақытпен берілген айнымалыны аппараттық регистрде сақтау-сақтамауды оның тиімділігіне байланысты өздері автоматты түрде анықтайтын болғандықтан **register** спецификаторы көп қолданылмайды.

Кең ауқымды уақытпен берілген статикалық жады кластары айнымалыларын жариялауда **extern** және **static** спецификаторларын пайдаланады.

extern спецификаторы статикалық жады кластары айнымалылары болып табылатын кең ауқымды айнымалыларды жариялау үшін қолданылады. Әдетте (по умолчанию) барлық программалардағы блоктан тыс жарияланған барлық кең ауқымды айнымалылар глобалдық уақытпен берілген статикалық жады кластары айнымалылары болып есептеледі де, сондықтан олардың алдына **extern** спецификаторы жазылмайды. **extern** спецификаторын көбінесе бірнеше файлдарда бір айнымалыны ортақ пайдалану үшін қолданады. Мысалы, қандайда болмасын a=5 деген бүтін айнымалыны бір жобаға қатысатын Unit1.cpp және Unit2.cpp файлдары ортақ пайдаланатын болса, онда Unit1.cpp файлындағы мәтінде *a* айнымалысы:

```
int a=5;
```

кең ауқымды айнымалысы түрінде жарияланады, ал оның Unit2.cpp файлындағы мәтіннің кең ауқымды айнымалысы түрінде жариялануы келесі түрде болады:

```
extern int a;
```

сонда компилятор екі жарияланууда да беріліп тұрған бір ғана a=5 айнымалысы деп қабылдайды.

Егер кең ауқымды уақытпен берілген статикалық жады кластары айнымалысы жергілікті айнымалы болатын болса, онда оны жариялау үшін **static** спецификаторын қолданады. Ереже бойынша жергілікті айнымалы блоктың ішінде ғана анықталады және сол блок орындалғанда пайда болып, блок біткесін ол да жойылып кететін болды. Ал блок ішінде **static** спецификаторын қолданып жарияланған жергілікті айнымалы блок біткесін де жойылмайтын болады, яғни ол программаның барлық орындалуы кезінде жадыда өз мәнін сақтап, бар болып тұрады.

2.7. Амалдар

Программа жазу барысында өрнектерді, шарттарды жазу, идентификаторларға белгілі бір шамаларды меншіктеу және т.б. арнаулы амалдардың көмегімен жүзеге асырылады. Амалдардың орындалуына қатысатын шамаларды әдетте операндалар деп атайды. Мысалы, $a+b$ өрнегіндегі a мен b шамалары «+» амалының операндалары болып табылады. Операндаларының санына байланысты амалдар, екі операндасы бар бинарлық амалдар және бір ғана операндамен жұмыс жасайтын унарлық амалдар болып бөлінеді. Бұлардан басқа атқаратын қызметтеріне қарай амалдарды келесі топтарға біріктіруге болады:

– арифметикалық амалдар;
– қатынас амалдары;
– логикалық амалдар;
– биттік амалдар;
– меншіктеу амалдары және т.б.

2.7.1. Арифметикалық амалдар

Арифметикалық амалдар нақты сандармен, бүтін сандармен және көрсеткіш деп аталатын типтермен жұмыс жасау үшін қолданылады. Арифметикалық амалдар бинарлық немесе унарлық болып келеді.

2.3-кесте

Бинарлық арифметикалық амалдар

Таңбасы	Аты	Операнданың және нәтиженің типтері	Мысалдар
+	қосу	бүтін, нақты, көрсеткіш	a+b
-	азайту	бүтін, нақты, көрсеткіш	a-b
*	көбейту	бүтін, нақты	a*b
/	бөлу	бүтін, нақты	a/b
%	қалдықты табу	тек қана бүтін	a%b

2.4-кесте

Унарлық арифметикалық амалдар

Таңбасы	Аты	Операнданың және нәтиженің типтері	Мысалдар
++	инкремент	бүтін, нақты, көрсеткіш	a++ ; немесе ++a;
--	декремент	бүтін, нақты, көрсеткіш	b--; немесе --b;

Унарлық арифметикалық амал- *инкремент* өзінің операндасының мәнін 1-ге арттырып отырады, ал *декремент* 1-ге кемітіп отырады. Инкремент және декремент амалдары қосу немесе азайту амалдарымен салыстырғанда, жылдамырақ орындалады. Егер ++ немесе -- белгілері операнданың алдында тұратын болса, мысалы, ++a немесе --b болса, онда мұны инкременттің немесе декременттің *префиксті* түрде жазылуы деп қабылдайды, ал ++ немесе -- белгілері операндадан кейін тұрса, онда ол *постфиксті* түрде жазылған болып есептеледі. Жазылу түріне байланысты инкременттің (немесе декременттің) орындалулары да өзгеше болады. Префикс түрінде жазылған инкремент (немесе декремент) орындалғанда ондағы операнда-айнымалының мәні әуелі 1-ге арттырылады (немесе кемітіледі), сонан соң барып ол өзін пайдаланатын өрнекте қолданылады. Ал постфиксті түрде жазылатын болса, онда икременттегі (немесе декременттегі) операнда-айнымалы әуелі өзін пайдаланатын өрнекте қолданылады да, сонан соң барып оның мәні 1-ге артады (немесе кемиді).

2.4-жаттығу. Келесі программаның

```
#include <vcl.h>
#include <iostream.h>
#include <conio.h>
int main()
{ float i=0.1, j=0.5, k;
  k = ++i + j++;
  cout<<"k="<<k<<endl;
  k = ++i + j++;
   cout<<"k="<<k<<endl;
  getch();
}
```

орындалуының нәтижесі:

```
k=1.6
k=3.6
```

Мұндағы k бірінші рет есептелгенде, k=1.6 шығады, себебі өрнектегі бірінші операнда префиксті түрде жазылған ++i болғандықтан, оның ағымдағы мәні 0.1 шамасы 1-ге арттырылып, яғни оның мәні 1.1 болғасын барып өрнекте қолданылады, ал екінші операнда j++ постфиксті түрде болғандықтан, өрнекте ол өзінің ағымдағы мәні 0.5 болып, қолданылып болған соң, сонан кейін барып өзгереді. Программада k екінші рет есептелгенде, ++i-дің мәні ендігі 2.1, ал j++-тің мәні 1.5 болып тұрады.

2.7.2. Қатынас амалдары

Қатынас амалдары шартты өрнектер құру үшін қолданылады. Бұл шартты өрнектердің мәні екеу-ақ болады, «ақиқат» – **true** (немесе 1) және «жалған» – **false** (немесе 0). Шартты өрнектегі қатынас амалы орындалатын болса, онда оның мәні «ақиқат», ал орындалмаса «жалған» болады.

2.5-кесте

Қатынас амалдары

Таңбасы	Аты	Операнданың типтері	Мысал
<	кем	бүтін, нақты, көрсеткіш	if (i<0)
<=	кем не тең	бүтін, нақты, көрсеткіш	if (i<=0)
>	артық	бүтін, нақты, көрсеткіш	if (i>0)
>=	артық не тең	бүтін, нақты, көрсеткіш	if (i>=0)
= =	тең	бүтін, нақты, көрсеткіш	if (c=='s')
!=	тең емес	бүтін, нақты, көрсеткіш	if (k!= 6)

2.7.3. Логикалық амалдар

Логикалық амалдар программада бірнеше шарттардан тұратын құрама шарттарды беруде жиі қолданылады. Логикалық амалдардың операндалары нәтижелері скаляр, яғни сандық шамалар болатын өрнектер болады. Бұл логикалық амалдардың нәтижесі **true** немесе **false** болады.

2.6-кесте

Логикалық амалдар

Таңбасы	Аты	Мысал
&&	Логикалық «және» (AND)	if (i<1 && i<5)
||	Логикалық «немесе» (OR)	if (c==0 || c==1)
!	Логикалық «емес» (N0T)	if (!(x>1 && x<5))

Мұндағы (2.6-кестедегі) && және || амалдары бинарлық амалдар болып табылады.

Логикалық «және» – && амалының мәні, егер оның екі операндасы да нольден өзгеше (немесе **true**) болғанда, тек сонда ғана **true** бола алады, ал қалған жағдайдың бәрінде **false** мән беретін болады.

Логикалық «немесе» – || амалының мәні, егер оның екі операндасы да 0-ге (немесе **false)** тең болғанда, тек сонда ғана **false** мәнге ие бола алады, ал қалған жағдайдың барлығында **true** мән беретін болады.

Логикалық «емес» – ! амалы – унарлық амал, яғни ол бір ғана операндамен жұмыс жасайды. Егер ол операнданың мәні 0-ге тең болса, онда ! –

амалының нәтижесі **true** мәнге, ал операнда нольден өзгеше болғанда, мәні **false** болады.

2.7.4. Меншіктеу амалдары

C++ -те меншіктеу амалының қызметін «=» белгісі атқарады, бірақ программа құруда арифметикалық амалдармен қоса меншіктеу операторлары жиі қолданылады және олар арифметикалық амалдарды ықшам түрде жазуға мүмкіндік береді (2.7-кесте):

2.7-кесте

Арифметикалық меншіктеу амалдары

Таңба	**Аты**	**Операнданың және нәтиженің типтері**	**Мысалдардың жазылуы**	
			қысқаша	**толық**
+ =	қосумен меншіктеу	бүтін, нақты, көрсеткіш, құрылым, бірігу	x+=5	x=x+5
– =	азайтумен меншіктеу	бүтін, нақты, көрсеткіш, құрылым, бірігу	x-=10+y	x=x-(10+y)
* =	көбейтумен меншіктеу	бүтін, нақты	min*=2	min= min*2
/ =	бөлумен меншіктеу	бүтін, нақты	z /= x+y	z=z/ (x+y)
% =	қалдықты меншіктеу	бүтін	z %= 2	z=z % 2)

C++-те меншіктеу амалының тағы бір ерекшелігі оны қатынас амалдарының көмегімен жасалатын логикалық өрнектерде араластыра қолдануға болады, мысалы, келесі программа фрагменті:

```
if (f=x-y>0)
cout <<"airma on san";
```

орындалғанда, алдымен x-y айырма есептеліп, f айнымалысына меншіктеледі де, содан кейін оның мәні 0-мен салыстырылады;

Сондай-ақ, C++-те меншіктеу амалын тіркестіріп те қолдануға болады, мысалы, мына түрде жазуға болады:

x=y=z= a+b, мұнда алдымен a+b өрнегінің мәні есептеледі де, ол мән бірінші z-ке, содан соң y-ке, ең соңынан барып x-ке меншіктеледі.

2.7.5. Биттік амалдар

C++ программалау тілінде бүтін сандық типтердің екілік формат-тағы түрімен жұмыс жасауға мүмкіндік беретін, яғни биттер түрінде берілген сандарға қолданылатын амалдар қарастырылған.

2.8-кесте

Биттік амалдар

Таңбасы	**Аты**	**Мысалдар**
&	Биттік AND	i & 128
|	Биттік OR	j | 25
^	Биттік XOR	k ^12
~	Биттік NOT	~ k
<<	Биттік разрядтарды солға қарай жылжыту	i<<2 (солға екі орын жылжыту)
>>	Биттік разрядтарды оңға қарай жылжыту	j>>3 (оңға үш орын жылжыту)

Биттік амалдардың арифметикалық амалдардағы сияқты қысқаша жазылуы да қолданылады, мысалы:

x=x&y немесе x&=y;
x=x|y немесе x|=y;
x=x^y немесе x^=y;
x=x>>3 немесе x>>=3 және т.б.

Бұл &, |, ^ биттік амалдарды бүтін сандарға қолдану үшін келесі кестеде (2.9-кесте) берілген мәндер қолданылады.

2.9-кесте

Биттік амалдардың ақиқаттық кестесі

x	**y**	**x&y**	**x|y**	**x^y**
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

Келесі «~» биттік амалы екілік форматтағы бүтін санның әрбір разрядын керісіншеге ауыстырып отырады, яғни 0 болса, 1-ге ауысады немесе керісінше 1 болса, 0-ге деген сияқты.

Биттік разрядтарды оңға қарай жылжыту амалы >> сол жақ операндамен берілген бүтін санның екілік форматындағы разрядтарды оң жақ операндамен берілген орынға жылжытады және мұнда оң жақтағы разрядтар жойылады. Егер сол жақтағы операнда таңбасыз берілген бүтін сан болса, онда сол жақтағы пайда болған бос орындар нольмен толтырылады, ал керісінше жағдайда, таңба белгісімен толады.

Биттік разрядтарды солға жылжыту амалы << орындалғанда, сол жақ операндадағы бүтін санның разрядтарды оң жақ операндамен берілген орынға солға қарай жылжиды, мұндағы сол жақтағы разрядтар жойылмайды, пайда болған оң жақтағы разрядтар нольмен толтырылады. Бүтін санның биттік разрядтарын солға қарай **n** орынға жылжытқанда, шығатын екілік сан сол бүтін санды $\mathbf{2^n}$ санға көбейткенде шығады.

2.5-жаттығу. Биттік амалдардың орындалуын көрсететін келесі

```
#include <vcl.h>
#include <iostream.h>
#include <conio.h>
int main()
{  int x=17, y=15;
cout<<"17 & 15 ="<<(17&15)<<endl;
 cout<<"17 | 15 = "<<(17|15)<<endl;
 cout<<"17 ^ 15 = "<<(17^15)<<endl;
 cout<<"15 >> 2 = "<<(15>>2)<<endl;
cout<<"-15 >>2 = "<<(-15>>2)<<endl;
cout<<"17 << 3 = "<<(17<<3)<<endl;
 cout<<"-17 << 3 = "<<(-17<<3)<<endl;
getch();      }
```

программаның нәтижесі төмендегідей болып шығады:

```
17 & 15 = 1
17 | 15 = 31
17 ^ 15 = 30
15 >> 2 = 3
-15 >>2 = -4
17 << 3 = 136
-17 << 3 = -136
```

Жаттығуды талдайтын болса, 17 & 15 деген 10001 & 1111 өрнегін береді, себебі $17=10001_2$ және $15=1111_2$, егер бұл разрядтарды 2.9-кестедегі биттік амалдардың ақиқаттық кестесіне сәйкес қосатын болсаңыз, 10000_2 деген сан шығады, ал бұл ондық жүйедегі 1 саны болады (Екілік сандарға мұндай амалдарды қолданғанда, сандарды сол жақтан бастап жазады, оң жақтағы жетпеген разрядтардың орнына 0 жазылады, мысалы, 10001 & 11110 сияқты). Тура сол сияқты 17 | 15 амалы орындалғанда, 11111_2 саны алынады, ол ондық жүйедегі 31 санына сәйкес келеді.

Программадағы (15>>2) жылжыту амалы орындалғанда, 15 санының екілік жүйедегі түрі 1111_2 санындағы разрядтар екі орынға алдыға жылжиды да, оң жақтағы екі разряд жоғалады, сонда қалатыны 11_2 немесе бұл ондық жүйедегі 3 саны.

Қалған амалдар да жоғарыда айтылған ережелер бойынша орындалады.

2.7.6. Шартты амал (?:)

Шартты амал (?:) – C++ тіліндегі үш операндамен жұмыс жасайтын бірден-бір амал. Синтаксисі :

```
шарт ? өрнек_1 : өрнек_2
```

мұндағы шарт – кез келген скаляр шамалы өрнек, ал өрнек_1 және өрнек_2 типтері бірдей болатын кез келген шамалар бола алады. Шартты амалдағы бірінші операнда – шарт **true** болғанда алынатын мән

өрнек_1 – екінші операндада сақталады, ал шарт **false** болғандағы мән бұл өрнек_2 болып табылады, мысалы:

```
(max < b) ? b : max;
```

Әдетте, бұл шартты амал орындалғанда шығатын нәтижені типі екінші, үшінші операндалардағы өрнектер типімен бірдей болатын айнымалыға меншіктейді, мысалы:

```
max = (max < b) ? b : max;
```

орындалғанда егер шарт орындалса max айнымалысының мәні b-ға тең болады, ал орындалмаса, max мәні ағымдағы өзінің мәніне тең болып шығады. Бұл шартты амалдың орындалуы кәдімгі **if** . . . **else** операторының қызметіне ұқсайды, сондықтан бұл амалды **if** . . . **else** операторын қолдану синтаксистік тұрғыдан мүмкін болмаған немесе бір тиімді жағдайларда қолданады.

2.6-жаттығу. Келесі программа

```
# include <iostream.h>
#include <conio.h>
int main()
{ int max=10, b=7;
   max=(max<b)? b : max;
  cout<<"max="<<max;
   getch();  }
```

b=7 немесе b=17 мәндері үшін орындалатын болса, онда нәтижеде сәйкесінше max=10 немесе max=17 шығатынын көруге болады.

2.7.7. Үтір амалы (,)

Бұл амал бірнеше өрнектерді бірінен кейін бірін тіркестіріп жазу үшін қолданылады, синтаксисі:

өрнек_1 , өрнек_2, . . . , өрнек_n,

мұнда өрнектердің мәнін есептеу сол жақтан басталады және олардың нәтижелері void болады, тек қана оң жақтағы соңғы өрнектің нәтижесінің типі сақталады да, бұл үтір амалының нәтижесі де сол соңғы өрнектің нәтижесімен бірдей болады. Үтір амалын көбінесе цикл операторларындағы параметрлердің мәндерін беруде қолданады, мысалы:

```
for (i=0;  i<4; i++ )
for (j=0;  j<5;  j++)
            {  . . .  };
```

программа фрагментін үтір амалын пайдаланып бір-ақ жолмен жазуға болады:

```
for (i=0, j=0; i<4, j<5; i++, j++)
                {. . . }.
```

2.7.8. ***sizeof*** *амалы*

C++ тіліндегі амалдар алдын ала анықталған белгілі бір функциялар сияқты жұмыс жасайтыны белгілі. Сондай унарлық амалдардың бірі **sizeof** өзінің операндасының, жадыдан алатын орнын байтпен шығарып бере алады, синтаксисі екі түрлі болып келеді:

```
sizeof  (айнымалы_аты);                немесе
sizeof өрнек;
```

Бұл амалдың операндасы айнымалы болса, онда оның нәтижесі сол айнымалы жататын типтің ұзындығын көрсететін байт болады. Ал операнда өрнек болатын болса, онда сол өрнекке қатысатын барлық айнымалылардың, тұрақтылардың типтерінің ұзындықтарының қосындысын береді.

2.7- жаттығу. Төменде мәтіні берілген

```
#include <vcl.h>
#pragma hdrstop
 #include <iostream.h>
 #include <conio.h>
int main()
{
  int a=2,b=3, c=5;
cout << "\n (unsigned)int = " << sizeof(int);
cout << "\n (unsigned)short =" << sizeof(short);
cout << "\n (unsigned)char = " << sizeof(char);
cout << "\n (unsigned)float = " << sizeof(float);
cout << "\n (unsigned)double = "<< sizeof(double);
cout << "\n (unsigned)long = "<< sizeof(long);
cout << "\n (unsigned)long double = "<< sizeof(long double);
cout << "\n a+b+c ornek yzindigi= "<< sizeof a+b+c;
  getch();  }
```

программаның орындалуының нәтижесі:

```
(unsigned)int = 4
(unsigned)short = 2
(unsigned)char = 1
(unsigned)float = 4
(unsigned)double = 8
(unsigned)long = 4
(unsigned)long double = 10
a+b+c ornek yzindigi= 12
```

Бұл мәндер компьютер құрамындағы процессордың немесе программалық жабдықтың версиясына байланысты өзгеруі мүмкін.

2.7.9. Ағынға қосу (<<) және ағыннан алу (>>) амалдары

Жалпы «ағын» (поток) ұғымы программаға деректерді енгізу және программадан деректерді алуға байланысты қолданылады. Осыған дейінгі қарастырылған программалардың барлығында дерлік кездесіп отырған #include <iostream.h> препроцессор директивасының қызметі де осы компиляцияланатын программаны ағындармен жұмыс жасайтын арнаулы кітапханалармен байланыстыру болып табылады. Бұл iostream.h (мұндағы «i» – input – «енгізу», «o» – output – «шығару» сөздерінің бірінші әріптерін, ал «stream»-ағын дегенді білдіреді) файлында сипатталған енгізу-алу объектілері арқылы программист ағынға деректі қосу және ағыннан деректі алу мүмкіндіктерін пайдалана алады. Ағынды әдетте байттардың (символдардың) тізбегі ретінде қарастырады. Бұл байттар тізбегі программаға деректі енгізуші немесе шығарушы белгілі бір құрылғыларға (мысалы, дискідегі файл, пернетақта, принтер, дисплей экраны және т.б.) байланысты қолданылып отырады.

C++ тілінде деректі *жіберушіден* оны *қабылдаушыға* жеткізуде стандарт ағындар ретінде анықталған cin және cout объектілері қолданылады. cin стандарт енгізу ағыны қолданылғанда, жіберушінің ролін пернетақта, ал қабылдаушының қызметін орындалып тұрған программа атқарады. cout стандарт шығару ағыны қолданылғанда, жіберушінің ролін программа, ал қабылдаушының қызметін дисплей экраны атқарады. Бұл cin стандарт енгізу ағыны мен cout стандарт шығару ағынын негізінен ағынға қосу (<<) және ағыннан алу (>>) амалдарымен пайдаланады.

Экранға шығарылатын деректерді шығару ағынына қосу «ағынға қосу» (<<) амалымен орындалады. Мұны бинарлық амал деп қарастыруға болады, синтаксисі:

```
ағын_объектісі << аргумент;   немесе
cout << аргумент;
```

бұл амал орындалғанда, бірінші операнданың орнында тұрған ағынға екінші операнданың орнында тұрған аргумент қосылатын болады.

Бұл амалды *манипуляторлар* деп аталатын endl (ағынды жаңа жолға бағыттау) және бүтін сандарды шығару үшін үшін **dec** (ондық форматта), **oct** (сегіздік форматта), **hex** (он алтылық форматта) қызметші сөздерімен бірге қолдануға да болады. Сондай-ақ, <iomanip.h> тақырыптық файлында функция түрінде анықталған setprecision() манипуляторын жылжымалы үтірмен берілетін нақты сандардағы үтірден кейінгі таңбалардың санын көрсету үшін қолданады.

2.8-жаттығу. Келесі

```
#include <vcl.h>
#include <iostream.h>
 #include <conio.h>
#include<iomanip.h>// setprecision() манипуляторы үшін қосылды
int main()
{   int a=2009;
cout<< "8-formatta ="<<oct<<a<<endl;
cout<< "16-formatta ="<< hex<<a<<endl;
float b=0.123456789;
cout<< "\n setprecision(3) bolganda, b ="<<
setprecision(3)<<b<<endl; //үтірден кейін үш сан қалдыру
cout<< "setprecision(5) bolganda, b ="<<
setprecision(5)<<b<<endl; //үтірден кейін бес сан қалдыру
getch();  }
```

программа орындалғанда, 777 санының *сегіздік* және *он алтылық* форматтағы түрлері шығатын болады. Мұндағы setprecision() манипуляторы **float** b=0.123456789 санындағы үтірден кейінгі қанша санды қалдыру керек екенін тағайындайды, яғни программа нәтижесі келесі болады:

```
8-formatta =3731
 16-formatta =7d9

 setprecision(3) bolganda, b =0.123
setprecision(5) bolganda, b =0.12346
```

Ағыннан алу (>>) амалы орындалғанда, пернетақтадан немесе басқа құрылғыдан енгізілетін символдар тізбегі программадағы сипатталуына сәйкес, белгілі бір базалық типтердің, мысалы, **int, long, double, char** немесе т.б. типтердің біріне түрленеді. Программадағы жазылуы:

```
ағын_объектісі >> аргумент;    немесе
cin >> аргумент;
```

Бұл бинарлық ағыннан алу (>>) амалы оң жақтағы операнда-аргументтің типіне байланысты үш түрлі орындалады, яғни бүтін, нақты және жолдық типтер үшін үш түрлі орындалады, себебі олардың ішкі берілу кодтары түрліше болады.

2.9-жаттығу. Төмендегі программаны

```
#include <vcl.h>
#include <iostream.h>
 #include <conio.h>
void main()
{
union birigu
  {long bytin;
  char gol[4];
  float nakti; } ainimali;
       cout<< "bytin san engiz ="<<endl;
cin>> ainimali.bytin;
cout<<"ainimali.bytin= "<< ainimali.bytin<<endl;
cout<< "ainimali.gol="<< ainimali.gol<<endl;
cout<< "ainimali.nakti="<< ainimali.nakti<<endl;
       cout<< "gol engiz ="<<endl;
cin>> ainimali.gol;
cout<< "ainimali.bytin= "<< ainimali.bytin<<endl;
cout<< "ainimali.gol="<< ainimali.gol<<endl;
cout<< "ainimali.nakti= "<< ainimali.nakti<<endl;
       cout<< "nakti san engiz ="<<endl;
   cin>> ainimali.nakti;
```

```
cout<< "ainimali.bytin= "<< ainimali.bytin<<endl;
cout<< "ainimali.gol= "<< ainimali.gol<<endl;
cout<< "ainimali.nakti= "<< ainimali.nakti<<endl;
getch();
}
```

орындағанда шығатын нәтиже мына түрде болады:

```
bytin san engiz =
777
ainimali.bytin= 777
ainimali.gol= ♥
ainimali.nakti= 1.08881e-42
 gol engiz =
777
ainimali.bytin= 3618615
ainimali.gol= 777
ainimali.nakti= 5.07076e-39
 nakti san engiz =
777
ainimali.bytin= 1145192448
ainimali.gol=
ainimali.nakti= 777
```

Бұл нәтижелер талданатын болса, онда 777 санын бүтін, жолдық және нақты типтер ретінде үш түрлі енгізгенде, оларға жадыдан берілетін 4 байт орын және символдарда (777) бірдей болғанымен, олар жадыға типіне қарай түрліше қабылданады және ағынға шығарғанда да түрліше болып шығады. Мұндағы **union** типі туралы 3-тараудың 3.9-бөлімінде қарастырылады.

*2.7.10. Жады ұяшығы адрестерімен жұмыс жасауға арналған амалдар (& және *)*

C++ Builder-де жады ұяшықтарына деректерді тікелей орналастыру және оны ұяшықтардан қайта алу үшін *көрсеткіштер* деп аталатын

арнайы айнымалылар қолданылады. Бұл көрсеткіштердің мәні жады ұяшықтарының адрестері болып табылады. Осы көрсеткіштермен жұмыс жасау үшін *&-жады ұяшығының адресін алу* және **- адресі бойынша ұяшықтан мәнді алу* амалдары қолданылады. Бұл амалдар унарлық амалдар болып табылады (Толығырақ 3-тараудың, 3.5-бөлімінде қарастырылады).

2.7.11. Объектілермен жұмыс жасауға арналған амалдар (. және ->)

Нүкте (.) және бағыттауыш (->) амалдары бірнеше бөлімдерден немесе элементтерден құралатын «құрылым» және «класс» секілді күрделі типтермен жұмыс жасау үшін қолданылады. Егер тип объект түрінде қарастырылатын болса, онда ол объектінің құрамындағы элементтерін пайдалану үшін « . » (нүкте немесе объектінің аты бойынша элементті таңдау) амалын қолданады, мысалы, student объектісінің fam элементіне мән беру үшін келесі түрде жазады:

```
student.fam="Асанова";
```

Егер элементті пайдалану объектіге сілтеме жасайтын көрсеткіштер арқылы жүргізілетін болса, онда «->» (бағыттауыш немесе объектіге көрсеткіш бойынша элементті таңдау) амалын қолданады, мысалы, C++ Builder-де түрлі кластарға жататын объект-компоненттерді пайдалану үшін осы амалды қолданады, мысалы:

```
Label1->Caption="Асанова";
```

Егер объектінің аты бойынша элементті таңдау үшін көрсеткіш қолданылатын болса нүкте (.) амалы «нүкте-жұлдызша» (.*), ал бағыттауыш (->) амалы «бағыттауыш-жұлдызша» (->*) түрінде жазылады.

Нүкте (.) және бағыттауыш (->) амалдарының қолданылуы туралы 6-тараудың 6.3-бөлімінде толығырақ қарастырылады.

2.7.12. Әрекет ету облысын тағайындау амалы (::)

Айнымалының немесе элементтің әрекет ету облысын тағайындау амалы (::) екі түрлі жағдайда қолданылады.

Біріншісі, кең ауқымды айнымалымен (глобальная переменная) аты бірдей болатын айнымалы жарияланғанына қарамастан, сол блокта кең ауқымды айнымалыны пайдалануға мүмкіндік беретін :: – унарлық амалы, оның жазылуы келесі түрде болады:

```
:: айнымалының_аты
```

Мысалы, ::I – бұл сол блокта немесе басқа да блоктарда сипатталған I айнымалылардың болуына қарамастан, ::I – айнымалының кең ауқымды айнымалы екенін көрсетіп тұрады.

Екіншісі, объектіге бағдарланған программалаудағы «кластар» типі үшін қолданылатын :: – бинарлық амалының жазылуы келесі түрде болады:

```
кластың_аты :: кластың_элементі
```

мұндағы :: – бинарлық амалы, «класстың_элементімен» атаулары бірдей болатын басқа кластардағы элементтердің немесе функциялардың болуына қарамастан, арнайы :: – бинарлық амалы арқылы беріліп тұрған «кластың_элементін» пайдалануды тағайындайтын болады (6.2-тақырыпта толығырақ қарастырылатын болады).

2.7.13. Амалдар приоритеті

Амалдар приоритеті бірнеше амалдар қатар қолданылатын күрделі өрнектердегі амалдардың, белгілі бір ретпен орындалу тәртібін анықтайды (2.10-кесте). Программалауда амалдар приоритетінің сақталуының үлкен маңызы бар, сондықтан программа құруда амалдар орындалуының тәртібі программист тарапынан ескеріліп отырғаны дұрыс болады.

2.10-кесте

Амалдар приоритеті мен ассоциативтілігінің кестесі

№	Амалдың таңбалары	Ассоциативтілігі
1	() [] ::	Солдан оңға қарай
2	! ~ + – ++ -- & * **sizeof new delete**	Оңнан солға қарай
3	.* ->*	Солдан оңға қарай
4	* / %	Солдан оңға қарай
5	+ -	Солдан оңға қарай
6	<< >>	Солдан оңға қарай
7	< <= > >=	Солдан оңға қарай
8	== !=	Солдан оңға қарай
9	&	Солдан оңға қарай
10	^	Солдан оңға қарай
11	|	Солдан оңға қарай
12	&&	Солдан оңға қарай
13	||	Оңнан солға қарай
14	?:	Солдан оңға қарай
15	= *= /= %= += -= &= ^= |= <<= >>=	Оңнан солға қарай
16	,	Солдан оңға қарай

Жоғарыдағы кестеде амалдардың орындалу тәртібі он алты деңгеймен анықталып тұр, ең жоғарғы деңгей 1-ші, ал төменгісі 16-шы болып тұр. Егер приоритеттері бірдей амалдар қатар келгенде, олар ассоциативтілігіне байланысты тұрған реті бойынша солдан оңға қарай немесе керісінше орындала береді. Кестеде кейбір амалдардың таңбалары бірнеше рет қайталанады, олардың біріншісі унарлық амалдарға, ал екіншісі бинарлық амалдарға сәйкес келеді.

2.10-жаттығу. Төмендегі программаны орындағанда

```
#include <iostream.h>
#include <conio.h>
int main()
{
  int a=12,b=5,c=3,d=0;
```

```
d=a+2*b%c*3;
  cout<<"d="<<d;
 getch(); }
```

шығатын нәтиже d=15 болады, себебі a+2*b%c*3 өрнегіндегі «*» және «%» амалдарының приоритеті (2.10-кесте, 4-орын) «+» амалынан (2.10-кесте, 5-орын) жоғары сондықтан олар бірінші орындалады, ал «ассоциативтілігі» бойынша солдан оңға қарай алдымен 2*b, сонан соң «%» амалы, одан кейін барып екінші «*» амалдары орындалады.

2.8. Операторлар

С++ тілінде жазылған программадағы әрекеттердің орындалу тәртібін басқару үшін барлық құрылымдық программалау тілдеріндегі сияқты негізгі құрылымдарды, яғни шартқа байланысты белгілі бір әрекеттің орындалуын тағайындайтын немесе белгілі бір әрекеттің қайталануын жүзеге асыратын немесе т.б. орындайтын арнайы операторлар қолданылады. Оларды шартты түрде келесі топтарға біріктіруге болады:

– басқару операторлары **(if , switch , goto)**;
– циклді ұйымдастыру операторлары **(for, while , do while);**
– циклмен жұмыс жасайтын операторлар **(break, continue).**

*2.8.1. **if** шешім қабылдау операторы*

Шешім қабылдауға арналған **if** операторының қызметі, бұл шартқа байланысты орындалатын операторлар арасының қайсы операторды орындалатынын тағайындайды, яғни программадағы басқаруды сол операторға жібереді. **if** операторының программадағы жазылуы шартқа байланысты альтернативалардың, яғни орындалатын операторлардың санына байланысты келесі түрде болуы мүмкін (2.11-кесте):

2.11- кесте

if *операторының жазылу түрлері*

Түрлері	**Синтаксисі немесе жазылуы**	**Мысалы**
Бір альтернативті	**if** (шарт) оператор;	**if** (x>=0) x+=1;
Екі альтернативті	**if** (шарт) оператор_1; **else** оператор_2;	**if** (max==i) max=i; **else** max=j;

Көп альтернативті	**if** (шарт_1) оператор_1;
	else if (шарт_2)
	оператор_2;
	. . .
	else if (шарт_N)
	оператор_N;
	else
	оператор_N+1;
		if (amal=='+') z=x+y;
		else if (amal=='-')
		z=x-y;
		else if (amal=='*')
		z=x*y;
		else
		z=x/y;

Шартқа байланысты бірнеше операторлар қатар орындалатын жағдайда, оларды операторлық блокқа { } алып жазады, мысалы:

```
if (max<min)
{ // операторлық блоктың басы
max=i;
min=j
}; // соңы
else
{
 max=j;
min=i
};
```

Операторлық блок бос, яғни онда бірде-бір оператор болмаса, онда мұндай бос блоктың орнына «;» /нүктелі үтір/ – белгісін қойып кетеді, мысалы:

```
if (max<min)
;// бос блок
```

```
else
{
max=j;
min=i
};
```

Шешім қабылдауға арналған **if** операторындағы шарттарды беру үшін қатынас амалдары (2.5-кесте) қолданылады, ал бірнеше шарттардың бір мезгілде қатар орындалуын тексеру үшін логикалық амалдар арқылы (2.6-кесте) логикалық өрнектер түріндегі шарттарды қолдануға тура келеді. Бұл логикалық амалдар мен қатынас амалдарын қолдану барысында амалдардың приоритетін (2.10-кесте) ескеріп отыру керек, мысалы, «!» амалының приоритеті «*» амалынан да жоғары болып келеді, ол тек «( )» – жақша амалына ғана бағына алады, сол сияқты логикалық «және» – **&&** амалының приоритеті «немесе» – || амалынан жоғары, бірақ бұл екеуінің де приоритеті «артық», «кем», «тең» және т.б. қатынас амалдарынан төмен болады, сондықтан келесі шарттың

```
a>b&&b>c||b>d
```

программадағы орындалу реті төмендегідей болады:

```
((a>b)&&(b>c))||(b>d),
```

2.11-жаттығу. Үш қабырғасы бойынша үшбұрыштың ауданын табатын программа құру керек. Программа пернетақтадан енгізілетін мәліметтердің, мысалы, сандардың үшбұрыш қабырғалары бола алатындығын тексере алатындай болуы керек. Үшбұрыштың ауданын табу үшін Герон формуласын пайдаланады.

```
#include "iostream.h"
#include "conio.h"
#include "math.h"
void main()
{ int  a,b,c; //үшбұрыштың қабырғалары
```

```
float s, p; // үшбұрыштың ауданы мен жарты периметрі
cout<<"\n a,b,c yshin sandar engiziniz \n";
cin>>a>>b>>c; //үшбұрыш қабырғалары үшін мәліметтер енгізу
                                    //енгізілген мәліметтерді тексеру
if (a<=0 || b<=0 || c<=0 ) //үшбұрыштың қабырғалары нольден өзгеше
                                    //және теріс емес сандар болуы керек
cout<<"\n nolden ozgeshe teris emes sandar bolyi kerek";
else
{
if ((a+b)>c&&(b+c)>a&&(a+c)>b) //кез келген екі қабырғасының
//қосындысы үшіншісінен үлкен болуы керек
{
  p= float(a+b+c)/2;
  cout<<"\n p="<<p;
  s=sqrt(p*(p-a)*(p-b)*(p-c));
  cout<<"\n s="<<s;
 }
 else
 {
    cout<<"\n"<<a<<','<<b<<','<<c;
 cout<<" yshburishtin kabirgasi bola almaidi";
 }
 }
   getch(); }
```

Программаны *a, b, c* үшбұрыш қабырғаларының әр түрлі мәндері үшін орындағанда келесі нәтижелерді көруге болады:

1-жағдай. Үшбұрыш қабырғаларының бірі теріс сандар болуын тексеру.

```
a,b,c yshin sandar engiziniz
1 2 -3
nolden ozgeshe teris emes sandar bolyi kerek
```

2-жағдай. «Үшбұрыштың кез келген екі қабырғасының қосындысы үшіншісінен үлкен болуы керек» шартының орындалуын тексеру.

```
a, b, c yshin sandar engiziniz
1 2 3
1,2,3 yshburishtin kabirgasi bola almaidi
```

3-жағдай. Барлық шарттарды қанағаттандыратын сандар беріп тексеру.

```
a, b, c yshin sandar engiziniz
2 3 4
p=4.5
s=2.90474
```

2.8.2. ***switch*** *таңдау операторы*

С++-те прогамма жазу барысында көп шарттар тексеріліп, олардың әрқайсысына сәйкес басқаруды жеке-жеке операторлардың орындалуына беру керек болған кезде **if** шарт операторын қолдану тиімді болып есептелмейді, сондықтан мұндай жағдайларда **switch** таңдау операторын қолданады. Программадағы жазылуы /синтаксисі/:

```
switch (P) {
case m1:
case m2:
...
case mN :
   оператор_1;
   оператор_2;
break;
case k1:
case k2:
```

```
...
case kN :
   оператор_3;
   оператор_4;
break;
. . .
   default:
  оператор_N;
}
```

мұндағы Р – селектор (немесе ауыстырып-қосқыш, переключатель) деп аталатын, мәні тек бүтін, логикалық, символдық, саналатын және диапазон типтердің біреуіне ғана жататын өрнек, ал m1, ... , mN, k1, ..., kN (i=1,2,3 ... N) – бұл типтері селектор-өрнек мәнінің типімен бірдей болатын тұрақтылар, оператор_1, ..., оператор_N- С++ -тің операторлары.

switch операторының программадағы орындалуы төмендегідей болады:

– Р-селектор өрнектің мәні есептеледі, анықталады;

– осы анықталған мәнге сәйкес **case** – тұрақты ізделінеді;

– сол тұрақтының тұсындағы оператор орындалады, сонан соң **break** операторы орындалып, таңдау операторы аяқталады;

– егер ондай тұрақты табылмаса – **default-тан** кейінгі оператор орындалады да, таңдау операторы аяқталады.

2.12-жаттығу. Келесі программа пернетақтадан енгізілетін жыл айларының реттік номеріне сәйкес олардың жыл мезгілдерінің қайсысына жататынын анықтап бере алады, мысалы, 12 (желтоқсан), 1(қаңтар), 2(ақпан) – қыс, 3,4,5-көктем және т.б.

```
#include <iostream.h>
#include<conio.h>
void main()
{
int nomerAi;
cout<<"\n Aidin nomerin engiz=";
```

```
cin>> nomerAi;
 switch (nomerAi)
 {
 case 12:
  case 1:
  case 2:
   cout<<"\n KIS";    break;
 case 3:
 case 4:
 case 5:
 cout<<"\n KOKTEM";  break;
 case 6:
 case 7:
 case 8:
 cout<<"\n GAZ"; break;
  default :
 cout<<"\n KYZ";
 break; }
 getch();
}
```

Программа орындалуының нәтижесі:

```
Aidin nomerin engiz=5
 KOKTEM
```

2.8.3. ***goto*** *көшу операторы*

Программада операторлардың орындалу тәртібін өзгерту үшін көшу операторы қолданылады, бірақ программаның жазылуы мен орындалуы сәйкес келуі үшін мүмкіндігінше бұл операторды қолданбай жазуға да болады. Көшу операторының программадағы жазылуы:

goto белгі; //метка

мұндағы белгінің – программа мәтінінде аты көрсетіледі де, одан кейін «қос нүкте » қойылады, содан кейін барып оператор жазылады. Мысалы:

```
goto belgi_1: cout<<"\n Byl simvol";     немесе

goto belgi_1;
... ... ...
belgi_1: cout<<"\n Byl simvol";
... ... ...
```

2.13-жаттығу. Төменде берілген программа орындалғанда, **goto** операторы қатысатын болғандықтан, ондағы операторлардың орындалу тәртібі қатаң сақталмайды. Осы программаның орындалуын бірнеше жағдайлар үшін карастырып көрелік.

```
#include <iostream.h>
#include<conio.h>
void main()
{  char c;
   cout<<"Kez-kelgen simvol engiz: ";
   cin>>c;
    switch (c)
    {
   case 'a':
   case 'b':
   case 'c':
   case 'd':
   goto belgi_1; break; //Басқару belgi_1-ге беріледі
   case '1':
   case '2':
   case '3':
   case '4':
    goto belgi_2; break; //Басқару belgi_2-ге беріледі
    default:
    goto  belgi_3;
    }
```

```
belgi_2: cout<<"\n Byl zifra"; goto belgi_3; //Басқару belgi_3-ке
                                              //беріледі
belgi_1: cout<<"\n Byl simvol";
belgi_3: cout<<"\n Programmanin soni"; getch();
}
```

1-жағдай. Мұнда пернетақтадан енгізілетін символ 'a', 'b', 'c', 'd' әріптердің бірі болсын, онда бірінші тұрған **goto** belgi_1; операторы орындалады да, басқару төменде тұрған belgi_1 ге беріледі, әрі қарай келесі жолда тұрған операторлар орындалып кете береді, нәтижесі:

```
Kez-kelgen simvol engiz: b
 Byl simvol
 Programmanin soni
```

2-жағдай. Егер пернетақтадан енгізілетін символ '1', '2', '3', '4' цифрларының бірі болса, онда екінші тұрған **goto** belgi_2; операторы орындалады да, басқару belgi_2 ге беріледі, мұндағы cout<<"\n Byl zifra"; орындалғаннан кейін әрі қарай келесі жолда тұрған **goto** belgi_3; орындалады, оның нәтижесінде басқару бір жолды аттап өтіп, belgi_3;-ке беріледі, нәтижесі:

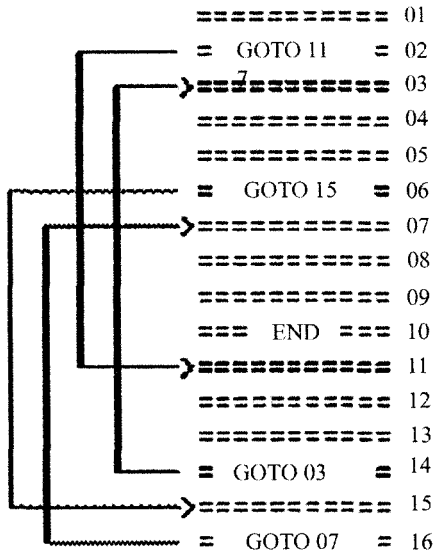
```
Kez-kelgen simvol engiz: 1
 Byl zifra
 Programmanin soni
```

3-жағдай. Программаға тексеру қарастырылмаған басқа бір символ енгізілетін болса, онда басқару **goto** belgi_3; операторы орындалғаннан кейін бірден belgi_3;-ке беріледі, нәтижесі:

```
Kez-kelgen simvol engiz: z
 Programmanin soni
```

Бұл жай ғана қарапайым программаның өзінен, **goto** операторының программадағы операторлардың орындалуын басқаруды қым-қиғаш,

ретсіз өзгерте алатынын байқауға болады, яғни, басқаша айтқанда, бірі – екіншісінің, үшіншісі біріншісінің және т.б. алдына немесе артына шығып жатады, егер программа үлкен көлемді болса және оған көптеген **goto** операторы қатысатын болса, онда тіпті де шатасуға болады (2.2-сурет), сондықтан болар программист-мамандар арасында мұндай программалар BS-программалар (аббревиатура «Bowl Spaghetti» немесе «блюдо спагетти») деген атқа ие болды.



2.2-сурет. *BS-программа листингісінің фрагменті*

Жалпы программистердің алдындағы міндет жадыдан аз орын алатын және тез жұмыс жасайтын программа құру әдістерін табу керек болды, мысалы қазіргі программалау тілдеріндегі **goto** операторын қолдану да соның бірі болып табылады. Соған қарамастан программа проектісін құрудағы негізгі міндеттердің бірі программаның статикалық күйі /бастапқы жазылуы немесе листингісі/ мен динамикалық күйі / компьютердегі орындалу тәртібі/ сәйкес келуі керек, яғни программа командалары жазылу реті бойынша орындалуы тиіс. Бұл мәселенің шешімі 70-жылдардың басында IBM корпорациясы ұсынған, теориялық негізін профессор Э. Дейкстра қалаған құрылымдық про-

граммалау технологиясын /технология нисходящего структурного программирования/ қолдану болып табылады, яғни кез келген программаны **goto** операторын пайдаланбай жазып шығуға болады. Дегенмен де кейбір алгоритмдерді жазуда **goto** операторын пайдаланудың тиімді жақтары бар екенін де ұмытуға болмайды.

*2.8.4. **while** цикл операторы*

Бастапқы шартпен берілетін цикл операторы **while** қайталану немесе цикл алдын ала тексерілетін шартқа байланысты орындалатын жағдайларда қолданылады. Жазылуы:

```
while(шарт)
  оператор;
```

мұндағы шарт – логикалық өрнек, мәні «ақиқат» болғанда, оператор қайталана береді, «жалған» болғанда – тоқтайды. Сондықтан да циклдің шарты дұрыс берілуі керек, әйтпесе берілген шартқа байланысты қайталануы тиіс оператордың тіпті бір ретте орындалмауы немесе тіпті де тоқтамай қайталана беруі мүмкін. Егер шартқа байланысты қайталанатын операторлар бір емес бірнешеу болса, онда оларды операторлық { } блокқа алып жазады, яғни:

```
while(шарт)
{
оператор_1;
оператор_2;
... ... ...
оператор_N;
}
```

2.14-жаттығу. Келесі программа $y=x^2/7$ функциясының [-2, 2] аралығындағы мәндерін 0.5 қадаммен есептеп шығарып береді:

```
#include "iostream.h"
#include "conio.h"
```

```
void main()
{ float i=-2,y; //i-дің бастапқы мәні 2-ге тең
//циклдің басталуы
  while (i<=2) // осы шарт ақиқат болса, цикл қайталана береді
{
    y= i*i/7; //функцияның мәнін есептеу
    cout<<"\n y("<<i; // аргументтің мәні i-ді баспаға шығару
    cout<<")= "<<y; // функцияның мәні y-ті баспаға шығару
    i=i+0.5 ; //қадамды өзгерту
  } // циклдің соңы
     getch();
  }
```

Программа орындалғаннан кейін шығатын нәтиже:

```
y(-2)= 0.571429
y(-1.5)= 0.321429
y(-1)= 0.142857
y(-0.5)= 0.0357143
y(0)= 0
y(0.5)= 0.0357143
y(1)= 0.142857
y(1.5)= 0.321429
y(2)= 0.571429
```

2.8.5. ***do while*** *цикл операторы*

Шарты кейін берілетін **do while** операторында алдымен оператор орындалады да, сонан соң барып шарт тексеріледі, яғни мұнда қайталанатын оператордың ең болмағанда бір рет болса да орындалуына мүмкіндік жасалады. Жазылуы:

```
do
{
     оператор_1;
     оператор_2;
```

```
        ... ... ...
        оператор_1;
    } while ( шарт);
```

мұнда да шарт – логикалық өрнек, мәні «ақиқат» болғанда, операторлар қайталана береді, «жалған» болғанда – тоқтайды.

2.15-жаттығу. Айталық, b>0 нақты саны берілсін. Ал мүшелері a_1, a_2, a_3, a_4 … болатын шексіз сандар тізбегі келесі заңдылық бойынша құрылады:

$$a_1 = b, \ \ldots \ , a_i = a_{i-1} - 1/\sqrt{i} \quad (i=2,3, \ldots)$$

Осы тізбектің ең алғашқы теріс, яғни $a_i<0$ мүшесін және оған дейінгі барлық мүшелерін табатын программа келесі түрде болады:

```
# include <iostream.h>
# include <conio.h>
#include<math.h> // sqrt функциясы үшін қосылады
void main()
{
int i=1; // i- тізбек мүшесінің реттік номері
 float a; //a- тізбек мүшесінің мәні
 float b=2.5; // b>0 нақты санының мәні
 a=b; //тізбектің бірінші a1= b мүшесінің мәнін анықтау
//циклдің басталуы
 do
 { cout<<"\n a("<<i<<")="<<a;  // i- тізбек мүшесін баспаға шығару
   i++; //келесі тізбек мүшесінің реттік номерін алу
   a=a-1/sqrt(i); // келесі тізбек мүшесінің мәнін есептеу
 }
 while (a>0); //қайталану шартын тексеру
//циклдің соңы
// тізбектің алғашқы теріс, яғни ai<0 мүшесін баспаға шығару
cout<<"\n Tizbektin algashki teris myshesi :";
 cout<<"\n a("<<i<<")="<<a;
      getch();    }
```

Бұл программаның орындалуының нәтижесі:

```
a(1)=2.5
a(2)=1.79289
a(3)=1.21554
a(4)=0.715543
a(5)=0.268329
Tizbektin algashki teris myshesi :
a(6)=-0.139919
```

Жоғарыдағы 2.15-жаттығуда шарты берілген есептер сияқты қайталанудың саны алдын ала белгісіз, бірақ ол белгілі бір шартқа ғана тәуелді болып келетін жағдайларда ең тиімдісі осы **while** немесе **do while** операторларының бірін қолдану болып табылады.

2.8.6. **for** *цикл операторы. Кері цикл. Іштестірілген циклдер*

Параметрлі цикл операторы **for** қандай да болмасын бір оператордың қайталану саны алдын ала белгілі болған жағдайда ғана қолданылады. Программадағы жазылуы:

```
for (i =I_бастапқы ; i <= I_соңғы ; i + +)
    оператор_1;
```

мұндағы i – цикл параметрі деп аталатын айнымалы, мәні бүтін, символдық және саналатын типтердің бірі бола алады, оператор_1 – бұл қайталанатын оператор. Қайталану циклдің шарты $i <= I_{соңғы}$ «ақиқат» болғанда, орындала береді де, ол «жалған» болғанда – тоқтайды. **for** параметрлі цикл операторының бұл $I_{бастапқы} <= I_{соңғы}$ болғандағы түрін тура орындалатын цикл деп қарастыруға болады.

2.16-жаттығу. Программа ‘A’ мен ‘Z’ аралығындағы барлық бас әріптерді экранға шығарып бере алады.

```
#include <iostream.h>
#include <conio.h>
```

```
void main()
{ for(char i='A';i<='Z'; i++)
   cout<<"\t "<<i;
   getch();
}
```

Программа орындалуының нәтижесі төмендегідей болады:

A	B	C	D	E	F	G	H	I	
J	K	L	M	N	O	P	Q	R	S
T	U	V	W	X	Y	Z			

Бұл программада, i- цикл параметрінің мәні символдық шама болып тұр, символдық шамалар реттеліп, номерленген болғандықтан, i++; операторы орындалғанда, келесі символдық шаманың реттік номері алынып отырады.

Параметрлі цикл операторы **for** жазылуындағы жақша ішіндегі үш өрнектің кез келгенін немесе үшеуін де жазбай кетуге болатын жағдайлар болады, айталық *бірінші жағдайда* жақша ішіндегі бірінші және екінші өрнек жоқ, яғни цикл параметрінің бастапқы мәні циклден тыс беріледі, ал цикл параметрінің келесі мәндері циклдің өз ішінде өзгеретін болады, мысалы:

```
int sum = 0; //қосындының бастапқы мәні
int i = 1; // i- дің бастапқы мәні берілді, енді ол жақшада көрсетілмейді
for (; i <= 100; ) {
sum = sum + i; //қосындыны есептеу
  i ++; // i- дің келесі мәнін алу, ол да жақшада берілмейді
}
```

Ал *екінші жағдайда* жақша ішіндегі үш өрнектің үшеуін де жазбай кеткенде қалай болатыны қарастырылсын, мысалы:

```
int sum = 0;
int i = 1; // i- дің бастапқы мәні берілді, ол енді көрсетілмейді
for (; ;) {
   if (i > 100) // шарт берілді, ол да көрсетілмейді
```

```
        break;
    sum = sum + i;
    i + +; // қадам берілді, ол да көрсетілмейді
}
```

мұндай үш өрнегі де берілмеген циклді «ашық цикл» немесе «бос цикл» (open loop) атайды.

Сондай-ақ, **for** параметрлі цикл операторының *керісінше* түрі де жиі қолданылады, оның жазылуы келесі түрде болады:

```
for (i =I_бастапқы ; i <= I_соңғы ; i--)
    оператор_1;
```

мұндағы қайталану, циклдің шарты i <= $I_{соңғы}$ «ақиқат» болғанда орындала береді және кері циклде $I_{бастапқы}$ >= $I_{соңғы}$ болатындығына назар аударылу керек, ал i- цикл параметрінің мәні i-- болғандықтан, біртіндеп кеміп отырады.

2.17-жаттығу. Келесі программа 0 мен 10 аралығындағы барлық бүтін сандарды экранға *кері* ретпен шығарып береді:

```
#include <iostream.h>
#include <conio.h>
void main()
{ for(int i=10;i>=0;i--)
   cout<<" "<<i;;
      getch();
}
```

Программа орындалуының нәтижесі төмендегідей болады:

```
10 9 8 7 6 5 4 3 2 1 0
```

Көптеген есептерді шешу іштестірілеген цикл операторларынсыз мүмкін болмайды. *Іштестірілген циклдер* деп бір цикл операторының екінші цикл операторының ішінде қолданылуын айтады. Мысалы, іштестірілген **for** цикл операторларын келесі түрде жазуға болады:

```
for (i =I_бастапқы ; i <= I_соңғы; i + +)
  for (j =J_бастапқы ; j <= J_соңғы; j + +)
     оператор_1;
```

мұндағы j – бойынша орындалатын *ішкі* цикл, і- бойынша *сыртқы* циклдегі і-дің әрбір мәні үшін толығымен $J_{соңғы}$ рет қайталанып отыратын болады, демек іштестірілген циклде орналасқан оператор_1; $I_{соңғы} * J_{соңғы}$ рет қайталанатын болады.

2.18-жаттығу. Бұл программа көбейту кестесін мектепке арналған торкөз дәптердің соңғы парағындағы секілді шығарып береді.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
void main()
{ int i,j; //цикл параметрлерін сипаттау
          // 2-ден 5-ке дейінгі сандардың көбейту кестесін
          // шығаратын іштестірілген циклдер
  for(i=1;i<=9;i++)
     { for(j=2;j<=5;j++ )
      cout<<"\t"<<j<<" x "<<i<<" = "<<(i*j);
     cout<<"\n";
     } ;
 cout<<"\n"; //
 // 6-дан 9-ға дейінгі сандардың көбейту кестесін шығаратын
 // іштестірілген циклдер
  for(i=1;i<=9;i++)
     { for(j=6;j<=9;j++ )
      cout<<"\t"<<j<<" x "<<i<<" = "<<(i*j);
     cout<<"\n";
     }
     getch();
}
```

Программа орындалуының нәтижесі төмендегідей болады:

```
2 x 1 = 2    3 x 1 = 3    4 x 1 = 4    5 x 1 = 5
2 x 2 = 4    3 x 2 = 6    4 x 2 = 8    5 x 2 = 10
2 x 3 = 6    3 x 3 = 9    4 x 3 = 12   5 x 3 = 15
2 x 4 = 8    3 x 4 = 12   4 x 4 = 16   5 x 4 = 20
2 x 5 = 10   3 x 5 = 15   4 x 5 = 20   5 x 5 = 25
2 x 6 = 12   3 x 6 = 18   4 x 6 = 24   5 x 6 = 30
2 x 7 = 14   3 x 7 = 21   4 x 7 = 28   5 x 7 = 35
2 x 8 = 16   3 x 8 = 24   4 x 8 = 32   5 x 8 = 40
2 x 9 = 18   3 x 9 = 27   4 x 9 = 36   5 x 9 = 45

6 x 1 = 6    7 x 1 = 7    8 x 1 = 8    9 x 1 = 9
6 x 2 = 12   7 x 2 = 14   8 x 2 = 16   9 x 2 = 18
6 x 3 = 18   7 x 3 = 21   8 x 3 = 24   9 x 3 = 27
6 x 4 = 24   7 x 4 = 28   8 x 4 = 32   9 x 4 = 36
6 x 5 = 30   7 x 5 = 35   8 x 5 = 40   9 x 5 = 45
6 x 6 = 36   7 x 6 = 42   8 x 6 = 48   9 x 6 = 54
6 x 7 = 42   7 x 7 = 49   8 x 7 = 56   9 x 7 = 63
6 x 8 = 48   7 x 8 = 56   8 x 8 = 64   9 x 8 = 72
6 x 9 = 54   7 x 9 = 63   8 x 9 = 72   9 x 9 = 81
```

2.8.7. ***break, continue*** *операторларын циклде қолдану*

Цикл ішінде қолданылатын **break** операторы циклдегі қайталануды толық орындамай, сол **break** операторы кездескен жерден жылдам аяқтап (досрочный выход из цикла) шығып кетуді қамтамасыз етеді. **break** операторын пайдалану программада циклді артық қайталамай, тиімді пайдалануға мүмкіндік береді, оны **for, while, do while** цикл операторларының бәрін де қолдануға болады. Әдетте, **break** операторын цикл ішінде қолданғанда, белгілі бір шарттар тексеріледі, осы шартқа байланысты **break** операторы орындалғанда, программаны басқару циклден шығарылып, циклден кейін келесі тұрған операторға берілетін болады. Циклді ұйымдастырудағы **break** операторын қолданудың тиімділігін келесі жаттығудан көруге болады.

2.19-жаттығу. Пернетақтадан енгізілген кез келген үш таңбалы бүтін санның 3-ке бөлінетін-бөлінбейтіндігін анықтайтын программа жұмысын қарастырады.

1-жағдай. **break** операторы қолданылады.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>  // abs() функциясы үшін қосылды
int main()
{int i,j,k, belgi=0; //i,j,k-цикл параметрлері,
                     //belgi- үш таңбалы санның цифрларының қосындысы
 int san,kol_k=0, kol_i=0, kol_j=0;
// san-пернетақтадан енгізілетін үш таңбалы сан
                 //  kol_k- k  бойынша циклдің қайталану саны
                 //  kol_j- j  бойынша циклдің қайталану саны
                 //  kol_i- i  бойынша циклдің қайталану саны
 cin>>san;
 for(i=1;i<=9;i++)   // үш таңбалы санның жүздігін алу
   { kol_i++;        // i  бойынша циклдің қайталану санын есептеу
     for(j=0;j<=9;j++ ) // үш таңбалы санның ондығын алу
     {kol_j++;       // j  бойынша циклдің қайталану санын есептеу
       for(k=0;k<=9;k++ )  //үш таңбалы санның бірлігін алу
         {kol_k++;  //- j  бойынша циклдің қайталану санын есептеу
          int tekS=i*100+j*10+k;  //кез келген үш таңбалы санды
                                  // цифрлары бойынша құру
     if (abs(san)==tekS) //құрылған санды енгізілген санмен салыстыру
           belgi = k+j+i;  //санның цифрларының қосындысын есептеу
           if (belgi!=0) // k бойынша циклден шығу шарты
             break;
         }
      if (belgi!=0) // j  бойынша циклден шығу шарты
        break;
       }
     if (belgi!=0) // i  бойынша циклден шығу шарты
       break;
    }
```

```
  if(belgi%3==0)  //цифрлар қосындысының 3-ке бөлінетіндігін тексеру
      cout<<" bolinedi"<<endl;
      else cout<<"bolinbeidi"<<endl;
     cout<<"kol_k="<<kol_k<<endl;
     cout<<"kol_j="<<kol_j<<endl;
     cout<<"kol_i="<<kol_i<<endl;
      getch();
}
```

Программаның нәтижесі келесі түрде болады:

```
 Ysh tanbali san engiz:  111
 bolinedi
kol_k=12
kol_j=2
kol_i=1
```

Бұл есептің алгоритмі «егер санның цифрларының қосындысы, үшке бөлінсе, оның өзі де үшке бөлінеді» деген қағидаға негізделеді, ендеше бірінші мәселе – пернетақтадан енгізілген санның цифрларын бөліп алу, мысалы, енгізілген сан «111» болса, онда оның цифрлары «1», «1», «1» болғаны. Бұл есепті шығарудың көптеген тиімді жолдары бар, мысалы, енгізілетін санды жолдық тип ретінде қарастырып, оның әрбір символын қиып алып, оны арнаулы функциялар көмегімен санға айналдыруға да болады. Бірақ бұл жолы іштестірілген циклдердің және **break** операторының жұмысын көрсету үшін басқаша шығару жолы, яғни мұнда үштаңбалы санды жүздік, ондық және бірлік разрядтарға жіктеп жазу пайдаланылды. Мысалы, барлық теріс емес бүтін үштаңбалы сандарды алуды келесі іштестірілген циклдерді пайдаланып шығарып алуға болады:

```
for(int i=1;i<=9;i++) //санның жүздігін алу
  for(int j=0;j<=9;j++) // санның ондығын алу
  for(int k=0;k<=9;k++) // санның бірлігін алу
   {
```

```
    int san=i*100+j*10+k; // санның өзін алу
     cout<<"  "<<san;
  }
```

2-жағдай. **break** операторын қолданбағанда циклдердің қайталану саны қандай болатынын көрсетеді.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
int main()
{int i,j,k, belgi=0;
 int san,kol_k=0, kol_i=0, kol_j=0;
  cout<<"Ysh tanbali san engiz: ";
  cin>>san;
  for(i=1;i<=9;i++)
    { kol_i++;
      for(j=0;j<=9;j++ )
      {kol_j++;
        for(k=0;k<=9;k++ )
          {kol_k++;
           int tekS=i*100+j*10+k;
           if (abs(san)==tekS)
             belgi = k+j+i;
          } // k  бойынша циклдің соңы
      } // j  бойынша циклдің соңы
   } // i  бойынша циклдің соңы
    if(belgi%3==0)
     cout<<" bolinedi"<<endl;
     else cout<<"bolinbeidi"<<endl;
    cout<<"kol_k="<<kol_k<<endl;
    cout<<"kol_j="<<kol_j<<endl;
    cout<<"kol_i="<<kol_i<<endl;
     getch();
}
```

Бұл программаның нәтижесі келесі болады:

```
Ysh tanbali san engiz: 111
bolinedi
kol_k=900
kol_j=90
kol_i=9
```

Бұл жаттығудың **break** операторы қолданылған *1-жағдайында* үш циклдің жалпы қайталану саны 12+2+1=15 болды, яғни қайталану «111» саны табылғанға дейін жүреді де, ол табылған соң басқару циклдерден шығып кетеді. Ал **break** операторы болмаған 2-*жағдайда* да есептің жауабы шығады, бірақ мұнда барлық үш таңбалы сандар қарастырылатындықтан, үш циклдің жалпы қайталану саны 900+90+9=999 болып тұр.

Келесі **continue** операторы цикл ішіндегі кейбір қадамдағы қайталануды (итерацияны) орындатпай тоқтатып, басқаруды келесі қадамға өткізіп жіберіп отырады.

2.20-жаттығу. 0-ден 50-ге дейінгі аралықтағы тек қана 7-ге бөлінетін сандарды шығарып беретін программаны келесі түрде жазуға болады:

```
#include <iostream.h>
#include <conio.h>
int main()
{
int i=0; // i циклдің қадамы
  while (i<=50) //циклдің орындалу шарты
    {
   i++; // циклдің келесі қадамын алу
   if (i%7!=0) // i-дің 7-ге бөлінбейтіндігін тексеру
  continue; // басқаруды келесі i++; қадамға өткізіп жібереді
   cout<<"\n"<<i;
   cout<<":7";
   cout<<"="<<i/7;
```

```
    }
 getch();
}
```

Программа орындалуының нәтижесі:

```
7:7=1
14:7=2
21:7=3
28:7=4
35:7=5
42:7=6
49:7=7
```

мұндағы **if** (i%7!=0) шарты орындалғанда, яғни кезектегі қадамдағы i саны 3-ке бөлінбейтін болса, онда continue; операторы орындалады да, ол өзінен кейінгі тұрған үш оператордың қайталануына жол бермей, басқаруды келесі i++; қадамға өткізіп жібереді.

Бақылау сұрақтары

1. C++ программалау тілінде жазылған программада міндетті түрде болуы тиіс функция қалай аталады?
2. Программада директивалар қалай жазылады және олардың қызметі не?
3. Идентификатор деген не? Идентификаторға қандай шектеу қойылады?
4. Программада тұрақтылар қандай қызмет атқарады және оның қандай түрлері бар?
5. Айнымалыларды жариялауда жады кластары қандай қызмет атқарады?
6. Деректердің қарапайым типтерін атаңыз.
7. Биттік амалдар қандай сандарға қолданылады?
8. **sizeof** амалы не үшін қолданылады?

9. Ағынға қосу (<<) және ағыннан алу (>>) амалдары қандай объектілермен бірге қолданылады?

10. Шешім қабылдау операторларының қандай түрлері бар?

11. Цикл операторларының түрлерін атаңыз.

12. **break** және **continue** операторлары қандай қызметтер атқарады?

Тапсырмалар

1. Екі нақты оң сан берілген. Осы екі санның арифметикалық ортасын, қосындысын, айырмасын және көбейтінділерін есептейтін программа жазыңыз.

2. Бастапқы жылдамдығы v_0 физикалық дене a тұрақты үдеуімен қозғалып келеді. Осы дененің t уақыт ішінде жүрген жолын есептейтін программа жазыңыз.

3. Массасы m, жылу сыйымдылығы c сұйықты t_1 температурасынан t_2 температурасына дейін қыздыру үшін қанша жылу керек.

4. Уақытша сақтауға үшінші R айнымалысын пайдаланып, A және B айнымалыларының мәндерін алмастырыңыз.

5. Пернетақтадан енгізілген екі бүтін санды бөлгендегі қалдықты есептеп, нәтижесін экранға шығаратын программа жазыңыз.

6. Келесі программа фрагментін орындаңыз және шыққан нәтижені талдап қорытынды жасаңыз:

```
{ . . .
double c;
   int a=13;
   int b=2;
    cout<<"a="<<a<<endl;
    cout<<"b="<<b<<endl;
     c= a/b;
        cout<<"a/b="<<c<<endl;
     c= (float)a/b;
        cout<<"(float)a/b="<<c<<endl;
     c= float (a/b);
        cout<<"float (a/b)="<<c<<endl;
     c= a/(float)b;
```

```
            cout<<"a/(float)b="<<c<<endl;
        getch();
    . . .
}
```

7. $M(x, y)$ нүктесі орналасқан ширек нөмірін анықтап баспаға шығарыңыз.

8. Келесі функцияның мәнін есептейтін программа жазыңыз:

$$y = \begin{cases} x^2, & \textit{егер } 0 < x < 2; \\ x + 4, & \textit{егер } -2 < x \le 0; \\ 0, & \textit{қалған жағдайда.} \end{cases}$$

9. $M(x, y)$ нүктесінің радиусы r, центрінің координатасы a, b болатын дөңгелекке тиісті болатынын немесе болмайтынын анықтыңыз.

10. Апта күндерінің нөмірін енгізіп, соған сәйкес қазақ және ағылшын тілдерінде апта күндерін шығарыңыз.

11. Уақытты енгізіңіз (тек сағаттарды). Осы уақытқа сәйкес келесі хабарламаны: «Қайырлы таң», «Қайырлы күн», «Қайырлы кеш», «Қайырлы түн» шығаратын программа жазыңыз.

12. Отырғызу орны санын енгізіңіз. Осы санға сәйкес көлік түрлерін шығарыңыз: «велосипед», «мотоцикл», «жеңіл автомобиль», «микроавтобус», «автобус» (Басқа нұсқалардың болуы да мүмкін)

13. 0,5 қадамымен 2-ден 10-ға дейінгі аралықта өзгеретін x үшін $y = 0{,}4x^2 - \dfrac{1}{x}$ функциясының мәнін есептеңіз.

14. Пернетақтадан енгізілген n үшін Y=1!+2!+3!+...n! қатарының қосындысын есептеңіз.

15. Нақты x саны берілген. Шексіз тізбектің берілген ε-нан аспайтын қосылғыштарының қосындысының жуық мәнін есептендер:

а) $x + \dfrac{x^2}{2} + \dfrac{x^3}{3} + ...(|x| < 1)$ b) $x - \dfrac{x^2}{2} + \dfrac{x^3}{3} - ....(|x| < 1)$

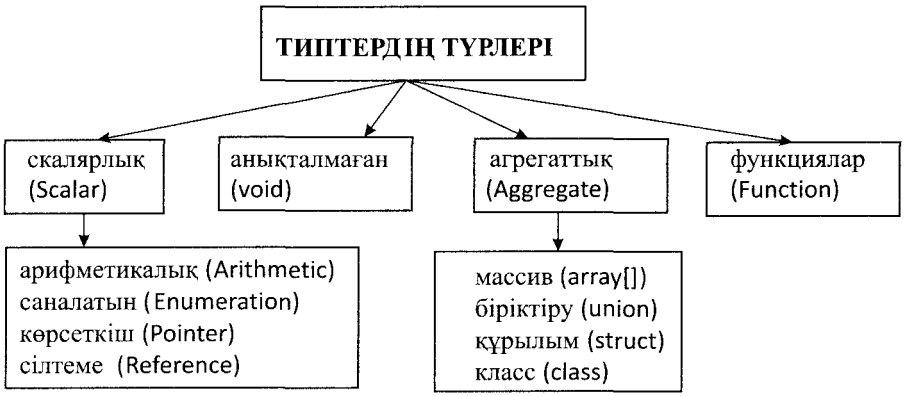
c) $x + \frac{x^3}{3} + \frac{x^5}{5} + ...(|x| < 1).$

16. Пернетақтадан енгізілген n үшін $\frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \frac{4}{5} + \frac{5}{6} + \frac{6}{7} + ...$ қатарының алғашқы n қосылғышының қосындысын есептеңіз.

3-тарау. C++ ТІЛІНДЕГІ ДЕРЕКТЕРДІҢ ТИПТЕРІ

3.1. Деректер типтерінің жалпы классификациясы

Программаға қатысатын барлық шамалардың қандай да болмасын әйтеуір бір типпен анықталатындығы белгілі болды. Программадағы қолданылатын деректердің табиғатына, атқаратын қызметтеріне, жадыдан алатын орындарына және т.б. байланысты C++ программалау тілінде деректер типтерін классификациялаудың бірнеше жолдары бар. Жалпы деректердің барлық типтерін атқаратын қызметтеріне қарай үлкен төрт топқа бөлуге болады (3.1-сурет).



3.1-сурет. *C++ программалау тіліндегі деректер типтерінің классификациясы*

Программаға қатысатын деректер типтерін жасалу жолына байланысты *негізгі типтер* және *туынды типтер* деп екіге бөледі. Деректердің негізгі типтеріне компилятор үшін алдын ала анықталып қабылданған **int, float, char, double, bool** және т.б. типтер жатады (негізгі типтерге қолданылатын **signed** және **unsigned** модификаторлары және типтерді келтіру туралы 2.4.1-пунктте қарастырылған).

Қолданушы негізгі типтерді пайдалана отырып, өз қажетіне қарай туынды типтерді құрастырып ала алады. Туынды типтерді келесі түрлерге бөледі:

– псевдоним типтер (**typedef**)

– саналатын типтер (**enum**)
– жиындар (**Set**)
– көрсеткіштер мен сілтемелер
– массивтер
– құрылым (**struct**) мен біріктіру (**union**)
– жолдық типтер (символдар массиві және **string, AnsiString**)
– файлдар
– класс (**class**)

3.2. Псевдоним типтер (typedef)

C++ -те жазылған программада негізгі (**int, float, char, double, bool** және т.б.) немесе өзіне дейін анықталған (массив, құрылым, саналатын және т.б.) белгілі типтер негізінде жаңадан псевдоним типті алу үшін **typedef** қызметші сөзін қолданады, жазылуы:

typedef белгілі_тип жаңа_псевдоним_ тип;

Мысалы,

```
typedef unsigned long ganaType;
ganaType x1,x2;
```

Программада **typedef** қызметші сөзін **signed және unsigned** модификаторлары арқылы берілетін атауы ұзын типтерді қысқартып жазуда, сондай-ақ C++-тегі типтерді мағынасына қарай Паскальдағы типтермен үйлестіріп қолдану үшін де жиі қолданады, мысалы, C++ -тегі **float** типін Паскальдағы сияқты **real** деп пайдалану үшін C++- те «real» деген псевдоним типті құрып пайдалануға болады:

```
typedef float real;
real a, b, c;
```

3.1-жаттығу. Программа **bool** негізгі типінің орнына *myType* деп аталатын псевдоним типті қолдануды көрсетеді.

```
#include <iostream.h>
#include <conio.h>
void main()
{ typedef bool myType; // typedef көмегімен bool типінің негізінде
                       // жаңа псевдоним тип myType құру
myType flag; // flag айнымалысы myType псевдоним типтің өкілі ретінде
             //анықталып тұр
int a,b;
cin>>a>>b;
if (a>b)
flag=true; // myType псевдоним типке жататын flag айнымалының мәні
else
  flag=false; // myType типке жататын flag айнымалының мәні
  cout<<"\n flag="<<flag;
  getch();
}
```

Программа орындалуының нәтижесі :

1-жағдай	*2-жағдай*
5 3 flag=1	7 9 flag=0

3.3. Саналатын типтер (enum)

Саналатын типтерді әдетте апта күндері, жылдағы айлар, белгілі бір тауарлардың тізімі және т.б. секілді тізбектей номерлеп санауға болатын деректермен жұмыс жасауда қолданады. Саналатын типті қолданушы өзі құрастырады. Оның мәні тек қолданушы анықтаған мәндерді ғана қабылдайды. Саналатын типтің өзі жеке-жеке идентификаторлар тізімінен тұрады, ал C++ компиляторы әрбір идентификаторға сәйкес бүтін санды мәндер тағайындайды немесе идентификаторларды номерлеп санап шығады, әдетте ол нольден басталады, оны өзгертуге де болады. Мәндердің қайталануы мүмкін, бірақ идентификаторлар қайталанбауы тиіс, жазылуы:

```
enum санал_Тип_аты {идентиф_0, идентиф_1, ... , идентиф_N};
```

мұнда идентиф_0-ге сәйкес мән «0», идентиф_1-ге сәйкес мән «1» және т.б. кете береді. Мысалы,

```
enum aptaKyni {dyisenbi,sisenbi,sarsenbi,bisenbi,
                guma,senbi,geksenbi};
```

мұнда әрбір идентификаторға 0-6 аралығындағы сандар сәйкестендіріледі, мысалы, guma-ға «4» сәйкес келіп тұр, егер қолданушы мұны өзгерткісі келсе, келесі саналатын типтегі идентификаторлардың мәнін өзі көрсетіп, келесі түрде жариялайды:

```
enum aptaKyni {dyisenbi=1,sisenbi=2,sarsenbi=3,bisenbi=4,
                guma=5,senbi=6,geksenbi=7};
```

әрі қарай осы типке жататын айнымалыны келесі түрде сипаттауға болады:

```
aptaKyni bygin;  немесе aptaKyni bygin  = sarsenbi;
```

3.2-жаттығу. Келесі программа арифметикалық калькулятордағы кейбір математикалық қателерді, мысалы, «санды нольге бөлуге болмайды», «қате амалдың» енгізілуі сықты қателерді тексеруді көрсетеді.

```
#include<iostream.h>
#include<conio.h>
int main()
{
//математикалық қателерді саналатын тип түрінде анықтау
enum mathKateler {kateGok,amalKate,nulgeBolu};

 char amal;  // мәні +, – , * , /  символдарының бірі болатын амал
 double x,y,z; //операндалар

mathKateler Kate=kateGok; //саналатын типке жататын
                          //айнымалыны және оның бастапқы мәнін жариялау
```

```
 cout<<"\n x -ti,  amaldi , y-ti  engizu :  \n\n";
 cin>>x>>amal>>y;

 switch (amal)
 {
case '+':
      z=x+y; break;
 case '-':
      z=x-y; break;
case '*':
    z=x*y; break;
case '/':
  if (y!=0) //нольге бөлуге болмайтынын тексеру
     z=x/y;
 else
    Kate=nulgeBolu; break;       // Kate айнымалысының
                                 // мәні өзгерді
default:
     Kate=amalKate; break; // Kate айнымалысының мәні өзгерді
 }

 //Нәтижені шығару
 if (Kate==kateGok) // Kate-нің  мәнін тексеру
   cout<<x<<" "<<amal<<" "<<y<<" "<<" = "<<z;
 else
   switch (Kate)  // Kate-нің  мәнін тексеру
     {
        case amalKate:
           cout<<"\n \n amal duris emes"; break;
        case nulgeBolu:
           cout<<"\n \n sandi nolge boluge bolmaidi "; break;
     }

getch();
}
```

Программа орындалуының әр түрлі жағдайлардағы нәтижелері:
1-жағдай. x-тің, amaldi-ң және y-тің мәндері дұрыс берілді:

```
x-sanin, amaldi , y-sanin engizu
:
78*46
78 * 46 = 3588
```

2-жағдай. x пен y-тің мәндері дұрыс, ал amaldi-ң мәні дұрыс емес берілді:

```
x-sanin, amaldi , y-sanin engizu :
789 ? 45
amal duris emes
```

3-жағдай. x пен amaldi-ң мәндері дұрыс, ал y-тің мәніне «ноль» берілді:

```
x-sanin, amaldi, y-sanin engizu :
12456 / 0
sandi nolge boluge bolmaidi
```

Бұл программада саналатын тип түрінде анықталған mathKateler типі қолданылды, оның сәйкесінше номерленген үш мәні: «kateGok», «amalKate», «nulgeBolu» анықталған, сондықтан типі mathKateler болатын Kate айнымалысының қабылдайтын мәні, осы үш мәннің біреуі болуы мүмкін, оның бұл мәндерден басқа мәндері болуы мүмкін емес.

3.4. Жиындар (Set)

Программалау тілдерінде жиынды, белгілі бір типке жататын және логикалық мағынасына қарай біріктірілген мәндердің, элементтердің

немесе объектілердің жиынтығы ретінде қолданушы анықтайтын туынды тип ретінде қарастырады. Бірде-бір элементі жоқ жиын *бос жиын* деп аталады. Жиындағы элементтердің саны шектеулі болады, ол 0-ден 255-ті қоса алғанға дейін өзгереді.

C++Builder-дегі программада жиынды пайдалану үшін программаға осы жиын анықталған vcl/ sysdefs.h тақырыптық файлы қосылуы керек. Жиынның программадағы жариялануы:

Set <*тип, төменгі_мән, жоғарғы_мән*> *айнымалы*;

мұндағы *тип* реттеліп, номерленген типтердің бірі, яғни **int, char** немесе саналатын типтердің біреуі ғана бола алады, *төменгі_мән, жоғарғы_мән* – бұл сәйкесінше жиындағы ең кіші және ең үлкен мәнді көрсетеді, *айнымалы* – ол осы жиын типіне жататын айнымалыны жариялау немесе оның идентификаторы болып табылады. Әдетте, Set жиындық типке жататын *айнымалыны* жариялағанда оның мәндері анықталмайды, сондықтан жиын типіне жататын айнымалының мәндерін анықтайды немесе инициализациялайды, ол үшін жиындық типке қолданылатын арнаулы << – жиынға жаңа элементті немесе мәнді қосу амалы қолданылады.

Мысалы, қабылдайтын мәндері символ түріндегі цифрлардан тұратын жиындық типке жататын zifr айнымалыны жариялау мен оның мәндерін тағайындау (инициализация) келесі түрде жазылады:

```
Set <char, '0','9'> zifr; //жиын типіне жататын zifr айнымалыны жариялау
 // zifr айнымалысына мәндер тағайындау немесе жаңа мәндер қосу
zifr<<'0'<<'1'<<'2'<<'3'<<'4' <<'5'<<'6'<<'7'<<'8'<<'9';
```

Егер жиын элементтерінің типі саналатын тип болса, онда ол *кең ауқымды* тип ретінде жиынды жариялағанға дейін анықталған болуы керек, мысалы:

```
    enum tys {ak, kizil, sari,gasil, kok}; // саналатын типті жариялау
void main()
{
```

```
    Set <tys, kizil, kok> boiau; // жиын типті айнымалыны жариялау
    boiau<<kizil<<sari<<gasil<<kok;
    ... ...  ...
}
```

Жиындық типтерге қолданылатын негізгі амалдарға келесі амалдар жатады:

– қиылысу амалы (*), яғни екі жиында да бар болатын ортақ элементтерді табу, егер болмаса бос жиын мәнді беру;

– біріктіру амалы (+), яғни жиындардың элементтерін біріктіру;

– айырма амалы (-), яғни азайғыш жиынның азайтқыш жиынға кірмейтін элементтерін беру;

– эквиваленттілікті анықтау (= =), егер А және В жиындары эквивалент болса, онда(А== В) логикалық өрнек **true** мән береді;

– эквивалент еместікті (!=) анықтау, егер А және В екі жиын эквивалент болмаса, онда А!= В логикалық өрнек **true** мән береді;

– жиынға элемент қосу амалы (<<) жиынға жаңа элементті тіркеу немесе жалғау үшін қолданылады;

– жиыннан элементті шығару амалы (>>) жиыннан элементті шығару немесе алып тастау үшін қолданылады және бұл элемент жиында бар болуы керек.

Сондай-ақ жиындық типтер үшін төмендегідей әдістер қолданылады:

Clrear(); – жиындық типті айнымалыны тазалайды, жазылуы:

```
айнымалы. Clrear();
```

Contains(Мән); – бұл берілген Мәннің жиындық типті айнымалының мәніне тиістілігін анықтау үшін қолданылады, егер ол тиісті болса, бұл әдіс **true** мән қайтарады, жазылуы:

```
айнымалы. Contains(Мән);
```

бұл әдісті қолданғанда тексерілетін Мәннің типі жиынды құрайтын негізгі типпен сәйкес болуы керек.

Жалпы жиындық типтерді қолдану көбінесе программада пайдаланылатын деректерге шектеу қоюда қолданылады.

3.3-жаттығу. Программа пернетақтадан енгізілетін символдарға шектеу қояды, яғни енгізілетін символ тек қана цифр болуы керек:

```
#include <conio.h>
#include <iostream.h>
#include <sysdefs.h> // Set үшін қосылды немесе
                     //#include <vcl.h> қосылады
int main()
{
Set <char, '0','9'> zifr; //жиын типіне жататын zifr айнымалыны
                     // жариялау
 zifr<<'0'<<'1'<<'2'<<'3'<<'4'
    <<'5'<<'6'<<'7'<<'8'<<'9'; // zifr айнымалысынының мәндерін
                              // тағайындау немесе жаңа мәндер қосу
char EngSimbol; // перентақтадан енгізілетін символды жариялау
 do
 { Beep(); //стандарт дыбыс шығаратын функция
  cout<<"\n\n kez kelgen zifr-simbol engiz:";
  cin>>EngSimbol;
if(!zifr.Contains(EngSimbol)) //енгізілген символдың zifr жиынға тиісті
                              //емес екенін тексеру
cout<<"\n\n Bul zifr emes !";
 }
 while (!zifr.Contains(EngSimbol)) ; //енгізілген символдың zifr жиынға
                                     // тиісті емес екенін тексеру
 cout<<"\n\n Bul zifr ="<<EngSimbol;
 getch();
 }
```

Программа орындалуының нәтижесі:

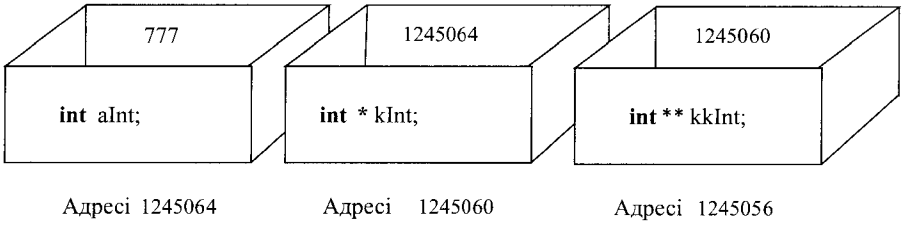
```
kez-kelgen zifr-simbol engiz: f
Bul  zifr  emes !

kez-kelgen zifr-simbol engiz: П
Bul  zifr  emes !

kez-kelgen zifr-simbol engiz: G
Bul zifr  emes !

kez-kelgen zifr-simbol engiz: 7
Bul zifr =7
```

мұнда қолданушы цифр болатын символ енгізгенше **do while** циклі қайталанып тұра береді, ал цикл ішіндегі **if** шарты «Bul zifr **emes** !» деген хабарламаны шығару үшін ғана пайдаланылып тұр.

3.5. Көрсеткіштер

Программалауда қолданылатын кез келген дерек, мейлі ол программаның коды немесе жарияланған айнымалы немесе тұрақты, немесе т.б. болсын, бәрібір ол компьютер жадысында белгілі бір адрес бойынша ұзындығы байтпен өлшенетін орын алатыны белгілі. Егер компьютер жадысы шартты түрде көптеген кішкентай ұяшықтардан немесе сол ұяшықтардан тұратын кішкентай жәшіктерден тұрады деп есептелсе, онда осы *ұяшықтың адресі* оның жадыдағы тұрған орнын білдіретін болады (3.2-сурет). Программада қандай да болмасын белгілі бір типке жататын айнымалы немесе т.б. жарияланған кезде, оның типіне сәйкес байт-ұзындықтағы орынды немесе жәшікті жадыдан алады, мысалы, **int** alnt; түрінде жарияланған бүтін типті alnt айнымалысы үшін қатар орналасқан адрестері 1245064, 1245063, 1245062, 1245061 болатын ұяшықтар алынады, егер ол айнымалының мәні берілсе, онда сол мән жаңағы ұяшықтардан құралған жәшікке барып түсті деп есептеуге болады, ал айнымалының идентификаторы (alnt) сол ұяшықтардан құралған жәшіктің сыртындағы *этикеткасы* секілді қызмет атқарады.

Демек, қолданушы программамен жұмыс жасағанда айнымалының нағыз адресімен емес, тек оның этикеткасымен ғана жұмыс жасайтын болып тұр, себебі бұл адрестермен жұмыс жасағаннан әлдеқайда жеңіл, яғни бұл жерде адрестермен жасалатын жұмысты компилятор өзі атқарып тұр, сондықтан бұл қолданушыға байқалмайды.



3.2-сурет. *Жады ұяшықтарының адрестері мен көрсеткіштердің байланысы*

С++ программалау тілінде айнымалылардың, объектілердің немесе т.б. деректердің жадыдағы орындарының адрестерімен тікелей жұмыс жасау үшін *көрсеткіш* деп аталатын арнаулы айнымалылар қолданылады. *Көрсеткіш* өз алдына жеке тип болып есептелмейді, ол тек типі анықталып, жарияланған айнымалының жадыдан алған орнын немесе адресін көрсету үшін ғана қолданылатын ерекше айнымалы болып табылады. Көрсеткіштің қабылдайтын мәні бұл белгілі бір айнымалының мәні сақталатын жәшіктің адресі болып тұр, сондықтан программада көрсеткішті жарияламас бұрын алдымен қажет айнымалыны жариялап алады, сонан соң барып сол айнымалының адресін сақтайтын көрсеткішті жариялайды, мұнда айнымалы мен оның көрсеткішінің типі сәйкес болуы керек. Көрсеткіштің программадағы жариялануының бірнеше түрі бар:

тип ***көрсеткіш**;

немесе

тип ***көрсеткіш**= *көрсеткіш_айнымалы;*

немесе

тип *айнымалы=мәні;*

тип *көрсеткіш=& *айнымалы;*

мұндағы **тип*** жарияланатын айнымалының берілген **тип**-ке жататын көрсеткіш екенін білдіреді, ал &-«*адресті алу амалы*» деп аталады, яғни ол өзінен кейін тұрған *айнымалы*-ның адресін алып беруді қамтамасыз етеді.

Бір типке жататын бірнеше көрсеткіштерді жариялағанда, «*» белгісі олардың әрқайсысының алдына қойылады.

Мысалы,

```
int aInt=777;       //aInt  айнымалыны жариялау
int * kInt=&aInt;   //aInt  айнымалының адресі сақталатын
                          // kInt  көрсеткішті жариялау
Int ** kkInt=&kInt; // kInt  көрсеткіштің адресі сақталатын
                           // kkInt  көрсеткішті жариялау
Int *** kkkInt=&kkInt;  //kkInt  көрсеткіштің адресі сақталатын
                                 // kkkInt  көрсеткішті жариялау
      char *kChar_a, *kChar_b, *kChar_c;  //бірнеше көрсеткішті
                                    // қатар жариялау
```

мысалы, мұндағы **int** aInt; түрінде жарияланған бүтін типті aInt айнымалысы үшін жадыдан бөлінген орын немесе жәшіктің адресі 1245064 болсын, сонда kInt көрсеткішке арналған жәшікте осы айнымалының адресі сақталатын болады (3.2-сурет), ал «көрсеткішке көрсеткіш» немесе «**» түрінде жарияланған kkInt жәшіктің ішінде kInt көрсеткіш адресі орналасады.

Көрсеткіштерге қолданылатын амалдар:

– адрес бойынша мәнді алу амалы (*) көрсеткіште адресі сақталып тұрған жәшіктің ішіндегі мәнді алу үшін қолданылатын унарлық амал, оның нәтижесі сол жәшіктегі **мән** болып шығады, мысалы, 3.2-суретке сәйкес, **int** natige= ***kInt**; орындалғанда, **int** natige-нің мәні «777» болып шығады. Сол сияқты **int** natige= ****kkInt**; орындалғанда да natige-нің мәні «777» болады;

– меншіктеу амалы (=), мысалы, **char** *kChar = 'ю'; Көрсеткішке нольді де меншіктеуге болады, мысалы:

```
int *k_nol=NULL;   немесе    int *k_nol=0;
```

– типтерді келтіру амалы (**void** типіне қолданылмайды), мысалы:

```
unsigned long L=0x12345678L;
char *k_Char=(char *)&L; // long типін char* типке келтіру
int *k_Int=(int *)&L; // long типін int * типке келтіру
char  *k_adr=(char *) 0xB8000000; // көрсеткішке жадыдағы
          //жәшіктің адресін анық түрде меншіктеу,
           //мұнда он алтылық түрдегі 0xB8000000-
           //тұрақты char типке келтіріледі
```

– инкремент (++) және декремент (--) амалдары, **int** *k_Int ++; немесе **int** *k_Int -- ; түрінде жазуға болады;

– жадыдағы жәшіктің адресін алу (&) амалы, программада жарияланған атауы бар, уақытша жадыда орналасқан шамалар үшін ғана қолданылатын унарлық амал, мысалы,

```
int a=45; // бүтін типті a айнымалыны жариялау
int *k_a=&a; // жадыдағы a айнымалысы орналасқан жәшіктің
                // адресін *k_a көрсеткішке алып беру.
```

Бұл амалды скаляр өрнектер үшін атауы жоқ тұрақтылар үшін қолдануға болмайды.

Көрсеткіштерді жариялауда оның мәнінің анықталып кетуіне назар аудару керек, көрсеткіш, жады ұяшықтарының адрестерімен жұмыс жасайтын болғандықтан, оның кездейсоқ адрестермен жұмыс жасауы қателерге соқтыруы мүмкін.

3.4-жаттығу. Программа деректер орналасқан жады жәшіктерінің адрестерін алу, жәшіктегі мәндерді алуда көрсеткіштерді пайдалануды көрсетеді (3.2-суретке сәйкес жасалады).

```
#include "iostream.h"
 #include "conio.h"
int main()
{
 int aInt=777; //бүтін типті aInt айнымалыны жариялау, мәнін беру
 int* kInt=&aInt;   //* kInt көрсеткішті жариялау, мәніне aInt
                    //айнымалының адресін алып беру
```

```
  int** kkInt=&kInt;//көрсеткішке көрсеткіш ** kkInt жариялау
                    //мәніне  * kInt көрсеткіштің адресін алып беру
 int*** kkkInt=&kkInt; //көрсеткішке көрсеткішке көрсеткіш  *** kkkInt
                    //жариялау мәніне ** kInt көрсеткіштің адресін алып беру
  cout<<"\n\n aInt-tin adresin alu="<<&aInt;
  cout<<"\n *kInt-tegi adres boinsha mandi alu ="<<*kInt;
  cout<<"\n *kInt-tin ozinin adresi="<<&kInt;
  cout<<"\n **kkInt-tegi adres boinsha mandi alu ="<<**kkInt;
  cout<<"\n **kkInt-tin ozinin adresi="<<&kkInt;
  cout<<"\n ***kkkInt-tegi adres boinsha mandi alu="<<***kkkInt;
  cout<<"\n ***kkkInt-tin ozinin adresi=="<<&kkkInt;
 //Көрсеткішті char типті айнымалыға қолдану
  char aChar='a';
  char bChar='b';
  char *k_aChar=&aChar;
  char *k_bChar=&bChar;
  cout<<"\n\n aChar adresi="<<int (&aChar);
  cout<<"\n *k_aChar adresi="<<&k_aChar;
  cout<<"\n bChar adresi="<<int (&bChar);
  cout<<"\n *k_bChar adresi="<<&k_bChar;
  getch();
 }
```

Программа орындалуының нәтижесі:

```
 aInt-tin adresin alu=1245064
*kInt-tegi adres boinsha mandi alu =777
*kInt-tin ozinin adresi=1245060
**kkInt-tegi adres boinsha mandi alu =777
**kkInt-tin ozinin adresi=1245056
***kkkInt-tegi adres boinsha mandi alu=777
***kkkInt-tin ozinin adresi==1245052

aChar adresi=1245051
*k_aChar adresi=1245044
bChar adresi=1245050
*k_bChar adresi=1245040
```

3.6. Сілтемелер

Сілтемелер немесе оларды сілтеуіш айнымалылар деп те атайды, негізінен көрсеткіштердің бір түрі болып табылады. Жазылуы:

тип & *сілтеме* (айнымалы);
тип & *сілтеме* = айнымалы;

мұндағы & (амперсанд) таңбасы бұл жерде сілтеу амалының қызметін атқарады, *сілтеме* – бұл дұрыс идентификатор. Мысалы:

```
int x=10; //x айнымалыны жариялау
int & cx=x;  // x айнымалысына сілтеме cx жариялау
```

Сілтеу амалының қызметін атқаратын & (амперсанд) таңбасы, көрсеткіштерде жады ұяшығының адресін алу амалын да білдіреді, бірақ онда & таңбасы айнымалының алдына қойылады. Мысалы:

```
int x=10;              //x айнымалыны жариялау
int & cx=x;            // x айнымалысына сілтеме cx жариялау
  int *korsX= &x;  // korsX көрсеткішке x айнымалысының адресін беру
```

Әдетте, көрсеткішті пайдаланғанда, онымен айнымалының өзі сияқты жұмыс жасауға шектеулер қойылады, ал сілтемені жай айнымалыларға қолданса, онда ол сол айнымалының псевдонимі сияқты жұмыс жасайды, яғни сілтememен айнымалының өзімен жұмыс жасаған секілді жұмыс жасай беруге болады.

3.5-жаттығу. Келесі программа сілтememен немесе сілтеуіш айнымалымен жұмыс жасауды көрсетеді:

```
#include <iostream.h>
 #include <conio.h>
int main()
{ int x=10; //x айнымалыны жариялау
int & cx=x; // x-қа сілтейтін cx сілтеуіш айнымалыны жариялау
cout<<"\n x="<<x; // x айнымалынының мәнін көру
```

```
cout<<"\n cx="<<cx; // cx сілтеуіш айнымалының мәнін көру
x=x*2; //x айнымалының мәнін өзгерту
cout<<"\n\n x=x*2; tan keingi cx="<<cx; //x айнымалының мәні
                                    //өзгергеннен кейінгі cx сілтеменің мәнін көру
cx=cx+7; //cx сілтемені өзгерту
cout<<"\n\n cx=cx+7;tan keingi cx="<<cx; //cx сілтеме өзгергеннен
                                    //кейінгі оның мәнін көру
 cout<<"\n x= "<<x; //cx сілтеме өзгергеннен кейінгі x-тің мәнін көру
getch();
 }
```

Программа орындалғанда шығатын нәтижелер келесі түрде болады:

```
 x=10

cx=10

x=x*2;   tan keingi     cx=20

 cx=cx+7;   tan keingi   cx=27

x=27
```

Программа нәтижесі көрсеткендей, егер **x**- айнымалы өзгерсе, онда ол өзгеріс оның **cx**-сілтеуіш айнымалысына да беріледі немесе керісінше **cx**-сілтеме өзгерсе, ол өзгеріс бастапқы **x**-айнымалыға берілетін болады.

Сілтеуіш айнымалылардың бұл мүмкіндіктері күрделі программалауда, күрделі кластарды құруда көптеген программалық трюктарды жүзеге асыруға мүмкіндіктер береді. Сілтемені көрсеткішке қолданатын болса, онда ол көрсеткішпен объект сияқты жұмыс жасауға мүмкіндік береді.

Сондай-ақ, программалауда сілтеуіштерді функция параметрлерін беруде жиі қолданады. Жады ұяшығында сақталған мәндерді өзгерту үшін ұяшыққа адрес арқылы қатынау міндетті болып саналады, сондықтан мұның бірден-бір тиімді жолы ретінде сілтеуіштерді пайдаланады (4.3.2-тақырыпты қараңыз).

3.7. Массивтер

Программалауда бір типке жататын бірнеше элементтерден тұратын объектілерді де пайдалануға болады. Массив осындай бір типке жататын элементтерден құралатын құрылымдық тип болып табылады, мысалы, бірыңғай **int** типті сандардан тұратын *бүтін сандар массиві* немесе бірыңғай **char** типті символдардан тұратын *символдық массив* деген сияқты.

Массивке тән негізгі үш қасиет мыналар:

– элементтері бір типке жатады;

– элементтері реттелген, яғни номерленген;

– массивтің элементі екі нәрсемен анықталады индексі (реттік номері) және мәні.

Массивтерді өлшеміне қарай, егер ол бір ғана жолдан (немесе бағаннан) тұрса, *бір өлшемді массив*, ал жолдар мен бағандардан тұрса, *екі өлшемді массив (матрица)* деп те атай береді. Жалпы массивтермен жасалатын әрекеттерді келесі топтарға бөледі:

– массивті жариялау;

– массивті инициализациялау (элементтерінің мәндерін тағайындау);

– массивті өңдеу немесе сұрыптау;

– массив-нәтижені баспаға шығару.

Программада массивтердің өлшемдері тұрақтылар түрінде алдын ала жарияланса (немесе а[10] деген сияқты көрсетіліп берілсе), онда оларды өлшемдері өзгермейтін *статикалық массивтер* деп қарастырады. Ал егер массивтің өлшемін алдын ала білу мүмкін болмайтын жағдайларда *динамикалық массивтер* деп аталатын, өлшемдері программаның орындалуы кезінде ғана анықталатын массивтер қолданылады (5.1-тақырыпты қараңыз).

3.7.1. Бір өлшемді массивтер

Бір өлшемді массивтер массивтердің ең қарапайым түрі болып есептеледі, программадағы сипатталуы:

тип *массивтің_аты* [*ұзындығы*];

мұндағы **тип** – кез келген типтердің бірі, *массивтің_аты* – дұрыс идентификатор, *ұзындығы* – массивтегі элементтер саны. Мысалы, бүтін сандар болатын 100 элементтен тұратын *a* – бір өлшемді массивтің сипатталуы келесі түрде болады:

```
int a[100]; //100 бүтін саннан тұратын a массивті жариялау
float mas[10]; //10 нақты саннан тұратын mas массивті жариялау
char sMas[25]; //25 символдан тұратын sMas массивті жариялау
```

Бір өлшемді массивтердің элементтері бір ғана индекспен анықталады және элементтерді номерлеу 0-ден басталады, мысалы, **int** a[100]; массивтің алғашқы элементін шақыру a[0], ал соңғы элементін пайдалану a[99] түрінде болады.

Массивті инициализациялаудың немесе оның элементтеріне мәндер берудің бірнеше тәсілдері қолданылады:

– массив элементтерінің мәнін массивті жариялау кезінде тікелей беруге болады, мысалы, массив келесі түрде жарияланса:

```
int m[3] = {1, 28, 103};
```

онда оның элементтерінің мәндері m[0] = 1, m[1] = 28, m[2] = 103 болғаны;

– массивтің әрбір элементінің мәнін пернетақтадан енгізіп отыру, мұнда әдетте параметрлі цикл операторы **for**-ды пайдаланады, мысалы, **int** m[3]; массивінің элементтерінің мәндерін пернетақтадан енгізу үшін келесі түрде жазады:

```
for (int i=0;i<3;i++)
cin>>m[i];                    немесе

for (int i=0;i<3;i++)
        { cout<<"a[ "<<i<<"]= ";
cin>>m[i]; }
```

– массив элементтерінің мәндерін беру үшін кездейсоқ сандар генераторын пайдалану, бұл кездейсоқ сандар генераторы функцияларының

бірі **int** random(**int** san) массив элементтерінің мәні үшін **0** мен **san-1** аралығынан кездейсоқ алынған кез келген бүтін санды бере алады, оны пайдалану үшін программа мәтініне #include <stdlib.h> директивасы қосылуы керек, мысалы, келесі программа фрагменті орындалғанда,

```
int a[5];
for (int i=0; i<5;i++)
a[i]= random(20);
```

массив элементтерінің мәні ретінде **0** мен **19** сандарының арасынан кездейсоқ алынған кез келген бүтін сандардың бірі алынады. Кездейсоқ сандар генераторының функцияларын пайдалану өте үлкен ұзындықтағы массивтерді инициализациялауда тиімді болып табылады.

3.6-жаттығу. Келесі программа элементтері нақты сандар болатын a[10] массивін, кездейсоқ сандар генераторын пайдаланып өзі құрады және олардың арасындағы [0,0.5] аралығында жататындарын тауып бере алады:

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h> // random(30) үшін қосылды
#include <iomanip.h> // setprecision() манипуляторы үшін қосылды
const n=10; //массивтің ұзындығын тұрақты түрінде жариялау
void main()
{ float a[n]; //a[n] нақты сандар массивін жариялау
            //массивті кездейсоқ сандармен толтыру
for (int i=0; i<n;i++)
a[i]= (float) random(30)/13;
//массивтің барлық элементтерін экранға шығару
cout<<"Massiv kyrildi: "<<endl;
for (int i=0; i<n;i++)
 cout<<" "<<setprecision(1)<<a[i];
 cout<<"\n\n";
//массивтің [0,0.5] аралығындағы элементтерін экранға шығару
 cout<<"[0,0.5] arasindagi elementter:"<<endl;
```

```
for (int i=0; i<n;i++)
 if(a[i]>=0&&a[i]<=0.5)
cout<<" "<<setprecision(1)<<a[i];
getch();
}
```

Бұл программада көрсетілгендей, массивтің ұзындығын, яғни массив элементтерінің санын **const** n=10; тұрақты түрінде жариялау программаның тиімділігін арттырады, мысалы бұл n шамасы программадағы циклдерде бірнеше рет қайталанады, егер массивтің ұзындығын 10 емес 20 қылып өзгерту керек болса, онда осы тұрақтыны **const** n=20; деп өзгертсе болды, қалған n-дардың барлығы өздері автоматты түрде өзгереді, ал егер n – тұрақты түрінде жарияланбай, басқаша **int** a[10]; түрінде жарияланып, циклдерде for (**int** i=0; i<**10**; i++) түрінде көрсетілсе, программаның барлық жеріндегі 10-дарды 20-ға ауыстырып отыру керек болады.

Программа орындалуының нәтижесі:

```
Massiv kyrildi:
 1  2  2  0.3  0.9  0.2  2  2  0.5  2

[0,0.5] arasindagi elementter:
 0.3  0.2  0.5
```

3.7-жаттығу. Массивтерге байланысты жиі кездесетін есептердің бірі бұл – массивтің ең үлкен немесе ең кіші элементін табуға арналған есептер. Келесі программа **int** a[20]; түрінде жарияланған массив элементтерінің арасынан ең үлкенін тауып береді.

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
const n=20; //массивтің ұзындығын жариялау
int main()
{ int massiv[n]; //массивті жариялау
```

```
 int i,j,k, max,Nmax;  //max – ең үлкен элемент, Nmax – ең үлкен
                       //элементтің номері, i,j,k,- цикл параметрлері
 cout<<"\n Bastapki massiv :" ;
 for (i=0; i<n; i++) //массивті кездейсоқ сандармен толтырып шығару
{ massiv[i]=random(30);
cout<<" " <<massiv[i];
}
max=massiv[0]; Nmax=0; //массивтің алғашқы элементін max деп алу
 for (i=1; i<n; i++)  // max элементті іздеу басталды
  if (max<massiv[i])
    { max= massiv[i]; // max элементті алу
      Nmax=i; // max элементтің номерін алу
    };
cout<<"\n\n max="<<max<<"\n";
cout<<"\n\n Nmax="<<Nmax<<"\n";
getch();
}
```

Программа орындалуының нәтижесі:

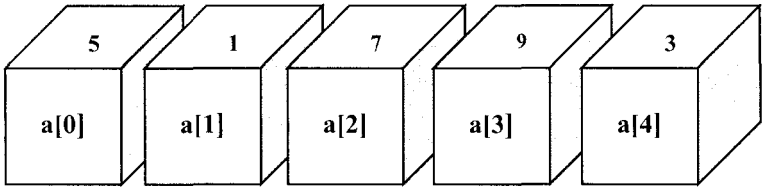
```
 Bastapki massiv : 19 24 26 4 12 3 26 24 6 21 1 14 10 2 23 9 13 28 16 15
max=28
Nmax=17
```

Программада массивтің алғашқы элементі massiv[0]-ді шартты түрде **max** элемент деп алады, оның реттік номері Nmax=0; болады, әрі қарай циклде **max**-ті массивтің қалған элементтерімен салыстырады, егер одан үлкен элемент кездессе, онда **max** – сол элементтің мәнін, ал **Nmax** реттік номерін алып отырады, осылайша массивтің барлық элементтері тексеріліп, ең үлкені табылады. Есептің жауабында **Nmax**=17 болып тұр, себебі массив элементтерін номерлеу «0»-ден басталды.

3.7.3. Массивтердің көрсеткіштермен байланысы

C++ программалау тілінде массивтер үшін көрсеткіштерді пайдаланудың өте үлкен маңызы бар. Бұл жалпы массивтің жадыдағы орна-

ласуына байланысты болып келеді. Массив элементтері жадыда бірінен кейін бірі қатар орналасқан ұяшықтарда немесе жәшіктерде сақталады. Мысалы, **int** a[5]= {5,1,7,9,3}; түрінде жарияланған массивтің жадыда орналасуы келесі түрде болар еді (3.3-сурет.)



3.3- сурет. *a[5] массиві элементтерінің жадыда орналасуы*

C++ -те *массивтің аты*, оның *алғашқы элементінің* (мысалы, **a[0]**-дің) *адресін* сақтап тұратын *көрсеткіштің* қызметін атқарады, сондықтан егер программада келесі түрде жазылса,

```
int a[5]={5,1,7,9,3}; //массивті жариялау
int *k_mas=a; // k_mas көрсеткішке массивтің атын меншіктеу
```

онда бұл жазу **int** *k_mas=&a[0]; дегенмен пара-пар болып шығады, сондықтан программалауда **int** *k_mas=&a[0]; деп жазудың орнына **int***k_mas=a; деп жаза береді. Массивтің келесі a[1] элементінің *адресін* алу үшін көрсеткіштерге қолданылатын декремент амалын k_mas++; немесе «қосу» амалын k_mas=+1; деп пайдалануға болады, ал бірден a[2] – үшінші элементтің *адресін* алу керек болса, онда k_mas=+2; деп жазуға болады. Ал сол адресі көрсетілген жәшіктегі массив элементінің *мәнін* алу керек болса, онда сәйкес көрсеткішке, *«*»-адрес бойынша мәнді алу* амалын қолданады, мысалы, келесі программа фрагменті орындалғанда:

```
int a[5]= {5,1,7,9,3}; //массивті жариялау
int *k_mas=a; // k_mas көрсеткішке массивтің атын меншіктеу
cout<<"\ *(k_mas+2)="<< *(k_mas+2); //k_mas+2 көрсеткішіне «*»-
                                    //амалын қолданып сол адрес бойынша мәнді алу
```

а массивтің үшінші жәшікте орналасқан үшінші элементінің (a[2]-нің) мәні *(k_mas+2)= **7** шығатын болады (3.3-сурет.).

3.8-жаттығу. Келесі программа **int** a[5]= {5,1,7,9,3}; түрінде жарияланған массив элементтері орналасқан жады жәшіктерінің адресін шығарып береді:

```
#include <iostream.h>
#include <conio.h>
int main()
{ int a[5]= {5,1,7,9,3}; //массивті жариялау және инициализациялау
  int *k_masBasi=a; //массивтің алғашқы элементінің көрсеткіші
  int *k_masSoni=&a[4]; //массивтің соңғы элементінің көрсеткіші
  int i=0; //массивтің алғашқы элементінің реттік номері
while (k_masBasi<=k_masSoni) //әзірге алдыңғы адрес, соңғыдан кіші
 {cout<<"\n\na["<<i<<"] adresi-> "<< k_masBasi; //ендеше сол
                                  //k_masBasi көрсеткіштегі адресті көрсету
  k_masBasi++; // келесі адреске көшу
  i++; // массивтің келесі элементінің реттік номерін алу
  }
   getch();
}
```

Программа орындалуының нәтижесі:

```
a[0] adresi-> 1245036
a[1] adresi-> 1245040
a[2] adresi-> 1245044
a[3] adresi-> 1245048
a[4] adresi-> 1245052
```

Бұл программаның нәтижесіне қарай отырып элементтерінің әрқайсысы 4 байт орын алатын (**int** типті) массив элементтерінің жадыда *кіші адрестен* басталып *үлкен адреске* қарай жәшіктерде қатар орналасатынын байқауға болады.

3.9-жаттығу. Келесі программа **int** a[5]= {5,1,7,9,3}; түрінде жарияланған массив элементтерін көрсеткіштерді пайдалана отырып кері ретпен шығарып береді:

```
#include <iostream.h>
#include <conio.h>
int main()
{ int a[5]= {5,1,7,9,3}; //массивті жариялау және инициализациялау
  int *k_masBasi=a; //массивтің алғашқы элементінің көрсеткіші
  int *k_masSoni=&a[4]; //массивтің соңғы элементінің көрсеткіші
while (k_masSoni>=k_masBasi) //әзірге соңғы адрес алдыңғыдан үлкен
   {cout<<" "<< *k_masSoni; // ендеше сол адрестегі мәнді алу
   k_masSoni--; // алдыңғы адреске түсу
    }
    getch();
}
```

Бұл есепте массив элементтерінің жадыдағы орналасуына сәйкес (3.8- жаттығу, 3.3-сурет), кейін тұрған жәшіктің адресі, алдында тұрған жәшіктің адресінен үлкен болғандықтан, шарт (k_masSoni>=k_masBasi) деп алынады.

Программа орындалуының нәтижесі келесі түрде болады:

```
3  9  7  1  5
```

Жалпы программалауда, аса қажет болмаса, массив элементтерімен жұмысты, көрсеткіштерді пайдаланбай-ақ жай индекстер арқылы да жасай береді. Бұл программаның түсінікті болуын арттыру мақсатында жасалады. Бірақ көптеген өндірістік есептерді шығаруға арналған программаларда массив элементтерінің санын, яғни массивтің өлшемін алдын ала беру мүмкін болмайды, сондықтан мұндай жағдайларда, *динамикалық массивтер* деп аталатын өлшемдері программаның орындалуы кезінде ғана анықталатын массивтер қолданылады (5.1-тақырыпты қараңыз).

3.7.3. Массивтерді сұрыптау

Кез келген ретпен орналасқан массив элементтерін белгілі бір ретпен сұрыптап орналастыруды *массивті сұрыптау* деп атайды. Массивті сұрыптау – бұл массив элементтерін өсу немесе кему ретімен орналастыру болып табылады. Массивті сұрыптаудың көптеген әдістері кездеседі, бірақ олардың барлығы да төменде берілген негізгі үш әдістен шығады деп есептеуге болады:

– тікелей таңдау (метод прямого выбора);

– алмастыру немесе көпіршік әдісі (метод обмена или метод «пузырька»);

– кірістіру әдісі (метод вставки).

3.7.3.1. Массивті сұрыптау. Тікелей таңдау әдісі

Тікелей таңдау әдісінде, егер сұрыптау өсу (кему) ретіне қарай жүрсе, алдымен массивтің ең кіші (ең үлкен) элементі табылады, келесі қадамда осы табылған элемент массивтің ең басына апарылады, ал массивтің басында тұрған элемент жаңағы ең кіші (ең үлкен) элементтің орнына барады, сонда бұл әрекеттердің нәтижесінде ең кіші (ең үлкен немесе) элемент массивтің алғашқы элементі болып орналасады. Әрі қарай тура осы әрекеттерді массивтің келесі элементінен бастап орындайды, осылай массивтің соңына дейін жүре береді.

3.10-жаттығу. Төменде берілген программа массив элементтерін кему ретімен сұрыптауда *тікелей таңдау* әдісін қолданады:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
const n=10;
int main()
{
  int massiv[n];
int i,j,k, max,Nmax,bufer;// bufer-аралық мән сақталатын айнымалы
 cout<<"\n Bastapki massiv :" ;
 for (i=0; i<n; i++)
```

```
{ massiv[i]=random(30)%13-7;
cout<<" " <<massiv[i];
}
 cout<<"\n\n SURIPTAU BASTALDI :\n";
 for (i=0; i<n; i++)
{ max=massiv[i]; Nmax=i;
    for (j=i+1; j<n; j++) // max пен Nmax-ті тауып алу
 if (max<massiv[j])
   {max= massiv[j];
    Nmax=j;
    } ;
 bufer=massiv[i]; //массивтің i-ші элементін сақтай тұру
 massiv[i]=max; // i-ші элементтің орнына max-ті жіберу
 massiv[Nmax]=bufer; // max орнына bufer-дегі сақтаулы i-ші
                          // элементті жіберу
 cout<<"\n "<<i<<"- kadam :";
  for (k=0; k<n; k++)
 cout<<" " <<massiv[k];
 }
 cout<<"\n\n suriptalgan massiv :";
  for (i=0; i<n; i++)
 cout<<" " <<massiv[i];
getch();
                }
```

Программа орындалуының нәтижесі:

```
Bastapki massiv : -1  4 -7 -3  5 -4 -7 4 -1 1
SURIPTAU BASTALDI :
0- kadam : 5 4 -7 -3 -1 -4 -7 4 -1 1
1- kadam : 5 4 -7 -3 -1 -4 -7 4 -1 1
2- kadam : 5 4 4 -3 -1 -4 -7 -7 -1 1
3- kadam : 5 4 4 1 -1 -4 -7 -7 -1 -3
4- kadam : 5 4 4 1 -1 -4 -7 -7 -1 -3
5- kadam : 5 4 4 1 -1 -1 -7 -7 -4 -3
6- kadam : 5 4 4 1 -1 -1 -3 -7 -4 -7
7- kadam : 5 4 4 1 -1 -1 -3 -4 -7 -7
8- kadam : 5 4 4 1 -1 -1 -3 -4 -7 -7
9- kadam : 5 4 4 1 -1 -1 -3 -4 -7 -7
suriptalgan massiv : 5  4  4  1  -1  -1  -3  -4  -7  -7
```

3.7.3.2. Массивті сұрыптау. Алмастыру әдісі

Массивті сұрыптаудың *алмастыру* әдісінде қатар тұрған екі элемент салыстырылады, мысалы, массивті өсу ретімен сұрыптағанда алмастыру әдісін қолданса, онда кіші немесе «жеңіл» элементтер массивтің басына қарай қалқып жылжыған сияқты болады, ал үлкен немесе «ауыр» элементтері массивтің түбіне қарай шөгіп отырғандай болады. Массив, керісінше кему ретімен сұрыпталса, «ауыр» элементтері жоғары қарай, ал«жеңілдері» төмен қарай кетеді. Осыған байланысты сұрыптаудың бұл әдісін «көпіршік» әдісі деп те атайды.

3.11-жаттығу. Төменде берілген программа массив элементтерін кему ретімен сұрыптауда *«көпіршік» (немесе алмастыру)* әдісін қолданады:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
const n=10;
int main()
{
 int i,j,k,bufer;
   int massiv[n];
  cout<<"\n Bastapki massiv :";
  for (i=0; i<n; i++) //Бастапқы массивті көру
{massiv[i]=random(30)%13-7;
cout<<" "<<massiv[i];
}
 cout<<"\n\n SURIPTAU BASTALDI :\n";
 int f=1;  // f алмастырудың бар-жоғын анықтайды, егер
           // f =0 болса алмастыру жоқ, f≠0  болса  бар
int kol=0; // kol – while циклдегі қадамдар саны
while (f)
{ f=0;   kol++;  // әзірге f =0 алмастыру болмады
 for (j=0; j<n-1; j++)
 if (massiv[j]<massiv[j+1]) //қатар тұрған элементтерді салыстыру
```

```
 {bufer=massiv[j];
  massiv[j]=massiv[j+1];
  massiv[j+1]=bufer;
  f++;  // алмастыру болды, f өзгерді
}
cout<<"\n "<<kol<<"- kadam :";
 for (k=0; k<n; k++) //Әрбір алмасудан кейінгі массивті көріп отыру
cout<<" "<<massiv[k];
 }
 cout<<"\n\n suriptalgan massiv :";
  for (i=0; i<n; i++)
 cout<<" "<<massiv[i];
getch();
}
```

```
 Bastapki massiv :
-1  4  -7  -3  5  -4  -7  4  -1  1
 SURIPTAU BASTALDI :
 1- kadam : 4  -1  -3  5  -4  -7  4  -1  1  -7
 2- kadam : 4  -1  5  -3  -4  4  -1  1 -7  -7
 3- kadam : 4  5  -1  -3  4  -1  1  -4 -7  -7
 4- kadam : 5  4  -1  4  -1  1  -3  -4 -7  -7
 5- kadam : 5  4  4  -1  1  -1  -3  -4 -7  -7
 6- kadam : 5  4  4  1  -1  -1  -3  -4  -7 -7
 7- kadam : 5  4  4  1  -1  -1  -3  -4  -7 -7
 suriptalgan massiv :
5  4  4  1  -1  -1  -3  -4  -7  -7
```

Алмастыру әдісін қолданғанда, массивтің қатар тұрған элементтерін салыстырулардың саны – «массивтің ұзындығы -1»-ге тең болады, мысалы, n-1 деген сияқты, себебі массивтің n -ші соңғы элементін салыстыратын n+1 элемент анықталмаған. Бұл жаттығудағы алмастырудың бар-жоғын анықтайтын *f* айнымалысына (f =0 болса, алмастыру жоқ, f ≠0 болса – бар) тәуелді **while** циклін пайдалану, салыстырушы j бойынша циклдің артық орындалмауын қамтамасыз етеді.

2.7.3.3. Массивті сұрыптау. Кірістіру әдісі

Массивтерді сұрыптаудың тағы бір жолы *кірістіру* әдісін пайдалану болып табылады. Бұл әдіс бойынша алдымен массивтің бастапқы (немесе соңғы) екі элементі сұрыпталып орналастырылады, сонан соң келесі тұрған элемент тексеріліп, ол алдыңғы сұрыпталып тұрған екі элементке қатысты алғанда орналасады, ал қалған элементтер оған орын босатып, кейін қарай жылжиды, сонан соң келесі элемент, осылайша барлық элемент өз орындарын тапқанша кете береді.

3.12-жаттығу. Төменде берілген программа массив элементтерін кему ретімен сұрыптауда *кірістіру* әдісін қолданады:

```
#include<iostream.h>
#include<conio.h>
const n=10;
int main()
{
 int massiv[n]={-1,4,-7,-3,5,-4,-7,4,-1,1};
int i,j,k, el;
cout<<"\n Bastapki massiv :"<<"\n" ;
for (i=0; i<n; i++)
cout<<"  " <<massiv[i];
cout<<"\n\n SURIPTAU BASTALDI :\n";
for(i=1;i<n;i++)
{
el=massiv[i]; j=i;  // el – кезектегі тексерілетін элемент
while(massiv[j-1]<el) // el – элементті тексеру
{
 massiv[j]=massiv[j-1]; // элементті кейін жылжыту
   j--;  // алдыңғы индексті алу үшін керек
 }
   massiv[j]=el; //  элементті өз орнына апарып кірістіру
   cout<<"\n "<<i<<"- kadam: ";
  for (k=0; k<n; k++) // кезекті тексеруден кейінгі массивті көру
 cout<<"  " <<massiv[k];
```

```
}
cout<<"\n";
cout<<"\n\n suriptalgan massiv :";
 for (i=0; i<n; i++)
cout<<"  " <<massiv[i];
getch();
}
```

```
Bastapki massiv  :
 -1  4  -7  -3  5  -4  -7  4  -1  1
SURIPTAU BASTALDI :
1- kadam:   4  -1  -7  -3  5  -4  -7  4  -1  1
2- kadam:   4  -1  -7  -3  5  -4  -7  4  -1  1
3- kadam:   4  -1  -3  -7  5  -4  -7  4  -1  1
4- kadam:   5  4  -1  -3  -7  -4  -7  4  -1  1
5- kadam:   5  4  -1  -3  -4  -7  -7  4  -1  1
6- kadam:   5  4  -1  -3  -4  -7  -7  4  -1  1
7- kadam:   5  4  4  -1  -3  -4  -7  -7  -1  1
8- kadam:   5  4  4  -1  -1  -3  -4  -7  -7  1
9- kadam:   5  4  4  1  -1  -1  -3  -4  -7  -7
suriptalgan massiv :  5  4  4  1  -1  -1  -3  -4  -7  -7
```

Егер де кезектегі элемент алдыңғы сұрыпталып тұрған элементтерге қатысты алғанда дұрыс, өз орнында тұрған болса, онда жылжу болмайды да, тексеру одан кейінгі тұрған элементке беріледі. Бұл әдеттегі карта ойнағандағы ойыншының қолындағы карталарын реттестіріп орналастыруы сияқты жасалады. Мұнда егер массивтегі элементтер саны n *болса* , онда әрбір $n-1$ элемент тексеріледі және ол элементті орнына қою үшін үнемі $n-1$ қалған элементтерді жылжытуға тура келеді, сондықтан массив үлкен болғанда, бұл әдіс көп уақыт алады, бірақ мұнда массив үшін қосымша жадының қажеті болмайды, массив жадыдағы өзі алып тұрған орынды ғана пайдаланады.

3.7.4. Екі өлшемді массивтер

Екі өлшемді массивтерді тік төртбұрышты кестелер немесе матрицалар деп қарастыруға болады, сондықтан мұнда «баған» және «жол» ұғымдары өз мағыналарында қолданылады. С++ программалау тілінде екі өлшемді массив әрбір элементінің өзі бір өлшемді массив болатын массив түрінде анықталады, программадағы сипатталуы:

тип *массивтің_аты [жол_саны] [баған_саны];*

мысалы,

```
int a[5][5]; //5 жолы, 5 бағаны бар бүтін сандар матрицасы
float mat [3][2]; //3 жолы,2 бағаны бар нақты сандар матрицасы
```

Екі өлшемді массивтің элементтері екі индекспен анықталады. Айталық, mat[0][2]=0.01; түріндегі оператор mat матрицасының 0-жолы мен 2-бағанының қиылысуындағы элементінің мәні 0.01-ге тең екенін білдіреді, ал mat[i][j] – mat матрицасының і-жолы және j – бағанының қиылысуындағы элементті береді.

Екі өлшемді массивтерді енгізудің бірнеше жолдары бар, мысалы, mat[3][3] матрицасын енгізудің бірнеше жолдарын қарастырайық:

– екі өлшемді массив элементтерін инициализациялау арқылы бірден анықтап кетуге болады, мысалы:

```
int mat [3][3] = {  {10,20,30},
                    {40,50,60},
                    {70,80,90}
                  };
```

мұнда массивтің әрбір жолы {}, фигуралы жақшаға алынады;

– массив элементтерін пернетақтадан енгізу үшін іштестірілген **for** цикл операторлары қолданылады:

```
for (int i=0; i<3;i++)
```

```
for (int j=0; j<3;j++)
cin>> mat [i][j];
```

– екі өлшемді массив элементтерінің мәнін кездейсоқ сандар генераторымен анықтау:

```
#include <stdlib.h>
... ... ...
for (int i=0; i<3;i++)
for (int j=0; j<3;j++)
mat [i][j] = random(50);
```

Екі өлшемді массивтермен жұмыс жасағанда, оны экранға дәстүрлі матрица түрінде де шығаруға болады, мысалы, mat[3][3] матрицасын экранға дәстүрлі түрде шығаратын программа фрагменті келесі түрде болады:

```
for (int i=0; i<3;i++)
{  cout<<"\n";
   for (int j=0; j<3;j++)
       cout<<"  "<< mat [i][j];
}
```

Екі өлшемді массивтерге байланысты есептерде матрицалардың келесі қасиеттері жиі пайдаланылады:

– матрицаның бас диагоналінде жатқан элементтің жолы мен бағандарының номері бірдей болады, яғни **i=j** шарты орындалады;

– матрицаның бас диагоналінен төмен орналасқан элементтің жол номері оның баған номерінен үлкен болады, **i>j** шарты орындалады;

– матрицаның бас диагоналінен жоғары орналасқан элементтің баған номері оның жол номерінен үлкен болады, **i<j** шарты орындалады;

– матрицаның қосымша диагоналында жататын элементтің индекстері үшін келесі **i + j -1 = n** шарты орындалады.

3.13-жаттығу. Келесі программа кездейсоқ сандар генераторын пайдаланып, бес жолдан және бес бағаннан тұратын **a [5][5]** нақты сан-

дар матрицасын құрады және оның бас диагональдан төмен орналасқан элементтерін шығарып береді:

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
#include<iomanip.h>
int main()
{float a[5][5];
 for (int i=0; i<5;i++)
 for (int j=0; j<5;j++)
 a[i][j]= (float) random(113)/171; //матрица элементтері мәндерін
                                   //генератормен анықтау
 for (int i=0; i<5;i++)
   { cout<<"\n";
              for (int j=0; j<5;j++)      //матрицаны дәстүрлі түрде
                                          //форматтап шығару
     cout<<"\t"<<setprecision(1)<<a[i][j];
   }
 for (int i=0; i<5;i++)
   { cout<<"\n";
     for (int j=0; j<5;j++)
        if(i>j) //элементтің бас диагональдан төмен екенін тексеру
           cout<<"\t"<<setprecision(1)<<a[i][j];
   }
 getch();
              }
```

Программа орындалуының нәтижесі келесі түрде болады:

```
0.2   0.09   0.6   0.6   0.4
0.4   0.5    0.4   0.6   0.6
0.2   0.4    0.06  0.2   0.4
0.6   0.2    0.4   0.2   0.1
0.5   0.08   0.05  0.2   0.4
0.4
0.2   0.4
0.6   0.2    0.4
0.5   0.08   0.05  0.2
```

С++ программалау тілінде екі өлшемді массивтің өзі, бір өлшемді массивтерден құралатын массивтердің массиві болғандықтан, компьютер жадында олар жол бойынша орналасады, мысалы, келесі екі өлшемді массив немесе матрица:

```
int a[3][3]={ {10,20,30},
              {40,50,60},
              {70,80,90} };
```

элементтерінің компьютер жадысындағы орналасуы төмендегіше болуы мүмкін, мысалы, a[0,0] элемент 1245020 адрестен басталатын (боялған ұяшықтың адресі) қатар тұрған төрт (**int** типі 4 байт) ұяшықты алады, бұл төрт ұяшық құрайтын жәшіктің сыртындағы этикеткасы a[0,0] болады, сонда сыртындағы этикеткалары a[0,0], a[0,1], a[0,2] болатын қатар тұрған кішкене жәшіктер сыртындағысы a[0] болатын тағы бір үлкен жәшікке кіретін болады (3.4-сурет).

1245020	1245024	1245028	1245032	1245036	1245040	1245044	1245048	1245052
a[0,0]	a[0,1]	a[0,2]	a[1,0]	a[1,1]	a[1,2]	a[2,0]	a[2,1]	a[2,2]
10	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**
a[0]			a[1]			a[2]		

3.4-сурет. *Екі өлшемді* **int** *a[3][3] массиві элементтерінің жадыда орналасуы*

Мұндағы бір өлшемді массив элементтері болып тұрған a[0], a[1], a[2]-лердің әрқайсысы тағы да үш элементтен тұратын бір өлшемді массивтер болып тұрғанын байқауға болады және олар жолдардың атын білдіріп тұр.

Екі өлшемді массив үшін жарияланған көрсеткіштің мәні, бұл массивтің алғашқы элементінің адресі болып табылады, мысалы, келесі берілген :

```
int a[3][3];
int *pa;
```

жариялануға сәйкес pa=a немесе бұл pa=&a[0][0]; дегенмен парапар, яғни pa көрсеткіштің мәні массив-матрицаның a[0,0] элементінің адресі болып тұр. Сондай-ақ, осы жариялануға сәйкес жолдардың аты болып тұрған a[0], a[1], a[2]-лердің әрқайсысы өздерінің алғашқы элементтерінің адресін алып тұрған көрсеткіштер болып табылады, яғни:

a[0]=&a[0][0]; a[1]=&a[1][0]; a[2]=&z[2][0];

3.14-жаттығу. Төменде мәтіні берілген программа көрсеткішті пайдалана отырып **int** a[3][3] матрицаның бас диагоналындағы, бірінші жолындағы және бірнші бағанындағы элементтерді шығарып бере алады:

```
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
void main()
{
int a[3][3]={{10,20,30},
             {40,50,60},
             {70,80,90}};
 int i,j;
int *pa[3]; //немесе бұл  *pa[3]={a[0],a[1],a[2]}; дегенді білдіреді
 for (i=0; i<3; i++)
     pa[i]=a[i]; //*pa[3] көрсеткіш-массив элементтері  мәндерін беру
int *p=a[0]; //мәні a[0] адресі болатын көрсеткішті жариялау
cout<<"\n DIAGONAL ELEMENTTERI: ";
for (i=0;i<9;i+=4) //келесі үш жәшіктен аттап, төртіншіге түсу үшін
                   //i- дің мәнін  бірден 4-ке өсіру
cout<<"\t"<<*(p+i); //Бас диагональдағы элементтерді шығару
cout<<"\n\n BIRINSHI GOL ELEMENTTERI: ";
for (i=0; i<3; i++) //келесі жәшікке көшу үшін i++ жеткілікті
cout<<"\t"<<*(p+i); //Бірінші жолдың элементтерін шығару
cout<<"\n\n BIRINSHI BAGAN ELEMENTTERI: ";
for (i=0; i<3; i++)
```

```
cout<<"\t"<<*pa[i]; //Бірінші баған элементтерін шығару
getch();
}
```

Бұл программаның орындалуының нәтижесі келесі түрде болады:

```
DIAGONAL ELEMENTTERI:      10   50   90
BIRINSHI GOL ELEMENTTERI:  10   20   30
BIRINSHI BAGAN ELEMENTTERI: 10   40   70
```

Мұнда берілген матрица элементтерінің компьютер жадысында жол бойынша тізбектей орналасатындығы негізге алынады:

a[0][0], a[0][1], a[0][2], a[1][0], **a[1][1]**, a[1][2], a[2][0], a[2][1], **a[2][2]**

сонда бас диагональ бойында орналасқан элементтер нолінші, төртінші және сегізінші орындарда тұр, яғни оларды алу үшін **i** – цикл қадамын 4-ке өсіріп отыру қажет болды.

Жалпы бұл тақырыпта программалауды оқытуда қарастырылатын бір өлшемді массивтер және екі өлшемді массивтер (матрица) қарастырылды. Бұл айтылғандардан басқа көп өлшемді массивтерді де программалауға болады. Өлшемдері екеуден артық болатын массивтердің де, жадыда орналастырылуы жоғарыда айтылған екі өлшемді массивтердегі сияқты жүзеге асырылады.

3.8. Құрылым (struct)

C++ программалау тілінде әр түрлі типтегі деректерді (мысалы, түрлі типтегі айнымалыларды, массивтерді, көрсеткіштерді және т.б.) біріктіріп, құрама бір тип ретінде қарастырып пайдалану үшін *құрылым* **(struct – структура)** деп аталатын тип қолданылады. Қазіргі кездегі объектіге бағдарланған программалаудың негізі болып есептелетін *кластың* өзін бастапқыда осы құрылымды жетілдіруден пайда болған деп те айтылады. Жалпы программалауда бір объектіге қатысты әр түрл

деректерді біріктіріп, біртұтас дерек ретінде қарастыруда құрылымды пайдалану өте тиімді болып табылады. Құрылымның программадағы жариялануы келесі түрде болады:

```
struct типтің_аты
 {
   тип1  1_элемент_аты;
   тип2  2_элемент_аты;
   …  …  …
   типN  N_элемент_аты;
};
```

мұндағы типтің_аты – кез келген дұрыс идентификатор, тип_1, тип_2, ..., тип_N – осы құрылымның өзінен басқа кез келген және сол құрылым үшін анықталған көрсеткіш тип бола алады, ал 1_элемент_аты, 2_элемент_аты, N_элемент_аты – кез келген дұрыс идентификаторлар. Құрылымның бұл элементтерін *өрістер* деп те атайды. Мысалы, студенттің сессиядағы үлгерімі туралы ақпаратты келесі құрылым түрінде жариялауға болады:

```
struct student // student  құрылымын жариялау басталды
  {
    string  fam; //Студенттің фамилиясы
     int baga[3]; // төрт сабақ бойынша бағалары
 }; //student  құрылымын жариялау аяқталды
```

Бұл жариялану сол программадағы student деп аталатын жаңа құрылымдық типтің анықталғанын білдіреді, сондықтан программада әрі қарай осы типтің өкілдері болатын айнымалыларды, массивтерді, көрсеткіштерді жариялай беруге болады. С++ программалау тілінде типі құрылым болатын айнымалыларды, массивтерді, көрсеткіштерді жариялаудың бірнеше әдістері кездеседі:

– типі жаңадан анықталған құрылым болатын айнымалыларды, массивтерді, көрсеткіштерді немесе т.б. құрылымның соңынан ілестіріп бірге жариялауға болады, яғни келесі түрде:

```
struct типтің_аты
 {
   тип1  1_элемент_аты;
   тип2   2_элемент_аты;
   …  …   …
   типN  N_элемент_аты;
} айнымалылар тізімі;
```

мысалы, student типінің өкілі болатын X айнымалысын, top[3] массивін және *ukazStr көрсеткішті жариялау төмендегідей болады:

```
struct student  // student құрылымын жариялау басталды
  {
    string  fam; //Студенттің фамилиясы
    int baga[3]; // төрт сабақ бойынша бағалары
    } X, top[3], *ukazStr ; //айнымалыны, массивті, көрсеткішті жариялау
```

Жариялаудың бұл түрін қолданғанда, құрылымның атын жазбай-ақ қоюға болады, бірақ міндетті түрде айнымалылар тізімі берілуі керек;

– екінші жағдайда типі құрылым болатын айнымалылар немесе т.б. программада стандарт типтер секілді жарияланатын болады:

```
struct типтің_аты
 {
   тип1  1_элемент_аты;
   тип2   2_элемент_аты;
   …  …   …
   типN  N_элемент_аты;
} ;
          типтің_аты   айнымалылар тізімі;
```

мысалы,

```
struct student // student   құрылымын жариялау басталды
 {
   string  fam; //Студенттің фамилиясы
    int baga[3]; // төрт сабақ бойынша бағалары
 }; // student құрылымын жариялау аяқталды
```

```
student X,pressa[3],*ukazStr; //айнымалыны, массивті, көрсеткішті жариялау
```

Программада анықталған құрылымның өкілі болатын бірде-бір айнымалы, массив немесе т.б. жарияланбаған болса, онда жадыдан ол құрылым үшін орын берілмейді, ол программадағы жай мәтін секілді болып қала береді, яғни нақтылап айтқанда, егер жарияланған өкілі болмаса құрылым өздігінен тип бола алмайды.

Типі құрылым болатын айнымалылармен немесе т.б. пайдалану үшін нүкте (.) және бағыттауыш (->) амалдары қолданылады. Егер амал айнымалылардың немесе т.б. объектілердің тікелей өзіне қолданылатын болса, онда ол нүкте (.) амалы арқылы жүзеге асырылады, ал егер типі құрылым болатын айнымалы немесе т.б. объектілер үшін анықталған көрсеткіштер пайдаланылатын болса, онда сәйкесінше бағыттауыш (->) амалы арқылы әрекет жасалады. Бұл амалдар, құрылымның элементтерін немесе *өрістерін* пайдалану үшін қолданылады, программадағы жазылуы:

```
    айн_аты.элем_аты= мәні;
немесе
    көрсеткіш_аты->элем_аты= мәні;
Мысалы,
        top[1].fam="Дулатов";    top[1].baga[2]=5;
немесе
        ukazStr-> fam=" Дулатов ";
```

Типі құрылым болатын айнымалылар, массивтер және т.б. үшін деректер беру, оның барлық элементтерінің (өрістерінің) мәндерін енгізу арқылы жүзеге асырылады. Мұның бірнеше тәсілдері бар:

– инициализациялау жолымен беру, яғни мұнда құрылым элементтерінің мәндері программа мәтінінде бірден беріледі, мысалы:

```
struct student  // student құрылымын жариялау басталды
 {
   string  fam; //Студенттің фамилиясы
    int baga[3]; // төрт сабақ бойынша бағалары
 }; // student  құрылымын жариялау аяқталды
```

```
student  X; //құрылым типті X айнымалыны жариялау
                ... ...   ...
X.fam=" Дулатов";
X.baga[0]=4;
X.baga[1]=5;
X.baga[2]=5;
X.baga[3]=4;
 ... ...   ...
```

мұнда student деп аталатын құрылым типті X айнымалысының мәндері инициализациялау жолымен беріліп тұр.

– құрылым элементтерінің (өрістер) мәндерін cin немесе scanf операторларының көмегімен пернетақтадан енгізуге болады, мысалы:

```
struct student // student  құрылымын жариялау басталды
  {
    string  fam; //Студенттің фамилиясы
     int baga[3]; // төрт сабақ бойынша бағалары
  }; // student  құрылымын жариялау аяқталды
       student  X; //құрылым типті X айнымалыны жариялау
  ... ...   ...
        cin >>X.fam;
  for(int j=0; j<=3;j++) //X студенттің бағаларын  енгізу
  cin>>X.baga[j];
              ... ...   ...
```

3.15-жаттығу. Топтағы студенттердің сессиядағы үлгерімі туралы ақпаратты құрылым түрінде жариялайтын және қолданушының сұрауы бойынша қажет студенттің орташа бағасын есептеп шығарып беретін программа мәтіні келесі түрде болады:

```
#include <vcl.h>
 #include <iostream.h>
 #include <conio.h>
 #include <string>
 struct student // student  құрылымын жариялау басталды
   {
```

```
    string  fam; //Студенттің фамилиясы
    int baga[3]; // сессиядағы төрт сабақ бойынша бағалары
  }; // student  құрылымын жариялау аяқталды
void main()
{  const int n=2; //топтағы студенттер саны
   student top[n]; //muni  student  болатын top массивті жариялау
   for(int i=0; i<=n;i++)  //массив элементтерін енгізу басталды
   {  cout<<"\n"<<i<<"- stud. Fam :  ";
  cin>>top[i].fam; //i-ші студенттің фамилиясын енгізу
  cout<<"\n"<<i<<"- stud. 4 sabaktan bagalari :  " ;
  for(int j=0; j<=3;j++)  //i-ші студенттің бағаларын  енгізу
  cin>>top[i].baga[j];
  }// массив элементтерін енгізу аяқталды
cout<<"\n";
    string  k;// қажет студенттің фамилиясы
    cout<<"\n Kaget studenttin fam. engiz : " ;
    cin>>k;
    cout<<"\n"<<"Stud. "<< k <<"- nin ortasha bagasi:  " ;
  for (int i=0; i<=n; i++)
   if(top[i].fam==k) //топтан k-фамилияны іздеу
  {  float s=0;
  for (int j=0; j<=3;j++)
  s=s+(float) top[i].baga[j]; //орташа бағаны есептеу
   s=s/4;  cout<<s;
   }
  getch();
 }
```

Программа орындалуының нәтижесі келесі түрде болады:

```
 0- stud. fam :  Абаева_Жансая
0- stud. 4 sabaktan bagalari :  4 5 5 4
1- stud. fam :  Булатов_Айбек
1- stud. 4 sabaktan bagalari :  4 4 3 3
2- stud. fam :  Асанов_Айболат
2- stud. 4 sabaktan bagalari :  5 5 5 5
 Kaget studenttin fam. engiz : Булатов_Айбек
Stud. Булатов_Айбек- nin ortasha bagasi:  3.5
```

3.9. union анықтайтын біріктірулер

С++ -те программа жазу барысында жады көлемін тиімді пайдалану үшін қолданылатын *«бірігулер» (объединения, смесь)* деп аталатын арнаулы тип қарастырылған. Программадағы жазылуы:

```
union біріктірудің_аты
 {   тип1  1_элемент_аты;
     тип2   2_элемент_аты;
     ...  ...   ...
     типN  N_элемент_аты;
 };
```

Мысалы,

```
union birge  //бірігуді жариялау басталды
  {
   short int a; //a-ның ұзындығы 2-ге тең
   char b; //b-ның ұзындығы 1-ге тең
   double c; //c-ның ұзындығы 8-ге тең
  } ;
        birge x; //типі birge болатын x айнымалыны жариялау
```

Мұнда біріктірудің құрылым сияқты анықталатынын байқауға болады, олардың айырмашылығы құрылым типті айнымалыны жариялағанда, жадыдан оның әрбір элементіне сәйкес жеке-жеке орын бөлінетін болады, ал *бірігу* типті айнымалыны жариялағанда, жадыдан бір-ақ орын бөлінеді. Бөлінген жады көлемі *бірігудің* құрамындағы ең үлкен элементтің ұзындығына (мысалы, с-ның ұзындығына) тең болады және бірігудің құрамындағы кез келген элемент үшін де жадыдағы сол орын пайдаланылатын болады.

Типі бірігу болатын айнымалылармен немесе т.б. жұмыс жасау үшін құрылымдағы секілді нүкте (.) және бағыттауыш (->) амалдары қолданылады.

3.16-жаттығу. Типі *бірігу* болатын **x** және типі *құрылым* болатын **y** айнымалыларының ұзындықтарын көрсетіп салыстыратын программа келесі түрде болады:

```
#include <iostream.h>
#include <conio.h>
 union birge //бірігуді жариялау басталды
                {
short int a; //a-ның ұзындығы 2-ге тең
  char b; //b-ның ұзындығы 1-ге тең
  float c; //c-ның ұзындығы 4-ке тең
   } ; //бірігуді жариялау аяқталды
  struct kuru  //құрылымды  жариялау басталды
    {
        short int a; //a-ның ұзындығы 2-ге тең
       char  b; //b-ның ұзындығы 1-ге тең
        float c; //c-ның ұзындығы 4-ке тең
    }; //құрылымды  жариялау аяқталды
void main()
{
    birge x; //типі бірігу  болатын x айнымалыны жариялау
    kuru y; //типі құрылым болатын x айнымалыны жариялау
                    //типі бірігу  болатын x-тың ұзындығын табу
                  cout<<"\n birigu:  sizeof(x) = "<< sizeof(x);
                   //типі құрылым  болатын y-тің ұзындығын табу
                  cout<<"\n kurulim:  sizeof(y) =  "<< sizeof(y);
   getch();
}
```

Программа орындалуының нәтижесі төмендегідей болады:

```
birigu : sizeof(x) = 4
kurulim : sizeof(y)=  8
```

Программа нәтижесіне қарағанда бірігу мен құрылымның элементтері бірдей болғанына қарамастан, олардың жадыдағы орындарында

айырмашылық бар екені көрініп тұр. Бірігу типті **x** айнымалы үшін оның ұзындығы ең үлкен элементі c-ның (c-ның ұзындығы 4-ке тең) ұзындығына тең орын бөлініп тұр, ал құрылым типті y айнымалысы үшін оның әрбір элементіне орын беріліп, олардың қосындысы 2+1+4=7 (a-ның ұзындығы – 2, b-ның ұзындығы – 1, c-ның ұзындығы – 4) болады, бірақ ұзындық 2^n байтпен өлшенетін болғандықтан, оның мәні 8 болып беріліп тұр. Бұл есепті орындағанда, дербес компьютерлердің мүмкіндіктеріне байланысты типтердің ұзындықтары басқаша болуы мүмкін екенін ескерген жөн.

Бақылау сұрақтары

1. Саналатын типтерді не үшін пайдаланады?
2. Саналатын типтер қалай құрылады?
3. **typedef** қызметші сөзін қолданып алынатын тип қалай аталады?
4. Программадағы көрсеткіштің атқаратын қызметі қандай?
5. Көрсеткіштерге қандай амалдарды қолдануға болады?
6. Жиын типті деректерді программада қалай жариялайды?
7. Жиын типті айнымалылар қалай сипатталады?
8. Бос жиын деген не және ол қалай беріледі?
9. Бір өлшемді массивті программада қалай жариялайды?
10. Бір өлшемді массив жадыда қалай орналасады?
11. Бір өлшемді массив элементтерін енгізудің қандай тәсілдері бар?
12. Бір өлшемді массивті сұрыптаудың қандай алгоритмдері бар?
13. Екі өлшемді массивті программада қалай жариялайды?
14. Екі өлшемді массив жадыда қалай орналасады?
15. Екі өлшемді массив элементтерін егізудің қандай тәсілдері бар?
16. Құрылым түріндегі тип қалай анықталады?
17. Құрылымның өрісі деп не аталады?
18. Құрылым мен біріктірудің айырмашылығы қандай?
19. Құрылым өрістеріне қандай операциялар қолданылады?
20. Құрылым мен массивтің қандай айырмашылығы бар?

Тапсырмалар

1. 1..100 бүтін сандар жиынынан 444 саны қалдықсыз бөлінетін сандар жиынын бөліп алыңыз. Осы жиынды экранға шығарыңыз.

2. Латын әріптері жиынынан сіздің атыңызды құрайтын символдар жиынына кірмейтін символдардан тұратын жиынды бөліп алыңыз. Осы жиынды экранға шығарыңыз.

3. Пернетақтадан енгізілген атыңызды, әкеңіздің атын және фамилияңызды құрайтын символдар жиынына бір мезгілде кіретін символдар жиынын анықтап, экранға шығарыңыз.

4. Пернетақтадан енгізілген тек қана атыңыз бен әкеңіздің атын немесе фамилияңызды құрайтын символдар жиынын анықтап, экранға шығарыңыз.

5. Пернетақтадан енгізілген туған жылыңызды құрайтын символдардың '0'..'9' ауқымына кіретін және кірмейтін символдар жиынын анықтап, экранға шығарыңыз.

6. 1..100 бүтін сандар жиынынан бүтін сандар квадраты болатын сандар жиынын бөліп алыңыз.

7. X(70) массиві берілген. Осы массивтің -1<X[i]<=1 шартын қанағаттандыратын элементтерін Y массивіне жазып және олардың санын есептеу қажет.

8. B(50) массиві берілген. Осы массивтің B[i]>0 шартын қанағаттандыратын элементтерінің арасынан ең үлкенін анықтау және оның реттік номерін табу қажет.

9. C(40) массиві берілген. Осы массивтің C[i]<0 шартын қанағаттандыратын элементтерінің арасынан ең кішісін анықтау және оның реттік номерін табу қажет.

10. Y(20) массиві берілген. Осы массивтің Y[i]>0 шартын қанағаттандыратын элементтерінің геометриялық ортасын табу қажет.

11. N(50) массиві берілген. Осы массивтің үш еселі элементтерінің қосындысын табу қажет.

12. Массивте пернетақтадан енгізілген 15 әр түрлі бүтін санды сақтауды ұйымдастыратын программа жазыңыз. Массив ішіндегілер өсу ретімен сұрыпталуы керек. Содан кейін массивте бар болуы тексерілетін бақылау саны пернетақтадан енгізіледі. Енгізілген санды массивке өсу реті бұзылмайтындай қылып орналастыру керек. Жаңа элемент қосылған массивті баспаға шығару.

13. Массивте әр түрлі 10 бүтін санды сақтауды ұйымдастыратын программа жазыңыз. Массив ішіндегілер өсуі бойынша сұрыпталады. Содан кейін пернетақтадан массивте бар болуы тексерілуі қажет бақылау саны енгізіледі. Егер массивте бақылау санына тең болатын массив элементі табылса, онда оны нольмен алмастырыңыз. Массив ішіндегілерді экранға шығарыңыз.

14. Берілген A(5,5) квадратты матрицасы бас диагоналіне симметриялды қатысты болып табылатынын анықтау қажет.

15. B(4,4) матрицасы берілген. Бірінші баған элементтерінің өсу бойынша сұрыпталғанын анықтау қажет.

16. C(5,5) матрицасы берілген. Әрбір қатардың бірінші элементі мен максималды элементтерінің орындарын ауыстыру қажет.

17. D(3,3) матрицасының әрбір қатарының 10-нан үлкен бірінші элементін B массивіне көшіріп жазу қажет.

18. Q(5,5) матрицасы берілген. Әрбір қатардың соңғы нолін 5-ке ауыстыру қажет.

19. D(4,4) матрицасы берілген. Оң сандар арасында максималды санды, теріс сандар арасында минималды санды анықтау және олардың орындарын ауыстыру қажет.

20. A(4,4) матрицасы берілген. Әрбір бағанның бірінші нолін сол бағандағы нольдер санына алмастыру қажет.

21. F(9,3) матрицасы берілген. Бірінші бағанның барлық элементтері бас диагональдағы сәйкес элементтерге тең болуын анықтау. Егер тең болмаса, олардың орнын ауыстыру қажет.

22. C(5,5) матрицасы берілген. Элементтері матрицаның бағанындағы нольдер санына тең B векторын алу қажет.

23. B(4,4) матрицасы берілген. Егер қатарда ең болмағанда бір 1 деген саны болса, онда сол қатарды ноль санына ауыстыру қажет.

24. 4x4 өлшемді бүтін сандар матрицасы берілген. Матрицаны 90 градусқа айналдырыңыз және матрица ішіндегілерін экранға шығарыңыз.

25. 4x4 өлшемді бүтін сандар матрицасы берілген. Осы матрицаның қатарлары мен бағандарының орнын ауыстырыңыз. Өңдеуге дейінгі және кейінгі матрица ішіндегілерін экран мониторына шығарыңыз.

26. 3x4 өлшемді бүтін сандар матрицасы берілген. Матрицаның әр бағанын кемуі бойынша сұрыптаңыз. Өңдеуге дейінгі және кейінгі матрица ішіндегіні экран мониторына шығарыңыз.

27. СТУДЕНТ құрылымын жариялаңыз және оған келесі деректерді енгізіңіз: тегі, есімі, әкесінің аты-жөні және информатика, программалау, ақпараттық жүйелер, программалау технологиясы пәндері бойынша бағалары. Қанша студенттің информатика пәніне «қанағаттанарлықсыз» баға алғанын анықтау қажет.

28. ТРАНСПОРТ құрылымын жариялаңыз және оған келесі деректерді енгізіңіз: жолаушының тегі, жолаушының есімі, жолаушының әкесінің аты, жүктері (заттар саны немесе салмағы кг). Салмағы 30 кг-нан артық жүгі бар жолаушылар санын және олардың аты-жөндерін анықтау қажет.

29. Телефон анықтамасы түріндегі құрылымдық массивті жариялайтын және мәліметтерді енгізуді, фамилия бойынша телефон номерін іздейтін, «компьютерлік ойындармен айналысушылар» критерийі бойынша барлық абоненттердің тізімін беретін программа жазыңыз.

30. Химиялық элементтер кестесін элементтердің атын, символдық белгілеулерін, атом массасын, атомдық ядро зарядын, негізгі химиялық қасиеттерін құрылым түрінде анықтайтын программа жазыңыз.

31. Құрылым түріндегі мектеп журналын сипаттаңыз. Оқушы фамилиясын, пән, баға жөніндегі ақпараттарды сақтауға арналған өрістерді қарастырыңыз. Сынып оқушыларының үлгерімі жөніндегі пернетақтадан енгізілген мәліметтерді оқитын және оқу озаттары, сыныптың орташа үлгерімі жөніндегі деректерді экранға шығаратын программа жазыңыз.

32. Мекеме қызметкерлерінің телефон номерлерін құрылым түрінде жариялаңыз және оған келесі деректерді енгізіңіз: фамилиясы, инициалы және телефон номері. Қызметкерлердің телефондарын іздеуді олардың фамилиясы мен инициалдары бойынша ұйымдастыратын программа жазыңыз.

33. Экспортқа шығатын тауар жөніндегі ақпараттарды құрылым түрінде жариялап және оған келесі деректерді енгізіңіз: тауар аты, тауарды импорттайтын ел және тапсырыстағы партия көлемі, данамен көрсетіледі. Осы тауарлар экспортқа шығарылатын елдер тізімін құрыңыз және оның экспортының жалпы көлемін анықтайтын программа жазыңыз.

4-тарау. С++ ТІЛІНДЕГІ ФУНКЦИЯ

4.1. Функция. Функцияны және прототипті жариялау

Практикада үлкен күрделі есептерді шығаруды орындайтын программалар жазу барысында ол күрделі есепті бірнеше жекелеген, шағын есептерге бөліп қарастырады. Есептерді мұндай ұсақтап қарастыруды программалау тілінде «декомпозиция» деп те атайды. Осы жекелеген кішкентай шағын есептерді шешуге арналған программаның бөліктерін *«функция»* деп атайды. Программада қолданушы немесе программист құратын әрбір *функция*, өзіне сәйкес есепті шешудің алгоритміне қатысты анықталып, сипатталатын типтерді, айнымалыларды тұрақтыларды және басқа орындалатын операторларды – барлығын бір атаумен біріктіріп тұрады және ол орындалғаннан кейін белгілі бір нәтижені қайтарады немесе береді. Программадағы қолданушы құратын әрбір функция бір ғана есепті шығарып, бір ғана мәнді қайтарғаны, яғни бір ғана нәтижені бергені дұрыс болады. Әдетте С++-тегі программалар осындай бірнеше функциялардан құралады. Осы уақытқа дейінгі қарастырылған программалар жалғыз ғана main() функциясынан тұрды, яғни ол программаларда программистің өзі құрған қолданушының функциялары болған жоқ. С++ программалау тіліндегі қолданушы құратын функцияның программадағы жалпы жариялануы келесі түрде болады:

```
тип функция_аты (тип параметр1, ... , тип параметрN)
    {
    айнымалыларды_жариялау;
     операторлар;
    return (қайтаратын мән);
    }
```

мұндағы **тип** функция орындалғаннан кейін қайтарылатын мәннің немесе функция нәтижесінің типін көрсетеді, бұл С++-тегі массив пен функциядан басқа кез келген тип бола алады. Егер функция орындалуының нәтижесі белгілі бір типті қайтармайтын болса, онда оның типін **void** деп анықтайды, бұл жағдайда **return** (қайтаратын мән)

сөзін жазбай тастап кетуге болады, ал қайтаратын мәннің типі тіпті де көрсетілмесе, компилятор оны әдеттегідей (по умолчанию) **int** типті деп қабылдап кетеді, дегенмен де программада түсінікті болу үші функция қайтаратын мәннің типін көрсетіп отырған дұрыс болады; *функция_аты* – бұл дұрыс идентификатор болып табылады; **тип** параметр1, …, **тип** параметрN бұл – функцияның жұмысына қатысатын, яғни функцияға сырттан деректер беру үшін немесе т.б. үшін қолданылатын айнымалы – аргументтер және олардың типтері. Бұл айнымалыларды функцияның *параметрлері* немесе функцияның *аргументтері* деп атайды. Сондай-ақ, функцияны жариялағанда бұл параметрлердің болмауы да мүмкін. Функция параметрлері берілмеген жағдайда функция келесі түрде жазылады:

```
тип функция_аты ()
{
айнымалыларды_жариялау;
  орындалатын_операторлар;
  return (қайтаратын мән);
}
```

Көптеген жағдайларда функция параметрлері берілмесе, онда программаның басқа орталарда орындалуында сәйкессіздіктер болмау үшін функцияны келесі түрде жариялайды:

```
тип функция_аты ( void)
{
айнымалыларды_жариялау;
 орындалатын_операторлар;
  return (қайтаратын мән);
}
```

Егер функцияда параметрлер көрсетілетін болса, онда оны *параметрлі функция* деп атайды.

С++ программалау тіліндегі функцияның жазылуындағы бірінші тұрған,

тип *функция_аты* (**тип** параметр1, … , **тип** параметрN)

жолды, шартты түрде функцияның тақырыбы немесе *прототипі* деп, ал қалған { } фигуралы жақшаларға алынған бөлігін *функцияның денесі* деп атайды, функцияның денесін есеп алгоритміне байланысты анықталатын айнымалылар, тұрақтылар, операторлар және т.б. құрайды.

Мысалы, екі санның үлкенін табатын **max** функциясының жазылуы келесі түрде болады:

```
int max(int x, int y) // параметрлі функция, x, y -парметрлер
   {
    int z; //айнымалыны жариялау
    if (x>y) //шарт операторы
      z=x;
    else
      z=y;
   return(z);
  }
```

Мысалы, жай ғана «Менің атым Қожа» деген мәтінді жаңа жолға шығаратын функцияның жазылуы:

```
void  gazu ()  //параметрі жоқ функция
   {
      cout<<"\n Менің атым Қожа"; // оператор
   }
```

C++ программалау тілінде функцияны жариялауды оның программадағы шақырылуынан кейін де немесе бұрын да жасай беруге болады. Осыған байланысты функцияның жариялануы екі түрлі жолмен берілуі мүмкін:

– Бірінші жағдайда функцияның *прототипін* жариялау қарастырылсын. Функция прототипін жариялау бұл функцияның тақырыбы мен денесін программада *функция шақырылғаннан* ***кейін*** анықтайтын жағдайларда қолданылады, яғни прототипі жарияланған функцияны жариялау программаның кез келген жерінде жүргізіле береді, сондайақ ол функцияның жариялануы басқа модульде немесе файлда болуы

да мүмкін. Бұл кезеңнің қызметі программадағы функция орындауға жіберілмей тұрып, яғни ол шақырылғанға дейін функция қайтаратын мәнге және оның параметрлеріне жадыдан қажет орын алып қоюды және функция кодының кейін анықталатынын негізгі орындалатын программаға хабарлауды қамтамазыз ету болып табылады. Функция прототипін жариялаудың синтаксисі:

тип *функция_аты* (**тип** параметр1 , ..., **тип** параметрN) ;

Программада функцияның *прототипі* функция бірінші рет шақырылғанға дейін жариялануы керек және де прототипі жарияланған функцияның тақырыбы да дәл сол прототиппен бірдей болуы керек, айырмашылық прототипті жариялау «;» белгісімен аяқталады, ал функция тақырыбының соңына мұндай белгі қойылмайды. Функция прототипі берілген жағдайда, қолданушы құратын функция жариялануының негізгі функция main() -ге қатысты орналасуын келесі түрде көрсетуге болады:

```
Функция_прототипі;  //прототипті көрсету
    void main()  //негізгі функция
{
  функцияны шақыру;
}
функцияны жариялау;
```

мысалы, екі санның үлкенін табатын max(**int** x, **int** y) функциясын прототипі арқылы жариялау төменде көрсетілген:

```
int max(int x, int y) ;  // max функциясының прототипін көрсету
int main()  //негізгі функция
   {
     функцияны шақыру;
   }
int max(int x, int y)  // max функциясының өзін жариялау
   {
     int z;   //айнымалыны жариялау
```

```
  if (x>y)   //шарт операторы
    z=x;
  else
    z=y;
 return(z);
        }
```

Функция прототипіндегі параметрлердің немесе айнымалылардың типін көрсету міндетті болып есептеледі, ал айнымалының аттарын жазбай кете беруге болады, мысалы:

```
int max(int , int ) ;  // max функциясының прототипін көрсету
```

– екінші жағдайда программада функцияның прототипі берілмейтін болады, сондықтан функция бірден толығымен жарияланады, яғни мұнда функцияның прототипі болмайды да, функцияның тақырыбы және денесі толығымен *функция шақырылғанға* ***дейін*** анықталып кетеді. Бұл жағдайдағы қолданушы құратын функция жариялануының, негізгі функция main()-ге қатысты орналасуын келесі түрде береді:

```
функцияны жариялау;
    void main()
{
     функцияны шақыру;
}
```

мысалы, жоғарыда келтірілген екі санның үлкенін табатын max(int x, int y) функциясын прототипсіз жариялау келесі түрде болады:

```
int max(int x, int y)  // max функциясын жариялау
  { int z; //айнымалыны жариялау
    if (x>y) //шарт операторы
      z=x;
    else
      z=y;
   return(z);
```

```
        }
int main()  //негізгі функция
   {
      функцияны шақыру;
   }
```

4.2. Функцияны шақыру

Жарияланған функцияны программада пайдалану үшін оны аты арқылы шақырады. Жарияланған функцияның программадағы шақырылуын келесі түрде жазады:

```
функция_аты (парам1,  парам2, ...,  парамN) ;
```

немесе

```
функция_аты () ;  // параметрсіз функцияның шақырылуы
```

Жарияланған функцияның тақырыбындағы параметрлерді – шартты түрде алынған *«формальды» параметрлер* деп, ал шақырылған функциядағы параметрлерді жұмысшы немесе *«нақты (фактические)» параметрлер* деп қарастырады. Функция шақырылғанда ең алдымен формальды параметрлер мен нақты параметрлердің типтерінің сәйкестігі тексеріледі, егер сәйкес келмесе қате туралы хабарланады.

Көпшілік жағдайларда функцияның шақырылуындағы параметрлердің саны және типтері функцияның тақырыбындағы немесе прототипіндегі параметрлер санымен және олардың типтерімен сәйкес болып келеді. Бірақ кейбір функцияларда (мысалы, printf) параметрлердің саны өзгере береді, типтері де өзгере береді. Мұндай функциялардың жариялануындағы немесе прототиптеріндегі параметрлердің орнына «...» көп нүкте белгісі қойылады, мысалы, прототипі:

```
void MyFunc(...) ;
```

болатын функцияның шақырылуында оның параметрлер саны да, типі де өзгере беруі мүмкін.

Функцияны шақыру барысында қайтарылатын тип көрсетілмейді. Мысалы, барлық программаларда дерлік қолданылып жүрген getch(); – бұл да стандарт функциялардың бірінің шақырылуы болып табылады.

4.1-жаттығу. Берілген екі бүтін санның үлкенін табатын *max* функциясын және екі санның кішісін табатын *min* функциясын жариялап және оларды шақырып пайдаланатын программаны құру қарастырылсын.

Функцияның прототипін жариялау тәсілі бойынша құру:

```
#include <iostream.h>
 #include<conio.h>
 int max(int x, int y);   // max функциясының прототипін жариялау
 int min (int x, int y);  // min функциясының прототипін жариялау
int main()  //негізгі функцияның басталуы
{
int a=4,b=-3;
cout<<"\n max="<<max(a,b);  //max функциясын шақыру
cout<<"\n min="<<min(a,b);  //min функциясын шақыру
    getch();
 }   //негізгі функцияның соңы
  int max(int x, int y)  // max функциясының өзін жариялау басталды
    {
    int z;
    if (x>y)
    z=x;
    else
   z=y;
   return(z);
    } //max функциясын жариялаудың соңы
 int min (int x, int y)  // min функциясының өзін жариялау  басталды
 { int k;
    if (x<y)
    k=x;
```

```
 else
 k=y;
return(k);
} // min функциясын жариялаудың соңы
```

Программаның нәтижесі келесі түрде болады:

```
max=4
min=-3
```

Бұл жаттығуда max (**int** x, **int** y); және min (**int** x, **int** y); функцияларының прототиптері алдын ала жарияланғандықтан, функциялардың өздерінің толық жарияланулары негізгі main() функциясынан кейін жасалды.

Прототипсіз, функцияның өзін алдымен жариялауды пайдаланып шығарудың программасы төмендегідей болады:

```
#include <iostream.h>
#include<conio.h>
     int max(int x, int y) // max- функциясын жариялау басталды
   {
   int z;
   if (x>y)
   z=x;
   else
  z=y;
  return(z);
   } // max функциясын жариялау аяқталды

    int min (int x, int y)  // min функциясын жариялау басталды
{
  int k;
   if (x<y)
   k=x;
   else
```

```
  k=y ;
  return(k);
 } // min функциясын жариялау аяқталды
    void main() // негізгі функция басталды
{
  int a=4,b=-3;
  cout<<"\n max="<<max(a,b); // max функциясын шақыру
  cout<<"\n min="<<min(a,b); // min функциясын шақыру
  getch();
} // негізгі функция аяқталды
```

Бұл программаның нәтижесі келесі түрде болады:

```
max=4
min=-3
```

Мұнда функциялар прототипсіз, бірақ негізгі main () функциясынан бұрын жарияланып тұр. Ескерілетін нәрсе, С және С++ программалау тілдерінің ертеректегі нұсқаларында функция немесе прототип, оның «ең алғашқы шақырылуына» дейін жариялануы тиіс деген қағида сақталады, ал қазіргі қолданылып жүрген программалау жүйелерінде, мысалы С++Builder 6 программалау ортасында да бұл қағида сақталмайды, яғни функцияны жариялау main () функциясынан бұрын да, кейін де жүргізіле береді, сондықтан бір программаның ішінде прототипті жариялауды пайдалану міндетті болмай қалды, бұл прототипті пайдалану арнайы кітапханаларда сақталатын функциялар прототиптерін тақырыптық файлдарда (*.h файлдар) сақтауда әлі де болса қолданылады.

4.3. Функция параметрлерінің берілуі

Параметрлер механизмін жалпы екі функция арасындағы (жарияланған және шақырылған) дерек алмасу ретінде қараған дұрыс болады. Функцияны программада шақырғанда, алдымен оның шақыры-

луындағы нақты параметрлердің орнында тұрған өрнектердің мәндері есептеледі, сонан соң барып жедел жадының *стек* деп аталатын арнайы бөлімінде функция жариялануындағы формальды параметрлер үшін олардың типтеріне сәйкес орындар бөлінеді, сол орындарға нақты параметрлердегі мәндер жіберіледі. Осыған байланысты функция параметрлерінің мәндерін беру екі түрлі жолмен жасалады:

– параметрлердің мәндер бойынша берілуі;

– параметрлердің адреске сілтеме бойынша берілуі.

4.3.1. Функция параметрлерінің мәндер бойынша берілуі

Параметрлердің мәндер бойынша берілуінде функция шақырылуындағы нақты параметрлердің орнында тұрған айнымалылардың көшірмелері жасалады. Функцияны шақыру кезінде стектегі формальды параметрлердің орнына мәндері анықталған айнымалылардың көшірмелері барады, яғни функция нақты параметрлердің көшірмелерімен ғана жұмыс жасай алатын болады, ал нақты параметрлердің орнында тұрған айнымалылардың мәндерін (немесе оригиналдарын) функция өзгерте алмайды.

4.2-жаттығу. Келесі программа функцияның нақты параметрлері ретінде берілетін x=1 және y='+' айнымалыларының мәндерін функцияның өзгерте алмайтынын көрсете алады:

```
#include <iostream.h>
#include <conio.h>
int ozgermeu(int i, char c);  //прототипті жариялау
int main()
{
int x=1;  // x-айнымалының бастапқы мәні
char y='+';  // y-айнымалының бастапқы мәні
cout<<"\n bastapki manderi: \n x="<<x<<";  y='"<<y<<"';";
ozgermeu (x,y);  // x, y-айнымалылардың мәндерін өзгертуші функцияны шақыру
cout<<"\n\n funcion- nan keingi mander : \n x="<<x<<";  y='"<<y<<"';";
getch();
}
```

```
int ozgermeu (int i, char c) //функцияны жариялау басталды
{
i++; //i-параметрдің мәніне 1-ді қосып өзгерту
c='-'; //-параметрдің мәніне '-' қойып өзгерту
} //функцияны жариялау аяқталды
```

Бұл программа орындалғанда келесі нәтиже алынады:

```
bastapki manderi:
x=1; y='+';

funcion- nan keingi mander:
x=1; y='+';
```

Мұндағы нәтижеге қарағанда, программадағы ozgermeu (x,y); функциясының шақырылуы бойынша оның жариялануындағы **int** i және **char** c формальды параметрлерінің орнына сәйкесінше нақты параметрлердің орнында тұрған **int** x және **char** y айнымалыларының мәндерінің көшірмелері баратын болады, яғни функция көшірмелерді ғана өзгерткендіктен, **int** x және **char** y айнымалыларының өздері өзгеріссіз қалатын болады.

4.3.2. Функция параметрлерінің сілтеме бойынша берілуі

Параметрлерінің адрестер бойынша берілуінде стектегі формальды параметрлердің орнына нақты параметрлер сақталған ұяшықтардың адрестері жіберілетін болады, ал адрес белгілі болатын болса, функцияның нақты парметрлердің мәндерін өзгертуге мүмкіндігі болатыны белгілі. Функция параметрлерін адрестері бойынша беруде көрсеткіштер мен сілтемелер қолданылады (3-тарау, 3.5-3.6-тақырыптарды қараңыз).

Әдетте сілтеме айнымалының қосалқы есімі немесе псевдонимі болып табылады. Сілтеме бойынша мәндерді параметрлерге беру барысында негізінен ол мәндер сақталып тұрған жады ұяшықтарының адрестері беріледі, мұнда көшірмелер жасалмайды. Функция параметрге берілетін мәндермен тікелей жұмыс жасайды, яғни функцияның

оригиналды өзгертуге мүмкіндігі бар болады. Егер параметрлер адрес сақталатын көрсеткіш түрінде немесе адреске сілтеме бойынша берілетін болса, онда функцияның тақырыбындығы, прототипіндегі параметрлердің алдына сәйкесінше көрсеткішті білдіретін * белгісін немесе сілтеменің & белгісін қою керек, мысалы,

```
int ozgeru( int *i, char &c); //функцияның прототипі, мұнда *i – көрсеткіш,
                              //&c – адреске сілтеме түрінде анықталған параметрлер
```

Функцияны шақыру барысында формальды көрсеткіш параметрге сәйкес нақты параметрдің алдына & "адресті алу" амалы қойылады, ал сілтеме параметрге сәйкес нақты параметрдің алдына бұл белгі қойылмайды, мысалы:

```
ozgeru(&x,y); // функцияны программада шақыру
```

мұндағы **int** *i көрсеткіш түрінде анықталған параметрдің орнына баратын х-нақты параметрдің алдына адресті алу амалының & белгісі қойылды.

4.3-жаттығу. Бұл программа функция параметрлерін адрестер бойынша беруді көрсеткіш және сілтеме арқылы орындауды көрсетеді:

```
#include <iostream.h>
#include <conio.h>
int ozgeru( int *i, char &c); //функцияның прототипі, мұнда *i – көрсеткіш,
                              //& c – адреске сілтеме түрінде анықталған параметрлер
int main()
{
int x=1; // х-айнымалының бастапқы мәні
char y='+'; // у-айнымалының бастапқы мәні
cout<<"\n bastapki manderi: \n x="<<x<<"; y='"<<y<<"';";
ozgeru(&x,y); // функцияны шақыру
cout<<"\n\n funcion- nan keingi mander : \n x="<<x<<"; y='"<<y<<"';";
getch();
}
```

```
int ozgeru (int *i, char &c) // функцияны жариялау басталды
{
(*i)++; // i көрсеткіштегі адрес бойынша мәнді алып, оған 1-ді қосу
c='-'; //c сілтеме бойынша берілген адрестегі мәнді '-'ке өзгерту
} // функцияны жариялау аяқталды
```

Бұл программа орындалуының нәтижесі төмендегідей болады:

```
bastapki  manderi:
x=1; y='+';

funcion- nan  keingi  mander:
x=2; y='-';
```

Бұл шыққан нәтиже талданатын болса, мұнда x-нақты параметрдің мәні x=2 болып өзгереді, себебі функция шақырылғанда, **int*** i көрсеткіш түрінде анықталған параметрге x-тің мәні сақталған жады ұяшығының адресі беріледі. Адресті беру үшін функцияның шақырылуында x-тің алдына &- «адресті алу» амалы жазылды немесе &x өрнегі қолданылды. Осы берілген адрес бойынша ұяшықтағы мәнді алу үшін функция денесінде (*i)++; берілген адрестен мәнді алу (операция разыменования) амалын пайдалануға тура келді. Сол сияқты y нақты параметрдің де мәні y='-'; болып өзгерді, себебі адреске сілтеме түрінде анықталған **int** &c параметрге, y-тің мәні сақталған ұяшықтың адресі сілтеме арқылы беріледі.

Параметрлерді адрес бойынша беруде, көрсеткішке қарағанда, сілтемені пайдалану тиімдірек болып келеді. Себебі, параметрді адреске сілтеме бойынша бергенде, жарияланған функциядағы формальды параметрге – функцияның шақырылуындағы нақты параметрдің, мысалы, y-тің адресі беріледі, ал адресі берілген ұяшықтан мәнді алу (разыменования, разадресация) функцияның ішінде өздігінен, анық емес (неявно) түрде жүреді, сондықтан сілтемені пайдалану программаның түсініктілігін арттырады және программисті, көрсеткіштерге қолданылатын &- «адресті алу», * – «адрестен мәнді алу» секілді амалдарды пайдаланудан босатады. Сілтемені пайдаланудың тағы бір тиімді жағы,

мұнда параметрлердің көшірмелерін жасаудың қажеті жоқ, бұл, әсіресе үлкен көлемдегі құрылымдық деректерді немесе параметрлерді беруде көп қолданылады.

4.3.3. Функция параметрлерінің массивтер түрінде берілуі

С++ программалау тілінде функцияның параметрі ретінде кез келген типті бере беруге болады, бірақ массив түріндегі және функция түріндегі параметрлерді беру үшін көрсеткіштерді пайдалануға тура келеді. Функцияның параметрі ретінде массив берілетін жағдайларда, параметр ретінде массивтің ең бірінші элементі үшін анықталған көрсеткіш қолданылады (С++-те массивтің аты оның ең алғашқы элементінің адресі сақталған көрсеткіш ретінде анықталады), демек бұл массив-параметрді беру тек қана адрес арқылы ғана мүмкін екенін көрсетеді. Сондай-ақ, функцияға массив-параметрді бергенде, массив элементтерінің саны жоғалып кетуі мүмкін, сондықтан массивтің ұзындығы қасында, бірақ мәні «өзгерістен қорғалған» бөлек параметрлер арқылы беріледі.

Жоғарыда айтылып кеткендей, егер параметрдің мәні адрес арқылы берілетін болса, онда функцияның сол берілген адрестегі немесе жады ұяшығындағы нақты мәнді «оригиналды» өзгерте алатыны белгілі болды. Кейбір жағдайларда бұл нақты мәндерді өзгертпей, қорғап қалу қажет болады. Ол үшін функция прототипіндегі және функцияның тақырыбындағы *«өзгерістен қорғалатын»* параметрлердің алдына **const** модификаторын жазады. Сондықтан функцияға массив-параметрді бергенде массив элементтерінің саны немесе ұзындығы өзгермеу үшін оның алдына **const** тұрақтылық модификаторын қойып жазады.

4.4-жаттығу. Программа массив-параметрмен жұмыс жасайтын функциялардың қызметін көрсетеді (Массивтің оң элементтерін баспаға шығаратын функция – *MasKoru*, ал жұп элементтерін көрсету мен олардың қосындысын есептеу *summa* функциясы түрінде анықталған).

```
#include<iostream.h>
#include <conio.h>
```

```
#include <stdlib.h> // random функциясы үшін қосылды
void  summa( int *mas,const int r); // summa функциясының прототипін
//жариялау, мұндағы *mas массивпараметр, r массив – ұзындығы
  void MasKoru(int*mas,const int r);//MasKoru функциясының прототипін
  //жариялау, мұндағы *mas массив – параметр, r – массив ұзындығы
 const r =10; //r-массив ұзындығын тұрақты ретінде жариялау
 int main()
{
 int MyMas[r]; // массивті жариялау
 int*p=MyMas; //нақты массив-параметр болатын p -көрсеткішті жариялау
                //және оған массивтің алғашқы элементінің адресін беру
     for (int i=0;i<r;i++)
      MyMas[i]=random(100)%27-3;   //массивті құру
       MasKoru(p,r); //MasKoru функциясын шақыру
      summa(p,r);  ////summa  функциясын шақыру
     getch();
}
 void summa(int *mas,const int r)  // summa  функциясын жариялау басталды
  {
  int s=0;
  cout<<"\n\n\n  gup elementterdi koru :";
   for (int i=0;i<r;i++)
     if(mas[i]%2==0)  //массив элементінің жұп сан болатынын тексеру
      {
      cout<<"  "<<mas[i]; //массивтің  жұп элементтерін баспаға шығару
     s=s+mas[i]; жұп элементтердің қосындысын есептеу
      }
      cout<<"\n\n  gup elementter kosindisi: ";
      cout<<"summa="<<s; // жұп элементтер қосындысын баспаға шығару
     } // summa  функциясын жариялау аяқталды
 void MasKoru(int *mas,const int r) //MasKoru функциясын жариялау
                                    // басталды
  {  cout<<"\n \n barlik  elementterdi koru: ";
     for (int i=0;i<r;i++)
     cout<<"  "<<mas[i];
  } //MasKoru функциясын жариялау аяқталды
```

Программа нәтижесі келесі түрде болады:

```
barlik elementterdi koru:   19 1 2 14 9 -1 -1 4 2 4
gup elementterdi koru :     2 14 4 2 4
gup elementter kosindisi:   summa=26
```

Программа талданатын болса, мұнда негізгі программада жарияланған **int** MyMas[r]; массиві MasKoru(**int***mas,**const int** r); және summa(**int** *mas,**const int** r); функцияларындағы **int***mas формальды параметрінің орнына баратын нақты массив-параметр болып табылады, бірақ ол массив болғандықтан, аталған функцияларға параметр ретінде массивтің өзі емес, оның ең алғашқы элементінің адресін көрсететін **int***p=MyMas; түрінде анықталған **p** көрсеткіш беріледі. Сондай-ақ, массив өлшемін көрсететін **int** r – тұрақтыны өзгерістен қорғау үшін оның алдына **const** модификаторының қойылғанын көруге болады.

4.4. Функцияның көру аймақтары

Программалау тілдерінде бірнеше функциялардың жиынтығынан құралатын программаның әр түрлі бөліктерінде (мысалы, функциялардың ішінде немесе одан тысқары, негізгі программада немесе одан да тысқары) жарияланған айнымалылардың сол программадағы қызметіне қатысты анықталған ережелері болады. Осы ережелер бір функцияда жарияланған айнымалыларды екінші бір функциядағы программалық кодтың пайдалана алу-алмау мүмкіндігін немесе *функцияның көру аймағын* анықтайды. Осы ережелер бойынша C++ -те функцияның көру аймағын екі түрге бөледі: *жергілікті* (локалдық) және *кең ауқымды* (глобалдық). Айнымалыларды жариялау жергілікті немесе кең аукымды аймақтардың кез келгенінде жасала береді. Функциялардың көру аймақтарына сәйкес ондағы жарияланған айнымалылардың «*өмір сүру*» немесе «*бар болу уақыты*» анықталады.

4.4.1. Жергілікті көру аймағы

Жергілікті көру аймағы блоктың, яғни { }- фигуралы жақшаларға алынған бөліктің көмегімен жасалады (2.6.2-тақырыпты қараңыз). Ереже бойынша, программада құрылған әрбір осындай блокка сәйкес оның өзінің көру аймағы анықталады. Мұндай блоктардың ең көп қолданылатын түрі бұл функциялар екені белгілі. Функцияның денесіндегі немесе оның фигуралы жақшаларға алынған бөлігіндегі коды оның тек қана өзінікі болып саналады, басқа ешбір функцияның ол кодта анықталған айнымалыларды немесе т.б. деректерді пайдалануға құқығы жоқ. Бұл ереже функцияның, программаның басқа бөліктеріне әсер етпеуін және керісінше басқа бөліктердің ол функцияға әсері тимеуін, яғни функция аймағындағы деректердің барынша қорғалғанын білдіреді. Мысалы, goto операторын пайдаланып, басқа бір функциядағы операторға көшу мүмкін емес, себебі ол функция goto-ны өз аймағына кіргізбейді. Функциялардың көру аймақтары әр түрлі болғандықтан, бір функция аймағында анықталған код пен деректер, екінші бір функцияның коды мен деректеріне тәуелсіз болады. Осыған байланысты бір программада, бірақ әр түрлі функциялардың аймағында жарияланған аттары бірдей айнымалылар кездесе береді.

Блоктың ішінде немесе функцияның жергілікті көру аймағында жарияланған айнымалыны осы блокка қатысты алғандағы *жергілікті айнымалы (локальная переменная)* деп атайды. Жоғарыда айтылған ережеге сәйкес, бұл жергілікті айнымалыны функцияның өзінен басқа ешкімнің пайдалануы немесе өзгертуі мүмкін болмайды. Жергілікті айнымалылардың *өмір сүру уақыты* өздері жарияланған функцияның өмір сүру уақытымен бірдей. Бұл программа орындалып тұрса да, әзір өздері анықталған функция шақырылғанға дейін жергілікті айнымалылардың жадыда болмайтынын білдіреді. Жергілікті айнымалылар жадыда функция шақырылғанда ғана барып пайда болады да, функцияның орындалуы аяқталған бойда жадыдан жойылып кетеді, яғни айнымалылар автоматты түрде құрылады және автоматты түрде жойылады. Сонымен, ереже бойынша жергілікті айнымалы блоктың ішінде ғана анықталады және сол блок орындалғанда пайда болып, блок біткесін ол да жойылып кететін болды. Бірақ блок ішінде **static** спецификаторын қолданып жарияланған жергілікті айнымалы блок біткесін

де жойылмайтын болады, яғни ол программаның барлық орындалуы кезінде жадыда өз мәнін сақтап, бар болып тұрады. Программалауда жергілікті айнымалыны айқын түрде инициализациалаған дұрыс болады.

4.4-жаттығу. Келесі программа a[m][n] матрица элементтерін енгізуді – matrEngizu(a,m,n); ал матрицаны баспаға шығарып көрсетуді matrKoru(a,m,n); функцияларының көмегімен жүзеге асыратын болады, мұнда матрица элементтерімен жұмыс жасау үшін қолданылатын циклдердің параметрлері **int** i,j; *жергілікті айнымалылар түрінде* жарияланады:

```
#include<conio.h>
#include<iostream.h>
const int m=3, n=3; //матрица өлшемдерін кең ауқымды тұрақтылар түрінде
                    //жариялау
int main()  //негізгі программа басталды
{
int a[m][n]; //матрицаның өзін жариялау
 void matrEngizu(int a[m][n],const int m,const int n); //matrEngizu функциясының
                                                       // прототипін жариялау
void matrKoru(int a[m][n],const int m,const int n); //matrKoru функциясының
                                                    // прототипін жариялау
matrEngizu(a,m,n); // matrEngizu функциясын шақырып, матрица
                   // элементтерін енгізу
matrKoru (a,m,n); // matrKoru функциясын шақырып, элементтерін баспаға
                  //шығару
getch();
} //негізгі программа аяқталды

void matrEngizu (int a[m][n],const int m,const int n)  //matrEngizu функциясын
                                                       // жариялау басталды
{
int i,j;  //цикл параметрлерін жергілікті айнымалы түрінде жариялау
cout<< "\n   matrizani engizu: \n\n ";
for(i=0;i<m;i++) // матрица элементтерін енгізу басталды
for(j=0;j<n;j++)
```

```
  {cout<< "a[ "<<i<< ", "<<j<< "]= ";
   cin>>a[i][j];
  } // матрица элементтерін енгізу аяқталды
} //matrEngizu функциясын жариялау аяқталды
void matrKoru ( int a[m][n],const int m,const int n) // matrKoru функциясын
                                                    // жариялау басталды
{
int i,j; //цикл параметрлерін жергілікті айнымалы түрінде жариялау
 cout<< " \n  matrizani baspaga shigaru :\n ";
 for(i=0;i<m;i++)   //матрицаны баспаға шығару басталды
       { cout<< "\n\n ";
        for(j=0;j<n;j++)
        cout<< "\t "<<a[i][j];
       } // матрицаны баспаға шығару аяқталды
} // matrKoru функциясын жариялау аяқталды
```

Программа орындалуының нәтижесі келесі түрде болады:

```
matrizani engizu:

 a[0,0]=1
a[0,1]=2
a[0,2]=3
a[1,0]=4
a[1,1]=5
a[1,2]=6
a[2,0]=7
a[2,1]=8
a[2,2]=9

  matrizani baspaga shigaru:
     1    2    3
     4    5    6
     7    8    9
```

Бұл программа орындалғанда алдымен бірінші шақырылатын matrEngizu(a,m,n); функциясы орындалады, яғни матрицаның эле-

менттері пернетақтадан енгізілетін болады, сонан соң барып екінші тұрған matrKoru(a,m,n); функциясы алдыңғы функцияның көмегімен енгізілген матрицаны дәстүрлі түрде баспаға шығаратын болады. Мұнда назар аударылатын мәселе, ол – matrEngizu(a,m,n); функциясының көру аймағында жарияланған, цикл параметрлерінің қызметін атқарып тұрған **int** i, j жергілікті айнымалылары. Тура осындай екі айнымалы matrKoru(a,m,n); функциясында тағы да қайталанып жарияланады, себебі алдыңғы matrEngizu(a,m,n); функциясының аймағында жарияланған жергілікті **int** i, j айнымалылары сол функция орындалған кезде ғана жадыда бар болып өмір сүре алады, ал функцияның жұмысы аяқталған соң олар да жадыдан жойылып кетеді.

Программалауда, деректер жергілікті айнымалылар түрінде жариялану арқылы сыртқы өзгертулерден және де рұқсатсыз пайдаланулардан қорғалады немесе жергілікті көру аймағының ережелері объектіге бағдарланған программалаудағы *инкапсуляция* қағидасының негізі болып табылады.

4.4.2. Кең ауқымды көру аймағы

Программаға бөлінген жады облысының функциялардан тысқары бос қалған жерінің барлығы кең ауқымды көру аймағына беріледі. Функциялардың сыртында кең ауқымды көру аймағында анықталған айнымалыларды *кең ауқымды айнымалылар* деп атайды. Кең ауқымды айнымалылар программадағы барлық функцияларда да көрінеді, сонымен қатар басқа файлдарда да көрінеді. Программалауда егер бір айнымалы бірнеше функциялардың жұмысына қатар қажет болатын болса, онда оны кең ауқымды айнымалы ретінде бір-ақ рет жариялайды. Әдетте, программада, кең ауқымды айнымалыларды main()-ға дейін, барлық функциялардан бұрын жариялап кетеді. Егер кең ауқымды айнымалылар айқын түрде инициализациаланбаса, онда компилятор оның мәнін «ноль» деп алады.

4.6-жаттығу. Келесі программа a[m][n] матрица элементтерін енгізуді – matrEngizu(a,m,n); ал матрицаны баспаға шығарып көрсетуді – matrKoru(a,m,n); функцияларының көмегімен жүзеге асыратын болады, мұнда матрица элементтерімен жұмыс жасау үшін қолданылатын

циклдердің параметрлері **int** i,j; *кең ауқымды айнымалылар* түрінде жарияланады:

```
#include<conio.h>
#include<iostream.h>
const int m=3, n=3; //матрица өлшемдерін кең ауқымды тұрақтылар түрінде жариялау
int i,j;  //цикл параметрлерін кең ауқымды  айнымалы түрінде жариялау
int main()  //негізгі программа басталды
{
int a[m][n]; //матрицаның өзін жариялау
 void matrEngizu(int a[m][n],const int m,const int n); //matrEngizu функциясының
                                                        //прототипін  жариялау
void matrKoru(int a[m][n],const int m,const int n); //matrKoru  функциясының
                                                     // прототипін  жариялау
matrEngizu(a,m,n);  // matrEngizu  функциясын шақырып, матрица элементтерін енгізу
matrKoru (a,m,n); // matrKoru функциясын шақырып, элементтерін баспаға шығару
getch();
} //негізгі программа аяқталды
void matrEngizu (int a[m][n],const int m,const int n)  //matrEngizu  функциясын
                                                        //жариялау  басталды
{
cout<<"\n   matrizani engizu: \n\n";
for(i=0;i<m;i++) // матрица элементтерін енгізу  басталды
for(j=0;j<n;j++)
  {cout<<"a["<<i<<","<<j<<"]=";
   cin>>a[i][j];
  } // матрица элементтерін енгізу аяқталды
} //matrEngizu  функциясын жариялау  аяқталды

 void matrKoru ( (int a[m][n],const int m,const int n) // matrKoru  функциясын
                                                       // жариялау  басталды
{
 cout<<"\n   matrizani baspaga shigaru :\n";
 for(i=0;i<m;i++) //матрицаны баспаға шығару басталды
       { cout<<"\n\n";
         for(j=0;j<n;j++)
```

```
        cout<<"\t"<<a[i][j];
      } // матрицаны баспаға шығару аяқталды
} // matrKoru функциясын жариялау аяқталды
```

Программа орындалуының нәтижесі келесі түрде болады:

```
matrizani engizu:
a[0,0]=1
a[0,1]=2
a[0,2]=3
a[1,0]=4
a[1,1]=5
a[1,2]=6
a[2,0]=7
a[2,1]=8
a[2,2]=9

matrizani baspaga shigaru:
   1   2   3
   4   5   6
   7   8   9
```

Бұл программада цикл параметрлерінің қызметін атқарып тұрған **int** i, j айнымалылары, барлық функциялардан, тіпті main()-нан да тысқары, кең ауқымды айнымалылар ретінде бір-ақ рет жарияланып тұр. Кең ауқымды айнымалылардың өмір сүру уақыты программаның орындалу уақытымен бірдей болғандықтан, бұл **int** i, j айнымалылары программаның орындалып тұрған кезінде бастан-аяқ жадыда орын алып тұрады, сондықтан оны кез келген функция, мысалы, matrEngizu(a,m,n); немесе matrKoru(a,m,n); функциялары кез келген уақытта пайдалана алады.

Кең ауқымды айнымалылар статикалық жады класына жатады (2.6.2-тақырыпты қараңыз), яғни олардың өмір сүру уақыты программаның өзімен бірдей болады немесе олар программаның орындалу барысында әр уақытта бар болады. Кең ауқымды айнымалылар өздері жарияланған орыннан бастап файлдың соңына дейін көрінеді. Статикалық кең аумақты айнымалылар бірнеше файлдардан тұратын

көп файлды программаларда да қолданылады. Кейбір жағдайларда, кең ауқымды айнымалы мен негізгі орындалатын программа немесе main() функциясы екеуі екі басқа файлда сақталған болуы мүмкін. Мұндай жағдайда, басқа файлдағы кең ауқымды айнымалыны негізгі программаға кірістіру үшін **extern** спецификаторын қолданады.

4.7-жаттығу. f=a*x^2+b*x+c; өрнегінің мәнін есептейтін esepteu() функциясын құру керек. Мұнда бүтін типті кең ауқымды a, b, c, x, f айнымалылары және esepteu() функциясының өзі main()-нан басқа бөлек файлда жарияланып сақталуы керек.

Бұл есепті шешу үшін жаңадан құрылатын жобада, мысалы, Project1 жобасында негізгі программа мәтіні жазылатын Unit1.cpp файлынан басқа, esepteu() функциясының мәтін жазылатын тағы да бір Unit2.cpp файлы құрылуы керек. Ол үшін ашық тұрған жобада, File -> New -> Unit командалары орындалады, нәтижесінде Unit2.cpp терезесі пайда болады, осы терезедегі файл мәтінінде кең ауқымды a,b,c,x,f айнымалылары **extern** арқылы жарияланып, esepteu() функциясының мәтіні теріледі, бұл Unit2.cpp файлын esepteu деген атпен сақтауға да болады, Unit2.cpp немесе esepteu.cpp файлының мәтіні:

```
#include "esepteu.h"
extern int a,b,c,x,f; //кең ауқымды айнымалыларды extern арқылы жариялау
esepteu() // функцияның өзін жариялау
{
f=a*x*x+ b*x+c ;
return(f);
}
```

Программаның немесе Unit1.cpp файлында сақталатын main()-нің мәтіні:

```
#include <iostream.h>
#include <conio.h>
#include "esepteu. cpp "// esepteu.cpp файлын программаға кірістіру
int a=5,b=7,c=10, x, f; // кең ауқымды a,b,c,x,f айнымалыларын жариялау
int main()
{ cin>>x;
```

```
f=esepteu(); // esepteu(); функциясын шақыру
cout<<"\n f="<<f;
getch();
}
```

Программа орындалуының нәтижесі төмендегідей болады:

```
 1
f = 22
```

Программалауда кең ауқымды айнымалыларды аса қажет болған жағдайларда ғана қолданады, мұның бірнеше себептері бар: бірінші ден кең ауқымды айнымалылар программаның орындалуы кезінде бастан-аяқ жадыдан орын алып тұрады, екіншіден программаларды модификациялаған кезде кең ауқымды айнымалылардың мәндері қорғалмағандықтан, кездейсоқ өзгерістерге ұшырауы мүмкін, ал бұл өз кезегінде күтпеген қателерге әкеп соқтырады, сондықтан программа көлемі неғұрлым үлкен болған сайын, ондағы кең ауқымды айнымалылардың саны соғұрлым аз болғаны дұрыс.

4.5. Функцияны қайыра жүктеу

Функцияларды қайыра жүктеу бұл қажеттіліктен туындаған нәрсе, мысалы, бастапқы С тілінде берілген санның модулін табу үшін abs(int x), labs(long x) және fabs(float x) деп аталатын үш функция қолданылды. Бұл үш функцияның орындайтын қызметі бірдей, айырмашылық тек қана параметрлерінің типтерін де ғана болып тұр. Олардың орындайтын амалдары біреу болғанына қарамастан, олардың әрқайсысына неке атау (уникальная имя) беріліп тұр, демек программист олардың үшеуін де жатқа білуі керек деген сөз, бұдан басқа мұндай нәрселер өз кезегінде программа құрылымының күрделенуіне және түсініктілігінің төмендеуіне де әкеліп соқтырады. Осындай себептерге байланысты программалау тілдерінде орындайтын әрекеттері бірдей немесе ұқсас болып келетін, тек қана параметрлерінің типтері мен сандары ғана өзгеше болатын функцияларды бір атаумен ғана беретін, «функ-

цияны қайыра жүктеу» механизмі қарастырылған. Сонымен, қайыра жүктелген функциялар атаулары бірдей, бірақ параметрлерінің типтері және сандарына қарай әр түрлі нәтижелер беретін әрекеттерді орындайтын функциялар.

4.8-жаттығу. Программада атаулары бірдей, бірақ параметрлерінің саны әр түрлі функцияларды қайыра жүктеу көрсетіледі.

```
# include <iostream.h>
# include <conio.h>
void perchar ( );  // параметрі жоқ функция прототипі
void perchar (char);   // бір ғана параметрі бар функция прототипі
void perchar (char, int);   // екі параметрі бар функция прототипі
int main()
{
  cout <<"\n\n";
  perchar ( );  // параметрі жоқ функцияны шақыру
  cout <<"\n\n";
  perchar ( '=' ); // бір ғана параметрі бар функцияны шақыру

cout <<"\n\n";
perchar ('+', 30); // екі параметрі бар функцияны шақыру
  getch();
}
void perchar ( )  // параметрі жоқ функцияны жариялау
{ for (int j=0; j<45; j++)
  cout << '*';   // функция «*» символын 45 рет басып шығарады
}
void perchar (char ch)  // бір  параметрі бар функцияны жариялау
{ for (int j=0; j<45; j++)
 cout<<ch;     //функция ch параметрімен берілген символды 45 рет шығарады
}
void perchar (char ch, int n) //екі параметрі бар функцияны жариялау
{ for (int j=0; j<n; j++)
 cout<<ch;    //функция ch параметрімен берілген символды n  рет шығарады
}
```

Программа нәтижесінде экранға келесі ақпарат шығарылады:

```
****************************************
========================================
++++++++++++++++++++++++++++++
```

Бұл программада perchar ( );, perchar (**char**); және perchar (**char**, **int**); деп аталатын үш функцияның қызметі көрсетілді. Бұл функциялардың барлығының атаулары бірдей екені көрініп тұр, сондай-ақ олардың әрқайсысы өз жариялануына сәйкес әр түрлі қызметтер орындауы керек. Бірақ, осыған қарамастан, аттары бірдей бұл үш функция бір-бірінен өзгеше үш түрлі нәтиже көрсетіп тұр, себебі бұл функциялардың параметрлерінің сандары және типтері әр түрлі болғандықтан ғана компилятор оларды ажырата алып тұр, яғни жарияланған әрбір функция өзіне тиесілі қызметті орындап тұр.

4.9-жаттығу. Программа бір атаумен берілген, бірақ параметрлерінің типтері әр түрлі, саны бірдей бірнеше функцияларды қайыра жүктеуді көрсетеді.

```
#include <iostream.h>
# include <conio.h>
struct tortburish  //tortburish құрылымдық типті жариялау
  { float a; //төртбұрыштың ені
    float b;  //төртбұрыштың ұзындығы
  };
       struct ushburish  // ushburish құрылымдық типті жариялау
  { float a; //үшбұрыштың табаны
    float h; //үшбұрыштың биіктігі
  };
      float audan (tortburish);  //төртбұрыш ауданын табатын
                                 // функция прототипі
     float audan (ushburish);    //үшбұрыш ауданын табатын
                                 //функция прототипі
```

```
   int main()
{
 tortburish  t={1.7,8.3}; //tortburish  типті t айнымалыны жариялау
ushburish   u={1.7, 8.3}; // ushburish типті u айнымалысын  жариялау
cout << " \n\n tortburish audani St="<<audan(t); // параметрі  tortburish
                                                  // типті t айнымалысы болатын
                                                  //audan(t); функцияны шақыру
cout << " \n\n ushburishtin audani Su="<<audan(u); // параметрі
                                        //ushburish  типті u  айнымалысы болатын
                                          //audan(u ); функцияны шақыру
 getch();
}
 float audan (tortburish k) //төртбұрыш ауданын табатын функцияны жариялау
{
  float s=k.a*k.b;
 return (s);
 }
float audan (ushburish k)  //үшбұрыш  ауданын табатын функцияны жариялау
{
float s=(k.a*k.h)/2;
return (s);
 }
```

мұнда функциялардың атаулары да, параметрлерінің саны да бірдей, бірақ ол параметрлердің типтері әр түрлі болып тұр, яғни бір параметрдің типі – tortburish, ал екінші параметрдің типі – ushburish, болғандықтан, компилятор оларды ажыратады және соған сәйкес функцияны іске қосатын болады.

Программалауда қайыра жүктелген функцияларды ұқсас әрекеттерді беру үшін қолданған дұрыс болады, оны өзара байланыспайтын есептер үшін қолдану программалау стилінің төмендігін көрсетеді, мысалы, sqr-ді бүтін санның квадратын табу үшін де және нақты саннан түбір табу үшін де қолдануға болатындай етіп жасауға болар еді, ал бұл дегенің тіпті де екі басқа мағынадағы есептер болып табылады, сондықтан мұндай жағдайларда функцияны қайыра жүктеу механизмін пайдаланудың мағынасы жоқ болады. Функцияларды қайыра жүк-

теу – бұл объектіге бағдарланған программалаудағы «бір интерфейс – бірнеше әдіс» немесе полиморфизм принципін жүзеге асырудың бір жолы ретінде қарастырылған.

4.6. Шаблон функциялар

Қайыра жүктеуді пайдаланып, ұқсас есептерді шешудің алгоритмдерін, атаулары бірдей бірнеше функциялар арқылы беруге болатыны белгілі болды. Бірақ мұнда есептердің алгоритмдері әр түрлі болғандықтан, оларға сәйкес функциялар да бірнешеу болды. Егер де есептерді шешудің алгоритмдері де бірдей болса, бірақ бұл алгоритм типтері әр түрлі болатын деректерге қолданылатын болса, мысалы, тікелей таңдау әдісі бойынша сұрыптау алгоритмін бүтін сандардан тұратын массив үшін де, нақты сандар массиві үшін де, әріптерден тұратын символдық массив үшін де қолдануға болатыны белгілі. Мұнда алгоритм біреу-ақ, бірақ оны пайдаланатын массивтердегі деректердің типі әр түрлі болып тұр. Бұл мәселені шешу үшін C++ те *функция шаблоны* немесе *шаблон функциялар* деп аталатын механизм қолданылады. Ол механизм бойынша әр түрлі типтегі деректер үшін қолданылатын алгоритм бірдей, яғни біреу болғандықтан, осы алгоритмге сәйкес функция, шаблон түрінде бір-ақ рет жазылады. Программист шаблон түрінде жазылатын функция параметрінің типін, **class** қызметші сөзін пайдаланып, шартты түрде (немесе формальды түрде) тағайындайды. **class** қызметші сөзімен анықталған тип, программист өзі құрған қолданушының типі болып есептеледі (6.2-ні қараңыз). Шартты түрде анықталған параметрдің орнына компиляциялау кезеңінде нақты параметрлер жіберіледі де, шаблон-функция сол нақты параметрге сәйкес орындалатын болады.

Функция шаблонының программадағы жазылуы келесі түрде болады:

```
template <class тип1, class тип2, ..., class типN>
        функция_аты (тип1 параметр1, ..., типN параметрN)
{ айнымалыларды_жариялау;
```

```
операторлар;
 return (қайтаратын мән);
}
```

мұндағы **template** функция шаблонын жариялаудың басталғанын білдіреді, ал **class**, функцияның формальды параметрлерінің типтерін шартты түрде анықтау үшін қолданылады, **тип1, тип2, ..., типN** дұрыс тағайындалған идентификаторлар болып табылады.

Функция шаблондарында **class** сөзімен анықталған типтерден басқа жай айнымалыларды да пайдалана беруге рұқсат етіледі, мысалы, келесі түрде:

template <class тип1, **class** тип2, **..., int** n**>**

4.10-жаттығу. Келесі программа тікелей таңдау әдісімен массивті өсу ретімен сұрыптауды шаблон-функция түрінде анықтайды және осы шаблонды бүтін сандар массиві, нақты сандар массиві және символдық массивтер үшін үш рет қолдануды көрсетеді.

```
#include <iostream.h>
 #include <conio.h>
template <class type1>  // шартты түрдегі type1 типін жариялау
void TikeSuriptau(type1 *b, int n) //шаблон-функцияны жариялау басталды
{
type1 buf;
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 ;
buf = b[i];
b[i]=b[imin];
b[imin]=buf;
}
for (int i = 0; i<n; i++)
```

```
cout<<" "<<b[i];
} //шаблон-функцияны жариялау аяқталды

int main()
{
const int n = 7;
int i, b[n]={7,-1,5,9,0,-6,3};
 cout<<"\n\n int – massivti suriptau :" ;
TikeSuriptau(b,n); //Шаблон-функцияны b[n] бүтін сандар массиві үшін шақыру
double a[]={0.22, 1.17, -0.08, 0.21, 42.5};
cout<<"\n\n  double – massivti suriptau :" ;
TikeSuriptau(a,5); //Шаблон-функцияны a[] нақты  сандар массиві үшін шақыру

char c[]="suriptau";
cout<<"\n\n  char – massivti suriptau :" ;
        TikeSuriptau(c,strlen(c));      //Шаблон-функцияны c[] символдық
                                        //массив үшін шақыру
getch();
}
```

Программа орындалуының нәтижесі келесі түрде болады:

```
 int - massivti suriptau :   -6  -1  0  3  5  7  9
double - massivti suriptau :  -0.08  0.21  0.22  1.17  42.5
char - massivti suriptau :  a  i  p  r  s  t  u  u
```

Программада жарияланған **void** TikeSuriptau(type1 *b, **int** n) шаблон-функциясында тікелей таңдау әдісін пайдаланып массивті сұрыптау алгоритмі анықталған (3.7.3.1-тақырыпты қараңыз). Шаблон-функцияның бірінші параметрінің типін (type1) қолданушы өзі анықтайды.

Әрі қарай қарастырылатын болса, онда программадағы **void** TikeSuriptau(type1 *b, **int** n) шаблон-функциясы типтері бүтін (**int**), нақты (**double**) және символдық тип (**char**) болатын әр түрлі үш түрлі массивті сұрыптау үшін пайдаланылды.

Шаблон-функцияны b[n] бүтін сандар массиві үшін бірінші рет шақырғанда, компилятор функцияның кодын бүтін тип үшін құратын болады, бұл процесті *шаблонды инстанциялandыру* (instantiation – инстанцирование) деп атайды. Инстанцияландыруда шаблон-функция параметрлерінің типін, компилятор, оның шақырылуындағы, мысалы, TikeSuriptau(b,n); шақырылудағы, b-ның типіне қарап, автоматты түрде анықтайды.

4.7. Рекурсивті функциялар

Рекурсивті функция деп өзін-өзі шақыра алатын функцияны айтады. Рекурсивті функцияның ең көп тараған классикалық мысалы ретінде факториалды есептеу функциясын айтуға болады. n санының факториалы деп әдетте 1-ден бастап n-ды қоса алғандағы барлық бүтін сандардың көбейтіндісін айтады және оны былай жазады:

$$n!=1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n .$$

немесе мұны келесі түрде де жазуға болады:

$$n!= n \cdot ((n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1) = n \cdot (n-1)!$$

демек n санының факториалын табу үшін n-ды (n-1) санының факториалына көбейтеді, сол сияқты (n-1) санының факториалы, оны (n-2)-нің факториалына көбейткенде шығады. Ал егер де n санының факториалын есептеу алгоритмін функция түрінде жазатын болса, онда бұл функцияда тағы да (n-1) санының факториалын есептейтін тура сондай функцияны шақыруға тура келеді, яғни функция өзін-өзі шақыратын болады. Шақырудың бұл тәсілін *рекурсия*, ал шақырылатын функцияны *рекурсивті функция* деп атайды.

4.11-жаттығу. Келесі программа перентақтадан енгізілген N бүтін санның факториалын есептейтін болады:

```
#include <iostream.h>
```

```
#include <conio.h>
long factorial(int n); //функцияның прототипін жариялау
int main()
{ int N;
cout<<"\n\n N>0 bolatin butin san engiziniz  N=";
cin>>N; // факториалы есептелетін бүтін санды пернетақтадан енгізу
 cout<<"\n\n  "<<N<<"!="<<factorial(N); //функцияның шақырылуы
 getch();
}
 long factorial(int n) //функцияны жариялау
 { cout<<"\n factorial("<<n<<") shakirildi\n"; // функцияның қандай сан
                                  //үшін шақырылғанын көріп отыру үшін жазылды
 if (n==0 || n==1) //ереже бойынша 0!=1 және  1!=1 болатынын көрсету
 return 1;
  return (n * factorial(n – 1)); //функцияның  n – 1 үшін өзін-өзі шақыруы
 }
```

Программа нәтижесін келесі түрде көруге болады:

```
N>0 bolatin butin san engiziniz  N=5

factorial(5)  shakirildi
factorial(4)  shakirildi
factorial(3)  shakirildi
factorial(2)  shakirildi
factorial(1)   shakirildi

5!=120
```

Бұл программадан **long** factorial(**int** n); функциясының өз денесінде өзін бес рет шақырғанын көруге болады. Функция өзін-өзі шақырғанда жай функцияның шақырылуындағы сияқты стекте нақты параметрлер мәндерінің көшірмесі жасалады, сонан соң басқару функцияның бірінші тұрған орындалатын операторына беріледі, функцияның әрбір шақырылуында осы процесс қайталана береді. Бұл процесті тоқтату үшін функциядағы алгоритмде ең болмағанда бір рекурсивті емес

тармақ болуы қажет, мысалы, жоғарыдағы 4.11-жаттығуда бұл келесі оператор болып тұр:

```
if (n==0 || n==1)
    return 1;
```

демек рекурсивті функция n==1 болғанда, соңғы рет шақырылып барып тоқтайтын болады. Функция қызметі аяқталған соң сәйкесінше стектегі орындар босатылады да, басқару рекурсивті функцияның шақырылуынан кейінгі тұрған бірінші операторға беріледі.

Программалауда рекурсивті функциялар рекурсивті алгоритмдерді барынша тиімді және қысқа түрде жазу үшін қолданылады. Бірақ кез келген рекурсивті алгоритмді рекурсияны қолданбай-ақ жазуға да болады. Мысалы, N! факториалды есептеу алгоритмін циклді пайдаланып келесі түрде де алуға болады:

```
long F=1; //факториалдың бастапқы мәні
if(N==0||N==1) // ереже бойынша 0!=1 және 1!=1 болатынын көрсету
  F=1;
  else
for (int i=1; i<=N;i++) // факториалды есептейтін циклді ұйымдастыру
F=F*i;
cout<<"\n\n "<<N<<"!="<<F;
```

Сонымен, программада рекурсивті функцияларды пайдаланудың артықшылығы – алгоритмді қысқаша түрде жазып шығуға болады, ал кемшілігі – функцияны қайта-қайта шақыруға кететін уақыт пен жадыдағы немесе стектегі параметрлер көшірмелері үшін алынатын орындар, бұл стектің толып қалу қаупін тудырады.

Бақылау сұрақтары

1. Программадағы функция түсінігі қалай анықталады?
2. Программада функцияны қалай жариялайды?
3. Функция қандай бөліктерден тұрады?

4. Функцияның прототипі деген не?

5. Функцияның параметрлеріне мәндер берудің түрлері қандай?

6. Тұрақты және айнымалы түріндегі параметрлерді беру қалай орындалады?

7. Функция параметрлерін адрестер түрінде беру не үшін қажет?

8. Массив параметрді берудің ерекшелігі қандай?

9. Функцияны қайыра жүктеудің қандай қажеттілігі бар?

10. Кең ауқымды параметрлер деген не?

11. Жергілікті параметр деген не?

12. Функция айнымалысының өмір сүру уақыты қалай анықталады?

13. Жады кластарының қандай түрлері бар?

14. Рекурсивті функция деген не?

15. Функция шаблоны не үшін қажет?

Тапсырмалар

1. Бір өлшемді массив элементтерін енгізуді және баспаға шығаруды функциялар түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

2. Бір өлшемді массивтің жұп элементтерінің қосындысын есептеуді функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

3. Бір өлшемді массивтің элементтерін «тікелей таңдау» әдісімен сұрыптауды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

4. Бір өлшемді массивтің элементтерін «көпіршік» әдісімен сұрыптауды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

5. Бір өлшемді массивтің элементтерін «кірістіру» әдісімен сұрыптауды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

6. Бір өлшемді массивтің элементтерінің арасынан мәні ең үлкен элементті табуды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

7. Бір өлшемді массивтің элементтерінің арасынан мәні ең кіші элементті табуды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

8. Бір өлшемді массивтің элементтерінің қосындысын табуды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

9. Бір өлшемді массивтің элементтерінің көбейтіндісін табуды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

10. Өлшемі 5x4 болатын бүтін санды екі өлшемді массив-матрица берілген. Матрицаны енгізуді және баспаға шығаруды функциялар түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

11. Өлшемі 3x5 болатын бүтін санды массив-матрица берілген. Матрица жолдарындағы мәні ең үлкен элементті табуды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

12. Өлшемі 5x5 болатын бүтін санды массив-матрица берілген. Матрицаның бас диагоналіндегі элементтерді нольмен алмастыруды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

13. Өлшемі 5x5 болатын бүтін санды массив-матрица берілген. Берілген матрица негізінде транспонирленген (сәйкесінше бағандары мен жолдары алмасқан) матрицаны алуды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

14. Өлшемі 5x5 болатын бүтін санды массив-матрица берілген. Матрицаның бас диагональдан төмен орналасқан бөлігін алуды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

15. Өлшемі 7x5 болатын бүтін санды массив-матрица берілген. Матрицаның бағанында орналасқан элементтерін сұрыптауды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

16. Өлшемі 5x7 болатын бүтін санды массив-матрица берілген. Матрицаның бағанында орналасқан элементтерінің қосындысын табуды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

17. $1+\frac{x^1}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots+\frac{x^n}{n!}$ қосындысын есептейтін программа жазыңыз.

18. $\frac{x^1}{1!}-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots-(-1)^n\frac{x^n}{n!}$ қосындысын есептейтін программа жазыңыз. n артқан сайын бұл қосынды sin (x) мәніне жақындайды.

19. $x\{i\} = \frac{2x\{i-1\}}{3} + \frac{a}{3x^2\{i-1\}}$, $x\{1\}=1$, $a>0$ рекуренттік қатынасын тізбектің элементтері i-дің мәні артқан сайын a-ның куб түбіріне тез жақындайтын болғандықтан, a-ның куб түбірін тез есептеу үшін қолдануға болады. a-ның куб түбір есептеу программасын жазыңыз.

Рекуррентті қатынасты анықтаңыз және тізбектің келесі үш мүшесін анықтайтын программа жазыңыз:

1, 2, 6, 24, 120, 720, 5040, ...

5-тарау. C++ -ТЕГІ ДИНАМИКАЛЫҚ ҚҰРЫЛЫМДАР

5.1. Динамикалық жады. new және delete амалдары

Программа құру кезіндегі ең басты мәселердің бірі – программаға қатысатын деректердің типін анықтау және оларды тиімді пайдаланудың жолдарын табу. Осыған байланысты программалауда деректерге қатысты мәселені екі үлкен топқа бөліп қарастыруға болады:

– біріншіден, қандай да болмасын объектіге қатысты, программада өңделетін ақпаратты барынша толық беру үшін типтің (деректің типінің) құрылымы дұрыс анықталуы қажет. Мысалы, программа, студенттер туралы ақпаратты өңдейтін болса, онда студент туралы деректі тек қана **int** типі арқылы немесе типі **char** болатын символдық массив арқылы беру мүмкін емес болар еді. Бұл мәселені шешу үшін программист **int, float, double, char, bool** және т.б. секілді стандарт типтерді пайдалана отырып, массивтерді, массив түріндегі немесе класс түріндегі жолдарды, саналатын типтерді, құрылымдарды, кластарды және т.б. типтерді өзі құрып алатыны белгілі. Деректердің құрылымдық типін анықтаудың бұл тәсілі «дәстүрлі» немесе стандарттық тәсілдердің бірі болып табылады;

– екінші мәселе, бұл программаға қатысатын деректердің жадыдан алатын орнына, яғни көлеміне байланысты туындайды. Компиляциядан кейінгі машиналық код түріндегі программаны орындауға жібергенде, бұл программа үшін жедел жадыдан орын, яғни бірнеше блоктардан тұратын ұяшықтар бөлінеді. Бұл блоктардың негізгілері статикалық және динамикалық блоктар болып табылады, сәйкесінше оларды *статикалық жады* және *динамикалық жады* деп атайды. Статикалық жадыда машиналық кодқа ауыстырылған программаның өзі, программада сипатталған айнымалылар мен тұрақтылар орналасады және программа орындалып тұрған кезде статикалық жады сақталып тұрады, сәйкесінше ондағы айнымалылардың да, тұрақтылардың да адрестері өзгермейтін болады. Бұл, әрине, программаға қатысатын деректердің көлемі немесе программа үшін қанша жады көлемі қажет болатыны алдын ала белгілі болса ғана мүмкін болады, ал алдын ала дерек көлемі аз ба, көп пе, қанша болатыны белгісіз болса, *динамикалық жады* және онымен жұмыс жасау үшін қажет *деректердің динамикалық құрылымдары* пайдаланылады.

Динамикалық жады – программаның орындалуы кезінде жедел жадыдан берілетін, бірақ программада әдейі қарастырылмаған болса, өздігінен пайдаланылмайтын бос тұратын жады бөлігі. Бұл пайдаланылмайтын бос жадыны программистердің арасында «*куча*» (heap) деп те атайтындар кездеседі. Программалауда динамикалық жадыны программа үшін қажет болған жағдайда пайдаланып, қажет болмай қалған кезде қайтадан босатып беру мүмкіндігі қарастырылған. Динамикалық жадыны пайдалану үшін тек қана көрсеткіштер түрінде жарияланатын *динамикалық айнымалылар* қолданылады. Динамикалық айнымалылар, жарияланғаннан бастап программа аяқталғанға дейін немесе динамикалық жады әдейі босатылғанға дейін уақытта бар болып, программа жұмысына қатыса алады.

C++ программалау тілінде динамикалық айнымалылармен жұмыс жасау үшін негізінен **new** – жадыдан орын алу, **delete** – жадыдан алынған орынды босатып беру амалдары қолданылады. Программалауда, динамикалық жадымен жұмыс жасау үшін C тілінен қабылданған <malloc.h> файлындағы функцияларды да қолдануға болады, бірақ **new – delete** амалдарының артықшылығы, оларды объектілер үшін де қолдана береді.

Программадағы деректер үшін динамикалық жадыдан алдын ала орындар алынады, ол үшін нақты көрсетілген типтерге **new** амалын қолданады, ол алынған орындағы алғашқы ұяшықтың адресі көрсеткіш түрінде анықталған динамикалық айнымалыға беріледі, мысалы:

```
int* n = new int;      // динамикалық жадыдан int типті шама үшін орын алады
                       // және оның адресін n-ға береді
int* m = new int (10); //динамикалық жадыдан int типті шама үшін орын алады,
                       //ол орынға 10 деген мәнді жазып инициализациялайды және
                       //оның адресін m-ға береді
int* mas = new int (10); //динамикалық жадыдан int типті он шамаға немесе
                         //массив элементтеріне орын алу, алғашқы ұяшық адресін
                         //массив аты болатын mas-ға беру
   int* nC = (int *)malloc(sizeof(int)); // malloc функциясын пайдаланып
                                         // динамикалық жадыдан int типті
                                         // шама үшін орын алады және оның
                                        //адресін nC-ға береді
```

Динамикалық жадыдан **new** амалының көмегімен алынған орынды босатып беру үшін **delete** амалы, ал malloc() функциясымен алынған орынды босату үшін free() функциясы қолданылады, мысалы:

```
delete n; //динамикалық жадыдан int типті шама үшін алынған
          //орынды босату
delete m; //динамикалық жадыдан int типті шама үшін алынған орынды
          //босату
delete [ ] mas; // int типті массивтің он элементіне алынған барлық орынды
                // босату
free (nC);// malloc функциясымен алынған орынды босату
```

Динамикалық жадыны **delete** амалының көмегімен босатуда, C++ тілінің бұрынғы нұсқаларында, егер динамикалық жадыдағы орын **new[ ]** түріндегі амалмен алынатын болса, онда ол **delete[]** түріндегі амалмен босатылуы қажет, себебі мұнда босату амалын квадрат жақшаларсыз **delete** түрінде жазса да қате болмайды, ол тек массивтің алғашқы элементі тұрған ұяшықты ғана босатып қайтарып береді, ал қалған элементтер үшін динамикалық жадыдан алынған орындар немесе ұяшықтар, сол күйі программа үшін қолданылуға жарамай қалып қояды, мұндай ұяшықтарды программалау тілдерінде *«қалдықтар» (мусор)* деп атады. Программа тиімділігінің бір көрсеткіші болып табылатын, жадыны қалдықтардан тазалап отыру мәселесі кейінгі кездерде маңызды орын алатындықтан, соңғы кездегі қолданылып жүрген C++ Builder 6 немесе тағы басқа да программалау орталарында **delete[]** mas; және **delete** mas; амалдарының нәтижелері бірдей болады, яғни екеуі де mas массиві үшін алынған орындардың барлығын қайтадан босатып бере алады.

5.1-жаттығу. Программа a[n] және b[n] динамикалық массивтерді динамикалық жадыға орналастыруда және оларға берілген орындарды босатуда **new-delete** амалдарын қолдануды көрсетеді.

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

```
 int n;
 cin>>n; //массивтердің өлшемін енгізу
 int* a = new int[n];   // a[n] массив элементтері үшін жадыдан орын алу
 cout<<"\n a massivi elementterinin  ";
 cout<<"\n ADRESTERI :    MANDERI : ";
 cout<<"\n _____";
 for(int i=0; i<n;i++)
 {
 a[i]=random(7);  //a[n] массивін құру
 cout<<"\n "<<&a[i]<<"      "<<a[i];  // a[n] массиві элементтері орналасқан
                                      //жады ұяшықтарының адрестерін
                                      // және элементтердің мәндерін
                                      // шығару
 }
 delete a; // a[n] массив элементтері үшін жадыдан алынған орынды босату
          // бұл амалды тоқтату үшін оның алдына екі слэш
          // таңбасы қойылса болды
 cout<<"\n\n  ";
 cout<<"\n b massivi elementterinin  ";
 cout<<"\n ADRESTERI :    MANDERI : ";
 cout<<"\n _____";
 int* b = new int[n];  // b[n] массив элементтері үшін жадыдан орын алу
   for(int i=0; i<n; i++)
 {
 b[i]=random(17); // b[n] массивін құру
 cout<<"\n  "<<&b[i]<<"  "<<b[i]; // b[n] массиві элементтері орналасқан
                                  // жады ұяшықтарының адрестерін және
                                  //элементтердің мәндерін шығару
 }
 getch();
}
```

Программа орындалуының нәтижесін екі түрлі жағдай үшін қарастыруға болады:

1-жағдай: динамикалық жадыны босату амалы амалы орындалмайтын жағдай, яғни //**delete** a; болғанда, программа орындалуының нәтижесі келесі түрде болады:

```
5
a massivi elementterinin
ADRESTERI:    MANDERI:

12076032      4
12076036      6
12076040      0
12076044      6
12076048      3

b massivi elementterinin
ADRESTERI:    MANDERI:

12076056      7
12076060      15
12076064      13
12076068      2
12076072      16
```

2-жағдай: динамикалық жадыны босату амалы амалы орындалатын жағдай, яғни **delete** a; болғанда, программа орындалуының нәтижесі келесі түрде болады:

```
5
a massivi elementterinin
ADRESTERI:    MANDERI:

12076032      4
12076036      6
12076040      0
12076044      6
12076048      3

b massivi elementterinin
ADRESTERI:    MANDERI:

12076032      7
12076036      15
12076040      13
12076044      2
12076048      16
```

мұнда a[n] және b[n] массивтерінің элементтерінің саны n = 5 болып, программаның орындалуы кезінде беріліп тұр, a[n] массивінің элементтері динамикалық жадыда 12076032-ден басталып, 12076052-ден аяқталатын аралықта қатар орналасып тұр. Бірінші жағдайда a[n] массиві үшін динамикалық жадыдан алынған орын босатылған жоқ, сондықтан жаңадан құрылатын b[n] массивінің элементтері 12076056 ұяшықтан бастап 12076076 ұяшықты қоса алғандағы аралықта орналасады. Ал екінші жағдайда, a[n] массиві үшін динамикалық жадыдан алынған орын **delete** a; амалының көмегімен босатылғандықтан, b[n] массивінің элементтері сәйкесінше сол босаған орындарға барып орналасып тұр. Сондай-ақ, бұл жаттығуда a[n] және b[n] массивтері динамикалық жадыда орналасатын динамикалық массивтер болғандықтан, олардың өлшемдері алдын ала анықталған тұрақтылар түрінде берілмейді (3.7-тақырыпты қараңыз).

5.2. Динамикалық құрылымдарды ұйымдастыру

Деректерді динамикалық жадыға орналастыру үшін арнайы бір әдістер бойынша ұйымдастырылатын құрылымдар қолданылады, программалауда бұл құрылымдарды *деректердің динамикалық құрылымдары* деп атайды. Бұл арнайы әдістер негізінен келесі түрде жүзеге асырылады, яғни программаның орындалуы кезінде деректерді орналастыратын динамикалық жады қажеттілікке қарай жеке-жеке блоктар немесе *элементтер* (компоненттер, түйіндер және т.б. атаулары бар) түрінде алынатын болады және бұл блоктар бірімен-бірі адрестер сақталатын көрсеткіштер арқылы байланысып тұрады (5.1-сурет).

Программаның орындалуы кезеңінде динамикалық құрылымдардың өлшемдерін өзгертіп пайдалана беруге болады, яғни программист қажет болған кезде көрсеткіштің көмегімен динамикалық жадыдан қатар-қатар орналасқан ұяшықтардан тұратын элементтерді немесе блоктарды ала береді, ал бұл блоктар қажет болмай қалғанда оларды қайтадан босатып береді. Осыған байланысты динамикалық құрылымдардың өлшемдері үнемі өзгеріп қозғалыста болатындықтан да оларды динамикалық деп атайды.

Деректердің динамикалық құрылымдары бұл тип емес, ол тек бұрыннан белгілі стандарт типтерді, қолданушы құратын типтерді және т.б.

типтерді жадыға белгілі бір ретпен орналастырудың әдістері немесе тәсілдері болып табылады.

Программалау тілдерінде динамикалық құрылымдардың блоктарын *түйіндер, элементтер немесе компоненттер* деп түрлі-түрлі атай береді. Динамикалық құрылымның *элементі (немесе түйіні, компоненті)* кем дегенде екі өрісі бар болатын құрылыммен (**struct**) анықталады. Өрістердің біреуінде стандарт типтер, қолданушы құратын типтер түріндегі немесе т.б. деректер сақталады, ал екінші өріс міндетті түрде көрсеткіш болады және ол көрсеткіште келесі элементтің адресі сақталады, егер келесі элемент жоқ болса, нольдік адрес NULL сақталатын болады. Динамикалық құрылымның элементін программада құрылым түрінде сипаттау келесі түрде болуы мүмкін:

```
struct elem //динамикалық құрылымның элементін жариялау
{
 int d;                  //деректер сақталатын өріс
 elem* adrK;             //келесі элементтің адресі сақталатын көрсеткіш
                         //немесе адрес-өріс
elem* adrA;              //алдыңғы элементтің адресі сақталатын көрсеткіш
                         // немесе адрес-өріс
};
```

мұнда өрістердің саны екеуден артық болса бола береді, бірақ одан кем болмауы керек. Мысалы, деректер сақталатын өрістер бірнешеу болуы мүмкін, олардың типтері де әр түрлі бола береді, сол сияқты адрес-өрістер де бірнешеу бола алады, мысалы, олардың біреуі алдыңғы элементтің, ал екіншісі кейін тұрған элементтің адресін көрсетіп тұрады.

Сонымен динамикалық құрылымдарды ұйымдастыруда белгілі бір тәсілдер қолданылатын болды, сол тәсілдерге байланысты оларды келесі топтарға бөледі:

– тізім (список);
– стек (стек);
– кезек (очередь);
– бинарлық ағаштар (бинарные деревья).

Бұл динамикалық құрылымдардың әрқайсысы белгілі бір ережелерге сәйкес құрылып жұмыс жасайтын болады. Жоғарыда аталған *тізім, стек, кезек және бинарлық ағаштар* түрінде анықталатын динамика-

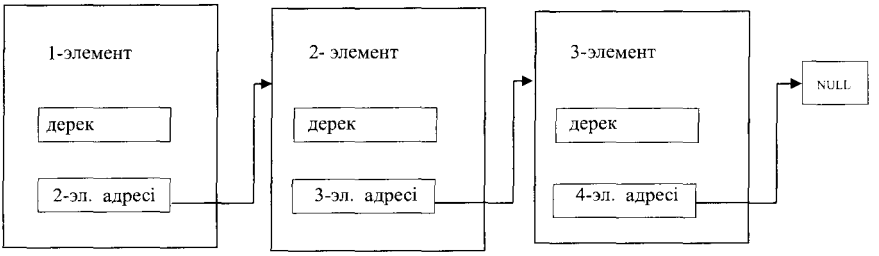
лық құрылымдарға тән ортақ нәрсе немесе олардың ұқсастығы, элементтерінің деректер және адрестер сақталатын өрістері бар құрылым түрінде анықталатындығында, ал айырмашылығы құрылым түрінде анықталған элементтердің бір-бірімен байланысу ережелерінде болады. Мысалы, тізімнің элементтері бірінен кейін бірі қатар орналасады және оның кез келген жерінде орналасқан элементімен жұмыс жасауға болатын болса, стек пен кезектің элементтерін пайдалану басқашалау болып келеді.

Программалауда динамикалық құрылымдарды пайдалану есептің алгоритміне байланысты таңдалатын болады. Мысалы, кәдімгі «санамақ» ойынының алгоритмін жүзеге асыру үшін ең тиімдісі, деректерді жадыда «тұйықталған тізім» немесе «сақина» түрінде ұйымдастыру болып табылар еді.

5.3. Тізім

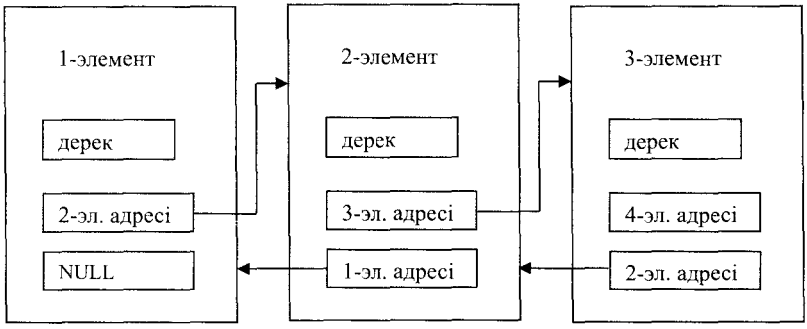
Тізім – бұл құрамдары бірдей болатын және бірінен кейін бірі тізбектей тізіліп орналасатын элементтерден тұратын динамикалық құрылым. Тізімдегі элементтердің санын *тізімнің ұзындығы* деп атайды. Программа орындалуы кезіндегі қажеттілікке қарай тізімнің ұзындығы өзгере береді. Тізбектей орналасқан тізім элементтерінің бір-бірімен байланысу түрлеріне қарай оларды үш түрге бөледі: *бір бағыттағы жай тізім, екі бағыттағы тізім және сақина.*

Бір бағыттағы жай тізім элементінің анықталуында көрсеткіш түріндегі адрестік өріс біреу ғана болады және ол келесі элементтің адресін көрсететін болады (5.1-сурет).



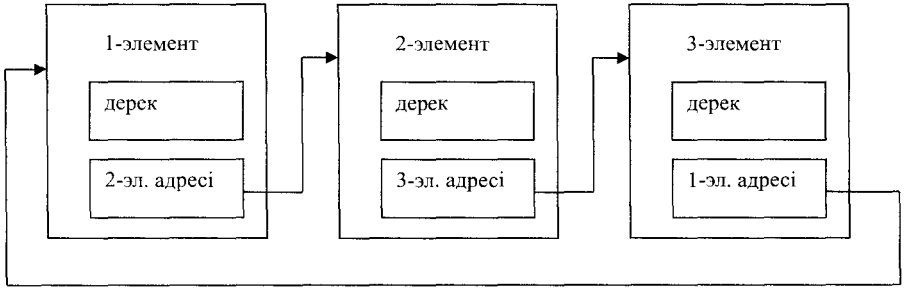
5.1-сурет. *Тізім элементтерінің байланысу схемасы /бір бағыттағы тізім/*

Екі бағыттағы тізімнің әрбір элементінде адрестік өріс екеу болады, оның біреуі сол элементтің алдындағы элементтің адресін, ал екіншісі одан кейін тұрған элементтің адресін сақтап тұрады (5.2-сурет).



5.2-сурет. *Тізім элементтерінің байланысу схемасы / екі бағыттағы тізім/*

Сақина түрінде ұйымдастырылатын тізім бұл да бір бағыттағы тізім болып есептеледі, бірақ мұнда тізімнің соңғы элементіндегі адрестік өрісте немесе көрсеткіште қайтадан тізімнің басындағы элементтің адресі сақталатын болады, сонда тізім тұйықталып, сақина түріндегі тізім болып шығады (5.3-сурет).



5.3-сурет. *Тізім элементтерінің байланысу схемасы /сақина немесе тұйықталған тізім /*

Программалауда тізімнің жұмысын ұйымдастыруды екі кезеңге бөледі: *тізімді құру және тізімді өңдеу.* Тізімді құру келесі алгоритм бойынша жүргізіледі:

– біріншіден, тізім элементінің жалпы құрылымы анықталады, мысалы,:

```
struct elem  // тізімнің  элементін жариялау
{
      int d;  //деректер сақталатын өріс
      elem* adrK;  //келесі элементтің адресі сақталатын
                   //көрсеткіш  өрісті жариялау
     elem *adrA; //алдыңғы  элементтің адресі сақталатын
                 //көрсеткіш  өрісті жариялау
};
```

– екінші, тізімнің басы анықталады, яғни тізімде алғашқы болып тұратын бірінші элементті құрып алады, мысалы, оны функция түрінде былай анықтайды:

```
elem* elem_1(int d)      //тізімнің бірінші элементін құратын функция
{
   elem* buf=new elem; //жаңа элемент үшін жадыдан орын алу және оның
                       //адресін buf көрсеткіш түріндегі айнымалыға жіберу
 buf->d=d;  //сол адрес бойынша  d-мәнді жіберу
 buf->adrK=0;   //келесі адрестің мәні NULL екенін немесе
                //тізімнің аяқталғанын көрсету
return buf;
}
```

Тізімді өңдеу амалдарына функциялар түрінде анықталған келесі амалдарды жатқызуға болады:

– тізімнің соңына жаңа элементті **тіркеу**, мысалы:

```
void kosuS(elem** tizSoni,int d) //тізімнің соңына жаңа
                                 //элементті қосу
 {
  elem *buf=new elem; //жаңа элемент үшін жадыдан орын алу және оның
                      //адресін buf көрсеткіш түріндегі айнымалыға жіберу

      buf->d=d;  //сол адрес бойынша жаңа элементтің мәні  d-ны жіберу
 buf->adrK=0;    //келесі адрестің мәні NULL екенін немесе
                 //тізімнің аяқталғанын көрсету
    buf->adrA=* tizSoni;  //жаңа элементке оның алдында тұратын
                          //элементтің адресін  беру
```

```
(*tizSoni)->adrK=buf; //енді,алдыңғы тұрған элемент үшін келесі адрес болып
                      //жаңадан қосылған элементтің buf- та сақталған адресі беріледі
    * tizSoni=buf; //көрсеткішке соңынан қосылған элементтің адресін
                   //беріп, оны тізімнің соңына жылжыту
}
```

– берілген деректің тізімде бар-жоғын **іздеуді** анықтайтын функция келесі түрде болуы мүмкін, мысалы:

```
elem* izdeu(elem* const tizBasi, int k)    //берілген элементті тізімнен
                                           //іздейтін функция
 {
   elem* buf= tizBasi;  //тізімнің басының адресін buf-қа беріп алу
   while(buf)  //әзірге buf-тағы адрес NULL болмаса,
     {
   if (buf->d==k)  //сол адрестегі d-ның мәнін берілген k-мен салыстыру
       break; //егер олар сәйкес болса шығып кету
    buf=buf->adrK; //әйтпесе келесі адреске көшіп, тексеруді жалғастыру
     }
   return buf; //функция қайтаратын мәнге  buf-тағы адресті беру
 }
```

– тізім элементтерінің арасындағы көрсетілген орынға жаңа элементті қосу функциясы, мысалы:

```
elem* ortKosu(elem *const tizBasi, int k, int k1)//жаңа k1 элементті тізімнен
                    //ізделетін k элементтен кейін орналастыратын функция
{
elem *berElemIzdeu=izdeu(tizBasi,k); //берілген  k элементті  тізімнен іздеу
    if (berElemIzdeu!=0)          // k элемент табылса, онда жаңа k1-ді
                                  //орналастыру басталады
      {
      elem* buf = new elem; //жаңа k1-элементке жадыдан орын алып,
                            //адресін  buf-қа беру
          buf->d=k1;                  //жаңа  адреске  k1дің мәнін жіберу
          buf->adrK=berElemIzdeu->adrK;    //жаңа элементтің кейінгі
                      //адресіне табылған элементтің кейінгі адресі беріледі
          buf->adrA=berElemIzdeu;          //жаңа элементтің алдыңғы
                     //адресіне табылған элементтің  өзінің  адресі беріледі
```

```
        berElemIzdeu->adrK=buf;  //табылған элементтің кейінгі адресіне
                                 //жаңа элементтің адресі беріледі
                }
}
```

– тізімнен элементті алып тастауды орындайтын функцияны келесі түрде құруға болады, мысалы:

```
bool aluEl(elem** tizBasi, elem** tizSoni,int k)//берілген k элементті тізімнен
                        //тауып, оны өшіріп тастайтын функцияны жариялау
  {
 elem* berElemIzdeu=izdeu(*tizBasi,k); //k үшін орындалатын izdeu
                                //функциясының нәтижесін berElemIzdeu-ге
                                //беру, егер ол k табылса, онда  berElemIzdeu-ге
                                //сол элементтің адресі, табылмаса ноль беріледі.
 if  (berElemIzdeu!=0) // k элемент табылса, онда
  {
    if (berElemIzdeu == *tizBasi)       // k элементтің орнын тексеру,
                                        //егер ол тізімнің басында тұрса, онда
      {
       *tizBasi = (*tizBasi)->adrK;
 (*tizBasi)->adrA=0; //тізімнің басы k-дан кейін тұрған элементке жылжиды
      }
     else {
       if (berElemIzdeu== *tizSoni)  // k элементтің орнын тексеру,
                                     //егер ол тізімнің соңында тұрса, онда
         {
          *tizSoni=(*tizSoni)->adrA;
  (*tizSoni)->adrK=0; //тізімнің соңы k-ның алдындағы элементке жылжиды
         }
              else    //егер ол  k элемент тізімнің ортасында  тұрса, онда
               {
                 (berElemIzdeu->adrA)->adrK=berElemIzdeu->adrK;
                  (berElemIzdeu->adrK)->adrA=berElemIzdeu->adrA;
               } //k-ның алдындағы және соңындағы элементтерді
                 //өзара байланыстыру
}
```

```
delete berElemIzdeu; //k элементі үшін алынған орынды жадыға қайтып беру
                     //немесе k элементі тізімнен алып тастау
    return true; //элемент табылып, жойылса, функция true мән қайтарады
 }
return false; //элемент табылмаса, функция false мән қайтарады
          }
```

Бұл аталғандар тізіммен жұмыс жасауда жиі қолданылатын амалдар болып табылады, бұлардан басқа тағы да басқа көптеген алгоритмдер кездеседі.

5.2-жаттығу. Программа тізімді құруды және ол тізімнен берілген элементті табуды және сол табылған элементтен кейін жаңа элементті тізімге орналастыруды көрсете алады:

```
#include <iostream.h>
#include <conio.h>
struct elem   //тізімнің элементін жариялау
   {
int d;          //деректер сақталатын өріс
elem* adrK;   //келесі элементтің адресі сақталатын
              //көрсеткіш өрісті жариялау
elem *adrA;  //алдыңғы элементтің адресі сақталатын
             //көрсеткіш өрісті жариялау
   };
elem* elem_1(int d); //тізімнің бірінші элементін құратын функция прототипі
void kosuS(elem **tizSoni,int d);  //тізім соңына жаңа элемент қосатын
                                   //функция прототипі
elem* izdeu(elem *const tizBasi, int k1);    //берілген элементті тізімнен
                                             //іздейтін функция прототипі
elem  *ortKosu(elem *const tizBasi, int k, int k1); //жаңа k1 элементті,
                                          // тізімнен ізделетін k элементтен
                                          //кейін орналастыратын функция
                                          //прототипі
int main() //негізгі программа басталды
{
int ganaEl;
cout<<"\n tizimnin birinshi elementinin mani: ";
cin>> ganaEl; // тізімнің бірінші элементі болатын жаңа элементтің мәнін енгізу
```

```
elem *tizBasi=elem_1(ganaEl); //тізімнің бірінші элементі құрылып, оның
                              //адресі тізімнің басы болатын
                              //көрсеткішке беріледі
elem *tizSoni=tizBasi;  //тізімнің соңы үшін жарияланған көрсеткішке
                        //тізімнің  басының адресі беріледі, яғни тізім бір
                        //ғана элементтен тұрады
    for(int i=2; i<6;i++)  // тізімнің қалған элементтерін енгізу басталды
 { cout<<"\n tizimnin kelesi "<<i<<"- gana  elementinin mani: ";
   cin>> ganaEl; //тізімнің келесі элементтерінің мәндерін енгізу
   kosuS(&tizSoni,ganaEl); //тізімнің соңына элемент қосатын функцияны
                           //шақыру
}
    elem *buf=tizBasi;  // buf  көрсеткіш – айнымалыға тізімнің басының
                        // адресін беру
   cout<<"\n TIZIM KURILDI: ";
   while (buf)        // тізімнің элементтерін баспаға шығару басталды,
                      //әзір buf-тің мәні
                      // нольге  тең болмаса бұл қайталанады
{
        cout<<"\t"<<buf->d;  // тізім элементінің мәнін баспаға шығару
buf=buf->adrK;          //buf-қа келесі элементтің адресін беріп жылжыту
}
    //тізімнен берілген элементті іздеу басталды
cout<<"\n\n izdeitin elementtin manin engizu :" ;
 int k;
 cin>>k;//ізделінетін элементтің мәнін енгізу
 buf=tizBasi; // buf  көрсеткішті қайтадан тізімнің басына апарып қою
elem*  izdEl= izdeu(buf, k); //izdeu функциясы  шақырылады, егер элемент
                             //табылса, izdEl  көрсеткішке сол табылған
                             //элементтің адресі, ал табылмаса, ноль беріледі
 if(!izdEl)
     cout<<"\n \n izdEl  gok";
     else
   { //берілген элемент тізімнен табылды
    cout<<"\n \n izdEl  bar=";
    cout<<"\n tabilgan elementten kein koilatin gana elementtin mani :" ;
```

```
   cin>> ganaEl; //табылған элементтен кейін қойылатын жаңа мәнді енгізу
    ortKosu( buf, k,ganaEl); //жаңа k элементті, тізімдегі көрсетілген орынға
                              //қоятын функцияны шақыру
      buf=tizBasi; // buf көрсеткішті қайтадан тізімнің басына апару
       cout<<"\n\n";
       while (buf) // тізімнің элементтерін баспаға шығару басталды
            {
             cout<<"\t"<<buf->d;
               buf=buf->adrK;
            }
   }
getch();
}
//---------------------------------------------------------------------------
elem* elem_1(int d) //тізімнің бірінші элементін құратын функцияны жариялау
  {
elem* buf=new elem; //жаңа элемент үшін жадыдан орын алу және оның
                    //адресін buf көрсеткіш түріндегі айнымалыға жіберу
buf->d=d;           //сол адрес бойынша d-мәнді жіберу
buf->adrK=0;        //келесі адрестің мәні NULL екенін немесе
                    //тізімнің аяқталғанын көрсету
return buf;
  }
//---------------------------------------------------------------------------
void kosuS(elem** tizSoni,int d)    //тізімнің соңына жаңа элементті
                                    //қосатын функцияны жариялау
  {
 elem *buf=new elem;  //жаңа элемент үшін жадыдан орын алу және оның
                      //адресін buf көрсеткіш түріндегі айнымалыға жіберу
 buf->d=d;          //сол адрес бойынша жаңа элементтің мәні d-ны жіберу
       buf->adrK=0;         //келесі адрестің мәні NULL екенін немесе
                            //тізімнің аяқталғанын көрсету
      buf->adrA=* tizSoni; //жаңа элементке оның алдында тұратын
                           //элементтің адресін беру
```

```
(*tizSoni)->adrK=buf;  //енді алдыңғы тұрған элемент үшін келесі адрес болып
                       //жаңадан қосылған элементтің
                       //buf-та сақталған адресі беріледі
     * tizSoni=buf; //тізімнің соңы-көрсеткішке соңғы қосылған элементтің
                    //адресін беріп, оны соңына жылжыту
   }
//-------------------------------------------------------------------------
elem* izdeu(elem* const tizBasi, int k) //берілген элементті тізімнен
                                        //іздейтін функцияны жариялау
   {
       elem* buf= tizBasi;    //тізімнің басының адресін buf-қа беріп алу
    while(buf)  // әзірге buf-тағы адрес NULL болмаса,
      {
     if (buf->d==k) //сол адрестегі d-ның мәнін берілген k-мен салыстыру
         break; //егер олар сәйкес болса шығып кету
      buf=buf->adrK; //әйтпесе келесі адреске көшіп, тексеруді жалғастыру
       }
  return buf; //функция қайтаратын мәнге buf-тағы адресті беру
       }
//-----------------------------------------------------
elem* ortKosu(elem *const tizBasi, int k, int k1)   //жаңа k1 элементті тізімнен
       //іздеретін k элементтен кейін орналастыратын функцияны жариялау
{
   elem *berElemIzdeu=izdeu(tizBasi,k); //берілген k элементті тізімнен іздеу
    if (berElemIzdeu!=0)          //k элемент табылса, онда жаңа k1-ді
                                  //орналастыру басталады
      {
     elem* buf = new elem; //жаңа k1-элементке жадыдан орын алып,
                           //адресін buf-қа беру
         buf->d=k1;        //жаңа адреске k1дің мәнін жіберу
        buf->adrK=berElemIzdeu->adrK;    //жаңа элементтің кейінгі
                  //адресіне табылған элементтің кейінгі адресі беріледі
        buf->adrA=berElemIzdeu;       //жаңа элементтің алдыңғы
                  //адресіне табылған элементтің өзінің адресі беріледі
    berElemIzdeu->adrK=buf; //табылған элементтің кейінгі адресіне
                            //жаңа элементтің адресі беріледі
            }
}
```

Программа орындалуының нәтижесі келесі түрде болады:

```
tizimnin birinshi elementinin mani: 7
tizimnin kelesi 2- gana elementinin mani: -2
tizimnin kelesi 3- gana elementinin mani: 1
tizimnin kelesi 4- gana elementinin mani: 4
tizimnin kelesi 5- gana elementinin mani: 23

TIZIM KURILDI:      7    -2    1    4    23

izdeitin elementtin manin engizu :  -2
izdEl bar =
tabilgan elementten kein koilatin gana elementtin mani :  17
     7    -2    17    1    4    23
```

мұнда тізім элементтерінің саны шектеулі болғандықтан, тізім элементтерін құруда **for(int** i=2; i<6;i++) түріндегі параметрлі цикл операторы қолданылды. Құрылатын тізімдегі элементтер саны алдын ала белгісіз болған жағдайларда **while** немесе **do_while** операторларының бірін қолдануға болады, мысалы, тізім элементтерін енгізіп, тізімді құруды келесі түрде де жазуға болады:

```
int i=2; //тізімнің келесі элементінің номері
 while (ganaEl != 13) //тізім соңының белгісін 13 саны білдіріп тұр
        { // тізімнің қалған элементтерін енгізу басталды
  cout<<"\n tizimnin kelesi "<<i<<"- gana elementinin mani: ";
  cin>> ganaEl; //тізімнің келесі элементтерінің мәндерін енгізу
 kosuS(&tizSoni,ganaEl);//тізімнің соңына элемент қосатын функцияны шақыру
        i++; //келесі элементтің реттік номерін алу
  } // бұл циклдің соңы болатын 13 саны кездескенше қайталана береді
```

5.3-жаттығу. Программа тізімді құруды, құрылған тізімнен берілген элементті тауып және оны тізімнен шығарып тастауды демонстрациялайды:

```
#include <iostream.h>
 #include <conio.h>
 struct elem              // тізімнің элементін жариялау
    {
```

```
int d;                  //деректер сақталатын өріс
elem* adrK;             //келесі элементтің адресі сақталатын
                        //көрсеткіш өрісті жариялау
elem *adrA;             //алдыңғы элементтің адресі сақталатын
                        //көрсеткіш өрісті жариялау
   };
elem* elem_1(int d); //тізімнің бірінші элементін құратын функция прототипі
 void kosuS(elem **tizSoni,int d);          //тізім соңына жаңа элемент қосатын
                                            // функция прототипі
 elem* izdeu(elem *const tizBasi, int k1);  //берілген элементті тізімнен
                                            //іздейтін функция прототипі
bool aluEl(elem**tizBasi, elem** tizSoni,int k); //берілген k элементті
                                    //тізімнен тауып, оны өшіріп
                                    //тастайтын функция прототипі
 int main()                         //негізгі программа басталды
{
int ganaEl;
cout<< "\n tizimnin birinshi elementinin mani:  " ;
cin>> ganaEl;  // тізімнің бірінші элементі болатын жаңа элементтің мәнін
               //енгізу
elem *tizBasi=elem_1(ganaEl); //тізімнің бірінші элементі құрылып, оның
                                    // адресі тізімнің
                                    //басы болатын көрсеткішке беріледі
elem*tizSoni=tizBasi;  // тізімнің соңы үшін жарияланған көрсеткішке
                       // тізімнің басының адресі беріледі, яғни тізім бір
                       // ғана элементтен тұрады
  int i=2;  // тізімнің келесі элементінің номері
  while (ganaEl != 13)  // тізім соңының белгісін 13 саны білдіріп тұр
       {   // тізімнің қалған элементтерін енгізу басталды
  cout<<"\n tizimnin kelesi "<<i<<"- elementinin mani: " ;
  cin>> ganaEl;  // тізімнің келесі элементтерінің мәндерін енгізу
kosuS(&tizSoni,ganaEl);  //тізімнің соңына элемент қосатын функцияны
                         // шақыру
        i++;  // келесі элементтің реттік номерін алу
  }  // бұл циклдің соңы болатын 13 саны кездескенше қайталана береді
```

```
elem *buf=tizBasi;    // buf көрсеткіш – айнымалыға тізімнің басының
                      // адресін беру
   cout<<"\n TIZIM KURILDI: ";
while (buf)    // тізімнің элементтерін баспаға шығару басталды,
               // әзір buf-тің мәні нольге тең болмаса бұл қайталанады
{
        cout<<"\t"<<buf->d; // тізім элементінің мәнін баспаға шығару
buf=buf->adrK;          //buf-қа келесі элементтің адресін беріп жылжыту
}
//тізімнен берілген элементті іздеу басталды
cout<<"\n\n izdeitin elementtin manin engizu :" ;
 int k;
 cin>>k;  //ізделінетін элементтің мәнін енгізу
 buf=tizBasi;  //buf көрсеткішті қайтадан тізімнің басына апарып қою
elem* buf2=tizSoni; // құрылған тізімнің соңғы эементінің адресін
                    // buf2-де сақтап тұру
elem*izdEl= izdeu(buf, k);  // izdeu функциясы шақырылады,егер элемент
                            //табылса, izdEl  көрсеткішке сол табылған
                           // элементтің адресі, ал табылмаса ноль беріледі
 if(!izdEl)
     cout<<"\n \n izdEl  gok";
     else
   {  // берілген элемент тізімнен табылды
     cout<<"\n \n izdEl  bar ol goiladi : ";
     aluEl(&buf,&buf2,k);  // табылған  k элементті тізімнен тауып, өшіретін
                           //функцияны шақыру
      buf=tizBasi;   // buf көрсеткішті қайтадан тізімнің басына апару
      cout<<"\n\n";
      while (buf)   // тізімнің элементтерін баспаға шығару басталды
            {
             cout<<"\t"<<buf->d;
               buf=buf->adrK;
            }
   }
getch();
 }
```

```
//---------------------------------------------------------------------------
elem* elem_1(int d) //тізімнің бірінші элементін құратын функцияны жариялау
  {
 elem* buf=new elem; //жаңа элемент үшін жадыдан орын алу және оның
                     //адресін buf көрсеткіш түріндегі айнымалыға жіберу
 buf->d=d;           //сол адрес бойынша  d- мәнді жіберу
 buf->adrK=0;        //келесі адрестің мәні NULL екенін немесе
                     //тізімнің аяқталғанын көрсету
return buf;
  }
//---------------------------------------------------------------------------
void kosuS(elem** tizSoni,int d)      //тізімнің соңына жаңа элементті
                                      //қосатын  функцияны жариялау
  {
  elem *buf=new elem;  // жаңа элемент үшін жадыдан орын алу және
                       // оның адресін buf көрсеткіш түріндегі
                       // айнымалыға жіберу
  buf->d=d;          // сол адрес бойынша жаңа элементтің мәні  d-ны жіберу
        buf->adrK=0;      //келесі адрестің  мәні NULL екенін немесе
                          //тізімнің аяқталғанын көрсету
       buf->adrA=* tizSoni;   //жаңа элементке оның  алдында тұратын
                              //элементтің адресін  беру
      (*tizSoni)->adrK=buf;   //енді алдыңғы тұрған элемент үшін келесі
                              // адрес болып жаңадан қосылған  элементтің
                              // buf-та сақталған адресі беріледі
   * tizSoni=buf;        // тізімнің соңы – көрсеткішке соңғы қосылған
                         // элементтің  адресін беріп, оны соңына жылжыту
    }
//---------------------------------------------------------------------------
elem* izdeu(elem* const tizBasi, int k)   // берілген элементті  тізімнен
                                          // іздейтін функцияны жариялау
    {
        elem* buf= tizBasi;    // тізімнің басының адресін buf-қа беріп алу
     while(buf)  // әзірге buf-тағы адрес NULL болмаса,
        {
   if (buf->d==k)   //сол адрестегі d-ның мәнін берілген k-мен салыстыру
```

```
            break; //егер олар сәйкес болса шығып кету
    buf=buf->adrK;  //әйтпесе келесі адреске көшіп тексеруді жалғастыру
          }
   return buf; //функция қайтаратын мәнге  buf-тағы адресті беру
        {
//-----------------------------------------------------
bool aluEl(elem**  tizBasi, elem** tizSoni,int k) //берілген элементті
                                    // тізімнен тауып, оны өшіріп тастайтын
                                    // функцияны жариялау
{
elem* berElemIzdeu=izdeu(*tizBasi,k); // k үшін орындалатын izdeu
                          // функциясының  нәтижесін berElemIzdeu-ге беру, егер
                          // ол k табылса, онда  berElemIzdeu-ге
                          // сол элементтің адресі табылмаса ноль беріледі.
 if  (berElemIzdeu!=0) // k элемент табылса, онда
   {
     if (berElemIzdeu == *tizBasi)      // k элементтің орнын тексеру,
                                        // егер ол тізімнің басында тұрса, онда
       {
        *tizBasi = (*tizBasi)->adrK;
 (*tizBasi)->adrA=0; // тізімнің басы k-дан кейін тұрған элементке жылжиды
       }
      else {
        if (berElemIzdeu == *tizSoni)  // k элементтің орнын тексеру,
                                        //егер ол тізімнің соңында тұрса, онда
           {
            *tizSoni =(*tizSoni)->adrA;
   (*tizSoni)->adrK=0; //тізімнің соңы k-ның алдындағы элементке жылжиды
           }
                else     //егер ол  k элемент тізімнің ортасында  тұрса, онда
                 {
                   (berElemIzdeu->adrA)->adrK= berElemIzdeu->adrK;
                    (berElemIzdeu->adrK)->adrA= berElemIzdeu->adrA;
                 } //k-ның алдындағы және соңындағы элементтерді
                    //өзара байланыстыру
     }
```

```
    delete berElemIzdeu; // k элементі үшін алынған орынды жадыға қайтып
                         // беру немесе k элементі тізімнен алып тастау
      return true; //элемент табылып, жойылса функция true мән қайтарады
    }
    return false; // элемент табылмаса, функция false мән қайтарады
    }
```

Бұл программаны тізімнен табылғаннан кейін жойылатын k элементтің тізімнің қай жерінде орналасатындығына байланысты үш түрлі жағдай үшін орындап көруге болады.

1-жағдай. Жойылатын элемент тізімнің басында орналасады:

```
tizimnin 1 – elementinin mani:          5
tizimnin kelesi 2 – elementinin mani:   8
tizimnin kelesi 3 – elementinin mani:   12
tizimnin kelesi 4 – elementinin mani:   -9
tizimnin kelesi 5 – elementinin mani:   17
tizimnin kelesi 6 – elementinin mani:   13
TIZIM KURILDI:   5    8    12    -9    17    13
izdeitin elementtin manin engizu:    5
izdEl  bar ol goiladi:
    8    12    -9    17    13
```

2-жағдай. Жойылатын элемент тізімнің арасында немесе ортасында орналасады:

```
tizimnin 1 – elementinin mani:          5
tizimnin kelesi 2 –  elementinin mani:   8
tizimnin kelesi 3 – elementinin mani:   12
tizimnin kelesi 4 – elementinin mani:   -9
tizimnin kelesi 5 – elementinin mani:   17
tizimnin kelesi 6 – elementinin mani:   13
TIZIM KURILDI:    5    8    12    -9    17    13
izdeitin elementtin manin engizu :    -9
izdEl  bar ol goiladi :
    5    8    12    17    13
```

3-жағдай. Жойылатын элемент тізімнің соңында орналасады:

```
tizimnin 1 – elementinin mani:          5
tizimnin kelesi 2 – elementinin mani:   8
tizimnin kelesi 3 – elementinin mani:   12
tizimnin kelesi 4 – elementinin mani:   -9
tizimnin kelesi 5 – elementinin mani:   17
tizimnin kelesi 6 – elementinin mani:   13

TIZIM KURILDI:      5    8    12    -9    17    13
izdeitin elementtin manin engizu :      13
izdEl  bar ol goiladi :
    5    8    12    -9    17
```

Сол сияқты бұл программа берілген элементті тізімнен іздеп таба алмаса, онда іздеген элементтің тізімде жоқ екенін хабарлайды:

```
tizimnin 1 - elementinin mani:          5
tizimnin kelesi 2- elementinin mani:    8
tizimnin kelesi 3- elementinin mani:    12
tizimnin kelesi 4- elementinin mani:    -9
tizimnin kelesi 5- elementinin mani:    17
tizimnin kelesi 6- elementinin mani:    13
TIZIM KURILDI:      5    8    12    -9    17    13
izdeitin elementtin manin engizu :      4
izdEl  gok
```

Бұл программада тізім элементтерінің саны алдын ала берілмейді, яғни қолданушы тізімнің соңын білдіретін таңбаны немесе шартты белгіні, мысалы, 13 санын енгізгенше тізімді құру жалғаса беретін болады.

5.4. Стек

Стек немесе *магазин* – бұл бір типке жататын деректердің тізбегінен тұратын және ол деректерді пайдалану, *стектің төбесі* деп аталатын

позициядан ғана жүргізілетін арнайы құрылым. Бұл құрылымды алу үшін деректерді жадыға белгілі бір ретпен орналастырады және белгілі бір ретпен шығарып алады. Программалауда жады түрлеріне байланысты стектің екі түрі қолданылады: статикалық және динамикалық стектер.

Стекпен жұмыс жасағанда *стектің тереңдігі* немесе *стектің сыйымдылығы* деген ұғым қолданылады. *Статикалық стек* үшін оның *тереңдігі* оған сиятын элементтердің санымен анықталады және программаның орындалуы кезінде өзгермейді, ал *динамикалық стек* үшін оның тереңдігі динамикалық жады көлемімен анықталады. Сондықтан оны шартты түрде шектелмеген деп те айтуға болады.

Стектегі әрбір дерек оның элементі деп аталады және ол элементтер қатаң түрде бірінен кейін бірі орналасады.

Стектің элементтері үшін екі-ақ амал анықталған: біреуі стекке *элементті енгізу,* екіншісі, стектен таңдаған *элементті шығарып алу.* Бұл амалдар стектің төбесі арқылы ғана орындалады. Стекке элементті енгізген кезде, оның алдындағы элементтер стектің төбесінен әрі қарай бір позицияға жылжып отырады да, ал керісінше стектен элементті алған кезде, оның артындағы элементтер бір позицияға стектің төбесіне жақындайды. Егер стекте ешқандай элемент болмаса, онда оны *бос стек* деп атайды. Стектің жұмысы *«соңғы келген бірінші болып шығып кетеді (LIFO- last in — first out)»* деген принципке негізделген. Бұл принципті немесе деректердің арнайы құрылымы стекті *L- F құрылым* (Last- First) деп те атайды.

Стекпен жұмыс істегенде кездесетін қателер: толық стекке элемент енгізуге болмайды. Мұны стектің толып кетуі деп атайды және бұл статикалық стекке тән нәрсе. Қандай да болмасын стек бос болса, одан элемент алу мүмкін емес. Осы себептерге байланысты программада стекті пайдаланған кезде оның мүмкіндігін, яғни толық емес пе, бос емес пе тексеріп алу керек.

5.4-жаттығу. Келесі программа бес элементтен тұратын массив элементтерін стек түрінде орналастырып және оларды стектен шығарып алуды көрсетеді:

```
#include <conio.h>
#include <iostream.h>
const int teren=5;   //стектің тереңдігі
void stakEng( int* stak , const int teren);  //стекке элементтерді
                                    // орналастыратын  функция прототипі
void stakAlu( int* stak , const int teren);  //стектен элементтерді шығаратын
                                    //функция прототипі
  int main()
      {
int s[teren]; //стектің қызметін атқаратын  массивті жариялау
int* ukS=s; //массивтің бірінші элементіне көрсеткішті жариялау
cout<<"\n stakke elementter engizu :  ";
stakEng(ukS,teren); //стекке енгізуді орындайтын функцияны шақыру
cout<<"\n stakten elementterdi shigarip alu: ";
stakAlu( ukS , teren); // стектен шығарып алуды орындайтын функцияны
                       //шақыру
        getch();
    }
//-------------------------------------------------------------------------
void stakEng( int* stak , const int teren)    //стекке енгізуді орындайтын
                                              //функцияны жариялау
    {
        int tobe=0;  //стектің төбесінде стектің бірінші элементінде тұр
        if (tobe==teren-1)  //стектің төбесінде соңғы элемент тұрса, онда
  cout<<"\n stak toldi";       //әрі қарай орын жоқ стек толып тұр
        else
   while (tobe<teren)        // әзір стек толмаған болса
        {
         int m;              // m  стекке енгізілетін элемент
         cin>>m;
        stak[tobe]=m; // m элементті  стекке  орналастыру
         tobe++;        //төбені кейінгі позицияға немесе орынға жылжыту
        }
    }
//-------------------------------
```

```
void stakAlu( int* stak , const int teren)   //стектен шығарып алуды
                                              // орындайтын функцияны жариялау
    {
       int tobe=teren-1;  //стектің төбесінде соңғы элемент тұр
     if (tobe==0)          // стектің төбесінде бірінші элемент тұрса,
                           // онда әрі қарай ештеңе жоқ,  стек бос
  cout<<"\n stak bos";
else
    while (tobe>=0)  //әзір стектің төбесі бос емес болса
        {
        int m; // m  стектен шығатын  элемент
         m=stak[tobe]; // элементті  стектен шығарып алып m-ға беру
         cout<<"  "<<m;
        tobe--;  //төбені алдыңғы  позицияға немесе орынға жылжыту
        }
    }
```

Бұл программаның нәтижесі келесі түрде болады:

```
stakke elementter engizu :          1   2   3   4   5
stakten elementterdi shigarip alu :  5   4   3   2   1
```

Бұл программадан стектің тереңдігі массивтің өлшемімен бірдей болатындығын және оның өзгермейтіндігін (**const int** teren = 5;) байқауға болады.

Программалауда динамикалық стекті тізімдер түрінде қарастырады, сондықтан мұнда стектің тереңдігі алынатын жады көлеміне байланысты өзгере береді. Динамикалық стек тізім түрінде құрылады да, одан элементтерді шығарып алу тізімнің басынан емес соңынан жүргізіледі.

5.5-жаттығу. Келесі программа элементтерді динамикалық стекке орналастыруды және одан шығарып алуды демонстрациялайды:

```
#include <iostream.h>
#include <conio.h>
 struct elem                  // stak-тізімнің элементін жариялау
```

```
    {
int d;
elem* adrK;
 elem *adrA;
    };
 elem* elem_1(int d);    // stak-тізімге бірінші элементті орналастыру
 void kosuS(elem **tizSoni,int d);   //stak-тізімге  элемент енгізу
 void aluS(elem** tizSoni) //stak-тізімнен элементті шығарып алу
 int main()                  // негізгі программа басталды
{
int ganaEl;
cout<<"\n  birinshi elementtin mani:  " ;
cin>> ganaEl;     //stak-тізімге баратын бірінші элементтің мәнін енгізу
elem *tizBasi=elem_1(ganaEl);  //stak-тізімге бірінші элементті
                                    // орналастыру
elem *tizSoni=tizBasi; //тізімнің соңы үшін жарияланған көрсеткішке
  //тізімнің  басының  адресі беріледі, яғни тізім бір ғана элементтен тұрады
 while  (ganaEl != 13)  // тізім соңының белгісін  13 саны білдіріп тұр
         {
           //stak-тізімге қалған элементтерді  енгізу басталды
          cout<<"\n kelesi elementtin mani: ";
          cin>> ganaEl;  // келесі элементтерінің мәндерін енгізу
   kosuS(&tizSoni,ganaEl); //stak-тізімге элемент енгізетін функцияны шақыру
        }
elem *buf=tizBasi;   // buf-қа  stak-тізімнің басының адресін беру
cout<<"\n ELEMENTTER STAK-TIZIMGE ORNALASTI  :  \n";
while (buf)          //stak-тізімге енген  элементтерді көру
        {
cout<<"\t"<<buf->d;
buf=buf->adrK;
         }
   cout<<"\n ELEMENTTER STAK-TIZIMNEN  SHIGARILDI  :  \n ";
  buf=tizSoni;  // buf-қа  stak-тізімнің соңының  адресін беру
  aluS(&buf);  // stak-тізімнен  элементтерді шығаратын функцияны шақыру
  getch();
```

```
}
//---------------------------------------------------------------------------
elem* elem_1(int d)  //stak-тізімге бірінші элементті орналастыру функциясы
  {
 elem* buf=new elem;
 buf->d=d;
 buf->adrK=0;
 return buf;
  }
//---------------------------------------------------------------------------
void kosuS (elem** tizSoni,int d)  //stak-тізімге элемент енгізетін функция
      {
 elem *buf=new elem;
 buf->d=d;
 buf->adrK=0;
 buf->adrA=* tizSoni;
 (*tizSoni)->adrK=buf;
       * tizSoni=buf;
  }
//---------------------------------------------------------------------------
void aluS (elem** tizSoni)  // stak-тізімнен элементтерді шығаратын функция
     {
        elem* buf=*tizSoni;
 while (buf)
      {
        cout<<"\t"<<buf->d;
        buf=buf->adrA;
      }
* tizSoni=buf;
     }
```

Программа орындалғанда келесі нәтижелерді көрсете алады:

```
birinshi elementtin mani:  1
kelesi elementtin mani: 2
kelesi elementtin mani: 3
kelesi elementtin mani: 4
kelesi elementtin mani: 5
kelesi elementtin mani: 13
ELEMENTTER STAK- TIZIMGE ORNALASTI:
    1    2    3    4    5    13
ELEMENTTER STAK- TIZIMNEN  SHIGARILDI:
    13    5    4    3    2    1
```

Жалпы программалауда деректерді стек құрылымы түрінде ұйымдастыру негізінен жүйелік программалауда, компиляторларды жасауда және де рекурсивті алгоритмдерді жүзеге асыруда қолданылады.

5.5. Кезек

Кезек (очередь) – бұл жұмыс істеу принципі кәдімгі кезек сияқты бір типке жататын арнайы деректер құрылымы. Бұл құрылым келесі түрде ұйымдастырылады, яғни кезекке элементті кіргізу немесе енгізу оның соңынан ғана, ал элементті шығарып алу басынан ғана жүргізіледі. Кезектегі деректер оның *элементтері* деп аталады. Элементтері бірінен кейін бірі тізбектей орналасады.

Кезекпен жұмыс істеу үшін екі амал қарастырылған: *элементті кезекке енгізу* және таңдаған *элементті кезектен шығарып алу.*

Кезекке енгізілген кезекті дерек соңғы тұрған элементтен кейін орналасады да, өзі соңғы элемент болады.

Кезектен элементті шығару алып алу тек *кезектің басынан* ғана мүмкін болады және оны шығарып алғаннан кейін қалған элементтер бір позицияға ілгері жылжиды да, шығарып алған элементтен кейін тұрған келесі элемент енді кезектің басында тұрады.

Егер кезекте бірде-бір дерек жоқ болса, онда мұндай кезекті *бос кезек* деп атайды.

Статикалық кезек үшін ондағы деректер санына тең болатын *«өлшем»* ұғымы анықталған. Динамикалық кезек үшін ондай ұғым жоқ, ол динамикалық жадының өлшеміне ғана тәуелді.

Кезек құрылымындағы сақталатын *«бірінші келген – бірінші болып шығып кетеді* FIFO (first in – first out)» деген принципке байланысты оны *F-F құрылымы* (First-First) деп атайды.

Кезекпен жұмыс жасағанда кездесуі мүмкін қателер: статикалық кезектің өлшемі алдын ала анықталғандықтан, егер ол толық болса, онда оған әрі қарай дерек енгізу мүмкін емес; кез келген бос кезектен дерек алуға болмайды.

5.6-жаттығу. Келесі программа бес элементтен тұратын массив элементтерін кезек түрінде орналастырып және оларды кезектен шығарып алуды көрсетеді:

```
#include <conio.h>
#include <iostream.h>
const int olshem=5;   // кезектің өлшемі
void kezekEng(int*kezek,const int olshem); // кезекке элементтерді
                                    // орналастыратын  функция прототипі
void kezekAlu( int* kezek , const int olshem);  //кезектен элементтерді
                                    // шығаратын  функция прототипі
int main()
      {
int s[olshem];  //кезектің қызметін атқаратын  массивті жариялау
         int* ukS=s;  //массивтің бірінші элементіне көрсеткішті жариялау
         cout<<"\n kezekke elementter engizu :      ";
        kezekEng(ukS,olshem);  //кезекке енгізуді орындайтын функцияны
                                    //шақыру
      cout<<"\n kezekten elementterdi shigarip alu :   ";
kezekAlu( ukS , olshem);  // кезектен шығарып алуды  орындайтын
                                // функцияны шақыру
   getch();
    }

//---------------------------------------------------------------------
```

```
void kezekEng( int* kezek , const int olshem)  // кезекке енгізуді
                                               // орындайтын функцияны жариялау
    {
        int basi=0;  //кезектің басында кезектің бірінші элементі тұр
        if (basi==olshem-1)  // кезектің басында соңғы элемент тұрса, онда
  cout<<"\n kezek toldi";   //әрі қарай орын жоқ кезек толып тұр

else
   while (basi<olshem)       // әзір кезек толмаған болса
        {
         int m;              // m  кезекке енгізілетін элемент
         cin>>m;
        kezek[basi]=m; // m элементті  кезекке  орналастыру
    basi++;      //кезектің басын  кейінгі позицияға немесе орынға жылжыту
        }
    }

//-------------------------------------------------------------------------------------
void kezekAlu(int*kezek ,const int olshem) //кезектен шығарып алуды
                                           // орындайтын функцияны жариялау
     {
      int  basi=0;  // кезектің басында бірінші элемент тұр
     if (basi== olshem-1) // кезектің басында соңғы элемент тұрса, онда әрі
                          // қарай ештеңе жоқ,  кезек бос
  cout<<"\n kezek bos";
else
    while (basi<=olshem-1) // әзір кезектің соңына жеткен жоқ болса
         {
          int m; // m  кезектен шығатын  элемент
           m=kezek[basi]; // элементті  кезектен шығарып алып m-ға беру
           cout<<"  "<<m;
           basi++; // кезекті кейінгі  позицияға немесе орынға жылжыту
         }
     }
```

Бұл программаның нәтижесі келесі түрде болады:

```
shiretke elementter engizu:          1  2  3  4  5
shiretten elementterdi shigarip alu: 1  2  3  4  5
```

Бұл программадан статикалық кезектің өлшемі массивтің өлшемімен бірдей болатындығын және оның өзгермейтіндігін (**const int** olshem=5;) байқауға болады.

Динамикалық кезек те тура динамикалық стектер сияқты тізімдер түрінде қарастырылады. Динамикалық кезектің де өлшемі программаға берілетін динамикалық жады көлеміне байланысты өзгере береді. Динамикалық кезек тура тізім сияқты жұмыс жасайды, яғни тізім түрінде құрылады да, одан элементтерді шығарып алу тізімнің басынан жүргізіледі.

5.7- жаттығу. Келесі программа элементтерді динамикалық кезекке орналастыруды және одан шығарып алуды көрсетіп бере алады:

```
#include <iostream.h>
#include <conio.h>
 struct elem                    // kezek-tizimnin  elementin zhariyalau
    {
int d;
elem* adrK;
 elem *adrA;
    };
 elem* elem_1(int d);  // kezek-tizimge birinshi elementti orналastyru
 void kosuS(elem **tizSoni,int d);  //kezek-tizimge element engizu
 void aluS(elem** tizBasi); //kezek-tizimnen elementti shygarip alu
 int main()                  //negizgi programma bastaldy
{
int ganaEl;
cout<<"\n  birinshi elementtin mani:  " ;
cin>> ganaEl;  //kezek-tizimge baratyn birinshi elementtin mәnin engizu
elem *tizBasi=elem_1(ganaEl); //kezek-tizimge birinshi elementti
                              // orналastyru
```

```
  while (ganaEl != 13) // kezek-tізімнің соңының белгісін 13 саны білдіріп тұр
        {  // kezek-tізімге қалған элементтерді енгізу басталды
          cout<<"\n kelesi elementtin mani: " ;
         cin>> ganaEl; // келесі элементтерінің мәндерін енгізу
 kosuS(&tizSoni,ganaEl); //kezek-tізімге элемент енгізетін функцияны шақыру
        }
elem *buf=tizBasi;  // buf-қа kezek-tізімнің басының адресін беру
cout<<"\n ELEMENTTER KEZEK- TIZIMGE ORNALASTI  : \n";
while (buf)       // kezek-tізімге енген элементтерді көру
        {
cout<<"\t"<<buf->d;
buf=buf->adrK;
        }
   cout<<"\n ELEMENTTER KEZEK- TIZIMNEN  SHIGARILDI  : \n ";
  buf=tizBasi; //buf-қа kezek-tізімнің басының адресін беру
  aluS(&buf); //kezek-tізімнен элементтерді шығаратын функцияны шақыру
  getch();
 }

//---------------------------------------------------------------------------
elem* elem_1(int d) //kezek-tізімге бірінші элементті орналастыру функциясы
   {
 elem* buf=new elem;
 buf->d=d;
 buf->adrK=0;
 return buf;
    }
//---------------------------------------------------------------------------
void kosuS (elem** tizSoni,int d)  //kezek-tізімге элемент енгізетін функция
        {
  elem *buf=new elem;
  buf->d=d;
 buf->adrK=0;
 buf->adrA=* tizSoni;
 (*tizSoni)->adrK=buf;
         * tizSoni=buf;
```

```
    }
//---------------------------------------------------------------------------
void aluS (elem** tizBasi) // kezek-tizimnen  elementterdi shygaratyn funkcija
      {
        elem* buf=*tizBasi;
 while (buf)
      {
        cout<<"\t"<<buf->d;
        buf=buf->adrK; //kezektin kelesi elementinin  adresin alu
       }
* tizBasi=buf;
      }
```

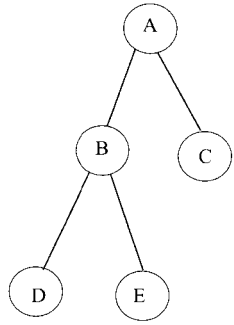
Программа орындалғанда келесі нәтижелерді көрсете алады:

```
birinshi elementtin mani:  1
kelesi elementtin mani: 2
kelesi elementtin mani: 3
kelesi elementtin mani: 4
kelesi elementtin mani: 5
kelesi elementtin mani: 13

ELEMENTTER KEZEK-TIZIMGE ORNALASTI:
    1    2    3    4    5    13

ELEMENTTER KEZEK-TIZIMNEN  SHIGARILDI:
    1    2    3    4    5    13
```

Программалауда деректерді кезектер түрінде ұйымдастыру, модельдеу есептерінде, операциялық жүйедегі командалардың орындалу тәртібін тағайындауда (диспетчеризация задач операционной системы), буферге енгізу-алу есептерінде және т.б. қолданылып жүр.

5.6. Бинарлық ағаштар

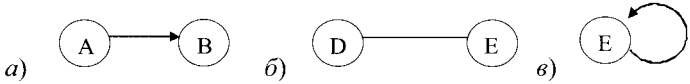
Граф-ағаш. Граф – бұл төбелердің және оларды өзара жалғастыратын немесе қосатын тікелей байланыстардың жиынтығы. Граф өзара байланысқан төбелердің қосары, мысалы, (A,C), (A,B), (B,E) және т.б. арқылы анықталады. Мысалы: бинарлық ағаш-графтың қарапайым түрі болып табылады (5.4-сурет).



5.4-сурет. *Қарапайым графтың мысалы-бинарлық ағаш*

Бұл мысалдағы A, B, C, D, E – графтың төбелері, ал (A-B), (A-C), (B-D) және (B-E) сәйкесінше төбелерді қосатын байланыстар болып табылады. Бұл арнайы құрылымдағы деректерді пайдалану сызықтық емес түрде жүреді деп айтуға болады және графты *сызықтық емес құрылым* деп атайды.

Графтағы төбелердің қосары берілген болса, онда ол төбелердің бірінен екіншісіне баратын тікелей жол да берілген болып есептеледі. Бұл жол (немесе байланыс) бір бағытта тура болуы мүмкін, бұл кезде оны доға деп атайды, яғни *графтың доғасы* деп аталады. Егер байланыс екі бағытта да мүмкін болса, онда мұндай байланысты қабырға, яғни жолды *графтың қабырғасы* деп атайды. Граф бір ғана төбеден де тұруы мүмкін. Бұл жағдайда ол төбенің өз-өзімен байланысын тұзақ, яғни *графтың тұзағы* деп атайды (5.5-сурет).



а) *б)* *в)*

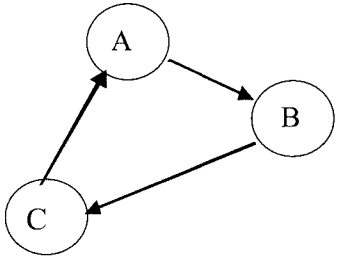
5.5-сурет. *Графтағы байланыстың түрлері*
а) доға; б) қабырға; в) тұзақ

Қабырғалар және тұзақ арқылы байланысқан *төбелер сыбайлас* деп аталады. Егер бір төбе басқа бірнеше төбелермен байланысқан болса, бұл төбелерде өзара сыбайлас төбелер болып табылады.

Егер графтағы барлық байланыстар (жолдар) қабырға түрінде анықталған болса, онда мұндай графты *бағдарланбаған граф* деп атайды, ал егер графтың байланыстарының барлығы доғалар түрінде болса, онда мұндай графты *бағдарланған граф* деп атайды. Мысалы, әуе жолдарының картасы бағдарланбаған графтың, ал алгоритмдердің блок-схемасы бағдарланған графтың мысалы бола алады.

Бір төбе мен екінші төбені байланыстыратын қабырғалар мен доғалардың тізбегі *маршрут* деп аталады. Сыбайлас екі төбе үшін маршрут қабырға немесе доға болып табылады.

Егер графтың қандайда болмасын төбесі үшін басқа төбелер арқылы болса да әйтеуір сол төбенің өзіне қайтып әкелетін маршрут болса, онда бұл маршрутты *цикл* деп атайды (5.6-сурет).



5.6-сурет. *Графтағы цикл-маршрут*

Егер графтың кез келген екі төбесін байланыстыратын маршрут бар болса, онда мұндай графты *байланысқан,* керісінше жағдайда *байланыспаған граф* деп атайды.

Графтар үшін төрт амал қарастырылған:

1. Деректі енгізу, яғни графқа дерегі бар төбе қосу;

2. Граф төбелеріндегі орналасқан деректерді іздестіру;

3. Деректі графтан алып тастау, яғни графтың сол дерек тұрған төбесін жою;

4. Байланыстарға сәйкес граф төбелерін айналып өту;

Графтың қарапайым мысалы ағаш-құрылым болып табылады. Графтар теориясында ағаш деп *тұзақ* және *цикл* түріндегі байланыстары жоқ

болатын бағдарланбаған, байланысқан графты айтады, яғни ағаштың төбелері бір-бірімен қабырғалар арқылы байланысады.

Ағаштың тармақтарын *бұтақтар* деп атайды да, бұтақтардың ұшындағы байланыспаған төбелерді *жапырақтар* деп атайды.

Белгілі бір төбеден басталатын ағашты *түбір ағаш* деп, ал төбенің өзін *түбір* деп атайды, әрі қарай түбір ағашты жай «*ағаш*» деп атау келісілген.

Сондай-ақ граф-ағаштың түбірін, яғни түбір болып табылатын төбесін ағаштың *түпкі тегі*, ал одан таралатын төбелерді *тума ұрпақтары* деп те атайды.

Ағаштың әрбір төбесі түпкі тегімен бірегей қабырғалар арқылы байланысады да, ал тума ұрпақтарымен бірнеше қабырғалар арқылы байланыса береді (5.4-сурет).

Ағаштың төбесін ұрпақтарымен байланыстарын қабырғаларымен қоса алғанда *түйін* деп атайды. Программалауда оны түйін немесе элемент деп атап жүр.

Граф-ағаш екі түрлі жолмен құрылады:

Граф-ағаштың барлық төбелеріндегі және жапырақтарындағы барлық деректер бір мағынаны білдіреді (семантика);

Граф-ағаштың төбелеріндегі деректер бір мағынаны, ал жапырақтарындағы деректер басқа мағынаны білдіреді.

Егер ағаштың әрбір *түйіндеріндегі* немесе элементіндегі ұрпақтарымен байланыстыратын қабырғалар саны екеуден аспаса, мұндай ағашты *бинарлық ағаш* деп атайды.

Бинарлық ағаштарға графтардағы сияқты *төрт амал қолданылады:*

1. Деректі енгізу;
2. Деректі іздеу;
3. Деректі жою;
4. Ағашты айналу.

Бір семантикалы ағаштар үшін бұл амалдардың бәрін де орындауға болады, ал әр түрлі семантикалы ағаштарда басқаша болуы мүмкін. Мысалы, «өрнектің» ағашында деректі қосу немесе алып тастау өрнектің мағынасын түгелімен өзгертіп жібереді, болмаса мүмкін емес жағдайларға соқтыруы мүмкін, сондықтан мұнда «іздеу» және «ағашты

айналу» амалдарын ғана қолдануға болады. Негізгі амалдардың сипаттамалары келесі түрде беріледі:

Ағашқа деректі енгізу: ереже бойынша ағаштың өзін құру «түбірден» басталады да, әрі қарай келесі деректі қосатын, енгізетін «түйін» ізделінеді. Бұл іздеудің нәтижесі екі түрлі болады: біреуінде «жапырақ» жасалады, яғни дерек қосылады, екіншісінде, яғни дәл осындай дерегі бар «түйін» табылса, ештеңе өзгермейді;

Берілген деректі ағаштан іздеу: Бұл амалдың операндасы – берілген деректің мәні; Амалдың нәтижесі – **true** немесе **false** болады, яғни **true** болады, егер дерекке сәйкес түйін немесе жапырақ табылса, **false** болады, егер ондай дерек табылмаса;

Ағаштан деректі жою: Деректі жою үшін әуелі оны іздейді, егер ондай дерек табылмаса, онда амал орындалмайды, ал дерек табылған жағдайда сол «түйін» немесе «жапырақ» қалған түйіндермен байланысын, ұрпақтармен байланысын қоса алғанда жойылады;

Ағашты айналу амалы: ағашқа байланысты көптеген есептерді шешуде «ағашты айналу» амалы қолданылады. «Түйіндерді» және «жапырақтарды» айналып өту маршруты түбірден басталады, ал айналудың өзі ең шеткі сол жақ бұтақтан басталады. Бұтақты айналуда екі жүріс қолданылады: пассив жүріс және актив жүріс. Ағашты айналуда оны құрудың ережесі міндетті түрде ескеріледі. Ағашты айналудың бірыңғай ортақ алгоритмі жоқ.

5.8-жаттығу. Берілген фамилияларды динамикалық ағашқа сұрыптап орналастыру керек. Ағаштағы деректерді алфавит бойынша өсу (тура айналу) және кему (кері айналу) ретіне қарай баспаға шығаратын программа құру керек.

```
#include <iostream.h>
#include <conio.h>
struct elem                    //ағаштың түйінін немесе элементін жариялау
 {
    string d;                  //элементтің дерек сақталатын өрісі
    elem* adrOn;               //элементтің оң жақ бұтағы
    elem *adrSol;              //элементтің сол жақ бұтағы
 };
```

```
elem* elem_1(string d); //алғашқы элемент – түбірді құру функциясының
                        // прототипі
void* kosuAgash(elem **tubir, string d);      //түбірге түйін қосатын
                                              // функциясының прототипі
void keri_ainalu(elem** tubir);       //ағашты кері айналу функциясының
                                      // прототипі
void tura_ainalu(elem** tubir); // ағашты тура айналу функциясының
                                // прототипі
int main()
{
string ganaEl; // түбірге немесе түйіндерге жіберілетін мән
cout<<"\n tybir-elementtin mani: " ;
cin>>ganaEl;
elem *tubir=elem_1(ganaEl); // түбірді құру
elem *buf=tubir;
for(int i=2; i<6;i++)
     { cout<<"\n  kelesi tuinnin  mani: " ;
cin>> ganaEl;
kosuAgash(&buf,ganaEl); // түбірге жаңадан түйіндер қосатын функцияны
                        // шақыру
}
cout<<"\n \n agashti tura ainalu: ";
tura_ainalu(&buf);        // ағашты тура айналу функциясын шақыру
cout<<"\n \n agashti keri ainalu: ";
keri_ainalu(&buf);        // ағашты кері айналу функциясын шақыру
getch();
}
//--------------------------------------------------------------------------
elem* elem_1(string d) // алғашқы элемент-түбірді құру функциясын
                       // жариялау
{
elem* buf=new elem;
buf->d=d;
buf->adrOn=0;
buf->adrSol=0;
return buf;
```

```
}
//-------------------------------------------------------------------------------
void* kosuAgash (elem** tubir, string d) // түбірге  түйін қосатын
                                          // функцияны жариялау
{
 elem*  buf= *tubir;
 elem*  buf1;
 bool  f =false; //енгізілген d элемент ағашта жоқ  болса f = false әйтпесе, f=true
while ( buf && !f)  // әзір d элементті түбірге қосудың бағыты анықталады
{
    buf1=buf;
    if  (d<buf->d)  // d  өзі жалғанатын түйіндегі мәннен кіші болса,
    buf= buf->adrSol;   // түйін түбірге ол сол жақтан қосылады
        else
       if (d>buf->d) // d  өзі жалғанатын түйіндегі мәннен үлкен  болса,
        buf=buf->adrOn;   // түйін түбірге оң жақтан қосылады
           else
   f=true; // d  өзі жалғанатын түйіндегі мәнмен бірдей болса, түбірге
           // қосылмайды
}
if (!f)  // енгізілетін d элемент түбірде  жоқ, яғни  f = false
   {
        buf=new (elem);  // түбірге қосылатын жаңа түйінге орын алу
         buf->d=d;
        buf->adrOn=0;
        buf->adrSol=0;
        if  (d<buf1->d)
           buf1->adrSol=buf; // түбірге  жаңа түйінді сол жағынан қосу
            else
           buf1->adrOn=buf;  // түбірге  жаңа түйінді оң  жағынан қосу
    }
}
//---------------------------------------------------------------
 void  keri_ainalu(elem** tubir)  // ағашты кері айналу функциясын жариялау
 {
// Фамилияларды алфавит бойынша кему ретімен шығару
```

```
 if (*tubir!=0)
 {
   keri_ainalu(&(*tubir)->adrSol);
   cout<<"\n"<<(*tubir)->d;
   keri_ainalu(&(*tubir)->adrOn);
 }
}
//---------------------------------------------------------------
 void tura_ainalu(elem** tubir) // ағашты тура айналу функциясын
                                // жариялау
 {   // Фамилияларды алфавит бойынша өсу ретімен шығару
     if (*tubir!=0)
       {
      tura_ainalu(&(*tubir)->adrOn);
      cout<<"\n"<<(*tubir)->d;
      tura_ainalu(&(*tubir)->adrSol);
         }
}
```

Программаның орындалуының нәтижесі келесі түрде болады:

```
tybir-elementtin  mani:  Ибраимов Ермахан
kelesi tuinnin mani: Саттарханов Бекзат
kelesi tuinnin mani: Сарсекбаев Бахыт
kelesi tuinnin mani: Утепбергенов Нуртас
kelesi tuinnin mani: Мукашев Хакназар

agashti tura ainalu:
   Утепбергенов Нуртас
   Саттарханов Бекзат
   Сарсекбаев Бахыт
   Мукашев Хакназар
   Ибраимов Ермахан
agashti keri ainalu:
   Ибраимов Ермахан
   Мукашев Хакназар
   Сарсекбаев Бахыт
   Саттарханов Бекзат
   Утепбергенов Нуртас
```

Бұл программада түйіндегі деректің типі string, яғни класс түрінде анықталған жолдық тип болып тұр (7.3-тақырыпты қараңыз). Бинарлық ағашта деректердің сызықтық түрде орналаспайтындығы 5.4-суреттен байқалып тұр, бірақ кез келген деректің жадыда сызықтық түрде бірінен кейін бірі орналасатыны белгілі. Бинарлық ағашта да солай, мұнда бар болғаны түйіндерге қатынау адрестер бойынша жүргізілетіндіктен, программистке ол қатынауды сызықтық емес, басқаша түрде ұйымдастыруға мүмкіндік беріледі. Демек, бинарлық ағаштар түріндегі құрылымдарды, «түйін – элементтеріне қатынау, сызықтық емес түрде жүргізілетін логикалық құрылым» деп қарастыруға болады. Бұл программада бинарлық ағашқа түйіндер сол жағынан алып қарағанда кему реті бойынша орналастырылады, сондықтан да **void** keri_ainalu(elem** tubir) функциясының нәтижесінде алфавиттегі кему реті бойынша сұрыпталған фамилиялар алынады, ал **void** tura_ainalu(elem** tubir) функциясын пайдаланып, оң жақтан қатынағанда, бинарлық ағашқа алфавит бойынша өсу ретімен орналасқан фамилиялар алынатын болады.

Программалауда, деректерді компьютер жадысында бинарлық ағаштар түрінде орналастыру, оларды жадыға бірден сұрыптап орналастыру үшін қажет болады. Сұрыпталып орналасқан деректерден қажет деректі іздеп, тауып алу, сұрыпталмай, кез келген ретпен орналасқан деректер арасынан іздегеннен әлдеқайда тез және жеңіл екенін байқауға болады. Жалпы, программалауда, деректерді бинарлық ағаштар түрінде ұйымдастыру программаның тиімділігін арттыратын болады.

Бақылау сұрақтары

1. Деректердің динамикалық құрылымдары не үшін қажет?
2. Динамикалық құрылымдармен жұмыс жасау үшін қолданылатын жады қалай аталады?
3. Динамикалық жадыдан орын алу үшін қандай амал қолданылады?
4. Динамикалық жадыдан алған орынды босату үшін қолданылатын амал қалай жазылады?
5. Динамикалық құрылымдардың элементтері қалай аталады?
6. Динамикалық құрылымдардың элементтерін қалай жариялайды?
7. Динамикалық құрылымдардың қандай түрлері бар?
8. Тізім деген не?

9. Тізімнің қандай түрлері бар?

10. Тізімнің бірінші элементін қалай құрады?

11. Тізімнің соңына жаңа элементті қалай қосады?

12. Тізімнің ортасына жаңа элементті қалай қосады?

13. Тізімнің басына жаңа элементті қалай қосады?

14. Сақина деген не?

15. Стек түрінде ұйымдастырылатын деректер үшін қандай қағида орындалады?

16. Кезек түрінде ұйымдастырылған деректер үшін қандай қағида орындалады?

17. Деректерді бинарлық ағаштар түрінде орналастыру не үшін қажет?

18. Тізім, стек немесе кезек түріндегі ұйымдастырудың бинарлық ағаш түріндегі ұйымдастырудан өзгешелігі неде?

19. Бинарлық ағаш түрінде ұйымдастырылған құрылымға қандай амалдар қолданылады?

20. Бинарлық ағаштың элементін қалай анықтайды?

Тапсырмалар

1. Топтағы студенттер және олардың үлгірімдері туралы деректерді тізім түрінде ұйымдастырыңыз. «Өте жақсы» және «жақсы» бағаларымен оқитын студенттердің фамилияларын көрсететін программа жазыңыз.

2. Фирма қызметкерлері туралы деректерді тізім түрінде ұйымдастырыңыз. Фамилиясы «А»-дан басталатын қызметкерлер тізімін және олардың орташа еңбекақысын көрсететін программа жазыңыз.

3. Аяқкиім туралы деректерді тізім түрінде ұйымдастырыңыз. Ең қымбат әйел адамының аяққиімінің бағасын, өлшемін және фирмасын көрсететін программаны жазу.

4. Телефонды пайдаланушы абоненттер туралы деректерді тізім түрінде ұйымдастырыңыз. Тізімді абоненттердің фамилиялары бойынша сұрыптайтын программа жазыңыз.

5. Телефонды пайдаланушы абоненттер туралы деректер, олардың фамилиялары бойынша сұрыпталған тізім түрінде берілсін. Тізімге

жаңадан енгізілетін абонентті алфавит бойынша орналастыратын программа жазыңыз.

6. Автомашинаны жөндеу шеберханасындағы машиналар жөніндегі деректер тізім түрінде берілсін. Тізімнен барлық қызыл машиналарды шығарып тастайтын программа жазыңыз.

7. Мекеменің тұрғын үй кезегіне тұрған қызметкерлері туралы деректерді стек түрінде ұйымдастырыңыз. Бастапқы үш адам үй алғаннан кейінгі тізімдегілердің кезегін шығаратын программа құрыңыз.

8. Кітаптар жөніндегі деректер тізім түрінде берілсін. Экранға таралымы бойынша кему ретімен сұрыпталған кітаптар тізімін шығаратын программа құрыңыз.

9. Жолаушы туралы деректер тізім түрінде берілсін. Көрсетілген бағытта жол жүретін жолаушылар тізімін шығарып беретін программа жазыңыз.

10. Поездар туралы деректер тізімде номерлері бойынша өсу ретімен сұрыпталып орналасқан болсын. Келесі жаңа поезды сұрыптау тәртібін бұзбайтын қылып тізімге енгізу программасын жазыңыз.

11. Топтағы студенттер туралы деректерді тізімде фамилиялары бойынша сұрыптап орналастырыңыз. Топқа жаңадан қосылған студентті осы сұрыптау сақталатындай қылып тізімге орналастыратын программа жазыңыз.

12. Дәрігерге қаралушы пациенттер туралы деректер тізім түрінде берілген. Пациенттер тізімінен фамилияларының бас әрпі «Б»-дан «Л»-ға дейінгі аралықта жататындарын туған жылы мен адресін көрсете отырып экранға шығаратын программа құрыңыз.

II БӨЛІМ. ОБЪЕКТІГЕ БАҒДАРЛАНҒАН ПРОГРАММАЛАУ

- Кластар
- Жолдар
- Ағындар мен файлдар

6-тарау. КЛАСС

6.1. Объектіге бағдарланған программалау түсінігі

Программалау тілдерінің пайда болу тарихына қарағанда, олар пайда болғаннан бастап қазіргі уақытқа дейін жылма-жыл дамып, жетілдіріліп келе жатыр. Программистер компьютерге арналған алғашқы программаларды, компьютердің құрылғыларына түсінікті болатындай қылып барынша машиналық тілдерге жақындатып жазатын болды. Ал қазіргі кездегі программалау тілдері программаларды барынша адамға түсінікті болатындай қылып жазуға бейімделген. Программалау тілінде жазылған кез келген программа процессорға арналған нұсқаулардың жиынтығы болып табылады. Программалау тілдерінің тиімділігі де, деңгейі де осы нұсқаулардың барынша қысқа да нұсқа түсініктілігімен, жөндеуге жеңіл болуымен және өзгертуге бейімділігімен анықталады. Компьютерлерге арналған қазіргі программалардың көлемдері өте үлкен екені белгілі, осындай үлкен программаларды жобалағанда және құрғанда, жөндегенде және өзгерткенде барлық ұсақ-түйектерді ескеріп, бақылап отыру үлкен қиындықтар туғызады, сондықтан алдын ала программаға қатысты барлық деректердің ең маңызды дегендерін қалдырып, аса қажет емес дегендерін алып тастап бір жүйеге келтіріп алу қажет болады. Бұл процесті *программаның абстракциялану дәрежесін* жоғарылату деп атайды.

Программаның абстракциялану деңгейін жоғарылатудың *бірінші шарты*, программадағы функцияларды бір рет құрып алғаннан кейін, қалған жағдайларда оның жариялануы мен анықталуына тәуелсіз болу, яғни қалған жерлерде тек оның шақырылуы (интерфейсі) ғана қолданылатындай болуы керек. Егер функцияларда кең ауқымды (глобальдық) айнымалылар қолданылмайтын болса, функцияның ин-

терфейсі оның тақырыбы сияқты анықталады, сондықтан қазір программалауда аса қажет болмаса, кең ауқымды (глобальдық) айнымалыларды пайдаланбауға тырысады.

Абстракциялау деңгейін жоғарылатудың *екінші шарты* программада өңделетін әр түрлі деректерді олардың нақты өмірдегі табиғатына барынша жақындататындай және бір құрылымға біріктіріп пайдалануға болатындай типтерді құрумен анықталады.

Жоғарыда аталғандай арнайы құрылатын деректерді өңдеу үшін арнайы функциялар қажет болады. Абстракциялау деңгейін жоғарылатудың *үшінші шарты* бұл арнайы құрылған типтер мен функцияларды бір модульге біріктіріп, программадағы басқа модульдермен араластырмай, бөлек сақтау болып табылады. Бұл, әрине, арнайы құрылған типтер мен функцияларды сыртқы өзгертулерден қорғау үшін жасалады.

Программалау тілдеріне осы үш шартты қанағаттандыратындай абстракцияланған ұғым қажет бола бастады. Программалаудағы *класс* (**class**) ұғымы осыған байланысты пайда болды. Программа құрған кезде, *класты* қолданушының өзі анықтайтын тип ретінде қарастырады. Класта деректердің түрліше құрылымдары мен типтері және оларды өңдеуге қолданылатын функциялар – бәрі бірге бір-ақ рет анықталады. Программист анықталған класты әрі қарай басқа программаларда қолданған кезде, оның тек қана интерфейсін ғана қолданады, ал оның анықталуы мен жариялануын қайталаудың қажеті болмай қалады.

Класс ұғымының пайда болуы программалауда «*объектіге бағдарланған программалау*» деп аталатын тұтас бір парадигманың қалыптасуына негіз болды. Мұның себебі, класты пайдаланып нақты өмірдегі шынайы объектілерді олардың барлық қасиеттерімен, іс-әрекеттерімен қоса алғанда модельдеуге және ол модельді компьютерді пайдаланып өңдеуге мүмкіндік туды. Объектіге бағдарланған программалаудың негізі қағидалары сонау Simula-67, Smalltalk тілдерінен бастап қабылданған, бірақ ол қағидаларды жүзеге асыру және түсініп пайдалану қиын болғандықтан, олар көп таралмады. Ал C++ программалау тілінде бұл қағидалар барынша тиімді және үйлесімді жүзеге асырылған, сондықтан да C++ объектіге бағдарланған программалау тілдерінің ең көп және ертерек таралғандарының біріне жатады. Объектіге бағдарланған программалаудың негізін *инкапсуляция, қабылдаушылық және полиморфизм* қағидалары құрайды.

Объектіге бағдарланған программалаудағы *инкапсуляция (encapsulation)* – бұл бір объектіге тән барлық қасиеттердің, әдістер мен әрекеттердің капсуланың қызметін атқаратын бір құрылымға немесе типке жинақталуымен сипатталады. Бұл капсуланың немесе құрылым-типтің қызметін *класс (***class***)* типі атқарады. Кластың ішінде деректер мен функциялардың жарияланулары, анықталулары сырткы өзгертулерден қорғалып, жасырылып тұрады. Сондықтан, сырттан қарағанда, класс ішінде не бар екені белгісіз «қара жәшік» сияқты болып тұрады және керісінше класта жасалатын өзгертулер (интерфейс өзгеріп кетпесе) негізгі программаға әсер етпейді.

Объектіге бағдарланған программалаудағы *қабылдаушылық* қағидасының мағынасы мынау болады: мұнда алдымен ең жоғарғысы болып есептелетін *түпкі класс* құрылады, кейін осы кластың негізінде тарайтын төменгі *тума кластар* құруға болады. Кейін құрылатын тума кластар алдыңғы түпкі кластың барлық қасиеттерін, әдістері мен әрекеттерін қабылдап алып, сақтап қалады. Тума кластың түпкі кластан берілген қасиеттерін, әдістері мен әрекеттерін өзгертуге болмайды, бірақ оны жаңа элементтермен толықтыруға болады және оның өзінен тарайтын тағы да жаңа тума-кластар алуға болады. Мұның нәтижесінде қай кластың қалай пайда болғанын, қайсы кластан тарайтынын көрсететін, бинарлық ағаштар сияқты *кластар иерархиясы* пайда болады, оның ең түпкісі немесе жоғарғысы болып әдетте **TObject** класы табылады.

Полиморфизм – бұл кластар иерархиясының түрліше кластарындағы өзара мағыналас, бірақ түрліше нәтижелер беруі мүмкін әрекеттерді (немесе функцияларды) бір атаумен беру. Мысалы, Windows терезелерінде ең жиі қолданылатын объектілердің бірі батырмаларды алатын болсақ, бір терезеде бірнеше батырма тұрады, олардың кез келгеніне «шерткенде немесе сырт еткізгенде», олар түрліше қызметтерді орындайтыны сияқты. C++ программалау тілінде полиморфизм кеңінен қолданылады. Мысалы, бірінші бөлімде қарастырылған, функцияны қайыра жүктеу, функцияның шаблондарын пайдалану полиморфизм қағидасын жүзеге асырудың қарапайым түрлері болып табылады (4.5, 4.6-тақырыптарды қараңыз).

Жоғарыда айтылған үш қағидаға сүйенетін және ол қағидаларды жүзеге асыру құралы ретінде *класс (*class*)* типін пайдаланатын объек-

тіге бағдарланған программалау ұстанымының қалыптасуы, программаның абстракциялану деңгейінің жоғарылағанына қарамастан, программадағы модельдің шынайы объектіге барынша жақындауына да мүмкіндік жасады. Айталық, программа адамға қатысты деректерді өңдейтін болса, онда «adam» деген түпкі класты құрып алып, қажет болса одан әрі қарай «student», «kizmetker», «okushi» және т.б. сияқты тума кластарды тарата беруге болады. Мысалы, программа көліктерге қатысты деректермен жұмыс жасайтын болса, онда «transport» сияқты түпкі класты құрып, одан «atArba», «avto», «poezd», «samolet» сияқты тума кластарды, ал бұл тума кластардан әрі қарай мысалы, «avto»-дан әрі қарай, «gukMashina», «genilMashina», «avtobus» және т.б. деген сияқты тума кластан тарайтын тума кластарды шығарып алуға болады. Бұл өз кезегінде программаның пәндік облысқа барынша жақындауына жағдай жасайды, программаның басқа да қолданушылар үшін түсініктілігін, жеңіл қабылдануын арттырады. Бірақ мұнда кластар иерархиясын құру ең күрделі кезең болып табылады, дұрыс құрылмаған иерархия кейін программадағы жүзеге асырылған кезде түсініксіз күрделі жағдайларға әкеп соқтырады. Сондықтан болар, егер кластар иерархиясы қажет болмаса, программаны модульдерді, яғни модульдік технологияны пайдаланып та жаза береді. C++ программалау тілінде қолданылатын модульдік технологияда кластарды құрып, стандарт кітапханалар сияқты сақтап пайдалану мүмкіндіктері де қарастырылған.

6.2. Класты жариялау

Класс – программистің (қолданушының) өзі құрып, өзі жариялайтын абстракт тип. Абстракт тип дейтін себебі, класты программаның мәтінінде жариялау ол класс жадыда пайда болып, тип ретінде орын алды деген сөз емес, дәлірек айтқанда, типі, сол жарияланған класқа жататын объект құрылмайынша, класс программадағы мәтін күйінде қала береді. Бұл тип, программалаудағы басқа да типтер сияқты, өзі арқылы жарияланатын деректің (мысалы, объектінің және т.б.) компьютер жадысындағы ішкі берілуін, қабылдайтын мәндерінің жиынтығын

және оларға қолданылатын амалдар мен функцияларды біріктіре алатындай болуы керек.

Сонымен класс өмірдегі шынайы нәрселерді модельдеу үшін қажет болғаны белгілі болды, бұл нәрселердің қасиеттері класта *деректер өрістері* түрінде беріледі (**struct** типіндегі сияқты, 3.8-тақырып), ал олардың осы қасиеттеріне байланысты іс-әрекеттері сол *деректер өрістерімен жұмыс жасайтын функциялар* түрінде беріледі.

Деректер сақталатын өрістерді *кластың қасиеттері,* ал функцияларды *кластың әдістері* деп атайды. Кластың өрістері мен әдістерін *кластың элементтері* деп атайды, осыған байланысты оларды сәйкесінше *өріс-элементтер* және *функция-элементтер* деп немесе *дерек-мүшелер* және *функция-мүшелер* деп те атай береді. Кластың программадағы жалпы сипатталуы келесі түрде жазылады:

```
class   кластың аты
        {
           private:
        < жабық қасиеттер >
        < жабық конструкторлар>
        < жабық  әдістер >
    protected:
       < қорғалған қасиеттер >
       < қорғалған конструкторлар>
        <  қорғалған әдістер >
    public:
       < ашық қасиеттер >
        < ашық конструкторлар>
        < ашық деструкторлар>
        < ашық әдістер>
     };
```

Мысалы, программадағы rectangle деп аталатын кластың жариялануын келесі түрде беруге болады:

```
  class rectangle          // rectangle класты жариялау басталды
{
private:                   //жабық бөлімді жариялау
double length;             // ұзындығы-жабық қасиет болып тұр
```

```
double width;          // ені-жабық қасиет болып тұр
                       //protected-қорғалған бөлім жоқ
public:                //ашық  бөлімді жариялау
rectangle()            //әдеттегі  конструктор  ашық болып тұр
  {
     assign(0,0);
  }
... ... ...
}; // класты жариялау аяқталды
```

мұндағы **private, protected, public** спецификаторлары немесе бөлімдері класс құрамындағы элементтерді пайдаланудың ережелерін немесе приоритеттерін анықтайды. Бұл бөлімдердің орналасу тәртібі, саны өзгере береді, бір бөлімнің бірнеше рет қайталануы да мүмкін, бөлімдердің бос болуы да мүмкін, ал егер бірде-бір бөлім сипатталмаған болса, компилятор кластың барлық элементтерін автоматты түрде жабық деп есептейді. Ол бөлімдердің әрқайсысының атқаратын ерекше қызметтері бар, яғни:

– кластың **private**-жабық бөлімінде анықталған элементтерін сол кластың өзінің ғана әдістері пайдалана алады. Мысалы, түпкі кластың жабық қасиеттерін, әдістерін, оның өзінен тарайтын тума кластары да пайдалана алмайды;

– кластың **protected**-қорғалған бөлігінде жарияланған қасиеттері мен әдістерін, сол кластың өзінің және оның өзінен ғана тарайтын тума кластарының пайдалануына ғана құқығы бар;

– кластың **public**-ашық бөліміндегі жарияланған қасиеттері мен әдістерін, программаның кез келген жерінде, барлығы да пайдалана алады.

Мысалы, C++ Builder 6 ортасындағы консолдық қосымшалар құрған кезде, қандайда болмасын бір класқа жататын объектіні жариялап, кейін оны программада пайдаланған кезде сол объектінің сол класқа тән **public** – ашық бөліміндегі жарияланған элементтерін автоматты түрде шығатын мәзірден таңдап алуға болады, бірақ **private, protected** бөліміндегілер бұл жерден табылмайды.

C++ программалау тілінде кластың қасиеттерін немесе өрістерін жариялауда келесі шарттар анықталған:

қасиеттердің типі C++ тегі кез келген типтердің бірі бола алады, тек қана сол кластың өзі бола алмайды, бірақ сол класқа көрсеткіш немесе сілтеме болуы мүмкін;

– қасиетті **const** спецификаторын пайдаланып жариялауға болады, бірақ мұнда оның мәнін анықтау немесе инициализациялау конструктордың көмегімен бір-ақ рет жасалады да, және ол өзгермейтін болып қалады;

– қасиетті **static** спецификаторын пайдаланып жариялауға болады, ал **auto, extern** және **register** спецификаторларын пайдалануға болмайды;

– класты жариялағанда, оның қасиеттерінің мәндерін жариялау кезінде бірден анықтап, инициализация жасауға тиым салынады.

Программада класты пайдалану оның интерфейсі арқылы жүргізіледі, *кластың интерфейсі* болып оның құрамындағы әдістердің (функция-элементтердің) тақырыптары алынады.

Жалпы *кластың әдістері* – бірнеше операторлардан тұратын, кластың өзінің ішінде немесе кластан тыс анықталатын функциялар. Әдетте, кластың әдісінде (немесе функция элементінде) анықталатын операторлар саны көп емес, шағын болса, онда оны кластың ішінде жариялай береді және оларды кіріктірілген (inline, встроенный) әдістер деп атайды, мысалы:

```
class rectangle      // кластың аты
    {
private:              //жабық элементтерді жариялау бөлімі
...    ...    ...
protected :           // қорғалған элементтерді жариялау бөлімі
        ...    ...    ...
      public:
rectangle()     // әдіс-конструкторды толығымен жариялау
{
assign(0,0);
}
...  ...  ...
   }; // класты жариялау аяқталды
```

мұнда аты кластың атымен бірдей болатын rectangle() әдісі, толығымен кластың ішінде анықталып жарияланып тұр. Класты жариялағанда, атауы өзі жарияланатын кластың атауымен бірдей болатын кем дегенде бір әдіс немесе функция – элемент болуы тиіс. Бұл әдісті *конструктор* деп атайды. Конструктор сол класқа жататын объектілерді құратын әдіс болып табылады.

Егер кластың әдісі кластан тыс анықталатын болса, онда оның прототипі міндетті түрде кластың ішінде жариялануы тиіс. Кластың жариялануында, прототипімен ғана берілетін әдістер, сол класс анықталған файлда болуы да мүмкін, бұл функция мен оның прототипінің берілуі сияқты болып келеді.

Әдетте, мұндай әдістер өз алдына бөлек * .h немесе *.cpp файлдарда жинақталады және оларды жариялағанда әуелі оның класының атын жазып, сонан соң әрекет ету облысын тағайындау амалын немесе «::» (екі рет қос нүкте, 2.7.12-тақырыпты қараңыз) белгісін қойып, сонан соң барып функцияның аты жазылады, мысалы:

```
class rectangle  // кластың аты
{
private:  // жабық элементтерді жариялау бөлімі
... ... ...
public:  // ашық элементтерді жариялау бөлімі
 ... ... ...
void assign(double Len, double Wide);      // әдістің немесе
                                          //функция-элементтің прототипі
};  // класты жариялау аяқталды
  void  rectangle :: assign(double Len, double Wide) // әдістің немесе
                                          //функция-элементтің өзін жариялау
       {
         length=Len;
          width=Wide;
       };
```

мұнда кластың ішінде assign(**double** Len, **double** Wide); функциясының прототипі ғана тұрғандықтан, оның толық жариялануы кластан тыс беріліп тұр.

Класты жариялау программадағы барлық блоктардан тыс жүргізілетін болса, онда мұндай кластарды *кең ауқымды кластар* (глобальные классы), ал керісінше класс кандай да болмасын функция немесе класс түріндегі басқа бір блоктың ішінде жарияланатын болса, онда мұндай класты *жергілікті кластар* (локальные классы) деп атайды.

6.3. Объектілерді жариялау

Объектіге бағдарланған программалауда типі алдын ала жарияланған класс болатын айнымалыларды кластың *объектілері* немесе *экземпляры* деп атау келісілген. Программада типі алдын ала жарияланған класс болатын бір немесе бірнеше объектілерді, массивтер түріндегі объектілерді, көрсеткіштерді және т.б. жариялауға болады, мысалы:

```
class rectangle // класты жариялау басталды
{
public: // ашық бөлімді жариялау басталды
double length;
double width;
rectangle()
{
assign(0,0);
}
rectangle (double Len, double Wide)
{
assign(Len,Wide);
}
... ... ...
void assign(double Len, double Wide); // assign() әдісінің прототипі
}; // класты жариялау аяқталды
void rectangle :: assign(double Len, double Wide) // assign() әдісінің
                                                  // өзін жариялау
{ length=Len;
  width=Wide;
}
```

```
   int main() // негізгі программа басталды
{
rectangle rect, rect1;          //rectangle класына жататын rect, rect1
                                // объектілерін жариялау
 rectangle masRect[10]; //rectangle класына жататын
                                // объектілер массивін жариялау
rectangle* korRect=& rect; //rectangle класына жататын объектіге
                                //көрсеткішті жариялау
rectangle* dinRect=new rectangle (); //rectangle класына жататын
                                //динамикалық объектіні жариялау
}
```

Айнымалылардағы сияқты программада жарияланған объектілердің бар болу уақыты мен көріну аймағы олардың жариялану облыстарына байланысты болады. Программа орындалған кезде жарияланған объектілер, яғни олардың барлық элементтері үшін жадыдан жеткілікті болатындай орын бөлінеді және ол объектілердің мәндерін беру үшін немесе инициализациялау үшін арнайы конструкторлар шақырылады. Объект өзінің әрекет ету аймағынан шыққаннан кейін автоматты түрде шақырылатын деструкторлар оны жойып жібереді де, жадыны босатады.

Объектінің элементтерін пайдалану үшін құрылымдағы сияқты «.» (нүкте) амалы қарастырылған (2.7.11-тақырыпты қараңыз). Нүкте амалы объектінің өзімен тікелей жұмыс жасау үшін қолданылады, ал егер объектіні пайдалану тікелей оның өзімен емес, керісінше сол объектіге сілтеме жасайтын көрсеткіш арқылы жүргізілетін болса, онда « → » бағыттауыш амалы қолданылады, мысалы, жоғарыда жарияланған объектілер үшін олардың элементтерін пайдалануды келесі түрде жазуға болады:

```
rect.length=45;
rect.width=10;                  немесе

korRect->length=45;
korRect->width=10;
```

мұндағы «нүкте» және «бағыттауыш» амалдарын объектінің **public**-ашық бөліміндегі жарияланған элементтеріне ғана қолданады, мысалы, жоғарыдағы жариялауда объектінің length және width қасиеттері ашық бөлімде жарияланып тұр.

Ал объектінің **private** – жабық бөліміндегі элементтерінің мәнін алу үшін немесе олардың мәндерін өзгерту үшін сол кластың арнайы анықталған assign() сияқты әдістерін ғана қолдануға болады, мысалы, объектінің сәйкесінше length және width қасиеттері, **private** – жабық бөлімде жарияланатын болса, онда ол объектінің элементтерін пайдалану келесі түрде болады:

```
class rectangle // класты жариялау басталды
{
private:                //жабық бөлімді жариялау басталды
double length;
double width;
public:                 //ашық бөлімді жариялау басталды
rectangle()
 {
    assign(0,0);
 }
rectangle (double Len, double Wide)
 {
   assign(Len,Wide);
 }
     ...  ...  ...
          void assign(double Len, double Wide); // assign() әдісінің
                                                // прототипі
        };  // класты жариялау аяқталды

 void  rectangle :: assign(double Len, double Wide) // assign () әдісінің
                                                    // өзін жариялау
{
  length=Len;
  width=Wide;
}
```

```
int main() // негізгі программа басталды
{
rectangle rect;          // rectangle класына жататын rect
                         // объектіні жариялау
rectangle* korRect=& rect; // rectangle класына жататын объектіге
                           //көрсеткішті жариялау
 rect.assign(450,10);    // rect1 объектісі үшін assign() әдісін шақыру
 korRect->assign(450,10); // korRect көрсеткіш үшін assign() әдісін шақыру
          }
```

мұнда length және width қасиеттері, **private** – жабық бөлімде жарияланғандықтан, оларға сәйкесінше 450 және 10 деген мәндерді беру үшін оның assign () арнайы әдісін шақыруға тура келеді (assign – мәндерді беру – передача значении).

6.1-жаттығу. Келесі программа объектіні жариялауды және оның элементтеріне амалдар қолдануды көрсете алады:

```
#include <iostream.h>
#include <conio.h>

class rectangle      // rectangle класты жариялау басталды
  {
private:             //жабық бөлімді жариялау басталды
double length;       // length қасиетті немесе өріс-элементті жариялау
double width;        //width қасиетті немесе өріс-элементті жариялау
public:
rectangle()          //rectangle () әдісін жариялау, бұл әдіс әдеттегідей
                     //конструктор ( конструктор по умолчанию) болады
{
        assign(0,0);    //assign () әдісін шақыру, мұнда length=0 , width=0
}

rectangle (double Len, double Wide) //rectangle (double Len, double Wide) әдісін
                                    //жариялау, бұл әдіс берілген мәндер бойынша
                                    // объект құрушы конструктор болады
```

```
{
assign(Len,Wide);  // assign () әдісін шақыру, мұнда length=Len , width=Wide
}

double Length()  // length қасиеттің мәнін қайтаратын функция
{
    return length;
}

double Width()  // width қасиеттің мәнін қайтаратын функция
{
    return width;
}

double   Area()
{
return length*width; // ауданды есептеп, оның мәнін қайтаратын функция
            }
        void assign(double Len, double Wide); // assign() әдісінің прототипі
};  // rectangle класты жариялау аяқталды

void  rectangle :: assign(double Len, double Wide) // assign () әдісінің
                                                   //өзін жариялау
{ length=Len;       // length қасиетке Len мәнді жіберу
  width=Wide;   //width қасиетке Wide мәнді жіберу
}

int main()                    // негізгі программа басталды
{
 rectangle rect, rect1;  // rectangle класына жататын rect, rect1
                         // объектілерді жариялау
    rectangle* korRect=&rect1;    //rect1  объектіге көрсеткішті жариялау

 double uzin, eni;
  cout<<"uzindigi men enin beriniz-" ;
  cin>>uzin>>eni;          // ұзындық пен енінің мәнін пернетақтадан енгізу
```

```
rect.assign(uzin,eni);          // енгізілген мәндерді объектінің қасиеттеріне
                                // жіберу
korRect->assign(450,10);        // қасиеттердің мәндерін тікелей бірден беру
cout<<"\n\n\n Gai object  rect-nin olshemderi : \n ";
cout<<"\n"<<" uzindigi – "<<rect.Length(); // length қасиеттің мәнін көру
cout<<"\n"<<" eni –      "<<rect.Width();  //width қасиеттің мәнін көру
cout<<"\n"<<" audani –   "<<rect.Area();   //ауданның мәнін көру

cout<<"\n\n\n  Korsetkish object  rect1-din olshemderi : \n";
cout<<"\n"<<" uzindigi – "<< korRect->Length(); // length қасиеттің мәнін көру
cout<<"\n"<<" eni – "<< korRect->Width(); //width қасиеттің мәнін көру
cout<<"\n"<<" audani –   "<<korRect->Area(); //ауданның мәнін көру
getch();
 }
```

Программа орындалғанда төмендегідей нәтижені көруге болады:

```
uzindigi men enin beriniz-      20  30

Gai object  rect-nin olshemderi:

uzindigi -      20
 eni -          30
 audani -       600

Korsetkish object  rect1-din olshemderi:
uzindigi -     450
 eni -          10
 audani -       4500
```

мұнда length және width қасиеттері жабық бөлімде жарияланғандықтан, ол қасиеттерге жоғарыда аталған «нүкте» және «бағыттауыш» амалдарын қолдану мүмкін болмайды, сондықтан ол қасиеттердің мәндерін беру үшін assign(Len, Wide), ал сәйкесінше мәндерін алу үшін Length() және Width() әдістерін қолдануға тура келеді.

6.3.1. **const** *объектіні жариялау*

Программада жарияланған класс үшін тұрақты түрінде берілетін объектілерді де пайдалануға болады. Тұрақты (**const**) түрінде жарияланған объектінің өріс-элементтерінің немесе қасиеттерінің мәндері өзгермейді және оларға тұрақты константалық әдістерді ғана қолданады. Мысалы:

```
class rectangle  // класты жариялау басталды
{
   ...  ...  ...
   void constAdis() const // константалық әдісті жариялау
        {
           cout<< "\n Rectangle-bul tortburish";
        }
              ... ... ...
} ; // класты жариялау аяқталды

 int  main()
    {
const rectangle constOb; // тұрақты немесе константа түріндегі
                         // объектіні  жариялау
    ...   ...        ...
    constOb.constAdis(); // тұрақты үшін константалық әдісті шақыру
     ...   ...        ...
    }
```

мұнда әдістің константалық екенін көрсету үшін оның жариялануындағы параметрлерді беру аяқталғаннан кейін **const** спецификаторының жазылатынын байқауға болады, мысалы:

```
void constAdis() const  // константалық әдісті жариялау
```

бұл константалық әдістер объектінің қасиеттерін немесе өріс-элементтердің мәндерін өзгерте алмайды. Сондай-ақ константалық әдістерді тұрақты емес объектілер үшін де шақыруға болады. Программа-

лауда константалық әдістерді негізінен қасиеттердің мәндерін алу үшін қолданады.

6.2-жаттығу. Келесі программа тұрақты түріндегі объектіні жариялауды және оған константалық әдісті қолдануды көрсетеді:

```
#include <iostream.h>
#include <conio.h>
class rectangle            //класты жариялау басталды
 {
public:                    // ашық бөлімді жариялау басталды
double length;
double width;
rectangle()   //әдеттегідей конструкторды ( конструктор по умолчанию)
              // жариялау
{
 length=20;
 width=30;
}

void constAdis() const   // константалық әдісті жариялау
{
     cout<< "\n Rectangle-bul tortburish";
}

}; // класты жариялау аяқталды

int main()
{
  const rectangle constOb; // тұрақты түріндегі объектіні жариялау
  constOb.constAdis();     // тұрақты үшін константалық әдісті шақыру
  cout<<"\n uzindigi – "<<constOb.length;
  cout<<"\n eni – "<<constOb.width;
  getch();
 }
```

Программа орындалғанда келесі түрдегі нәтижені көруге болады:

```
Rectangle - bul tortburish
 uzindigi - 20
 eni - 30
```

мұнда **const** rectangle constOb; – тұрақты түріндегі объектіні жариялағанда, rectangle() конструкторының көмегімен length=20 және width=30 болатын constOb объектісі құрылады. Сонан соң барып бұл тұрақты түріндегі объект үшін constAdis(); константалық әдісі шақырылады. Сондай-ақ, rectangle класындағы length және width қасиеттері ашық бөлімде жарияланғандықтан, ол қасиеттерге жоғарыда аталған «нүкте» амалдарын constOb.length; және constOb.width; түрінде қолдана беруге болады.

6.3.2. **this** *көрсеткішінің қызметі*

Программада жарияланған бір класқа жататын әрбір объектінің әрқайсысы үшін олардың қасиеттері немесе өріс-элементтері жеке-жеке сақталады, ал әдістер болса, барлық объектілер үшін жадыда бір-ақ рет беріледі және оны барлығы ортақ пайдаланылады. Жадыда бір-ақ рет берілетін ортақ әдісті әрбір жеке объект үшін шақырып пайдалану ды әдістің жасырын түрде берілетін **this** параметрі жүзеге асырады Бұл **this** параметрінде әдіс қай объект үшін шақырылатын болса, тура сол объектіге сілтейтін көрсеткіш-тұрақты сақталады.

Жасырын түрде берілетін **this** көрсеткішті, программада, көрсеткішті әдістен шығарып қайтарып алу үшін, мысалы, (**return this**;) не месе (**return *this**;) түрінде анық айқын қолдануға болады. Мысалы

6.3-жаттығу. Келесі программа **this** көрсеткішті, (**return *this**;) түрінде айқын қолдануды демонстрациялайды:

```
#include <iostream.h>
#include <conio.h>
```

```
      class rectangle
         {
public:
double length;
double width;
rectangle()
  {
 length=20;
 width=30;
   }
rectangle  &kaitaru (rectangle &T)
 {
if( length >T.length )
return *this;          // this  көрсеткішті айқын қолдану
return T;
 }
         };

int main()
{
   rectangle rect, rect1;
rect.length=17;
rect.width=25;
rectangle kaitMan=rect.kaitaru(rect1);        // rect объектісі үшін
                                              // kaitaru() әдісін шақыру
cout<<"\n  kaitMan="<< kaitMan.length;
 getch();
}
```

Программа орындалғанда шығатын нәтиженің түрі мынадай болады:

```
kaitMan=20
```

Бұл программа талданатын болса, мұнда класта анықталған келесі әдіс:

```
rectangle  &kaitaru (rectangle &T)
```

```
 {
if( length >T.length )
return *this;          // this көрсеткішті айқын қолдану
return T;
 }
```

типі сол класс болатын rect объектісі үшін шақырылады. rect объектісінің қасиеттері rect.length=17; және rect.width=25; болып программада анықталып кетеді. Бұл шақырылуда әдістегі T параметрдің орнына rect1 объектісі барады. rect1-дің length және width қасиеттерінің мәндері конструктордағы (конструктор по умолчанию) мәндерге сәйкес 20 және 30 болып тұрады. Содан кейін келесі:

```
kaitMan=rect.kaitaru(rect1);
```

орындалғанда, яғни rect объектісі үшін kaitaru әдісі шақырылғанда

```
rect .length > rect1.length
```

шарты тексеріледі, яғни, 17>20 салыстырылады, нәтижесінде шақырылған әдіс T параметрдің орнында тұрған rect1 объектіні қайтарып және ол мәнді kaitMan объектісіне береді. Керісінше rect.length=27; және rect.width=25; болса, онда **return *this**; орындалады да, kaitMan объектісіне ***this** көрсеткіш беріледі, ал бұл көрсеткіш rect объектісіне сілтейтін болады немесе онда rect объектісінің адресі сақталатын болады.

6.4. Конструктор және деструктор

6.4.1. Конструктор

Класты жариялағанда, атауы өзі жарияланатын кластың атауымен бірдей болатын кем дегенде бір әдіс немесе функция – элемент болуы тиіс. Бұл әдісті *конструктор* деп атайды. Конструктор сол класқа жататын объектілерді құратын әдіс болып табылады, яғни объект

жарияланғанда, объектінің типіне сәйкес кластың конструкторы автоматты түрде шақырылып орындалатын болады. Конструкторларды пайдалану, мәндері анықталмаған немесе инициализацияланбаған объектілерді болдырмауды қамтамасыз етеді.

Конструктордың негізгі қызметі – ол объектіні құру үшін автоматты түрде шақырылады және объектінің мәнін анықтайды (инициализациялайды).

Төменде конструкторға қойылатын негізгі шарттар мен талаптар берілген:

– конструкторларда ешқандай мән қайтарылмайды, тіпті void типті мәнді де қайтармайды, сондықтан конструкторларға көрсеткіш арқылы сілтеме жасауға болмайды;

– конструкторлардың аттары кластың атымен бірдей болуы керек;

– бір кластың бірнеше конструкторлары болуы мүмкін, оларды бір-бірінен параметрлері арқылы ажыратады, мысалы, rectangle класында rectangle() және rectangle (Len, Wide) деп аталған екі конструктор болды. Бұл бірнеше конструкторлардың қолданылатын, себебі олар құрылатын объектілердің мәндерін әр түрлі анықтайды. Параметрі болмайтын, мысалы, rectangle() сияқты конструкторды *әдеттегідей берілген конструктор (конструктор по умолчанию)* деп атайды. Конструкторлар бірнешеу болғанда, олардың аттары бірдей, бірақ орындайтын қызметтері әр түрлі болатындықтан, оларды беру үшін «қайыра жүктеу» механизмі қолданылады (4.5-тақырыпты қараңыз).

– конструктордың өзінің типі болмайды, ал оның параметрлері сол кластың өзінен басқа кез келген типтер бола алады;

– егер программист класты жариялағанда, бірде-бір конструкторды көрсетпейтін болса, онда компилятор параметрі болмайтын *әдеттегідей берілген конструкторды* өзі автоматты түрде тағайындайтын болады. Бұл әсіресе тума кластар үшін маңызды. Мұндай конструкторлар, қажет болған жағдайда, тума кластың қасиеттерінің (өріс-элементтерінің) мәндерін де оның түпкі класындағы әдеттегідей берілетін конструкторда көрсетілген мәндерге сәйкес өзі автоматты түрде тағайындай алады. Сондай-ақ, мұнда түпкі кластың жариялануында тұрақтылар (**const**) мен сілтемелер болған жағдайда тума кластың объектілерін құруда қателер болуы мүмкін, себебі автоматты түрде тағайындалатын конструкторлар олардың мәндерін анықтай алмайды.

– тума кластар үшін түпкі кластың конструкторлары қабылданбайды, яғни әрбір тума кластың өзінің конструкторы болуы керек және онда түпкі кластың конструкторларын шақыруға болады. Бірақ тума кластың конструкторында түпкі кластың конструкторын шақыру айқын, анық көрсетілмесе, онда компилятор түпкі кластың әдеттегідей берілетін конструкторын автоматты түрде шақыратын болады;

– конструкторларға **const, virtual** және **static** спецификаторларын қолдануға болмайды.

Объектінің көшірмесін алу үшін қолданылатын *көшіру конструкторының* жалғыз ғана параметрі болады, ол параметр, көшірмесі алынатын объектінің адресі сақталатын көрсеткіш болып табылады:

```
rectangle :: rectangle (const rectangle & korOb)
 {
     ... ...      ....
 }
```

мұндағы rectangle – кластың аты. Бұл конструкторды бұрыннан бар болып есептелетін объектінің көшірмесін алу үшін шақырады, егер программист бірде-бір көшіру конструкторын көрсетпесе, компилятор көшіру конструкторын өзі автоматты түрде тағайындайды.

6.4.2. Деструктор

Деструктор бұл қандай да болмасын бір класқа жататын объект құрылған кезде, алынатын жадыны қайтадан босатып беруді қамтамасыз ететін арнайы әдіс немесе оны объектіні жоятын әдіс деп те айтады. Объект өзінің көріну (немесе әсер ету) аймағынан шыққан кезде деструктор автоматты түрде шақырылуы тиіс. Бұл автоматты түрде шақырылған деструктор өзінің көріну аймағынан шығып, өмір сүру уақыты аяқталған объектінің жадыдан алған орнын кейін қайтарып беруі керек. Объектінің түріне байланысты деструкторларды шақыру программаның әр түрлі бөліктерінде жүзеге асырылады, мысалы, бұл шақырулар келесі бөліктерде болуы мүмкін:

– жергілікті объектілер үшін деструкторлар, олар өздері жарияланған блоктан шыққан кезде шақырылады;

– кең ауқымды объектілер үшін деструкторлар, олар main функциясынан шыққан кезде шақырылады;

– көрсеткіш арқылы берілген объектілер үшін деструкторлар **delete** амалында айқын емес, жасырын түрде шақырылады.

Сондай-ақ программада шақырылатын деструкторлар келесі шарттарды қанағаттандыруы қажет:

– деструктордың аты кластың атымен бірдей болады және тильда (~) белгісінен басталады, мысалы:

```
~ rectangle ();
```

– деструктордың параметрлері болмайды;

– деструктор ешқандай мән қайтармайды;

– деструкторларға **const** және **static** қолданылмайды;

– тума кластар үшін түпкі кластың деструкторлары қабылданбайды;

– деструкторға көрсеткіш жариялауға болмайды.

Егер класты жариялағанда деструктор көрсетілмеген болса, онда компилятор деструкторды да әдеттегідей берілген конструкторлар секілді автоматты түрде өзі тағайындайды. Программалауда аса қажет болмаса, деструкторларды анық түрде, айқын тағайындамаған дұрыс болып саналады.

6.5. Кластың статикалық элементтері

Программада бір класқа жататын объектілердің барлығына бірдей ортақ деректер болады. Кластың әрбір объектісінде қайталанатын осындай деректер үшін программа орындалған кезде жадыдан қайта-қайта орын алуды болдырмау қажет екені белгілі, сондықтан кластарды жариялаған кезде *статикалық қасиеттер (өріс-элементтер)* мен *статикалық әдістер* қолданылады.

6.5.1. Статикалық қасиеттер (өріс-элементтер)

Статикалық қасиеттер (немесе кластың өріс-элементтері) – кластың барлық объектілеріне тән, ортақ деректерді программада бір-ақ рет беру үшін қолданылады. Статикалық қасиеттер кластың барлық объек-

тілері үшін бастапқыда бір-ақ рет анықталады да, кейін қайталанбайтын болады. Кластың жариялануында статикалық қасиеттерді көрсету үшін **static** спецификаторын қолданады. Мысалы:

```
class rectangle
    {
     public:
double length;
double width;
static int count;  // статикалық қасиетті немесе өріс-элементті жариялау
rectangle()
 {
 length=20;
 width=30;
  }
    };
```

Статикалық қасиеттің мәнін программада кең ауқымды айнымалы сияқты анықтайды, мәні анықталмаса, компилятор оны «ноль» деп автоматты түрде анықтап кетеді (инициализациялайды). Мысалы:

```
int  rectangle :: count=10;  // count статикалық қасиетті инициализациялау
```

Статикалық қасиеттермен жұмыс жасағанда келесі талаптарды ескеру керек:

– статикалық қасиет үшін жады оның мәнін анықтау (инициализация) кезінде бір-ақ рет бөлінеді;

– статикалық қасиеттің мәнін беру үшін оны кең ауқымды айнымалы сияқты жариялап, оған әрекет ету облысын тағайындау амалын (немесе «::» таңбасын) қолданады;

– **private** – жабық бөлімде анықталған статикалық қасиеттің мәнін беру үшін әрекет ету облысын тағайындау амалын қолдануға болмайды, ол тек статикалық әдістердің көмегімен ғана жасалады;

– статикалық қасиетті пайдалану кластың атынан немесе объект атынан жүргізіле береді;

– объектіге бөлінген жады көлемін алу үшін **sizeof** амалын қолданғанда статикалық қасиеттің жадыдағы көлемі ескерілмейді.

6.4-жаттығу. Программа кластың статикалық қасиетін жариялауды және пайдалануды көрете алады:

```
#include <iostream.h>
#include <conio.h>
class rectangle                    // класты жариялау басталды
 {
public:                            // ашық бөлімді жариялау басталды
int length;
int width;
static int count;                  // статикалық қасиетті жариялау
rectangle()  // конструкторды ( конструктор по умолчанию) жариялау
{
 length=20;
 width=30;
}
};                                 //класты жариялау аяқталды
int  rectangle :: count=10;  // count статикалық қасиетті кең ауқымды
                           // (глобальды) айнымалы ретінде жариялап, оның
                           // мәнін анықтап алу
int main()                 // негізгі программа басталды
{
    rectangle rect1;       // кластың объектісін жариялау
cout<<"\n rect1.count="<<rect1.count; // rect1 объектісі үшін статикалық
                                      // қасиеттің мәнін көру
    rectangle rect2 ;                 // кластың объектісін жариялау
    rect2.count=21;                   // rect2 объектісінің статикалық
                                      // қасиетінің мәнін өзгерту
 cout<<"\n rect2.count="<<rect2.count; // rect2 объектісінің статикалық
                                       // қасиеттің мәнін көру
cout<<"\n\n sizeof(rect1)="<< sizeof(rect1); // rect1 объектінің көлемін қарау
    getch();
}
```

Бұл программа орындалғанда келесі нәтижелерді көруге болады:

```
rect1.count=10

rect2.count=10

sizeof(rect1)=8
```

Программа нәтижесі көрсеткендей, статикалық қасиеттің мәнін алдын ала кең ауқымды айнымалы сияқты жариялап анықтап алу керек, әйтпесе бұл статикалық қасиет үшін қажет, жады көлемі бөлінбей қалады. Статикалық қасиетті кең ауқымды айнымалы сияқты жариялап, жадыдан оған қажет орынды алып алғаннан кейін, программада ол қасиеттің мәнін өзгерте беруге болады. Мысалы, rect2.count = 21; деген сияқты, бірақ мұндай өзгерту жасау статикалық қасиет, кластың **public** – ашық бөлімінде жарияланса ғана мүмкін болады. Сондай-ақ, нәтижеден объектіге бөлінген жады көлемін алу үшін **sizeof** амалын қолданғанда, статикалық қасиеттің жадыдағы көлемі ескерілмей тұрғанын көруге болады.

6.5.2. Статикалық әдістер

Статикалық әдістер кластың статикалық қасиеттерімен жұмыс жасау үшін енгізілген. Статикалық әдістерді басқа да статикалық әдістерді шақыру үшін қолдануға болады, ал қалған статикалық емес әдістерге олар қолданылмайды, себебі мұнда **this** көрсеткіш қолданылмайды.

Статикалық әдістерді пайдалану үшін оны кластың атынан немесе объектінің атынан шақырады, мысалы:

```
rect1. inc_count(); // inc_count();  статикалық әдісін rect1
                    // объектісі үшін шақыру
```

Статикалық әдістерді тұрақты (**const**) және виртуалды (**virtual**) түрде анықтауға болмайды.

6.5-жаттығу. Программа кластың статикалық әдісін жариялауды және пайдалануды көрете алады:

```
#include <iostream.h>
#include <conio.h>
class rectangle                    //класты жариялау басталды
 {
private:                           //жабық  бөлімді жариялау басталды
    static int count;              //жабық статикалық қасиетті жариялау
 public:                           // ашық  бөлімді жариялау басталды
int length;
int width;
rectangle()
  {
 length=20;
 width=30;
  }
static int  inc_count()   // inc_count() статикалық әдісті  жариялау
  {
     return count++;
  }
static void  cout_count()  // cout_count () статикалық әдісті  жариялау
  {
    cout<<" count= "<<count;
  }
};                                 //класты жариялау аяқталды
int  rectangle :: count;           // count статикалық қасиетті жариялау,
                                   //оның мәні автоматты түрде ноль болады
int main()                         // негізгі программа басталды
{
    rectangle rect1;               // кластың объектісін жариялау
    for (int i=1; i<=5; i++)
     {
rect1.inc_count();          // inc_count();  статикалық әдісін  rect1
                            //объектісі үшін шақыру
        cout<<"\n"<< i<<"- ret: ";
```

```
    rect1.cout_count();        // cout_count(); статикалық әдісін rect1
                                                  // объектісі үшін шақыру
    }
  getch();
}
```

Бұл программа орындалғанда келесі нәтижелерді көруге болады:

```
1- ret: count=1

2- ret: count=2

3- ret: count=3
```

Бұл программада rectangle класының count статикалық қасиеті **private** – жабық бөлімде анықталып тұр, сондықтан оны пайдалану, ереже бойынша кластың өзінде ғана анықталған әдістер арқылы жүргізіледі. Осыған байланысты **rectangle** кластың жарияланууында inc_count() және cout_count() деп аталатын екі статикалық әдіс анықталды, олардың біріншісі count қасиеттің мәнін 1-ге арттырып көбейтіп отырады, ал екіншісі count қасиеттің мәнін баспаға шығарып көрсетіп отырады.

6.6. Амалдарды қайыра жүктеу

C++ программалау тілінде бұрыннан белгілі амалдарды кластың объектілері үшін басқаша әрекеттерді орындай алатындай етіп қайта тағайындауға болады. Ол үшін функцияны қайыра жүктеу механизмін пайдаланады (4.5-тақырыпты қараңыз). Бұл – амалдарды қайыра жүктеу тәсілі, қолданушының өзі анықтайтын класс (**class**) типін стандарт типтерге барынша жақындатуға мүмкіндік береді.

C++ программалау тіліндегі нүкте (.), әрекет ету облысын тағайындау (::), көрсеткішке қолданылатын «нүкте-жұлдызша» амалы(.*), шартты амал (**?:**), **sizeof** амалдарынан басқа барлық амалдарды қайыра жүктеуге болады (2.7-тақырыпты қараңыз).

Амалдарды қайыра жүктеу *амал-функция* деп аталатын арнайы әдістің көмегімен жүзеге асырылады. Амал-функциялардың басқа функ-

циялардан айырмашылығы, мұнда функция тақырыбында міндетті түрде **operator** деп аталатын қызметші сөз қатысады және амал-функциялар үшін келесі шарттар сақталады:

– стандарт типтерге қолданылатын амалдарды қайыра жүктегенде, олардағы параметрлердің (немесе аргументтердің немесе операндалардың) саны, амалдардың приоритеттері мен ассоциативтілігі өзгермейді;

– стандарт типтерге қолданылатын амалдарды олардың өздері үшін қайыра жүктеуге болмайды;

– қысқаша көбейту (=*) амалынан басқа амал-функциялардың барлығы тума кластар үшін қабылданады;

– амал-функцияларды статикалық әдістер **(static)** түрінде анықтауға болмайды.

Программада амал-функциялар үш түрлі тәсілмен: кластың әдісі, кластың үйлесімді функциясы немесе жай ғана функция түрінде анықталады. Амал-функциялардың программадағы жариялануы:

```
нәтиженің_типі operator амал_таңбасы  (параметрлер тізімі)
{
   ...     ... ...
}
```

Амал-функцияларды параметрі жоқ функциялар түрінде де жариялауға болады. Мысалы, стандарт сандық типтерге қолданылып келген «+» амалын, комплекс сандарға, матрицаларға немесе т.б. қолдануға болады. Ол үшін қолданушы өзі анықтайтын типке қолдануға болатындай қылып «+» амалын қайыра жүктеп алуы керек және мұнда сол қайыра жүктелетін амалдың ең болмағанда бір операндасы, сол қолданушы анықтайтын типке жататын объект болуы тиіс.

6.6.1. Унарлық амалдарды қайыра жүктеу

Унарлық амалдарға сәйкес анықталатын амал-функцияларды кластың ішінде де немесе кластан тысқары да жариялай беруге болады. Унарлық амал-функция параметрсіз әдіс түрінде анықталатын болса, онда оның операндасы сол амал қолданылып тұрған немесе оның өзін шақырып тұрған объект болып табылады. Мысалы, «емес» (!)

логикалық амалын triangle класы үшін қайыра жүктейтін болсақ, онда амал-функцияны кластың ішінде келесі түрде жариялайды:

```
class triangle // класты жариялау басталды
      {
  private:
      ... ... ...
  public:
  int operator ! () //«емес » (!) логикалық амалын жариялау басталды
{
  if ( a!=0 && b!=0 && c!=0)
        {
       if(a+b>c && a+c>b && b+c>a)
          return true;
       else
          return false;
        }
  else
     return false;
    }; //«емес » (!) логикалық амалын жариялау аяқталды
... ... ...
      }; // класты жариялау аяқталды
```

6.6-жаттығу. Программа triangle класы үшін қайыра жүктелген (!)-«емес» логикалық және (++)- инкремент амалдарының қызметтерін көрсете алады. Мұнда (!)-«емес» логикалық амалы кластың ішінде, ал (++) – инкремент амалы кластың сыртында параметрі жоқ әдістер түрінде жарияланатын, яғни қайыра жүктелетін болады:

```
#include <iostream.h>
 #include <conio.h>

class triangle          // класты жариялау басталды
  {
private:
int a, b,c;  // үшбұрыштың a,b, c қабырғаларын жариялау
public:
```

```
triangle ()  // әдеттегідей берілген конструкторды жариялау
   {
      a=2; b=3; c=4;
   }
triangle (int x, int y, int z)  // объектіні қолданушы берген мәндер бойынша
                                // құратын  конструкторды жариялау
  {
a=x;  b=y;  c=z;
  }
void print ()                   // үшбұрыштың a,b, c қабырғаларының мәндерін
                                // баспаға шығаратын әдісі
  {
cout<<"\n a=" <<a;
cout<<"\n b=" <<b;
cout<<"\n c=" <<c;
  }
int  operator ! ()      // ! – логикалық амалын қайыра жүктеу басталды
 {
                if ( a!=0 && b!=0 && c!=0) // егер a≠0, b≠0, c≠0 болса, онда
                  { // келесі a+b>c, a+c>b және  b+c>a шарттары тексеріледі
           if(a+b>c && a+c>b && b+c>a)// шарты орындалса, онда
                    return true;  // ! – логикалық амалы true мән қайтарады
                 else
                    return false; // әйтпесе false мән қайтарады
           }
       else
          return false; // әйтпесе false мән қайтарады
};   // ! – логикалық амалын қайыра жүктеу аяқталды
    triangle operator ++ ();      // triangle класы үшін (++) – инкремент амалын
                                  // қайыра жүктейтін әдістің  прототипі
}; //класты жариялау аяқталды
triangle triangle :: operator ++ () // triangle класы үшін (++) – инкремент амалын
                                  // қайыра жүктейтін әдістің өзін жариялау
    {
++a;  ++b;  ++c; // үшбұрыштың a,b, c қабырғаларының мәндерін
                 // 1-ге өсіріп отыру
 return *this;
```

```
    }; // қайыра жүктейтін әдістің өзін жариялау аяқталды
 int main()  // негізгі программа басталды
{
    int a,b,c; // үшбұрыштың a,b, c қабырғаларын жариялау және
                // мәндерін енгізу
    cout<<"\n a="; cin>>a;
    cout<<"\n b="; cin>>b;
    cout<<"\n c="; cin>>c;
    triangle T1, T2(a,b,c); // triangle класына жататын объектілерді жариялау
    if( !T2 )  // T2 объектіге! – логикалық амалын қолдану, егер ол true болса, онда
        {
 cout<<"\n Ushburish bar, sebebi !T2=true";
 T1=++T2;  // T2 объектіге (++) – инкремент амалын қолдану
 cout<<"\n\n  T1=++T2; amalinin natigesi :";
 T1.print();
     }
 else // ! – логикалық амалының мәні false болса, онда
     cout<<"\n Ushburish jok, sebebi !T2= false ";
 getch();
 }
```

Программа орындалуының нәтижесін екі түрлі жағдай үшін қарастыруға болады.

1-жағдай. Мұнда үшбұрыштың a, b, c қабырғаларының мәндері, үшбұрыш бар болатындай қылып енгізіледі:

```
a=5
b=6
c=7
Ushburish bar, sebebi !T2=true
 T1=++T2;  amalinin natigesi :
a=6
b=7
c=8
```

2-жағдай. Мұнда үшбұрыштың a, b, c қабырғаларының мәндері, үшбұрыштың қабырғалары бола алмайтындай қылып енгізіледі:

```
a=1
b=2
c=3
Ushburish jok, sebebi !T2=false
```

Бұл программада triangle класы үшін қайыра жүктелген «емес » (!) логикалық амалының қызметі бұл қолданушы енгізетін a, b және c сандары үшбұрыштың қабырғалары бола ала ма, жоқ па соны тексереді. Ол үшін мектеп геометрия курсынан белгілі «үшбұрыштың кез келген екі қабырғасының қосындысы, үшіншісінен артық болуы керек» және «үшбұрыштың қабырғасының мәні нольден өзгеше оң сан болуы керек» деген қағидалар тексерілді, егер бола алса «емес» (!) логикалық амалының мәні **true,** ал бола аламаса мәні **false** болды. Ал triangle класы үшін қайыра жүктелген инкремент (++) амалының қызметі, бұл үшбұрыш қабырғаларының берілген мәндерін сәйкесінше 1-ге арттырып өсіріп отыру болды. Бұл жаттығуда қайыра жүктелген амалдар – *параметрі жоқ* әдістер, яғни

```
int operator ! ();
triangle operator ++ ();
```

түрінде анықталды. Ал кейбір жағдайларда унарлық амал-функцияларды кластың ішінде, *параметрі бар әдістер* немесе функциялар түрінде жариялау қажет болады. Мұндай жағдайларда оларды класпен үйлесімді функциялар түрінде жариялайды. Мысалы, «емес» (!) логикалық және инкремент (++) амалдарын triangle класы үшін қайыра жүктеу параметрлі әдіс арқылы жүзеге асырылатын болса, онда оларды сәйкесінше келесі түрде жариялайды:

```
class triangle  //класты жариялау басталды
{
        ... ... ...
```

```
friend int operator ! (triangle &T) // «емес » (!) логикалық амалын қайыра
                                    // жүктеу параметрлі әдіс арқылы жасалды
            {
 if (T.a!=0 && T.b!=0 && T.c!=0)
   {
          if (T.a+T.b>T.c && T.a+T.c>T.b && T.b+T.c>T.a)
                  return true;
            else
                                  return false;
       }
    else
       return false;
};
     friend triangle & operator ++ (triangle &T); // инкремент (++) амалын
                                    // қайыра жүктеу параметрлі әдіс арқылы жасалды
   }; // класты жариялау аяқталды
   triangle & operator ++ (triangle &T)
{
++T.a;
 ++T.b;
 ++T.c;
 return T;
              };
```

мұнда «емес » (!) логикалық және инкремент (++) амалдарын triangle класы үшін қайыра жүктеу параметрлі әдістер арқылы беріліп тұр, параметр ретінде екі функцияда да triangle класының T объектісіне сілтеме немесе &T пайдаланылды. Сондай-ақ, қайыра жүктеу параметрлі әдіс арқылы берілгендіктен, олар кластың үйлесімді функциялары ретінде яғни **friend** – функциялар ретінде жарияланып тұр (6.9-ды қараңыз).

6.6.2. Бинарлық амалдарды қайыра жүктеу

Бинарлық амалдарды қайыра жүктеуді жүзеге асыратын бинарлық амал-функцияларды да кластың ішінде немесе сыртында да жариялауға болады. Бинарлық амал-функцияны кластың ішінде кем дегенде бір

параметрі болатын статикалық емес әдіс түрінде жариялайды. Мұндай жариялауда бинарлық амалдың бірінші операндасы амал-функцияны шақырып тұрған объектінің өзі болады, ал екінші операнда, жариялауда көрсетілген параметрдің орнындағы объект болып табылады. Егер бинарлық амал-функция кластан тыс, оның сыртында жарияланатын болса, онда оның сол класқа жататын (типі сол класс болатын) екі параметрі болуы шарт, кейін амал-функцияны шақырғанда бұл екі параметрдегі көрсетілген объектілер екі операнданың қызметін атқаратын болады. Қайыра жүктеуде жиі қолданылатын, бинарлық амалдар ретінде «меншіктеу» (=) , « қосу» (+), «азайту» (-), «көбейту» (*) және «бөлу» (/) амалдарын айтуға болады. Бұл амалдарды қайыра жүктеп қолдану, әсіресе класс түрінде анықталатын математикалық объектілер үшін қолданғанда тиімді болып табылады. Мысалы, программалауда, комплекс сандарды, векторлар мен матрицаларды кластар түрінде анықтап және оларға қолданылатын амалдарды стандарт амалдарды қайыра жүктеу арқылы жасап алу жиі қолданылады. Сол секілді динамикалық жадымен жұмыс жасайтын **new** және **delete** амалдарын, массивтермен жұмыс жасауда қолданылатын «индекстеу» ([ ]) амалын және т.б. қайыра жүктеп, пайдалану да программалауда жиі кездеседі.

6.7-жаттығу. Программа матрицаны немесе екі өлшемді динамикалық массивті matrix класы түрінде анықтауды және осы класс үшін қайыра жүктелген «меншіктеу» (=), «қосу» (+) және «көбейту» (*) амалдарының қызметтерін көрсете алады:

```
#include <iostream.h>
#include <conio.h>
class matrix  // matrix класты жариялау басталды
{
   private:
   unsigned int GOL;        //матрицаның жолдарының саны
   unsigned  int BAGAN;  //матрицаның бағандарының саны
   float **elemMat;        //матрица элементіне көрсеткіш
  public:
 matrix( unsigned int gol, unsigned int bagan);   //матрицаны құрушы
                                  //конструктор-әдістің прототипін жариялау
```

```
 matrix engizu();        // құрылған матрицаны толтыратын немесе
                         // элементтерінің мәнін енгізетін әдістің прототипі
 void koru ();       //матрицаны баспаға шығарып көрсететін әдіс прототипі
matrix operator =(matrix &M);  // matrix класы үшін меншіктеу (=) амалын
                               // қайыра жүктейтін әдістің  прототипі
     matrix operator +(matrix &M);  // matrix класы үшін  қосу  (+)  амалын
                                    // қайыра жүктейтін әдістің  прототипі
   matrix operator *(matrix &M); // matrix класы үшін көбейту  (*)  амалын
                                 // қайыра жүктейтін әдістің  прототипі
 };                                  // matrix класын  жариялау аяқталды
//-------------------------------------------------------------------------------
matrix:: matrix( unsigned int gol, unsigned int bagan) // матрицаны құрушы
                                        // конструктор-әдістің өзін  жариялау
{
 unsigned int i;
 GOL=gol; BAGAN= bagan;
 elemMat = new float* [GOL];  // матрица элементтері үшін жадыдан
                              // GOL орын алу
 for(i=0;i<GOL;i++)
   elemMat[i]= new float [BAGAN];  //матрица элементтері үшін жадыдан
                                   // GOL* BAGAN орын алу
}
//-------------------------------------------------------------------------------
matrix matrix:: engizu()  // құрылған матрицаны толтыратын немесе
                          // элементтерінің мәнін енгізетін әдістің өзін жариялау

 {
 unsigned int i,j;
for(i=0; i<GOL; i++)
 for(j=0; j<BAGAN; j++)
 elemMat[i][j]=random(100)%13-3;    //матрица элементтерінің мәндерін
                                    // кездейсоқ сандар генераторы анықтайды
 return * this;
 }
//-------------------------------------------------------------------------------
 void matrix:: koru()  // кездейсоқ сандар генераторы анықтаған матрица
```

```
// элементтерінің мәндерін баспаға шығарып көрсететін әдістің өзін жариялау
 {
  unsigned int i,j;
  cout<<"\n matrixti koru :" ;
  for (i=0; i<GOL; i++)
    {
          cout<<"\n";
 for (j=0; j<BAGAN; j++)
    cout<<"\t"<< elemMat[i][j];
    }
 }
//-------------------------------------------------------------------------------
matrix  matrix:: operator = (matrix &M) //matrix класы үшін меншіктеу (=)
                        // амалын  қайыра жүктейтін әдістің  өзін жариялау
   {
    unsigned int i,j;
    GOL=M.GOL;  BAGAN=M.BAGAN;
    for (i=0; i<GOL; i++)
    for (j=0; j<BAGAN; j++)
    elemMat[i][j]=M.elemMat[i][j];
    return * this;
 }
// ------------------------------------------------------------------------------
matrix  matrix:: operator + (matrix &M) // matrix класы үшін қосу  (+)  амалын
                              // қайыра жүктейтін әдістің  өзін жариялау
{
    unsigned int i,j;
if((GOL!=M.GOL)||(BAGAN!=M.BAGAN))     //қосу амалы орындалатын
                                    //матрицалардың өлшемдерін тексеру
   {
 cout<<"\n matrixterdin olshemderi saikes emes!!!";
 getch();
 exit(-1);  // программаның орындалуы аяқталды
   }
 matrix S(GOL,BAGAN);     // matrix типті S қосынды матрицаны жариялау
   for(i=0; i<GOL; i++)
```

```
    for(j=0; j<BAGAN; j++)
 S.elemMat[i][j]=elemMat[i][j]+M.elemMat[i][j];
    return S ;
}
//-------------------------------------------------------------------------------
matrix matrix::operator *(matrix &M) //matrix класы үшін көбейту (*) амалын
                                     // қайыра жүктейтін әдістің өзін жариялау
 {
 unsigned int i,j,k;
 if((BAGAN!=M.GOL))   //көбейту орындалатын матрицалардың бағандары
                                     //мен жолдарының саны бірдей болуын тексеру
    {
 cout<<"\n matrixterdin bagandari men goldarinin sani birdei emes!!!";
 getch();
 exit(-1);
    }
matrix P(GOL,M.BAGAN); // matrix типті P көбейтінді матрицаны жариялау
for(j=0; j<M.BAGAN; j++)
for(i=0; i<GOL; i++)
    for(k=0; k<M.GOL; k++)
 P.elemMat[i][j]=P.elemMat[i][j]+elemMat[i][k]*M.elemMat[k][j];
 return P ;
 }
//-------------------------------------------------------------------------------
int main()                          // негізгі программа басталды
    {
  matrix A(4,4), B(4,4); // matrix типті A және B объектілерін жариялау
  A.engizu(); cout<<" \n A "; A.koru(); // A объектісі үшін әдістерді шақыру
  B.engizu(); cout<<" \n B " ; B.koru(); //B объектісі үшін әдістерді шақыру
  cout<<" \n A + B ";(A+B).koru(); // A+B қосу амалын орындап, нәтижені көру
  cout<<" \n A * B ";(A*B).koru(); // A*B көбейту амалын орындап,
                                     // нәтижені көру
  matrix C(3,2), D(3,3);  // matrix типті C және D объектілерін жариялау
  C.engizu(); cout<<" \n C "; C.koru(); // C объектісі үшін әдістерді шақыру
  D.engizu(); cout<<" \n D " ; D.koru(); // D объектісі үшін әдістерді шақыру
  cout<<" \n C + D ";(C+D).koru(); // C+D қосу амалын орындап, нәтижені көру
```

```
cout<<" \n C * D ";(C*D).koru(); //C*D көбейту амалын орындап, нәтижені көру
getch();
}
```

Программа орындалуының нәтижесі келесі түрде болады:

```
A   matrixti koru :
    7    1    5    2
    9    2    1    5
    5    6    6    2
    -3   3    1    -2

B   matrixti koru :
    -3   4    3    3
    6    -3   -2   8
    1    9    -1   4
    4    9    -2   0

A + B   matrixti koru :
    4    5    8    5
    15   -1   -1   13
    6    15   5    6
    1    12   -1   -2

A * B   matrixti koru :
    -2   88   10   49
    6    84   12   47
    35   74   -7   87
    20   -30  -12  19

C   matrixti koru :
    5    -1
    1    5
    2    7

D   matrixti koru :
    3    5    3
    4    8    7
    2    2    9

C + D   matrixterdin olshemderi saikes emes!!!
```

Бұл программадағы

```
matrix:: matrix( unsigned int gol, unsigned int bagan);
```

конструктор-әдістің қызметі gol X bagan өлшемді болашақ матрицаның элементтері үшін жадыдан орын алу болып табылады. Сондықтан негізгі программадағы matrix типті A(4,4); немесе C(3,2), объектілерінің жарияланулары орындалғанда, сәйкесінше 16 немесе 6 элемент үшін **new** амалының көмегімен жадыдан орын алынады. Бұл алынған орындарға немесе ұяшықтарға матрица элементтерінің мәндерін жіберетін matrix engizu() әдісіндегі келесі команда

```
elemMat[i][j]=random(100)%13-3;
```

болып табылады.

Бұл программадағы matrix класы үшін қайыра жүктелген «меншіктеу» (=), « қосу» (+) және «көбейту» (*) амалдарының барлығы да бинарлық амалдар, сондықтан да осы аталған амалдарға сәйкес қайыра жүктелген амал-функциялардың үшеуінде де,

```
matrix matrix::operator *(matrix &M)
matrix  matrix:: operator + (matrix &M)
matrix  matrix:: operator = (matrix &M)
```

екінші параметр ретінде matrix типті M объектіге сілтеме беріліп тұр, яғни осы сілтеме бойынша адрестегі мән, амалдардың екінші операндалары болып есептеледі. Бұл амалдарды қайыра жүктеудегі тағы бір назар аударатын нәрсе, қосынды матрицаны немесе көбейтінді матрицаларды жариялау болып табылады, мысалы, қосу амалын қайыра жүктеген кезде қосынды матрица

```
matrix S(GOL,BAGAN);
```

түрінде функция денесінде жергілікті айнымалы немесе объект түрінде жарияланды. Сондай-ақ, қосу немесе көбейту амалдары орындалмас

бұрын операндалардың орындарындағы матрицалардағы жолдар мен баған сандарының сәйкес болуы да тексерілуі қажет. Бұл тексеруді негізгі программада да жасауға болады, ал осы жолы тексеру амалдарды қайыра жүктеу кезінде тексеріліп кетеді.

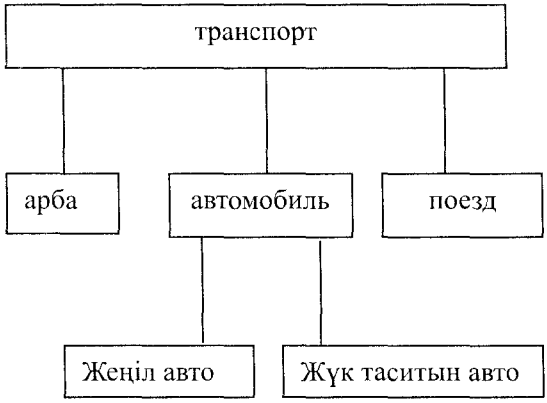
Бұл программада matrix класы мен оның барлық әдістері негізгі программамен бірге бір файлда жарияланып тұр және де жоғарыда келтірілген жаттығуда матрицаға қолданылатын қосу, көбейту және меншіктеу амалдарын қайыра жүктеу ғана қарастырылды. Жалпы матрицаны анықтайтын matrix класында матрицамен жасалатын барлық амалдарды, мысалы, анықтауышын табуды, минорларды есептеуді, бірлік матрицасын алуды және т.б. амалдардың барлығын анықтап және оны өз алдына жеке модуль ретінде құрып алса, онда матрицаларға қатысты кез келген есептерді шешу программаларында matrix класын жариялауды қайталап жатпай-ақ, сол құрылған модульді пайдалана беруге болады.

6.7. Қабылдаушылық

Объектіге бағдарланған программалаудағы негізгі қағидалардың бірі – *қабылдаушылық механизмі* программада кластарды пайдалануды реттеп отыру үшін қажет болады. Мысалы, «транспорт» саласына немесе пәндік облысына жататын «арба», «автомобиль», «поезд», «самолет» сияқты транспорт түрлерінің әрқайсысы үшін бір-бірімен байланысы жоқ тәуелсіз кластар құруға болар еді, бірақ онда бұл транспорт түрлерінің барлығына ортақ жылдамдық, салмақ, және т.б. сипаттамалар әрбір класта қайталанып отыратын болады және оларға қолданылатын амалдарды, салыстыруларды және т.б. әдіс-әрекеттерді, яғни кластар арасындағы байланыстарды да бірыңғай бір түрге келтіру қажет болар еді.

Ал программадағы кластардың және ол кластарға жататын объектілердің саны жүздеген, мыңдаған болған кезде бұл байланыс қалай жүзеге асырылар еді? Міне, бұл мәселенің шешуі кластар үшін қолданылатын *қабылдаушылық механизмі* болып табылады. *Қабылдаушылық механизмі* бойынша, пәндік облыстың түрліше өкілдеріне тән барлық

ортақ қасиеттері мен әдістері бір класқа біріктіріледі және ол класс негізгі, *түпкі кластың* (базовый класс, родительский класс) қызметін атқаратын болады, ал олардың өздеріне ғана тән, қалған қасиеттері мен әдістері үшін анықталатын өз кластарын, түпкі класқа бағынатын *тума класс* (производный класс, дочерний класс) ретінде жариялайды. Осылайша, түпкі кластар мен олардың тума кластарының арасындағы бір-біріне бағыныштылығы немесе кластар иерархиясы пайда болады (6.1-сурет).



6.1-сурет. *Кластар иерархиясының мысалы (жай қабылдаушылық)*

Кластар иерархиясы бойынша тума класс өзінің түпкі класының элементтерін (қасиеттері мен әдістерін) қабылдап алып отырады, бірақ оларды пайдалануға, яғни өзгертуге рұқсат, белгілі бір ережелерге сәйкес беріледі (6.1-кесте). Сондай-ақ, C++ программалау тілінде тума класс үшін бір емес бірнеше түпкі кластың болу мүмкіндігі де қарастырылған. Осыған байланысты қабылдаушылықтың келесі екі түрі пайдаланылады:

– *жай қабылдаушылық* (одиночное или простое наследование), мұнда тума класс үшін түпкі класс біреу ғана болады;

– *көп қабылдаушылық* (множественное наследование), мұнда тума класс үшін түпкі класс екі немесе одан да көп болады.

Тума кластың түпкі кластың элементтерін (қасиеттері мен әдістерін) пайдалануын реттеп немесе пайдалану статусын анықтап отыру үшін **private, protected, public** спецификаторлары қолданылады, яғни тума класты жариялағанда түпкі кластарды осы үш спецификатордың біреуінің көмегімен көрсетуге болады, яғни:

```
class тума_класс: private  түпкі_класс
  {
     … … …
  };
немесе
class тума_класс:  protected  түпкі_класс
  {
     … … …
  };
немесе
class тума_класс:   public  түпкі_класс
  {
     … … …
  };
```

Көп қабылдаушылықта немесе тума класс үшін түпкі кластар бірнешеу болған кезде, оны жариялауда түпкі кластарды «үтір» арқылы тізіп көрсетеді және де әрбір түпкі кластың алдына оның элементтерін тума класта пайдалану статусын анықтайтын спецификаторлар жекежеке жазылатын болады, мысалы:

```
class тума_класс:   түпкі_класс1, protected түпкі_класс2, public
түпкі_класс3
  {
     … … …
  };
```

жоғарыда көрсетілген жариялауда түпкі_класс1 үшін спецификатор берілмеген, себебі класты (тума класты) жариялағанда **private, protected, public** спецификаторларының бірде-біреуі көрсетілмесе,

онда компилятор әдеттегідей (по умолчанию) оның спецификаторын **private** деп өзі автоматты түрде тағайындайды. Егер тума ретінде класс емес құрылым (**struct**) алынатын болса, онда әдеттегідей (по умолчанию) спецификатор ретінде **public** тағайындалады.

Бұл **private, protected, public** спецификаторлары осыған дейін кластың ішіндегі элементтеріне қолданылып келді. Ендігі жерде олар қабылдаушылыққа байланысты тума кластардың түпкі кластардағы элементтерді пайдалану реттерін анықтау үшін тағы да қолданылатын болып отыр, сондықтан спецификаторлар арасында белгілі бір ережелер сақталады, оны келесі кестеден көруге болады (6.1-кесте):

6.1-кесте

Тума класс үшін түпкі кластың элементтерін пайдалану ретін тағайындау

Пайдалануды реттеуші спецификаторлар	**Түпкі кластағы спецификаторлар**	**Тума кластың пайдалану статусы**
private	private	пайдалана алмайды
	protected	private
	public	private
protected	private	пайдалана алмайды
	protected	protected
	public	protected
public	private	пайдалана алмайды
	protected	protected
	public	public

Бұл кесте бойынша, мысалы, тума класс, реттеуші спецификаторлардың қайсысын пайдаланса да бәрібір, түпкі кластың **private** бөліміндегі элементтерін пайдалана алмайды, оларға тек қана сол, түпкі кластың өзінің әдістері ғана қолданылады. Сол сиякты реттеуші спецификатор **private** болса, онда түпкі кластың **protected** және **public** бөліміндегі элементтері, тума класта private статусына ие болады.

Реттеуші спецификатор ретінде **public** қолданылса, сонда ғана тума класс түпкі кластың **public** бөліміндегі элементтерін **public** статусымен пайдалана алады екен. Сондықтан кластарды жариялағанда қандай элементті қай бөлімде жариялау керектігі алдын ала анықталып, зерттеліп дұрыс жасалуы керек.

Сондай-ақ, қабылдаушылық механизмі бойынша кластың түрлі қызмет атқаратын әдістері де оның тума кластарында түрліше қабылданатын болады:

– конструкторлар қабылданбайды, сондықтан әрбір тума класс үшін оның өзінің конструкторы анықталған болуы керек, егер тума класта түпкі кластың конструкторын пайдалану ашық түрде берілмесе, түпкі кластың әдеттегідей конструкторы (конструктор по умолчанию без параметров) автоматты түрде шақырылатын болады. Егер тума класс үшін түпкі кластар бірнешеу болатын болса, онда олардың конструкторлары жариялану реті бойынша шақырылады;

– деструкторлар да қабылданбайды, олардың конструкторларымен салыстырғандағы өзгешелігі тума кластарда түпкі кластардың деструкторларын ашық түрде шақырудың қажеті жоқ олар автоматты түрде өздері шақырылатын болады. Егер тума класс үшін түпкі кластар бірнешеу болатын болса, онда барлық түпкі кластардың деструкторлары конструкторлардың орындалу ретіне кері ретпен шақырылады;

– меншіктеу амалы да қабылданбайды, сондықтан тума класта ол ашық түрде анықталуы керек;

– түпкі кластың қалған әдістерін тума кластарда пайдалануға және оларды қайтадан анықтап (переопределение метода) қолдануға болады. Түпкі кластың тума класта қайта анықталған әдісін қолдану үшін әрекет ету облысын тағайындау амалын (::) пайдаланады.

6.7. 1. Жай қабылдаушылық

Кластар иерархиясы бойынша анықталатын тума класс үшін түпкі класс біреу ғана болса, онда мұндай қабылдаушылықты *жай қабылдаушылық* деп атайды.

6.8-жаттығу. Келесі программа transport класынан тарайтын avtomobil тума класының қызметін көрсететін болады.

```
#include <iostream.h>
#include <conio.h>
 #include <iomanip.h>
//---------------------------------------------------------------------------
class transport          // түпкі класты жариялау басталды
{
  private:
      string ati, marka;  // транспорттың аты және маркасы
  public:
      unsigned int gili;  // транспорттың шыққан жылы
       float bastBaga;   // транспорттың бастапқы бағасы

transport()  // transport класының әдеттегідей (по умолчанию) конструкторы
      {"0","0",0,0;};
void engizu (string ATI, string MARKA,unsigned int GILI, float BASTBAGA
)
 {          // transport класының транспорттың  атын, маркасын, жылын,
            // бастапқы бағасын енгізетін әдісі
   ati=ATI; marka=MARKA; gili=GILI;  bastBaga=BASTBAGA;
 };

string Ati()  { return ati; }  // транспорттың  атын көрсететін әдіс
 string Marka() { return marka; } // транспорттың маркасын көрсететін әдіс
 unsigned int Gili() { return gili;} //транспорттың жылын  көрсететін әдіс
float  BastBaga() {return bastBaga; } // бастапқы бағасын көрсететін әдіс
};                                    // түпкі класты жариялау аяқталды

//-------------------------------------------------------
class avtomobil: public transport   //тума  класты жариялау басталды
{
   private :
```

```
    unsigned int reiting; // автомобильдің рейтингісі
  public:
 void reitEng (unsigned int REITING ) // avtomobil класының автомобильдің
                                      // рейтингісін енгізетін әдісі
            { reiting= REITING; }
 unsigned int  Reiting () // автомобиль рейтингін көрсететін әдіс
            {return reiting;}
 float  avtoBagasi()  // автомобильдің сатылу бағасын есептейтін әдіс
    {
     float s=0;
     if( gili>=2010 && reiting >=10)
        s =bastBaga;
     if( gili>=2010 && reiting <=10)
        s =bastBaga-0.05*bastBaga;
     if( gili>=2005 &&  gili<=2010)
        s =bastBaga-0.1*bastBaga;
     if( gili<=2005)
        s =bastBaga-0.3*bastBaga;
    return s;
    }
}; // тума класты жариялау аяқталды

const n=5;
int main()              // негізгі программа басталды
{
    avtomobil a[n]; // avtomobil класының массив-объектісін жариялау
    string ati, marka;
    unsigned int  gili,reiting;
    float BastBaga;
for(int i=0;i<n;i++)
  {
        // автомобильдің атын, маркасын, жылын, бастапқы бағасы және
        // рейтингін енгізу
        cout<<i<<" – avtomobildin ati :\n"; cin>>ati;
```

```
cout<<"\n markasi :\n"; cin>>marka;
cout<<"\n gili :\n"; cin>>gili;
cout<<"\n  BastBaga:\n"; cin>>BastBaga;
cout<<"\n reiting :\n"; cin>>reiting;
a[i].engizu(ati, marka,gili,BastBaga );
a[i].reitEng(reiting);
 }
cout<<"\n  Ati    : Marka   : Gili: Reiting :  BastBaga : SatiluBagasi";
cout<<"\n ----------------------------------------------------------------------";
                    // автомобильдің атын, маркасын, жылын, рейтингін және
                    // сатылу бағасын шығару
for(int i=0;i<n;i++)
cout<<"\n"<<a[i].Ati()<<"    "<<a[i].Marka()<<"    "<<a[i].Gili()<<"  "
<<a[i].Reiting()<<"  "<<a[i].BastBaga()<<"$  "<<a[i].avtoBagasi()<<"$";
getch();
}
```

Мұнда алдымен түпкі кластың қызметін атқаратын transport класы құрылады, бұл класта транспорттың «аты», «маркасы», «шыққан жылы», «бастапқы бағасы» сияқты барлық транспорт түріне тән ортақ қасиеттер анықталды. Транспорттың «аты» мен «маркасына» сәйкес string ati, marka; қасиеттері түпкі кластың **private** бөлімінде жарияланады, сондықтан бұл қасиеттермен жұмыс жасауға сол түпкі кластың өзінің әдістеріне ғана рұқсат етіледі, ал тума кластың әдістері бұл қасиеттермен жұмыс жасай алмайды. Сондай-ақ, транспорттың «шыққан жылы» мен «бастапқы бағасына» сәйкес, **unsigned int** gili; және **float** bastBaga; қасиеттері, кейіннен анықталатын тума класта, автомобильдің сатылу бағасын анықтау үшін қолданылатындықтан, сәйкесінше **public** бөлімде жарияланатын болады. Программа орындалуының нәтижесі келесі түрде болады:

0 - avtomobildin
ati: Прадо
markasi: Тойота
gili: 2011
BastBaga: 70000
reiting: 10

1 - avtomobildin
ati: Лексус
markasi: Тойота
gili: 2007
BastBaga: 65000
reiting: 10

2 - avtomobildin
ati: Тюксон
markasi: Хундай
gili: 2009
BastBaga: 27000
reiting: 7

3 - avtomobildin
ati: Тауерик
markasi: Фольксваген
gili: 2005
BastBaga: 50000
reiting: 9

4 - avtomobildin
ati: Майбах
markasi: Мерседес
gili: 2011
BastBaga: 200000
reiting: 20

Ati:	Marka:	Gili:	Reiting:	BastBaga:	SatiluBagasi
Прадо	Тойота	2011	10	70000$	66500$
Лексус	Тойота	2007	10	65000$	58500$
Тюксон	Хундай	2009	7	27000$	24300$
Тауерик	Фольксваген	2005	9	50000$	35000$
Майбах	Мерседес	2011	20	200000$	200000$

Программа ары қарай талданатын болса, онда тума класс ретінде құрылатын avtomobil класы үшін конструктор анықталған жоқ, сондықтан бұл кластың объектілерін түпкі кластың **public** бөлімінде әдеттегідей анықталған, параметрлері жоқ

```
transport()
{"0","0",0,0;};
```

конструкторы құрады. Сол сияқты avtomobil класында анықталған, автомобильдің сатылу бағасын есептейтін келесі әдіс:

```
float avtoBagasi()  // автомобильдің сатылу бағасын есептейтін әдіс
   {
    float s=0;
    if( gili>=2010 && reiting >=10)
       s =bastBaga;
    if( gili>=2010 && reiting <=10)
       s =bastBaga-0.05*bastBaga;
    if( gili>=2005 &&  gili<=2010)
       s =bastBaga-0.1*bastBaga;
    if( gili<=2005)
       s =bastBaga-0.3*bastBaga;
   return s;
   }
```

автомобильдің шыққан жылы мен рейтингіне байланысты бастапқы бағасына өзгерістер енгізіп, оның сатылу бағасын анықтайды, мысалы, шыққан уақыты 2006 жылдан төмен болатын автомобильдердің бағасы бастапқы құнынан 30 пайызға (%) арзан бағамен сатылатынын есептеп бере алады.

6.7.2. Виртуал әдістер

Объектіге бағдарланған программалауда класс объектілерімен жұмыс жасау негізінен көрсеткіштер арқылы жүргізіледі, себебі кластар мен олардың объектілері көбейген сайын оларға қажет орынды ди-

намикалық жадыдан алған әлдеқайда тиімді болып келеді. Сондай-ақ, C++ -те түпкі класс үшін жарияланған көрсеткішке оның кез келген тума кластары объектілерінің адрестерін беру механизмі қарастырылған. Жұмыс жасау қағидасы осы механизмге негізделген, түпкі класта virtual қызметші сөзінің көмегімен арнайы жарияланып, сол кластың бір немесе бірнеше тума кластарында қайыра жүктелетін әдістерді (функцияларды) *виртуал әдістер* деп атайды. Бұл механизм бойынша түпкі класта анықталған виртуал әдістер, тума кластар объектілерінің бәрінде бірдей, сол түпкі класс үшін жарияланған көрсеткіш арқылы шақырылады, ал оның әрбір тума кластың объектілері үшін дәл қай вариантын орындау керек екені сол көрсеткішке тұрған адрес арқылы анықталады. Яғни, көрсеткішке тұрған адрес қай объектінікі болса, онда виртуал әдістің сол объект жататын тума кластағы қайыра жүктелген варианты орындалатын болады. Мұндай қайыра жүктеу көрсеткіштер арқылы жасалатын болғандықтан, ол программаның орындалуы кезінде ғана мүмкін болады, сондықтан мұны *динамикалық полиморфизм* деп атайды және виртуал әдістердің қарапайым функцияларды қайыра жүктеуден ерекшелігі де осы болып табылады (4.5-тақырыпты қараңыз). Сол сияқты виртуал әдістердің объектіге бағдарланған программалаудағы полиморфизм қағидасын жүзеге асырудағы басты құрал болып табылатынын да ескерген жөн. Қарапайым функцияларды қайыра жүктеуден ажырату үшін кластардағы қайыра жүктелетін әдістердің алдына virtual қызметші сөзі жазылады, оны бір-ақ рет түпкі класта жазып кетсе жеткілікті, тума кластарда қайталаудың қажеті жоқ.

6.9-жаттығу. Келесі программа виртуал әдістің қызметін демонстрациялайды.

```
#include <iostream.h>
#include <conio.h>
class rectangle // түпкі класты жариялау
  {  public:
 virtual void ati()  // түпкі класта виртуал әдісті  жариялау
  { cout<<"\n Bul rectangle ";}
  };
 class romb :public rectangle   // тума  класты жариялау
```

```
 {  public:
 void ati()  // тума класта виртуал әдісті жариялау
   {cout<<"\n Bul romb";}
  };
 class parallelogram :public rectangle  // тума класты жариялау
  {  public:
 void ati()  // тума класта виртуал әдісті жариялау
 {cout<<"\n Bul parallelogram";}
  };
int main()
{
 rectangle R;              // түпкі класс объектісін жариялау
 rectangle *ukR=0;         // типі түпкі класс болатын көрсеткішті жариялау
 parallelogram P;          // тума класс объектісін жариялау
 romb Ro;                  // тума класс объектісін жариялау
  ukR=&R;                  // көрсеткішке түпкі класс объектісінің адресін беру
 ukR->ati();               // түпкі класс объектісі үшін виртуал әдісті шақыру
  ukR=&P;                  // көрсеткішке тума класс объектісінің адресін беру
 ukR->ati();               // тума класс объектісі үшін виртуал әдісті шақыру
 ukR=&Ro;                  // көрсеткішке тума класс объектісінің адресін беру
 ukR->ati();               // тума класс объектісі үшін виртуал әдісті шақыру
 getch();
}
```

Программа орындалуының нәтижесі төмендегідей болады:

```
Bul rectangle
Bul parallelogram
Bul romb
```

Бұл қарапайым программадағы бір ғана атаумен **void** ati() {...}; түрінде анықталған виртуал әдіс, бір ғана көрсеткіштің көмегімен үш түрлі нәтиже көрсетіп тұрғанын байқауға болады.

Виртуал әдістері бар кластарды *полиморфты кластар*, ал оның объектілерін *полиморфты* объектілер деп атайды.

Виртуал әдістер үшін келесі ережелер орындалуы тиіс:

– виртуал әдістердің түпкі және тума кластардағы жарияланулары-рындағы атаулары мен параметрлері бірдей болуы керек, егер олардың параметрлері бір-бірінен өзгеше (типінде немесе санында айырмашылық) болса, онда бұл жай ғана функцияны қайыра жүктеу болып табылады, яғни әдіс өзінің «виртуал» деген мәнін жоғалтады;

– түпкі класта анықталған виртуал әдістер оның тума кластарына да беріледі, сондықтан тума кластарда виртуал әдістерді жаңа, өзгеше бір әрекеттерді орындау үшін аса қажет болмаса қайыра жүктемейді;

– виртуал әдістегі маңызды деген әрекеттерді тума класта анықтау керек болған жағдайларда, оны (виртуал әдісті) түпкі класта *таза виртуал әдіс* түрінде анықтайды, оның жазылуы келесі түрде болады:

virtual типі әдіс_аты (параметрлер тізімі)=0;

6.7.3. Абстракт кластар

Кластың кем дегенде бір *таза виртуал әдісі* бар болса, онда мұндай кластарды *абстракт кластар* деп атайды. Абстракт кластар маңызды деген мәселелер тума кластарда нақтыланып анықталатын жағдайларда, тума кластар үшін жалпылама деп есептелетін деректерді беретін түпкі кластардың қызметін атқарады. Абстракт кластар түпкі класс ретінде ғана анықталады. Типі абстракт класс болатын объектілерді жариялауға болмайды, себебі мұндай объект үшін таза виртуал әдісті тікелей шақыру болсын не жанама шақыру болсын, бәрібір ол қатеге әкеп соқтырады.

Түпкі класс абстракт класс болғанымен, оның объектісін жариялау мүмкін емес болғанымен, типі абстракт класс болатын көрсеткіштерді жариялауға болады. Бұл көрсеткіштерді виртуал әдістерді шақыру үшін абстракт кластың тума кластары пайдалана алады.

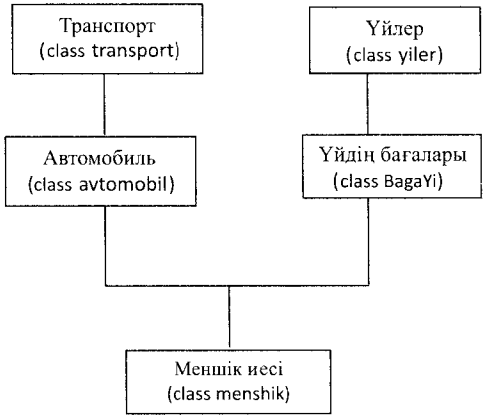
6.7.4. Көп қабылдаушылық

Кластар иерархиясы бойынша анықталатын тума класс үшін түпкі класс екі немесе одан да көп болса, онда мұндай қабылдаушылықты *көп қабылдаушылық* деп атайды. Мысалы, 6.2-суреттегі menshik класы avtomobil және BagaYi деп аталатын кластардың екеуінің де ұрпағы бо-

лып есептеледі. «Көп қабылдаушылық» қағидасына сәйкес анықталған, бұл menshik класының программадағы жариялануы, мысалы, келесі түрде болуы мүмкін:

```
class menshik :  public avtomobil, public BagaYi
    {
      ... ... ...
     };
```

мұндағы **public** спецификаторының орнына, **private** немесе **protected** спецификаторларының кез келгенін пайдалана беруге болады, бірақ олардың түпкі класс элементтерін тума класта пайдалануды реттеп отыратыны ескерілуі керек (6.7.1-кесте).



6.2-сурет. *Кластар иерархиясының мысалы (көп қабылдаушылық)*

6.10-жаттығу. Келесі программа «көп қабылдаушылық» қағидасына негізделіп құрылған «menshik» тума класын құруды және оның мүмкіндіктерін қолдануды көрсете алады. Бұл есеп бойынша автомобилі және үйі бар меншік иесінің осы заттарының құны есептеледі.

```
#include <iostream.h>
 #include <conio.h>
 #include <iomanip.h>
```

```
 class transport    // transport класын жариялау басталды
{
private:
 string ati, marka;
 public:
  unsigned int gili;
  float bastBaga;
   transport()
     {"0","0",0,0;};
  void engizu (string ATI, string MARKA,unsigned int GILI,float BASTBAGA )
     {
 ati=ATI; marka=MARKA; gili=GILI;  bastBaga=BASTBAGA;
     };
 string Ati()   { return ati; }
 string Marka() { return marka; }
 unsigned int Gili() { return gili;}
float  BastBaga() {return bastBaga; }
 } ;   // transport класын жариялау аяқталды

class avtomobil: public transport   // transport класының тума класы  avtomobil -ді
                                    // жариялау басталды
{
 private:
   unsigned int reiting;
 public:
   void reitEng (unsigned int REITING )
      {reiting= REITING;  }
  unsigned int  Reiting () {return reiting;}
   float  avtoBagasi()
     {
       float s=0;
      if( gili>=2010 && reiting >=10)
         s =bastBaga;
      if( gili>=2010 && reiting <=10)
         s =bastBaga-0.05*bastBaga;
      if( gili>=2005 &&  gili<=2010)
```

```
        s =bastBaga-0.1*bastBaga;
      if( gili<=2005)
        s =bastBaga-0.3*bastBaga;
     return s;
     }
};   // тума класс  avtomobil -ді  жариялау аяқталды
 class yiler   // yiler класын жариялау басталды
{
 public:
  string tyri;  // үйдің түрі коттедж, квартира және т.б.
   unsigned int salgili;  // үйдің салынған жылы
   float audani;  // үйдің ауданы немесе шаршы метрі
  yiler()
    {"0",0,0;};  // yiler класының конструкторы
void engizuYi (string TYRI, unsigned int SALGILI,float AUDANI )
 {
    tyri=TYRI; salgili=SALGILI;  audani=AUDANI;
 };
 string Tyri()   { return tyri; }
 unsigned int SalGili() { return salgili;}
float  Audani() {return audani; }
 } ;  // yiler класын жариялау аяқталды
class BagaYi: public yiler   // yiler класының тума класы BagaYi -ді
                             // жариялау басталды
{
  public:
     float metrKv;  // бір шаршы метрдің бағасы
     float b;
  float  YiBagasi()   // үйдің жалпы құнын  есептейтін әдіс-функция
      {
         b=metrKv*audani;  // үйдің бағасын есептеу
         return b;
      }
};  //  тума класс  BagaYi-ді жариялау аяқталды

 // -------- КӨП ҚАБЫЛДАУШЫЛЫҚ БАСТАЛДЫ --------
```

```
class menshik : public avtomobil, public BagaYi      // avtomobil және BagaYi
                // кластарының тума класы болатын menshik класын
                // жариялау басталды
{
  private:
string fam;    // меншік иесінің фамилиясы
public:
void engFam (string FAM)
 {
  fam=FAM;
 };
   string Fam()  { return fam; }
}; // menshik класын жариялау аяқталды
int main()
{
 menshik menIesi;  // menshik класының объектісін жариялау
 string fami,ati, marka,tyri;
 unsigned int  gili,reiting, salgili;
 float BastBaga,audani, kbMetrBaga;
 // ------Меншік иесінің фамилиясын енгізу ----
cout<<"menshik Iesinin fam-sin engizu:\n ";
 cin>>fami;
 menIesi.engFam(fami);
 // ------ Меншік иесінің автомобилі туралы деректерді енгізу басталды ----
cout<<"\n\n menshik Iesinin avtomobili turali engizu:\n ";
 cout<<" – avtomobildin ati : "; cin>>ati;
 cout<<"\n markasi : "; cin>>marka;
 cout<<"\n gili : "; cin>>gili;
 cout<<"\n  BastBaga: "; cin>>BastBaga;
 cout<<"\n reiting : "; cin>>reiting;
 menIesi.engizu(ati, marka,gili,BastBaga);
 menIesi.reitEng(reiting);
 // ------Меншік иесінің үйі туралы деректерді енгізу басталды----
cout<<"\n\n menshik Iesinin yileri turali engizu:\n ";
 cout<<" – yidin tyri : "; cin>>tyri;
cout<<"\n yidin audani: "; cin>>audani;
```

```
    cout<<"\n salgan gili : "; cin>>salgili;
    cout<<"\n  KbMetrdin bagasi: "; cin>>kbMetrBaga;
    menlesi.engizuYi(tyri,salgili,audani);
    menlesi.metrKv=kbMetrBaga;
    // ------Меншік иесінің дүние-мүлкі бағасын есептеу басталды----
     cout<<"\n\n azamat(sha) "<<menlesi.Fam()<<" \n\n  geke mensik
dyniesinin  kuni " << (menlesi.avtoBagasi()+menlesi.YiBagasi())<< "$"<<"\n
onin ishinde  avtonin bagasi- "<< menlesi.avtoBagasi()<<"$, kvartira bagasi-
   "<<menlesi.YiBagasi()<<"$";
      getch();
   }
```

Программаны орындағанда шығатын нәтиже келесі түрде болады:

```
menshik lesinin fam-sin engizu:
Сапарова

menshik lesinin avtomobili turali engizu:
– avtomobildin ati :  Лексус
          markasi :  Тойота
             gili :   2005
         BastBaga:  70000
           reiting :  10

menshik lesinin yileri turali engizu:
–  yidin tyri :  квартира
 yidin audani:  150
   salgan gili :   2010
 KbMetrdin bagasi:  1400
azamat(sha) Сапарова
 geke mensik dyniesinin kuni 259000$
onin ishinde avtonin bagasi – 49000$, kvartira bagasi- 210000$
```

Бұл программадағы түпкі класы екеу болатын menshik класында, меншік иесінің фамилиясы ғана жарияланды, ал қалған мүмкіндіктердің барлығын, ол өзінің түпкі кластары болатын avtomobil және BagaYi кластарынан және сәйкесінше олардың арғы аталары болатын transport және yiler кластарынан қабылдап тұрғанын көруге болады.

Көп қабылдаушылықты пайдаланып анықталатын кластарда кездесетін негізгі мәселелерге мыналар жатады:

– идентификаторлар арасындағы конфликт, бұл түпкі кластарда атаулары бірдей идентификаторлардың кездесуінен болады, яғни компилятор қай кластағы идентификаторды пайдалану керек екенін ажырата алмайды, сондықтан бұл мәселені шешу үшін әрекет ету аймағын тағайындау амалын қолданады;

– тума кластың түпкі кластарының өздерінің, арғы түпкі класы біреу немесе ортақ болса, онда жаңадан құрылып жатқан тума класта арғы түпкі кластың бірдей элементтері қосарлана қайталанып кетуі мүмкін, сондықтан бұл жағдайды болдырмау үшін арғы түпкі класты виртуал класс түрінде анықтайды.

6.8. Шаблон кластар

Программалауда есептерді шешудің алгоритмдері бірдей болып, бірақ бұл алгоритм типтері әр түрлі болатын деректерге қолданылатын жағдайда, мұндай ортақ алгоритмді шаблон-функцияның көмегімен бір-ақ рет анықтауға болатыны белгілі (4.6-тақырыпты қараңыз) болды. Кластар үшін де осы шаблон механизмін қолдану өте тиімді болып келеді. Мұнда да *шаблон класта* жарияланатын элементтер мен әдістер қандай да болмасын бір типтерге тәуелсіз болу үшін алдын ала шартты түрде анықталатын формальды типтер немесе параметрлер анықталып алынады, сондықтан шаблон кластарды *параметрленген кластар* деп те атайды. Шаблон кластың программадағы жариялануы келесі түрде болады:

```
template <class типАты> class классАты
{
  ... ... ...
}
```

мұндағы **template** кластың шаблон класс екенін хабарлайды, типАты – бұл шартты түрде құрылатын типтің аты, яғни кез келген дұрыс идентификатор. Мысалы:

```
template <class T1> class rectangle
{
 ... ... ...
};
```

Әрі қарай шаблон кластың объектілерін жариялағанда міндетті түрде шаблон кластың қандай типке қолданылатыны анық көрсетілуі керек, программадағы жазылуы:

```
шаблонКлассАты <типі> объектАты;
```

6.11-жаттығу. Программа rectangle деп аталатын шаблон класс құруды және оны пайдалануды көрсетеді. Төртбұрыш қабырғаларының мәндері бүтін сандар (int) болсын, мейлі нақты сандар (double) болсын, бәрібір ол төртбұрыштарды бір ғана rectangle шаблон класының объектілері деп қарастыруға болатынын көрсете алады.

```
#include <iostream.h>
#include <conio.h>

template <class T1> class rectangle  // шаблон класты жариялау басталды
   {
private:
    T1 length;
    T1 width;
public:
rectangle() {assign(0,0);}
rectangle (T1 Len, T1 Wide)
{ assign(Len,Wide);}
T1 Length(){return length;}
T1 Width(){return width; }
T1   Area(){ return length*width;}
void assign(T1 Len, T1 Wide)
{ length=Len;
  width=Wide;
}
```

```
    };    // шаблон класты жариялау аяқталды
int main( )
{
 rectangle <int> rect1 (5,6); // қабырғалары 5 және 6 бүтін сандары болатын
                              // rect1 объектіні (төртбұрышты) жариялау
 cout<<"\n <int> mandi tortbyrish audani = "<<rect1.Area();      // шаблон
                              // кластың Area(); әдісін rect1 үшін шақыру
rectangle <double> rect2 (3.5,7.11);   // қабырғалары 3.5 және 7.11 нақты
                // сандары болатын rect2 объектіні (төртбұрышты) жариялау
cout<<"\n <double> mandi tortbyrish audani = "<<rect2.Area(); // шаблон
                              // кластың Area(); әдісін rect2 үшін шақыру
    getch();
}
```

Программа орындалуының нәтижесі келесі түрде болады:

```
 <int> mandi       tortbyrish audani = 30
<double> mandi   tortbyrish audani = 24.885
```

Программадағы келесі операция,

```
rectangle <int> rect1 (5,6);
```

орындалғанда, rectangle шаблон класында жарияланған T1 шаблон типтің орнына сәйкесінше **int** типі беріледі де, шаблон класта жарияланған барлық қасиеттер мен әдістердің және т.б. элементтердің типі **int** болып өзгереді, тура сол сияқты

```
rectangle <double> rect2 (3.5,7.11);
```

орындалғанда, T1 шаблон типтің орнына **double** типі орналасатын болады.

Жалпы шаблон кластар көбінесе *контейнер кластарды* құруда қолданылады. Контейнер кластарда белгілі бір әдістер бойынша ұйымдастырылған деректер құрылымы, мысалы, массив, тізім, ширет немесе т.б. түріндегі деректер және сол деректермен жұмыс жасауға арналған әдіс-функциялар сақталады. C++ программалау тілінің стан-

дарт кітапханаларында (STL, Standart Template Library) көптеген шаблон кластардың сипаттамалары бар.

6.9. Үйлесімді функциялар мен кластар

Кластың private бөлімінде жарияланған жабық элементтерін, сол кластың өзінде ғана жарияланған өз әдістері ғана пайдалана алатыны белгілі. C++- те кластың осы жабық элементтерін, басқа бір функциялар пайдалана алатындай мүмкіндіктер қарастырылған және мұндай функцияларды *класпен үйлесімді функциялар* деп атайды. Үйлесімді функциялар өзі жабық элементтерін пайдаланатын кластың **public** бөлімінде **friend** қызметші сөзінің көмегімен толық түрде немесе прототипі арқылы жарияланады. Үйлесімді функцияның параметрі кластың объектісі немесе объектіге сілтеме болуы керек.

Программадағы жазылуы:

```
friend функцияТипі функцияАты (объектАты)
{
    ... ... ...
}
```

Мысалы, rectangle класына үйлесімді Area(rectangle rect); функциясын программада жариялау келесі түрде болуы мүмкін:

```
class rectangle
{
 private:
double length; //жабық элемент
double width; //жабық элемент
public:
    friend double Area(rectangle rect); //үйлесімді функцияның прототипі
}
double Area(rectangle rect)
 {return rect.length*rect.width; }   //үйлесімді функцияның жариялануы
```

6.12-жаттығу. Келесі программа rectangle класына үйлесімді Area(rectangle rect); және Perim(rectangle rect) функцияларын жариялауды және олардың қолданылуларын демонстрациялайды:

```
#include <iostream.h>
#include <conio.h>

class rectangle  // класты жариялау басталды
 {
 private:
double length; // жабық элемент
double width; // жабық элемент
      public:
        rectangle() {assign(0,0);}
rectangle (double Len, double Wide)
    { assign(Len,Wide);}
double Length()
    {return length; }
double Width()
{ return width;}
friend double Area(rectangle rect);  // үйлесімді функция прототипі
friend double Perim(rectangle rect);  // үйлесімді функция прототипі
void assign(double Len, double Wide);
 };  //класты жариялау аяқталды
void  rectangle :: assign(double Len, double Wide)
   {
length=Len;
width=Wide;
    }
//---- Үйлесімді функцияларды жариялау басталды-------
double Area(rectangle rect)        //үйлесімді функцияны жариялау
  {
return rect.length*rect.width;
  }
double Perim(rectangle rect)      //үйлесімді функцияны жариялау
   {
```

```
    return 2*(rect.length+rect.width);
  }

int main()
{
  rectangle rect(13,3);
  cout<<"\n"<<" audani – "<<Area(rect);   // үйлесімді функцияны шақыру
  cout<<"\n"<<" perimetr – "<<Perim(rect); // үйлесімді функцияны шақыру
   getch();
}
```

Бұл программа орындалғаннан кейін шығатын нәтиже төменде көрсетілген.

```
audani – 39
perimetr – 32
```

Программада rectangle класымен үйлесімді Area(rectangle rect); және Perim(rectangle rect); екі функцияның жұмысы қарастырылды, бұл екі функцияның прототиптері кластың **public** бөлімінде **friend** қызметші сөзінің көмегімен жарияланды. Функциялардың өздері кластан тыс кәдімгі жай функциялар сияқты жарияланды. Үйлесімді функциялардың программадағы шақырылуы да жай функциялардағы сияқты болады, мысалы,

```
cout<<"\n"<<" audani – "<<Area(rect);
cout<<"\n"<<" perimetr – "<<Perim(rect);
```

Жалпы программалауда класпен үйлесімді функцияларды пайдалану аса тиімді әдіс болып саналмайды, себебі үйлесімді функцияны пайдалану класс элементтерінің оңашалануын бұзады, яғни инкапсуляция қағидасының сақталмауына әкеп соқтырады. Ал екінші жағынан үйлесімді функцияларды пайдалану класс интерфейсін жеңілдетуге, жетілдіруге мүмкіндік береді.

C++ -те үйлесімді функциялармен қатар, *үйлесімді кластар* да қолданылады. Егер қандай да бір кластың барлық әдістері, екінші

бір берілген кластың жабық элементтерін пайдаланатын болса, онда алғашқы класс кейінгі класпен *үйлесімді класс* түрінде жариялануы керек.

Программадағы жазылуы:

```
class классАты
  {
        private:
       ... ... ...
       public:
    friend class  үйлесімдіКлассАты;
   };

class  үйлесімдіКлассАты
 {
   ... ... ...
  void  funk ()
     {
        ... ... ...
      };
 };
```

мұндағы үйлесімді кластың функциялары, мысалы, **void** funk (); берілген класпен үйлесімді функциялар болып есептеледі. Сондай-ақ, бұл функциялардың алдарына **friend** қызметші сөзі жазылмаса да, олар берілген кластың барлық жабық элементтерін пайдалана алады.

Бақылау сұрақтары

1. Объектіге бағдарланған программалаудың қандай қағидалары бар?
2. Инкапсуляция қағидасы не үшін қолданылады?
3. Полиморфизм қағидасының мағынасы неде?
4. Қабылдаушылық қағидасы не үшін енгізілген?
5. Объектіге бағдарланған программалаудағы кластар иерархиясы қалай құрылады?

6. Класс қандай бөлімдерден құралады?

7. Кластардағы private және public спецификаторлары қандай қызмет атқарады?

8. Кластардағы protected спецификаторы қандай қызмет атқарады?

9. Кластың қасиеттері деген не және ол қалай анықталады?

10. Кластың әдісі деген не және ол қалай анықталады?

11. Конструктор және деструктордың қызметі қандай?

12. Оқиға әдіс бола алады ма?

13. Оқиға өңдеуші деген не?

14. Кластың статикалық элементтерін қалай анықтайды?

15. Класс объектісі мен this көрсеткіштің қандай байланысы бар?

16. Кластардағы унарлық және бинарлық амалдарды қайыра жүктеу қалай жүзеге асырылады?

17. Жай қабылдаушылық бойынша жасалған тума кластың жариялануы қалай болады?

18. Көп қабылдаушылық бойынша жасалған тума класс қалай жарияланады?

19. private, protected, public спецификаторларының қабылдаушылық кезіндегі қызметтері қандай?

20. Виртуал әдістер деген не?

21. Таза виртуал әдістер не үшін қолданылады?

22. Виртуал әдістер мен полиморфизм арасында байланыс болады ма?

23. Динамикалық полиморфизм деген не?

24. Полиморфты кластар деп қандай кластарды айтады?

25. Абстракт кластар не үшін қажет?

26. Шаблон кластар қандай қызмет атқарады?

27. template қызметші сөзі не үшін қажет?

28. Контейнер кластар деп қандай кластарды айтады?

29. Үйлесімді функция деген не?

30. Үйлесімді функцияны қалай жариялайды?

31. Үйлесімді класты не үшін қолданады?

Тапсырмалар

1. «Аэропорт» тақырыбына класс құрыңыз. Оның ұшу бағытына және салонның түріне (мысалы, VIP, бизнес, эконом және т.б.) байла-

нысты билет құнын анықтайтын әдісін құрыңыз. Жолаушының фамилиясы, ұшу бағыты, салон түрі және бағасы көрсетілген билетті экранға шығаратын программа құрыңыз.

2. «Архитектуралық нысандар» тақырыбына класс құрыңыз. Оның әдістері мен тума кластарын анықтаңыз. Архитектуралық нысандарды салынған уақыты бойынша сұрыптап экранға шығаратын программа құрыңыз.

3. «Көліктер» тақырыбына класс құрыңыз. Оның әдістері мен тума кластарын анықтаңыз. Қолданушының сұрауы бойынша көлікті іздеп тауып, оны барлық деректерімен қосып экранға шығаратын программа құру керек.

4. «Геометриялық фигуралар» тақырыбына класс құрыңыз. Оның фигуралардың аудандары мен периметрлерін есептейтін әдістерін анықтаңыз. Қажет фигураның ауданы мен периметрін экранға шығаратын программа құру керек.

5. «Геометрия. Стереометрия» тақырыбына класс құрыңыз. Оның денелердің толық беті мен көлемдерін есептейтін әдістерін анықтаңыз. Қажет дененің бетінің ауданы мен көлемін экранға шығаратын программа құру керек.

6. «Компьютердің құрылғылары» тақырыбына класс құрыңыз. Оның әдістерін және «ноутбуки», «дербес компьютер» секілді тума кластарын анықтаңыз. Ең қымбат ноутбукты табатын программа құрыңыз.

7. «Компьютердің құрылғылары» тақырыбына класс құрыңыз. Оның жинақталған компьютердің (сборки) бағасын құрылғылардың бағаларына байланысты анықтап беретін әдісін анықтаңыз. Құрылғылардың бағаларына компьютер бағасын есептейтін программа құрыңыз.

8. «Поликлиника» тақырыбына класс құрыңыз. Оның әдістерін және «Дәрігерлер», «Қаралушылар» секілді тума кластарын анықтаңыз. Бір дәрігерге жазылған қаралушылар тізімін шығаратын программа жазыңыз.

9. «Футбол клубтары» тақырыбына класс құрыңыз. Оның әдістерін және тума кластарын анықтаңыз. Экранға футбол клубтарының рейтингін шығаратын программа жазыңыз.

10. «Кинопрокат» тақырыбына класс құрыңыз. Оның әдістерін және тума кластарын анықтаңыз. Қолданушының сұрауы бойынша белгілі

бір жанрдағы (мысалы, мультфильм, боевик, фантастика және т.б.) фильмдер тізімін шығаратын программа құрыңыз.

11. «Кинофильмдер» тақырыбына класс құрыңыз. Оның әдістерін және тума кластарын анықтаңыз. Бас рольде Х актер ойнайтын фильмдер тізімін экранға шығаратын программа құрыңыз.

12. «Азық-түлік дүкені» тақырыбына класс құрыңыз. Оның әдістерін және тума кластарын анықтаңыз. Дүкендегі сүт өнімдерін сұрыпталған түрде экранға шығаратын программа құрыңыз.

13. «Азық-түлік дүкені» тақырыбына класс құрыңыз. Кластың азық-түлік бағасын оның сақталу мерзіміне байланысты анықтайтын әдісін құрыңыз. Жылдам бұзылатын азық-түліктерді олардың бағасымен және сақталу мерзімімен қоса экранға шығаратын программа құрыңыз.

14. «Кітаптар» тақырыбына класс құрыңыз. Оның әдістерін және тума кластарын анықтаңыз. Аттары бойынша сұрыпталған кітаптар тізімін экранға шығаратын программа құрыңыз.

15. «Мебельдер салоны» тақырыбына класс құрыңыз. Оның әдістерін және тума кластарын анықтаңыз. «Жаңғақ» түсті ас үй гарнитурын іздейтін программаны құрыңыз.

16. «Бір өлшемді массив» тақырыбына класс құрыңыз. Оның массив элементтерін енгізетін және баспаға шығаратын әдістерін құрып, ол әдістердің программада қолданылуын көрсетіңіз.

17. «Автосалон» тақырыбына класс құрыңыз. Оның әдістерін және тума кластарын анықтаңыз. Бір маркаға жататын барлық автомобильдер тізімін толық сипаттамаларымен экранға шығаратын программа құру керек.

18. «Мұражай» тақырыбына класс құрыңыз. Оның мұражайдағы залдарды қарау санына байланысты мұражайға кіру билетінің құнын анықтайтын әдісін құрыңыз. Мұражайға келушінің аты-жөнін, қарайтын залдарын және бағасы көрсетілген билетті экранға шығаратын программа құрыңыз.

19. «Теміржол кассасы» тақырыбына класс құрыңыз. Оның билет құнын жүру бағытына және вагон түріне (мысалы, фирменный, купейный, плацкартный және т.б.) байланысты анықтайтын әдісін құрыңыз. Жолаушының аты-жөні, поезд бағыты, вагон түрі және бағасы көрсетілген билетті экранға шығаратын программа құру керек.

20. «Кітаптар» тақырыбына класс құрыңыз. Оның кітап құнын белгілі бір шарттарға байланысты (мысалы, балалар кітаптары өзінің негізгі құнымен салыстырғанда 10% жеңілдікпен сатылады) анықтайтын әдісін құрыңыз. Белгілі бір автордың кітабын құнымен қоса экранға шығаратын программа құрыңыз.

21. «Мебельдер салоны» тақырыбына класс құрыңыз. Оның әдістерін және тума кластарын анықтаңыз. Барлық мебельдерді бағалары бойынша сұрыптап шығаратын программа құру керек.

22. «Касса» тақырыбына класс құрыңыз. Оның әдістерін және тума кластарын анықтаңыз. Бір тәуліктегі түскен ақшаны есептеу программасын құрыңыз.

23. «Бір өлшемді массив» тақырыбына класс құрыңыз. Оның массив элементтерін өсу реті бойынша сұрыптайтын әдісін құрыңыз және пайдаланып программа жазыңыз.

24. «Студент» тақырыбына класс құрыңыз. Оның студенттің үлгеріміне байланысты шәкіртақы көлемін анықтайтын (мысалы, үздікке – 10% үстемеақы, бір ғана төрті барларға 5% үстемеақы қосылады және т.б.) әдісін құрыңыз. Стипендия алатын студенттердің тізімін стипендия мөлшерімен қоса шығаратын программа құрылсын.

25. «Кітапхана» тақырыбына класс құрыңыз. Оның «Кітапханашылар» және «Оқырмандар» деген тума кластарын анықтаңыз. Белгілі бір кітапханашының қызмет көрсеткен оқырмандарының тізімін шығаратын программа құру керек.

7-тарау. ЖОЛДАР

7.1. Жолдардың түрлері

C++ Builder-де жолдық типтер мәтін түріндегі деректермен жұмыс жасау үшін қолданылады. C++ программалау тілінде берілген деректің мәтін екенін компиляторға хабарлау үшін оны екі жағынан «тырнақша» (мысалы, «бұл_мәтін») белгісіне алынып жазылады.

C++ -те жолдық типтермен жұмыс жасау программалау тілі жетілдірілген сайын өзгерістерге ұшырап, толықтырылып отырды. Бұл әсіресе жолдық типтердің өзін тип ретінде анықтауға байланысты анық байқалады.

Жолдық типтерді пайдалануда C++ Builder- ге дейінгі C++ және Си тілдерінің бастапқы нұсқаларында жолдарды берудің бірнеше тәсілдері қолданылды:

– жолдық типтерді *символдық массивтер* түрінде қарастыру бастапқы Си тілінде қолданылды, бастапқыда жолдарды көрсету үшін арнайы тип болған жоқ, сондықтан жолдар **char** типті символдардан құралатын символдық массивтер ретінде қарастырылды немесе оларды *с-жолдар* деп атады, мұндағы *с* – «**char**»-дың бірінші символы. Си тілінде осы жолдық массивтермен жұмыс жасауға арналған функциялардың өте үлкен және бай кітапханасы қалыптасты және бұл функциялар осы күнге дейін өзінің қажеттілігін жоғалтпай отыр;

– C++ тіліндегі объектіге бағдарланған программалаудың дамуымен байланысты жолдық типті *string* класы түрінде анықтау жүзеге асырылды. *string* класының жолдық массивтермен салыстырғанда біраз артықшылықтары болғанымен, жолдық масивтерді өңдеу функцияларының қуаттылығы программистер үшін маңызды болып қала берді. *string* класы да бірнеше рет қосымша кітапханалар арқылы жетілдіріліп отырды;

– C++ Builder ортасының пайда болуымен қатар жолдарды ANSI стандартына сәйкес анықталған AnsiString типі түрінде қолдану C++-те кеңінен қоданыла бастады.

Программалауда қазіргі уақытқа дейін мәтін түріндегі деректермен жұмыс жасауда бұл тәсілдердің тиімді жақтары аралас қолданылып жүр, яғни программистердің есеп алгоритмінің ерекшелігіне қарай өзіне тиімді тәсілді таңдап алуына мүмкіндік жасалған.

7.2. Жолды символдық массив түрінде анықтау

Си тіліндегі жолдар соңы '\0' (нуль – символ) таңбасымен аяқталатын, символдардан тұратын бір өлшемді массивтер түрінде қарастырылды. Бұл «нуль-символды» нуль-терминатор деп те атайды. Символдық массив түрінде анықталған жол міндетті түрде осы '\0' символымен аяқталған болуы керек, сондықтан символдық массивті немесе жолды жариялағанда '\0' символына берілетін орынды ескеру қажет. Қолданушы жолдың қай жерден аяқталатынын, яғни '\0' символды өзі анық көрсетпесе, мұны компилятор автоматты түрде өзі қоятын болады. Жолдың символдық массив түрінде жариялануы:

char *жолдың_аты[ұзындығы]*;

Мысалы, **char** gol[30]; *//29 символдан және '\0' символынан тұратын жол*

Мұнда жол массив түрінде жарияланатын болғандықтан, жолдың аты да массивтің аты сияқты өзінің алғашқы элементінің мәнін сақтап тұратын көрсеткіш болып табылады. Сондықтан жолды **char*** түріндегі көрсеткіш айнымалы түрінде де жариялауға болады, мысалы:

```
char gol [ ] = "bul_gol";
char *kGol = "bul_gol";
```

түріндегі екі жариялану бәрібір бір нәрсені анықтайды, яғни екі жағдайда да жолдық айнымалыларға "bul_gol" деген мәтіннің алғашқы символының адресін береді.

Жолды символдық массив түрінде жариялағанда, массивтің ұзындығын көрсетпей кетуге де болады, мұндай жағдайда жол ұзындығын енгізілген немесе берілген мәтінге сәйкес компилятор өзі анықтайды, мысалы:

```
char gol [ ] = "bul_gol";
```

жариялануы 'b', 'u', 'l', '_', 'g', 'o', 'l' және '\0' символдарынынан тұратын ұзындығы 8-ге тең массивті анықтайды.

7.2.1. с-жолдарды енгізу-шығару

Программада жолдарды енгізудің бірнеше әдістері қолданылады: жолдың мәнін инициализациялау арқылы анықтау, мысалы:

```
char gol [ ] = "bul_gol";
```

жолдың мәнін пернетақтадан енгізу, ол үшін бұрынғыдай cin объектісі мен ағыннан алу (>>) амалын қолданғанда, жолдың құрамындағы «бос_орын» таңбасы ескерілмей қалатын болады, сондықтан жолды енгізгенде, cin объектісінің getline әдісін пайдаланады, оның жазылуы келесі түрде болады:

```
cin. getline(жол_Аты,sizeof(жол_Аты)-1);     мысалы,
        cin.getline(gol,sizeof(gol)-1);
```

пернетақта секілді енгізуші құрылғыдан енгізілген символдарды оқитын gets(жол_Аты); функциясын пайдалану, бұл функцияның анықталуы <stdio.h> кітапханасында сипатталады. Мұндағы gets(жол_Аты); функциясы Enter пернесі басылғанда пайда болатын «жаңа жол» ('\n') символы кездескенге дейінгі символдардың бәрін оқып жол_Аты айнымалысына береді және оның соңына «нуль-символды» ('\0') тіркеп қояды, мысалы:

```
#include <stdio.h>
... ... ...
gets ( gol );
```

мұнда пернетақтадан дұрыс дерек енгізілмесе немесе EOF символы кездессе, бұл функция NULL мәнді қайтаратын болады;

Жолдарды баспаға шығаруда әдеттегідей cout объектісі мен ағынға қосу (<<) операторын пайдаланады немесе <stdio.h> кітапханасында сипатталған puts(жол_Аты); функциясын да пайдалануға болады. Жолдарды шығаруда бұрынғы С тілінде қолданылған, кейбір программистер әлі де болса қолданатын форматтап шығаруды жүзеге асыратын **int**

printf(**const char** *format[, argument, ...]); функциясын да пайдалануға болады, мұндағы format – шығарылатын жолды қамтитын символдық массив, ал argument – бұл міндетті емес аргументтер. Бұл printf функциясы үшін қолданылатын көптеген форматтаушы спецификаторлар қарастырылған, оны C++ Builder ортасының анықтамалығынан қарауға болады.

7.1-жаттығу. Келесі программа A ... Z дейінгі әріптерді alphabet айнымалысымен анықталған жол түрінде шығарып бере алады және нуль-символ ('\0') көмегімен жолдың соңын өзгертіп отыруға болатынын көрсетеді:

```
#include <iostream.h>
#include <conio.h>
void main()
{
 char alphabet[26];   // 25 әріптен және '\0' символынан тұратын
                      // жолды символдық массив түрінде жариялау
  int index;
  for (char letter ='A', index = 0; letter <'Z',index<26;
  letter++, index++)
  { //A ... Z дейінгі әріптерді alphabet массивіне немесе жолға жазу
   alphabet[index]= letter ;
   }
   cout<<"gol soni  korsetilmegen:  ";
   cout <<alphabet<<"\n\n";
  cout<<"goldin soni [26]-da korsetildi:   ";
   alphabet[26]= '\0'; // жолдың соңын көрсету
   cout <<alphabet<<"\n\n";
 cout<<"goldin soni [20]-da  korsetildi:  ";
  alphabet[20] = '\0'; // жолдың соңын көрсету
  cout <<alphabet;
  getch();
}
```

```
gol soni korsetilmegen: ABCDEFGHIJKLMNOPQRSTUVWXYZ→[ ☺
goldin soni [26]-da korsetildi: ABCDEFGHIJKLMNOPQRSTUVWXYZ
goldin soni [20]-da korsetildi: ABCDEFGHIJKLMNOPQRST
```

Бұл мысалда символдық массив түрінде анықталған жолдық айнымалы alphabet үшін жолдың соңы, '\0' таңбасы көрсетілмеген жағдайда жолдың соңы тексерілмейді, жолды баспаға шығарғанда оның соңы түсініксіз болып шығуы мүмкін, сондықтан да символдық массив түрінде анықталған жолға символдарды меншіктеуде жолдың соңына '\0' берілуі керек.

7.2.2. с-жолдың функциялары (string.h)

Символдық массив түрінде анықталған жолмен массив сияқты жұмыс жасай беруге болады, бұл жолдармен жұмыс жасауда көптеген мүмкіндіктерді пайдалануға жол ашады. Дегенмен, символдық массив түрінде анықталған жолдар үшін қолданылатын арнайы стандарт кітапханаларда жарияланған өте көп функциялар бар (прототиптері string.h тақырыптық файлында анықталған). Төменде кейбір жиі қолданылатын функциялардың қызметі мен прототипі көрсетілген (7.1-кесте):

7.1-кесте

Жолдық функциялар қызметі мен прототипі

Функцияның аты	Қызметі мен прототипі
strlen (str)	str жолдағы символдар (нуль-символдан басқа) санын немесе жолдың ұзындығын береді, жазылуы: size_t strlen(**const char** *str);
strcat (str1, str2)	str2 жолды str1 жолмен біріктіреді және бұл бірігу str1 жолдың жаңа мәні болады, жазылуы: **char** *strcat(**char** *str1, **const char** *str2);

strncat(str1,str2,kol)	str2 жолдың kol символын апарып str1 жолға біріктіріп қосады, алынған жаңа мәнді str1 жолға меншіктейді, жазылуы: **char** *strncat(**char** * str1, **const char** * str2, size_t kol);
strcpy (str1, str2)	str2 жолды str1 жолға көшіреді, жазылуы: **char** *strcpy(**char** * str1, **const char** * str2);
strncpy (str1,str2,kol)	str2 жолдың kol символын str1 жолға көшіреді, жазылуы: **char** *strncpy(**char** * str1, **const char** * str2, size_t kol);
strchr (str,c)	с параметрінде берілген символдың str жолдағы ең алғашқы табылуын анықтайды, жазылуы: **char** *strchr(**const char** *str, **int** c);
strrchr (str,c)	с параметрінде берілген символдың str жолдағы ең соңғы кездесуін анықтайды; **char** *strrchr(**const char** *str, **int** c);
strpbrk(str1,str2)	str2 жолдың кез келген символының str1 жолдағы алғашқы кездесуін анықтайды, жазылуы: **char** *strpbrk(**const char** *str1, **const char** *str2);
strset (str,c)	str жолдағы барлық символды с параметрінде берілген символмен алмастырады, жазылуы: **char** *strset(**char** *str, **int** c);
strnset(str,c,kol)	str жолдағы kol символдарды с параметрінде берілген символмен алмастырады, жазылуы: **char** *strnset(**char** *str, **int** c, size_t kol);
strcspn (str1,str2)	str1 жолдың str2 жолдың символымен сәйкес келетін алғашқы символының орнын анықтайды, жазылуы: size_t strcspn(**const char** *str1, **const char** *str2);
strrev (str)	str жолдың символдарын кері ретпен орналастырады, жазылуы: **char** *strrev(**char** *str);

7.2-жаттығу. Келесі программа символдық массив түрінде енгізілген жолдағы «бос орын» белгісінің қанша рет кездесетінін санап шыға алады және кейбір функциялардың қызметін көрсете алады:

```
#include <iostream.h>
 #include <string.h>
```

```
#include <conio.h>
void main()
{
char str1[20],str2[20]; // str1, str2 жолдарды жариялау
int k=0;  // k- str1 жолдағы «бос орындар»  саны

cout<<"str1- goldi engiz: \n";
cin.getline(str1,sizeof(str1)-1); // str1  жолды енгізу

for (unsigned  i=0;i<strlen(str1);i++)
  if ( str1[i]==' ' ) // str1-дің әрбір символын «бос орынға» тексеру
  k++; // шарт орындалса k-ның мәнін 1-ге өсіру

cout<<"\n\n str1-dagi bos orin sani ="<<k;
 cout<<" \n\n str2- goldi engiz: \n";
cin.getline(str2,sizeof(str2)-1); // str2  жолды енгізу

//strcat(), strcpy(), strrev() – функциялардың қызметтерін көрсету
cout<<"\n\n strcat ( str1, str2 ) natigesi -> "<<strcat ( str1, str2 );
cout<<"\n\n strcpy ( str1, str2 )  natigesi -> "<<strcpy ( str1, str2 );
cout<<"\n\n strrev (str2)  natigesi -> "<<strrev (str2) ;
getch();
}
```

Программа орындалуының нәтижесі келесі түрде болады:

```
str1- goldi engiz:
ASTANA BAS KALA
 str1-dagi bos orin sani=2
str2- goldi engiz:
GAINAI BER
 strcat ( str1, str2 ) natigesi -> ASTANA BAS KALAGAINAI BER
 strcpy ( str1, str2 )  natigesi -> GAINAI BER
 strrev (str2)  natigesi -> REB IANIAG
```

7.3. string класы

Символдық массивтер түрінде берілген жолдармен жұмыс жасаудағы қолданылатын функциялар өте тиімді және көптеген мүмкіндіктерді жүзеге асыра алатын қуатты құралдар болып есептелгеніне қарамастан, мұнда жолдың соңын тексеріп отыру, жады көлемін, көрсеткіштердің жылжуын бақылап отыру және т.б. сияқты төменгі деңгейдегі программалау элементтері қолданылатын болды. Бұл мәселелерді шешу мақсатында кейінірек, шаблондардың стандарт кітапханасында (Standard Template Library, STL) анықталған string класын қолдану ұсынылды. Бұл класс ANSI/ISO C++ (American National Standards Institute/International Standards Organization) стандарты бойынша ANSI комитетінің string класымен сәйкес келеді. string типі түрінде анықталған жолдар соңына нуль-символ автоматты түрде қойылады. string класы STRING тақырыптық файлында жарияланған.

string класы basic_string шаблон-класының негізінде құрылған, сондықтан C++ Builder ортасының help-файл анықтамалығындағы basic_string шаблон-класс туралы деректерді string класы үшін де пайдалануға болады.

7.3.2. **string** *жолдарды жариялау, құру, біріктіру және салыстыру*

Программада string класының *объектісін немесе жаңа экземплярын* жариялау немесе құру үшін келесі конструкторлар қолданылады:

string(); – әдеттегідей конструктор, ұзындығы нольге тең жолды береді;

string(**const** string& s); – s-ке сілтеме бойынша алынатын жолмен бірдей болатын жолды құрады;

string(**const** string& s, size_t startIndex, size_t numChars = npos); – берілген s жолдың startIndex орнынан бастап numChars символдан тұратын бөлігіне сәйкес келетін жолды береді;

string(**const char*** cp); – *cp көрсеткіште адресі сақталған жолмен бірдей жолды береді;

string(**const char*** cp, size_t startIndex, size_t numChars=npos); – адресі бойынша алынатын cp жолдың startIndex орнынан бастап numChars символдан тұратын бөлігіне сәйкес келетін жолды береді;

string(size_t numChars, **char** c); – саны numChars болатын c символдан тұратын жолды береді.

Бір немесе бірнеше символдан тұратын жолдың көшірмелерін алу үшін қолданылатын конструкторлар (немесе меншіктеу амалы сияқты орындалатын конструкторлар):

```
string& operator=(const string& s);
string& operator=(const char* s);
string& operator=(char c);
```

string класының жолдарды біріктіруге (конкатенация) арналған әдістері немесе амалдары:

```
string  operator+ ( const string& s1, const string& s2);
string  operator+ ( const string& s, const char *cp);
string  operator+ (const char *cp, const string& s);
string  operator+ ( const string& s,  char c);
string  operator+ ( char c, const string& s);

string operator+=(const string& s);
string operator+=(const char * cp);
string  operator+=(char c);
```

string класының жолдарды салыстыруға арналған әдістері немесе амалдары:

```
bool operator== (const _string& s1, const string& s2);
bool operator== (const _string& s, const char *cp);
bool operator== (const char *cp, const _string& s);

bool operator !=(const _string& s1, const string& s2);
bool operator !=(const _string& s, const char *cp);
```

```
bool operator != (const char *cp, const _string& s);

bool operator > (const _string& s1, const string& s2);
bool operator > (const _string& s, const char *cp);
bool operator > (const char *cp, const _string& s);

bool operator < (const _string& s1, const string& s2);
bool operator < (const _string& s, const char *cp);
bool operator < (const char *cp, const _string& s);

bool operator >=(const _string& s1, const string& s2);
bool operator >= (const _string& s, const char *cp);
bool operator >= (const char *cp, const _string& s);

bool operator <= (const _string& s1, const string& s2);
bool operator <= (const _string& s, const char *cp);
bool operator <= (const char *cp, const _string& s);
```

7.3.1. **string** *жолдардың функциялары*

Төменде string класының кейбір жиі қолданылатын функция-әдістерінің қызметі мен прототипі көрсетілген, олар STRING тақырыптық файлында жарияланған (7.2-кесте):

7.2-кесте

string *класы функцияларының қызметі мен прототипі*

Функцияның аты	**Қызметі мен прототипі**

append	бастапқы берілген жолға s жолды немесе оның startIndex-тен басталатын numChars символдарын қосады:
	string& append(const string& s);
	string& append(const string& s, size_t startIndex, size_t numChars=npos);
	string& append(const char * cp, size_t startIndex, size_t numChars=npos)

assign	бастапқы берілген жолды s жолға немесе оның startIndex-тен басталатын numChars символына алмастырады:
	string& assign(const string& s);
	string& assign(const string& s, size_t startIndex, size_t numChars=npos);
	string& assign (const char * cp, size_t startIndex);

at	берілген жолдың символдарын индексі бойынша бір-бірлеп пайдалануды орындайды:
	const_reference at(size_t index);
	reference at(size_t index);

compare	бастапқы берілген жолды s жолмен немесе оның numChars символымен салыстырады,
	int compare(const string& s);
	int compare(size_t orig, size_t numChars=npos, const string& s);

copy	бастапқы берілген жолдың startIndex-тен басталатын numChars символын cp жолға көшіреді: size_t copy (char *cp, size_t numChars=npos, size_t startIndex);
c_str	бастапқы берілген жолдың (немесе оның көшірмесінің) адресін көрсеткіш түрінде береді: const char * c_str() const;

find	s буынсөздің (подстрока) бастапқы берілген жолдағы startIndex позициядан бастап алғашқы табылуын анықтайды:
	size_t find(const string& s);
	size_t find(const string& s, size_t startIndex);

find_first_of	бастапқы берілген жолдың (startIndex-тен бастап), s жолдың символдары кездесетін алғашқы позициясын көрсетеді:
	size_t find_first_of(const string& s);
	size_t find_first_of(const string& s, size_t startIndex);

find_last_of	бастапқы берілген жолдың (startIndex-тен бастап), s жолдың символдары кездесетін соңғы позициясын көрсетеді: size_t find_last_of (const string& s); size_t find_last_of (const string& s, size_t startIndex);

find_first_not_of	бастапқы берілген жолдың (startIndex-тен бастап), s жолда жоқ символы кездесетін алғашқы позициясын көрсетеді:
	size_t find_first_not_of(const string& s);
	size_t find_first_not_of(const string& s, size_t startIndex);

find_last_not_of	бастапқы берілген жолдың (startIndex-тен бастап), s жолда жоқ символы кездесетін соңғы позициясын көрсетеді: size_t find_last_not_of (const string& s); size_t find_last_not_of (const string& s, size_t startIndex);

insert	бастапқы берілген жолға pos позициядан бастап s жолдың startIndex-тен басталатын numChars символын кірістіреді:
	string& insert(size_t startIndex, const string& s);
	string& insert(size_t pos, const string& s, size_t startIndex, size_t numChars = npos);

length	бастапқы берілген жолдағы символдар санын береді: size_type length();

reserve	бастапқы берілген жолдан pos позициядан бастап n (n1) символды өшіріп, олардың орнына s жолдың startIndex-тен басталатын n2 символын қояды:
	string& replace(size_t pos, size_t n=npos, const string& s);
	string& replace(size_t pos, size_t n1, const string& s, size_t startIndex, size_t n2);

resize	жол үшін жадыдан берілетін орынның көлемін береді:
	size_t reserve();
	жүйеге, жол үшін ic байт орын қажет болатынын хабарлайды:
	void reserve(size_t ic);

substr	бастапқы берілген жолды m-символға өзгертеді, яғни m символды алып тастайды немесе **char** c болатын m символды қосады:
	void resize(size_t m, char c);
	void resize(size_t m);

reserve	бастапқы берілген жолдан startIndex-тен бастап барлық символдарын (немесе numChars символдарын) көшіріп алады:
	string substr(size_t startIndex);
	string substr(size_t startIndex, size_t numChars = npos);

7.3-жаттығу. Келесі программа string класының жолды құрушы конструкторларының, жолдың көшірмесін беретін конструкторының және біріктіру мен салыстыруларды орындайтын амалдарының қызметтерін демонстрациялайды:

```
#include <iostream.h>
#include <conio.h>
#include <string>
int main()
{
string s1="Almagul", s2="Student";
cout<<"Goldi kyratin Konstruktorlar kizmetin korsetu:\n\n";
```

```
cout<<"string(s1);->"<<string(s1)<<"\n";
cout<<"string(s1,0,4);-> "<<string(s1,0,4)<<"\n";
char c[]="Apatai";
char *cp=c;
cout<<"char *cp; string(cp);-> "<<string(cp)<<"\n";
char a='*';
cout<<"char a='*'; string(5,a);-> "<<string(5,a)<<"\n";
cout<<" \n\n Goldi koshiretin Konstruktor kizmetin korsetu:\n\n";
string k=s2; // k -га s2 жолдың көшірмесін меншіктеу
cout<<"string k=s2; -> k= "<<k<<"\n";
cout<<"\n\n Goldardi biriktirudi, salistirudi korsetu:\n\n";
cout<<"s1+s2="<<s1+" "+ s2<<"\n"; // s1 мен s2 жолдарды біріктіру
if (s1==s2) // s1 мен s2 жолдарды салыстыру
cout<<" s1 gane s2 goldar birdei";
else
cout<<" s1!=s2 nemese s1 gane s2 goldar birdei emes";
getch();
}
```

Программа орындалуының нәтижесі төмендегідей болады:

```
Goldi kyratin Konstruktorlar kizmetin korsetu:
string(s1);-> Almagul
string(s1,0,4);-> Alma
char *cp; string(cp);-> Apatai
char a='*'; string(5,a);-> *****

Goldi koshiretin Konstruktor kizmetin korsetu:
string k=s2; k-> k= Student

Goldardi biriktirudi, salistirudi korsetu:
s1+s2=Almagul Student
 s1!=s2 nemese s1 gane s2 goldar birdei emes
```

7.4-жаттығу. Келесі программа string класы функцияларының қызметін пайдаланып элементтері string типті жолдар (фамилиялар) болатын массивті сұрыптауды және массивтен қажет жолды (фамилияны) тауып бере алады:

```
#include <iostream.h>
#include <conio.h>
#include <string>
const n=5; // массивтің ұзындығын беру
int main()
{
string a[n]; // string типті массивті жариялау
cout<<"Familalardi engiz:"<<"\n";
 for(int k=0; k<n;k++)
 cin>>a[k]; // массив элементтері – фамилияларды енгізу
 cout<<"\n\n Suriptalgan Familalar"<<"\n";
 for(char i='A'; i<='Z';i++) // массивті сұрыптау басталды
 for(int k=0; k<n;k++)
 if (a[k].at(0)==i) //массивтегі k-шы фамилияның алғашқы 0-ші символын
                    // i-мен салыстыру
 cout<<a[k]<<"\n";
cout<<"Kerek Familani  engiz:";
 string kerek_Fam;
 cin>> kerek_Fam;
 int sanagish=0; // kerek_Fam-ды санап отыру үшін қажет
 for(int k=0; k<n;k++)
 if (a[k]==kerek_Fam) //  k-шы фамиляны kerek_Fam- мен салыстыру
 sanagish++;
if(sanagish!=0)
{ cout<<" Ondai adamdar bar, olar bastapki tizimde :";
for(int k=0; k<n;k++)
 if (a[k]==kerek_Fam)
 cout<<k<<",\t";
cout<<"-oriindarda tur \n";
 }
 else
 cout<<" Ondai adamdar gok"<<"\n";
 getch();}
```

Программа орындалуының нәтижесі:

```
Familalardi engiz:
Nazaryli
Tabinyli
Keteyli
Nazaryli
Gappasyli
Suriptalgan Familalar
Gappasyli
Keteyli
Nazaryli
Nazaryli
Tabinyli
Kerek Familani  engiz: Nazaryli
 Ondai adamdar bar, olar bastapki tizimde :0,  3,     -oriindarda tur
```

Мұндағы (at(0)==i) шартындағы at функцияның орнына жолдағы символдарды, жолдық массивтің элементтері сияқты пайдалануға мүмкіндік беретін *индексациялау амалын* қолдануға болады, мұнда символдың индексін көрсету үшін жай жақша () қолданылады, мысалы, жоғарыдағы шартты келесі түрде, (a[k].operator[](0)==i) жазады.

7.4. AnsiString типі

C++Builder -де объектіге бағдарланған программалауға негізделген қосымшалар құруда жолдармен немесе ұзын мәтіндермен жұмыс жасау үшін AnsiString типі енгізілген. Бұл AnsiString типі класс түрінде анықталып, *vcl/dstring.h* тақырыптық файлында жарияланған. AnsiString типі түрінде анықталған жолдар соңына да нуль-символдар автоматты түрде қойылады.

AnsiString типіне жататын объектіні жариялау немесе құру осы кластың келесі

_fastcall AnsiString*(аргумент);*

конструкторын шақыру болып табылады. Мұндағы аргументтердің түрліше болуына қарамастан, конструктор сол аргументте берілген мәндерді AnsiString типті жолға аударып құрып береді:

– AnsiString str; – бұл конструктордың әдеттегідей шақырылуы, мұнда конструктор аргументсіз беріліп тұр, құрылатын str жол «бос жол» болып табылады;

AnsiString(**const char*** str); – нульдік символмен аяқталатын символдар массиві түріндегі жолды құрады;

AnsiString(**const** AnsiString& str); – AnsiString типті str жолдың көшірмесін береді;

AnsiString(**const char*** str, **unsigned char** len); – str жолдың алғашқы len символынан тұратын және соңы нуль символмен аяқталатын жолды береді;

AnsiString(**int** san); – бүтін сандарды цифрлық символдардан тұратын жолға айналдырып береді;

AnsiString(**double** san); – жылжымалы үтірмен берілген сандарды цифрлық символдардан және «үтір» белгісінен тұратын жолға айналдырып береді, егер берілген сан ұзын болса, онда бастапқы 15 символы (үтірден басқа) түрленеді.

AnsiString типі үшін = =, !=, >, <, >=, <= салыстыру амалдары анықталған және олар өз мағынасында қолданылады. Салыстырулар регистрлерді ескере отырып жүргізіледі. Салыстыру символдардың коды арқылы жүргізіледі, мысалы, салыстырылатын екі жолдың бірінші символдарынан бастап сәйкес символдары салыстырылып отырады, егер сәйкес символдардың кодтары бірдей болса, онда жолдар өзара тең болып шығады, ал кодтар бірдей болмаса, кіші кодты беріп тұрған жол да кіші болады. Егер жолдардағы бастапқы символдар сәйкес келіп, бірақ жолдардың біреуіндегі символдар көп болса, онда сол ұзын жол үлкен болып есептеледі.

AnsiString типті жолдар үшін « = », « += » – меншіктеу және «+» – жолдарды біріктіру (конкатенация) амалдары қарастырылған. AnsiString типті жолдардың символдарын пайдалану үшін [ ] – индексациялау амалы қолданылады, мұнда символдарды номерлеу 1-ден басталады.

7.4.1. **AnsiString** *класының әдістері*

AnsiString типті жолдарға қолданылатын негізгі әдістер туралы төменде көрсетілген кестеде қарастырылған (7.3-кесте):

7.3-кесте

AnsiString *класының әдістері*

Әдістің аты және str жол үшін шақырылуы	**Қызметі мен синтаксисі**
str.AnsiCompare(rhs)	берілген str жолды әдіс-функцияның аргументінде берілген rhs жолмен регистрді ескере отырып салыстырады, егер мұнда str > rhs болса, функцияның қайтаратын мәні «1», str < rhs болса, функцияның қайтаратын мәні «-1», str = rhs болса, функцияның қайтаратын мәні «0»-ге тең болады: **int** fastcall AnsiCompare(**const** AnsiString& rhs) **const**
str.AnsiCompareIC(rhs)	берілген str жолды әдіс-функцияның аргументінде берілген rhs жолмен AnsiCompare әдісі сияқты салыстырады, бірақ мұнда регистр ескерілмейді: **int** _fastcall AnsiCompareIC(**const** AnsiString& rhs)**const**
str.AnsiLastChar()	берілген str жолдың соңғы символының адресін сақтайтын көрсеткішті береді: **char***_fastcall AnsiLastChar() **const**
str.AnsiPos(subStr)	subStr буынсөздің берілген str жолда бірінші кездесетін алғашқы символының индексін береді: **int** fastcall AnsiPos(**const** AnsiString& subStr) **const** мұнда индекстеу 1-ден басталады, subStr буынсөзді символдары str жолдан табылмаса 0-ді қайтарады, Pos әдісінен айырмашылығы көпбайтты символдар үшін де қолданылады.
str.c_str();	берілген str жолды, символдарын сол күйінде сақтай отырып соңы нуль-символмен аяқталатын символдық массив түріндегі жолға ауыстырып және сол жолдың адресін көрсететін көрсеткішті қайтарады: **char***_ fastcall c_str()**const** мысалы, келесі түрде жазуға болады: AnsiString str ="Bul gol"; cout<<str.c_str();
CurrToStr(value)	Currency типті value-ні AnsiString типті жолға айналдырады: **static** AnsiString_ fastcall CurrToStr(Currency value)

CurrToStrF (value, ffFixed, digits)	Currency типті value-ні жылжымалы үтірмен берілетін сандарды түрлендіру форматын пайдалана отырып AnsiString типті жолға айналдырады: **static** AnsiString_ fastcall CurrToStrF(Currency value, TstringFloatFormat format, **int** digits)
FloatToStrF (value, format, precision, digits)	функцияның аргументі ретінде жылжымалы үтірмен берілген сан value-ні көрсетілген format -тағыдай жолға түрлендіреді: **static** AnsiString_ fastcall FloatToStrF(**long double** value, TstringFloatFormat format, **int** precision, **int** digits) мұндағы format – жылжымалы үтірмен берілген санды жол түрінде шығарудың форматтарын көрсетеді, оның мәні ретінде келесілер алынады: *ffGeneral* – жалпы түрге келтіру, мысалы, 3.1416; *ffExponent* – экспоненциалдық түрге келтіру, мысалы, 3.1416E+00; *ffFixed* – жылжымайтын үтір арқылы жазылатын түрге келтіру, мысалы, 3.14; *ffNumber* – разрядтары ажыратылып жазылған түрге келтіру, мысалы, 7 333 141,60; *ffCurrency* – ақша бірлігімен жазылатын түрге келтіру, мысалы, 3 141.60 р.; precision – дәлдікті (цифрлардың жалпы санын), digits-разрядтың санын (үтірден кейінгі цифрлар санын) көрсетеді. Дәлдік **float** типі үшін 7-ден, **double** үшін 15-тен, **Extended** үшін 18-ден аспауы қажет. Разрядтың саны форматқа байланысты болады.
str .ToDouble()	берілген str жолды жылжымалы үтірмен берілетін санға түрлендіреді, егер жолдағы символдар сандық форматқа сәйкес келмесе, EconvertError ерекше жағдай шақырылады: **double_** fastcall ToDouble() **const**
str .ToInt()	берілген str жолды бүтін санға түрлендіреді, егер жолдағы символдар сандық форматқа сәйкес келмесе, EconvertError ерекше жағдай шақырылады: **int_** fastcall ToInt() **const**
str .ToIntDef()	берілген str жолды бүтін санға түрлендіреді, егер жолдағы символдар сандық форматқа сәйкес келмесе defaultValue мән қайтарылады: **int _** fastcall ToIntDef(int defaultValue) **const**

str.IsDelimiter (delimiters, index)	егер str жолдың index-ші символы delimiters-те берілген символдардың бірі болса, онда true мән қайтарылады және көп байтты символдар үшін қолданылады: **bool** fastcall IsDelimiter(**const** AnsiString& delimiters, **int**_ index) **const**
str .Length()	берілген str жолдағы символдардың санын береді: **int**_ fastcall Length() **const**
str .SetLength(newLen)	берілген str жолды newLen символға дейін қысқартады: **void**_ fastcall SetLength(**int** newLen)
str .LowerCase()	берілген str жолдағы барлық символдарды төменгі регистрге (кіші әріптерге) ауыстырады: AnsiString_ fastcall LowerCase() **const**
str .UpperCase()	берілген str жолдағы барлық символдарды жоғарғы регистрге (үлкен әріптерге) ауыстырады, бастапқы жол өзгермейді: AnsiString _ fastcall UpperCase() **const**
str .Substring(index, count)	берілген str жолдан index-тен басталатын, count символдан тұратын буынсөзді қиып алады: AnsiString_ fastcall SubString(**int** index, **int** count) **const**
str.Insert(subStr, index)	берілген str жолға index-ші позициядан бастап, subStr буынсөзді кірістіреді: **void**_ fastcall Insert(**const** AnsiString& str, **int** index)
str.Delete(index, count)	берілген str жолдан index-ші позициядан бастап count символды өшіреді: **void**_fastcall Delete(**int** index, **int** count)

AnsiString типті жолдарға осы әдістерді қолдана отырып кейбір күрделі алгоритмдерді қарапайым жолмен шығаруға болады.

7.5-жаттығу. Келесі программа пернетақтадан енгізілген сан палиндром сан (оң жағынан да, сол жағынан да бірдей оқылатын сандар) болса «YES», болмаса «NO» деген жауапты бере алады:

```
#include <vcl.h> // AnsiString үшін қосылды
#include <iostream.h>
#include <conio.h>
int main()
```

```
{ float san;
cin>>san;
AnsiString str=FloatToStr(san); // san-ды str-жолға түрлендіру
bool f=1; // san палиндром болса, f=1 , болмаса f=0
for(int i=1,j=str.Length(); i<=str.Length(),j>=1;i++,j--)
  if (str[i]!=str[j]) //жолдың оң жағы мен сол жағынан сәйкес
                      //символдарды салыстырып отыру
  {f=0; // сәйкес символдар немесе цифрлар бірдей емес екенін білдіреді
  break; // циклді аяқтамай шығып кету
  }
if (f)
  cout<<" Yes";
  else
  cout<<" No";
   getch();    }
```

Программа орындалуының нәтижесі келесі түрде болады:

1-жағдай.

```
123321

Yes
```

2-жағдай.

```
123321

No
```

7.6-жаттығу. Келесі программа, пернетақтадан енгізілген бүтін санның цифрларының қосындысын тауып бере алады:

```
#include <vcl.h> // AnsiString үшін қосылды
#include <iostream.h>
#include <conio.h>
int main()
{int san;  cin>>san;
 AnsiString str=IntToStr(san); // бүтін san-ды str-жолға түрлендіру
 int S=0; // қосындының бастапқы мәні
 for(int i=1; i<=str.Length();i++)
S=S+  str.SubString(i,1).ToInt(); // str-жолдағы символ-цифрларды бір-
                    //бірлеп қиып алып санға айналдырып қосып отыру
```

```
cout<<"S="<<S;
getch(); }
```

Программа орындалуының нәтижесі келесі түрде болады:

```
123456789
S=45
```

Бақылау сұрақтары

1. C++ Builder 6 программалау ортасында мәтін түріндегі деректерді жолдық типтер түрінде берудің қандай тәсілдерін қолдануға болады?

2. Символдық массив түрінде берілген жолдың соңын білдіретін символ қалай аталады?

3. Символдық массив түрінде берілген жолды программада қалай жариялайды?

4. Символдық массив түрінде берілген жолды енгізуде қандай операторлар қолданылады?

5. cin объектісінің cin. Getline (жол_Аты, **sizeof**(жол_Аты)-1); түрінде жазылатын әдісінің қызметі қандай?

6. Символдық массив түрінде берілген жолдармен жұмыс жасайтын функциялардың прототиптері жинақталған тақырыптық файл қалай аталады?

7. string класының сипаттамасы жарияланған тақырыптық файл қалай аталады?

8. string класының қандай конструкторлары бар?

9. string(**const char*** cp); – конструкторының қызметі не?

10. string класы үшін қандай салыстыру амалдары анықталған?

11. string-жолдың ұзындығын көрсететін әдіс қалай аталады?

12. string класы үшін конкатенацияның қандай түрлері анықталған?

13. AnsiString класының қандай конструкторлары бар?

14. AnsiString класы үшін қандай амалдар анықталған?

15. AnsiString типті жолдар үшін символдарды индексациялау амалын ([]) қолдануға болады ма?

16. FloatToStrF (value, format, precision , digits) әдісіндегі format параметрдің қызметі қандай?

Тапсырмалар

1. Егер берілген мәтінде «*а*» символы «*и*» символына қарағанда жиі кездессе, онда «*ақиқат*», әйтпесе «*жалған*» сөзін экранға шығарыңыз.

2. Берілген мәтіннің ішіндегі цифрларын алып тастап, ал "+" және "–" таңбаларын екі еселеп жазып, қайта шығарыңыз.

3. Берілген мәтіндегі жол қойылған бірнеше пробелдің біреуін ғана қалдырып, текстті қайтадан экранға шығару программасын жаз.

4. Берілген мәтінде "*а*" әрпі бар екені белгілі. Сол текстің бірінші "*а*"-дан кейінгі бөлігін шығарыңыз.

5. Берілген мәтіндегі "+" символынан кейін цифрлар тұрса, онда оны алып таста.

6. Берілген мәтіндегі *ph* жұбын *f* әрпіне өзгертіп текстті қайта шығарыңыз.

7. Латын әрпінен және сөздерінен тұратын тізбек берілген "*а*" әріпінен тұратын сөздер санын анықтаңыз.

8. Key сөзіндегі әріптер берілген мәтінде кездессе, онда "иә " әйтпесе "жоқ " сөзін баспаға шығарыңыз.

9. Берілген мәтіндегі алдында "с" әрпі орналасатын "в" әріпінің бәрін алып тастап, текстті қайта шығару қажет.

10. *C0,C1,C2* символдық айнымалыларының мәндері – цифрлар. *K* сандық бүтін айнымалысына осы үш цифрдан тұратын санды меншіктеңіз. (мысалы, C0="3",C1="4",C2="9",K=349).

11. Берілген мәтінде дөңгелек жақшалар дұрыс қойылған ба? Яғни, әрбір ашылған жақша жабылғандығын тексеретін программа құрыңыз.

12. Мәтіндегі "W" әріпінен аяқталатын сөздер санын анықтаңыз.

13. Мәтіндегі сөздер санын анықтаңыз.

14. Мәтінде "–" символдар жұбының бар-жоқтығын анықтаңыз.

15. Тізбекті шығарыңыз, abbccc…..zzz…..z

16. Текст берілген. Үш "с" әрпі кездесетін сөздерді шығарыңыз.

17. Енгізілген жолдағы тақ орындарда тұрған барлық символдарды жойыңыз.

18. Енгізілген жолда қай әріп жиі кездеседі? Біріншісі ме, әлде соңғысы ма? Анықтаңыз.

19. Пернетақтадан енгізілген жолдағы барлық артық бос орынды (пробел) жойыңыз.

20. Енгізілген жолдағы '*' және '!' символының санын шығарыңыз.

21. Енгізілген жолдағы '+', '– ', '*' символдарының жалпы санын шығарыңыз.

22. Пернетақтадан енгізілген жолдағы дауысты дыбыстар әрпінің (латын) санын шығарыңыз.

23. Пернетақтадан енгізілген жолдағы барлық X символын Y символына алмастырыңыз.

24. Пернетақтадан енгізілген А жолын кері ретпен В жолына жазыңыз. В жолын экранға шығарыңыз. Ең болмағанда екі қос нүктесі бар символдар жолы берілген. Экранға бірінші және екінші қос нүкте арасындағы барлық символдарды шығарыңыз.

25. Енгізілген жолдағы ең үлкен сөзді анықтап, ұзындығын экранға шығарыңыз.

26. Енгізілген жолдағы сөздердің санын анықтап экранға шығарыңыз.

27. Енгізілген жолдағы барлық '123' символдарын '45'-ке алмастырыңыз.

28. Енгізілген жолдағы әр цифрды реті бойынша өзінен кейінгі санмен алмастырыңыз. 9 цифрын 0 цифрымен алмастырыңыз.

29. Жолдағы әр нүктені көп нүктемен (...) алмастырып түрлендіріңіз. Енгізілген жолдағы барлық кіші латын әріптерін бас әріптеріне алмастырыңыз.

8-тарау. АҒЫНДАР ЖӘНЕ ФАЙЛДАР

8.1. Ағындар

C++ программалау тілінде деректі программаға енгізу және алу, ағындармен тікелей байланысты. *Ағын (stream, поток)* – программаға енгізілетін немесе программадан алынатын ақпараттардың жиынтығын, компьютердің құрылғыларымен байланысты қарастыру үшін енгізілген абстракциялы ұғым. C++ -те программалауда, ағынмен байланысатын пернетақта, монитор, принтер, модем, дискідегі файл сияқты физикалық құрылғылардың барлығын бір ғана «*файл*» деп аталатын терминге біріктіріп қарастыратындықтан, ағындарды *файлдың логикалық интерфейсі* деп те атайды. Түрлері немесе мүмкіндіктері бойынша файлдардың әр түрлі болуына қарамастан, олармен жұмыс жасайтын ағындар бірдей болады, яғни ағын барлық құрылғыларға ортақ бір интерфейстің болуын қамтамасыз етеді.

Енгізілетін немесе шығарылатын деректің түріне (файлдағы) байланысты ағындарды екі түрге бөледі: *мәтіндік ағын* және *екілік ағын*. *Мәтіндік ағын* символдар түрінде берілетін деректер үшін ғана қолданыла алады. Мәтіндік ағын мен файлдың арасындағы байланыс бір мәнді болмауы мүмкін, мұның себебі кейбір басқарушы символдардың түрленіп кетуімен байланысты, мысалы, ағынға шығару кезінде «\n» (жаңа жол) басқарушы символы, «#13» (ENTER) символына түрленіп кетеді. *Екілік ағын* деректердің түріне тәуелсіз болады және кез келген типті дерекпен жұмыс жасауға қолайлы болып келеді. Екілік ағынды пайдалану кезінде символдардың түрленуіне жол берілмейтіндіктен, мұнда ағын мен файлдың арасындағы байланыс бір мәнді болады, яғни ағында не болса, файлға сол жазылады немесе керісінше, файлдағы дерек қандай болса, ағынға оқылатын дерек те сондай болады.

Деректердің берілу бағытына байланысты ағындар үш топқа бөлінеді:

– *енгізуші ағыны (input stream-потоки ввода),* файлмен байланысқан мұндай ағындардан деректер оқылады (немесе алынады) да, ол деректер программадағы айнымалыларға беріледі;

– *алушы немесе шығарушы ағыны (output stream)*, мұнда керісінше программадан алынған деректер, файлмен байланысқан ағынға жазылады (немесе қойылады);

– *екі бағыттағы ағындар (input- output stream)*, мұндай ағындар деректі оқу мен жазуды қатар орындайды.

Программадағы құрылу (немесе жариялану) тәсілдеріне байланысты ағындарды екі топқа бөледі:

– автоматты түрде құрылатын стандарт ағындар, стандарт енгізу-алу құрылғылармен жұмыс жасау үшін қолданылады;

– айқын түрде құрылатын файлдық ағындар, файлдармен жұмыс жасау үшін қолданылады.

8.1.1. Стандарт ағындар

C++ -тегі <iostream> тақырыптық файлы қосылған программа орындалған кезде автоматты түрде құрылатын стандарт ағындар болады. Бұл стандарт ағындарды пайдалану үшін программа мәтініне стандарт ағындар кластарының жарияланулары және алдын ала анықталған cin, cout, cerr, clog объектілерінің сипаттамалары сақталатын <iostream> тақырыптық файлын кірістіреді. Программа орындалған кезде бұл тақырыптық файлға байланысты автоматты түрде төрт түрлі ағындар объектілері құрылады:

– cin – бұл деректі стандарт құрылғыдан (әдетте пернетақтадан) енгізуге арналған буферлік ағын объектісі;

– cout – бұл деректі стандарт құрылғыға (әдетте мониторға) шығаруға арналған буферлік ағын объектісі;

– cerr – бұл қате туралы деректі стандарт құрылғыға (әдетте мониторға) шығаруға арналған буферлік емес ағын объектісі;

– clog – бұл қате туралы деректі стандарт құрылғыға (әдетте мониторға) шығаруға арналған буферлік ағын объектісі.

Қалыпты жағдайда стандарт ағындардың бағыты консольмен (пернетақта – монитор) байланысты болады. Бірақ қажет болған жағдайда программалық жолмен немесе операциялық жүйе командаларының көмегімен стандарт ағындардың бағытын басқа құрылғыларға не болмаса дискілік файлдарға қарай өзгертуге болады.

Арнаулы құрылғылармен жұмыс жасайтын орталардың жұмысын C++-те программалағанда, cerr және clog стандарт ағындарын қателер туралы хабарламаларды бірден шығару үшін қолданады және бұл ағындар программалауды біраз жеңілдетеді.

– cerr және clog стандарт ағындарының бір-бірінен айырмашылығы, cerr стандарт ағынына жіберілген қате туралы хабарлама, мониторға бірден шығарылады, яғни буферде сақталмайды, ал clog стандарт ағынын пайдаланғанда, қате туралы хабарламалар буфер толғаннан кейін барып шығарылады.

Жалпы, *буфер (buffer)* бұл – компьютердің сыртқы құрылғылары мен іске қосылып тұрған программаның, бір-біріне жіберетін деректерін уақытша сақтай тұруға арналған жедел жадының бір бөлігі. Егер ағын буферді пайдаланбаса, онда дерек файлға (немесе құрылғыға) бірден шығарылады, ал ағын буферді пайдаланатын болса, ол кезде файлға шығару буфер толғаннан кейін ғана орындалады.

8.1.2. Ағын кластары

Жалпы C++ программалаудың енгізу-алу жүйесі, <iostream> тақырыбымен берілген файлда анықталған күрделі кластар иерархиясының негізінде жүзеге асырылады. Программа жазу барысында автоматты түрде құрылатын стандарт ағындардан басқа, файлдармен жұмыс жасау үшін айқын түрде құрылатын ағындар да жиі қолданылады. Көптеген есептерді шығаруда бұл ағындарды белгілі бір кластардың объектілері ретінде қарастырады. Программалаудағы енгізу-алуды жүзеге асыруда жиі қолданылатын кластар негізінен <iostream> файлында сипатталған ios және streambuf класынан таратылады. Бұл ios класы жоғары деңгейдегі енгізу-алу операцияларын қамтамасыз ететін, форматтауды және енгізуші-алушы ағындарымен байланысты қателер мен статустық деректерді беруді жүзеге асыра алатын қасиеттер мен әдістерді қамтиды, ал streambuf класы ағындарды буферизациялауды және сыртқы құрылғылармен байланысты қамтамасыз етеді.

Төменде программалауда жиі қолданылатын ағын кластары көрсетілген:

– ағындардың түпкі класы – ios;
– енгізуші ағынның класы – istream;
– алушы немесе шығарушы ағынның класы – ostream;
– екі бағыттағы ағын класы – iostream;
– енгізуші файлдық ағынның класы – ifstream;
– алушы немесе шығарушы файлдық ағынның класы – ofstream;
– екі бағыттағы файлдық ағынның класы – fstream.

Бұл кластардың барлығына ортақ негізгі жарияланулар ios класында беріледі, сондықтан программаға <iostream> тақырыбымен берілген файлға сәйкес тақырыптық файл iostream.h қосылып жазылады. Сол сияқты ifstream, ofstream және fstream файлдық ағын кластарын пайдалану үшін программаға fstream.h тақырыптық файлын кірістіреді.

Программалауда, жоғарыда айтылғандардан басқа жедел жадымен дерек алмасу үшін құрылатын istringstream – енгізуші жолдық ағын, ostringstream – алушы жолдық ағын және stringstream – екі бағыттағы жолдық ағын кластары қолданылады. Бұл кластарды пайдалану үшін программаға <sstream> файлына сәйкес тақырыптық файл sstream.h қосылуы керек.

8.2. Файлдар

Жалпы *файл* деп қандай да болмасын деректер сақталған нақты бір құрылғыны қабылдау қалыптасқан, мысалы, қатты дискінің деректер сақталған белгілі бір атауы бар облысын да *файл* ретінде қарастырады. Ал программалауда файлды байттардың немесе символдардың реттелген тізбегі деп қарастырады. Бұл тізбектің басталуын, яғни бірінші байттың алдындағы позицияны *файлдың басы*, соңғы байттан кейінгі позицияны *файлдың соңы* деп атайды. Файлдың басы және бірінші байт нольдік позицияда болады, ал келесісінен бастап позициялар бірге өсіп, жылжып отырады, ал файлдың соңының позициясы, байтпен алғандағы файл ұзындығының шамасына тең болады. Файлға деректі жазу және файлдан деректі оқу, осы айтылған позициялар бойынша жүргізіледі.

Файлға қатынау немесе файлдағы байттарды не символдарды оқу мен жазу позицияларға қатысты алғанда екіге бөлінеді:

– *файлға біртіндеп қатынау*. Мұнда файлдағы байттарды немесе символдарды пайдалану бірінен кейін бірі тізбектелген ретпен жүріп отырады;

– *файлға тікелей қатынау*. Мұнда файлдағы байттарды немесе символдарды пайдалану файлдың кез келген көрсетілген жерінен, яғни позициясынан жүргізіле береді.

Жадыдағы берілу тәсіліне байланысты файлдарды екіге бөліп қарастырады:

– *мәтіндік файлдар.* Мәтіндік файлдар – бұл «бос орын», «жаңа жол», табуляция символдарымен ажыратылған символдар тізбегінен тұратын мәтін түрінде берілетін файл.

– *бинарлық немесе екілік файлдар.* Мұнда файлға жазылатын немесе файлдан оқылатын дерек байттардың порциясы түрінде берілетін болады.

Программалауда файлмен жасалатын негізгі операцияларға келесілер жатады:

– файлды құру;
– файлға деректі жазу;
– файлды ашу;
– файлды жабу;
– файлдан деректі оқу және т.б.

C++ -те программалауда файл ретінде дискідегі файлдар ғана емес, сондай-ақ енгізу-алу операцияларын орындайтын кез келген құрылғы да «файл» деп қарастырылатынын ескеру керек.

8.2.1. Файлды құру, ашу және жабу

Программада файлға деректі жазу немесе файлдан оқу үшін алдымен программа мен файлды, файлдық ағындар арқылы байланыстыру керек екені және мұндай ағындардың үш түрі қолданылатыны белгілі (8.1-тақырыпты қараңыз). Деректі файлға жазу үшін файл мен программаны байланыстыратын алушы файлдық ағын класының (ofstream) объектісін жариялайды. Программадағы жариялануы:

ofstream file_object (*"файлдыңАты"*);

мұндағы ofstream – типі, file_object – объектінің аты, *файлдыңАты* – дискіде, ағымдағы бумада құрылатын файлдың аты, мысалы:

```
ofstream file_object ("FILENAME.TXT"); // FILENAME.TXT мәтіндік файлын құру
ofstream file_object ("FILENAME. DAT"); // FILENAME. DAT бинарлық файлын құру
```

Программада ofstream типіне жататын объектіні жариялағанда, дискіде көрсетілген атпен файл құрылады, егер дискіде дәл сондай атпен құрылған файл бұрыннан бар болса, онда ол жойылып, оның орнына соңғы құрылған жаңа файл баратын болады.

Бұрыннан құрылған файлға әрі қарай толықтырып деректер жазу үшін немесе ондағы деректерді оқып, программаға беру үшін ол файлды ашу керек болады. Файлды ашу программада енгізу файлдық ағыны класының объектісін жариялаумен орындалады, яғни программадағы жариялануы келесі түрде болады:

ifstream file_object ("*файлдыңАты* ", ашу режимі);

мұндағы ifstream – типі, file_object – объектінің аты, *файлдыңАты* – оқу үшін ашылатын файлдың аты, ашу режимі – файлдың қандай максат үшін ашылатынын тағайындайтын параметр (8.1-кесте). Мысалы:

```
ifstream file_object ("FILENAME.TXT", ios::app); //FILENAME.TXT файлын
                                                 //қосымша дерек жазу үшін ашу
ifstream file_object ("FILENAME.TXT", ios::out | ios::noreplace);
                        // FILENAME.TXT /файлын өзгерістен қорғай отырып ашу
```

Дискідегі сақталған файл түрлі мақсаттар үшін ашылуы мүмкін, мысалы, ондағы деректерді өзгертпей экранға шығару керек болуы мүмкін немесе ол файлға қосымша тағы да деректер жазу үшін ашу керек болады және т.б. жағдайлар болуы мүмкін. Осыған байланысты файлды ашу кезінде режимді тағайындайтын параметр көрсетіледі (8.1-кесте).

8.1-кесте

Файлды ашу режимдері

Ашу режиміне сәйкес параметр мәні	**Орындайтын қызметі**
ios::app	файлды дерек қосу үшін ашу және файлдық көрсеткішті файлдың соңына апару
ios::ate	файлдық көрсеткішті файлдың соңына апару
ios::binary	Файлды екілік (бинарлық) режимде ашу

ios::in	файлды дерек енгізу үшін ашу
ios::nocreate	файл табылмаған жағдайда, файлды жаңадан құрмай, қатені хабарлау
ios::noreplace	файл табылған жағдайда, файлды ашу операциясын үзіп, қатені хабарлау
ios::out	файлды шығару үшін ашу
ios::trunc	Файлдағы деректі жою

Әдетте файлды ашу кезінде ios::binary режимі тағайындалмаған болса, файлдар мәтіндік режимде ашылған болып есептеледі.

C++ программалау тілінде жалпы файлды ашу кезінде open(); функциясын қолдану қарастырылған, жазылуы:

```
ifstream file_object;                    //файлдық объектіні жариялау
file_object. open ("файлдыңАты ", ашу режимі);    //файлды ашу
```

Бірақ ofstream, ifstream және fstream кластарында файлды ашуды автоматты түрде орындайтын конструкторлар болғандықтан, программистер open(); функциясын қолданбай-ақ, файлдарды жоғарыда келтірілген әдіспен аша береді.

Файлды жабу үшін close (); функциясын қолданады. Бұл close (); функциясының параметрлері болмайды және ол ешқандай мән қайтармайды.

8.2.2. Мәтіндік файлға жазу және оқу

Деректі алушы немесе шығарушы файлдық ағын класының (ofstream) объектісі көмегімен ағынмен байланысқан файлға жазу үшін ағынға қосу амалын (<<) пайдаланады, жазылуы келесі түрде болады:

```
file_object << X;
```

мұндағы X – файлға жазылатын дерек, мысалы,

```
ofstream file_object ("FILENAME.TXT ");
file_object << "Дерек файлға жазылып жатыр"<<"\n";
```

Деректі енгізуші файлдық ағын класының (ifstream) объектісі көмегімен ағынмен байланысқан файлдан оқу үшін ағыннан алу амалын (>>) пайдаланады, жазылуы келесі түрде болады:

```
file_object >> X;
```

мұндағы X – файлдан оқылатын дерек, мысалы:

```
ifstream file_object ("FILENAME.TXT ");
file_object >>X;
```

Файлдан оқу кезінде ағыннан алу амалы (>>) пайдаланылса, онда алғашқы бос орындарға дейінгі символдар ғана оқылады, ал бір жол тұтас оқылуы үшін файлдық объектінің getline( ); функциясы немесе әдісі қолданылады, жазылуы:

```
file_object.getline(str1, sizeof(str1));
```

Мәтіндік файлға жазуды немесе мәтіндік файлдан оқуды программалауда файлдың соңын және жолдың соңын білдіретін символдарды қолданады. Файлдың соңын білдіретін символ ретінде EOF (ASCII кодтағы саны – 26 немесе Ctrl+Z) қабылданған, осыған сәйкес файлдық ағындар кластарында анықталған бульдік eof(); функциясы программада файлдың соңын көрсету үшін қолданылады. Файлдың соңы кездескенге дейін бұл функцияның мәні 0 болып тұрады, ал файлдың соңына жеткенде оның мәні 1-ге тең болады. Мысалы, келесі программа фрагменті файлдың соңын анықтауды білдіреді:

```
ifstream file_object;
while (!file_object.eof())
{
... ... ...
};
```

8.1-жаттығу. Программа алушы немесе шығарушы файлдық ағынды пайдаланып, мәтіндік файлға дерек жазуды көрсете алады.

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
 int main()
  {
ofstream file_object1 ("111.txt"); //алушы файлдық ағын объектісін жариялау
                                   //және программаны файлмен байланыстыру
 file_object1<<" Menin atim Koga!"; // алушы файлдық ағынға деректі қосу
file_object1.close();  // файлды жабу
 }
```

Бұл программа орындалғанда шығатын нәтиже жоба сақталған ағымдағы бумада 111.txt файлы құрылады және бұл файлға "Menin atim Koga!" деген мәтін жазылып қалады.

8.2-жаттығу. Программа енгізуші файлдық ағынды пайдаланып мәтіндік файлдан деректі оқуды көрсете алады.

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
int main()
{ char X[20];
ifstream file_object2 ("111.txt"); // енгізуші файлдық ағын объектісін жариялау
                                   // және  программаны файлмен байланыстыру
 file_object2.getline(X, sizeof(X)); //енгізуші файлдық ағыннан деректі алып,
                                     //X-ке беру
cout<<X;
file_object2.close(); // файлды жабу
 getch();
 }
```

Бұл программа орындалуының нәтижесі келесі түрде болады:

```
Menin atim Koga!
```

Программада файлдағы жолды тұтасымен бірден оқып шығу үшін getline( ); функциясы шақырылды, егер оның орнына file_object2>>X; ағыннан алу амалын қолданса, онда ол бірінші бос орынға дейінгі сөзді, яғни «Menin» деген мәтінді ғана шығарған болар еді.

8.3-жаттығу. Программа саны алдын ала белгісіз, string түрінде анықталған фамилияларды мәтіндік файлға жазуды көрсетеді.

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
int main()
{
string fam;
ofstream FileGazu("spisok.txt"); // файлды программамен байланыстыру
cout<<"File-ga gazu bastaldi: \n ";
 while (fam!="!")  // әзірге енгізілген фамилия "!" болмаса цикл қайталанады
 {
 cin>>fam;
FileGazu <<fam<<"\n"; // fam-дағы мәнді алушы файлдық ағынға қосу
  }
     FileGazu.close();
     cout<<"File-ga gazu toktaldi \n ";
      getch();
       }
```

Бұл программаның орындалуы нәтижесінде ағымдағы бумада

```
File-ga gazu bastaldi:
 Asanov
Saparova
Bulatov
Kim
Petrov
Ospanov
!
File-ga gazu toktaldi
```

spisok.txt мәтіндік файлы құрылып, оған жоғарыдағы терезедегі енгізілген фамилиялар жазылады.

Бұл программада файлға жазылатын фамилиялардың саны алдын ала берілмеген, яғни қолданушы қанша фамилия енгізгісі келсе, сонша фамилияны енгізе алады, ал енгізуді аяқтау үшін «!» белгісін басуы қажет болды.

8.4-жаттығу. Программа 8.1-жаттығу барысында құрылған spisok.txt мәтіндік файлдағы фамилияларды оқып, экранға шығарып бере алады.

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
int main()
{
   string fam;
   fstream FileOku("spisok.txt");     //файлды программамен байланыстыру
   cout<<"File-gan oku bastaldi: \n ";
  while (!FileOku.eof())  //әзірге файлдың соңы 1 болмаса цикл қайталанады
 {
   FileOku>>fam;          //файлдық ағыннан фамилияны алып, fam-ға беру
   cout<<fam<<"\n";
 }
 FileOku.close();
 getch();
}
```

Бұл программа орындауға жіберілгенде, spisok.txt мәтіндік файлдағы фамилиялар оқылып экранға шығарылады:

```
File-gan oku bastaldi:
 Asanov
Saparova
Bulatov
Kim
Petrov
Ospanov
!
```

Бұл программада файлдағы жолдар саны немесе символдардың саны алдын ала белгісіз болғандықтан, файлдың соңын іздеуді орындайтын eof(); функциясы қолданылды.

8.2.3. Екілік файлға жазу және оқу

Деректерді екілік (бинарлық) файлға жазу үшін және оқу үшін файлдық ағын объектілерінің сәйкесінше write (); және read (); әдістері қолданылады. Бұл функцияларды пайдаланғанда дерек сақталатын аралық буферлік айнымалы және оның байтпен есептелген ұзындығы көрсетілуі тиіс, жазылуы:

```
FileGazu.write(buffer, sizeof(buffer));
FileOku.read(buffer, sizeof(buffer));
```

Мысалы, файлға жазу келесі түрде болады:

```
ofstream FileGazu("777.dat");
FileGazu.write(buffer, sizeof(buffer));
```

Ал файлдан оқу төмендегідей түрде анықталады:

```
ifstream FileOku("777.dat");
FileOku.read(buffer, sizeof(buffer));
```

8.5-жаттығу. Келесі программа құрылымдық массивті екілік файлға жазуды орындайды:

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
struct adam
 { string fam;    //фамилия
 int stag;        //еңбек өтілі
 float zarpl;  //жалақысы
 };
```

```
int main()
{
 adam gruppa[3];                          //құрылымдық массивті жариялау
   ofstream FileGazu("777.dat");          // алушы файлдық ағын объектісін
                                  //жариялау және оны екілік файлмен байланыстыру
 for(int i=0;i<3;i++)
 {
cin>>gruppa[i].fam>> gruppa[i].stag>>gruppa[i].zarpl; // деректерді енгізу
 FileGazu.write((char*)& gruppa[i], sizeof (adam));   // екілік файлға жазу
 }
 FileGazu.close();
}
```

Бұл программа орындалуының нәтижесінде пайда болған терезеге келесі деректер енгізіледі:

```
Asnova
20
100000
Mukasheva
25
75000
Muratova
15
50000
```

Бұл терезедегі енгізілген деректер 777.dat екілік файлға жазылып сақталып қалады.

8.6-жаттығу. Келесі программа екілік файлдағы деректі құрылымдық массив түрінде оқуды демонстрациялайды:

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
struct adam
 {
```

```
 char fam[30];    //фамилия
 int stag;        //еңбек өтілі
 float zarpl;     //жалақысы
 };
int main()
{
  adam gruppa[3];                          //құрылымдық массивті жариялау
        cout <<"\n\n  FILE- dan oku bastaldi :  \n" ;
       ifstream FileOku("777.dat");        //енгізуші  файлдық ағын объектісін
                                  //жариялау және оны екілік  файлмен байланыстыру

 for(int i=0;i<3;i++)
 { FileOku.read((char*)& gruppa[i], sizeof (adam)); //ағыннан деректі алу
    cout<<"\n";
  cout<<gruppa[i].fam<<"  ";
  cout<<gruppa[i].stag<<"  ";
    cout<<gruppa[i].zarpl;
 }
    FileOku.close();
 getch();
}
```

Программа орындалуының нәтижесі келесі түрде болады:

```
FILE-dan oku bastaldi:
Asanova  20  100000
Mukasheva  25  75000
Muratova  15  50000
```

8.3. Файлға еркін қатынау

Файлға деректер жазуды немесе файлдан деректер (байттар) оқуды біріншісінен кейін екіншісіне көшіп, біртіндеп немесе файлдың кез келген көрсетілген позициясынан бастап жүргізуге болатыны мәлім (8.2-қараңыз). Файлдағы байттарды біртіндеп пайдалану – бұл көп қолданылатын, кең тараған әдіс. Осы кезге дейінгі файлға қатысты

қарастырылған жаттығулардың барлығында дерлік файлға қатынау біртіндеп жүргізілді. Бірақ программалауда көптеген есептерді шешуде файлға біртіндеп қатынау тиімді болып табылмайды, мысалы, дискідегі бумалардың өзі файл сияқты сақталады, ал бумадағы қажет файлды тауып, оны ашу үшін оған дейінгі файлдардың бәрін біртіндеп қарамайды, бірден сол қажет файлға түсу керек болады. Мұндай есептер өте көп, сондықтан бұл мәселені шешуде «*файлға еркін қатынау*» әдісін қолданады. Бұл әдісті жүзеге асыру үшін C++ программалау тілінде файлмен байланысты get-көрсеткіш және put-көрсеткіш деп аталатын екі көрсеткіш анықталған. Бұл көрсеткіштердің қызметі файлға жазу немесе файлдан оқудың қай жерден (позициядан) басталатынын көрсету болып табылады. Осы аталған екі көрсеткіштер негізінде құрылған, файлға еркін қатынауда көп қолданылатын функцияларға seekg() және seekp() функциялары жатады. Функциялардың программадағы жазылулары келесі түрде болады:

seekg(off_type *байтНомері*, seekdir *көрсеткішАғымдағыМәні*);
seekp (off_type *байтНомері*, seekdir *көрсеткішАғымдағыМәні*);

мұндағы *байтНомері* – байттың файлдағы позициясын көрсетеді, оның типі off_type, ios класында анықталған бүтін санды мән болып келеді, ал *көрсеткішАғымдағыМәні* – байтНомерін қай жерден бастап санауды көрсетеді. Мысалы, файлдың басынан бастап немесе соңынан санағанда және т.б. деген сияқты. Оның типі seekdir саналатын тип түрінде анықталған, келесі мәндерді қабылдай алады: ios::beg – файлдың басы, ios::cur – ағымдағы позиция және ios:: end – файлдың соңы.

Сонымен seekg() функциясы файлдың get-көрсеткішін *көрсеткішАғымдағыМәнінде* берілген орынға қатысты алғанда *байтНомері* позицияға жылжыта алады.

seekp() функциясы файлдың put-көрсеткішін *көрсеткішАғымдағыМәнінде* берілген орынға қатысты алғанда *байтНомері* позицияға жылжыта алады.

Бұл seekg() және seekp() функцияларын бір мәнділік сақталатын екілік файлдар үшін ғана қолданады.

Төменде seekp() функциясын қолдануды көрсететін жаттығулар берілген. Бұл жаттығуларды берілу реті бойынша жеке-жеке орындау керек.

8.7-жаттығу. Келесі программа пернетақтадан енгізілген бүтін сандарды massiv.dat екілік файлға жазуды орындайды:

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
int main()
{
 int k;
    ofstream FileGazu("massiv.dat");
 for(int i=0;i<5;i++)
  {
 cin>>k;
 FileGazu.write((char*)& k, sizeof (int));
 }
    FileGazu.close();
}
```

Бұл программа орындалуының нәтижесі төмендегі консол терезесінде

```
1
2
3
4
5
```

енгізілген бүтін сандардың massiv.dat файлына жазылуымен аяқталады.

8.8-жаттығу. Жоғарыдағы жаттығуда құрылған massiv.dat екілік файлына жаңа мәнді көрсетілген позициядан бастап жазуды көрсетеді:

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

```
{
    cout<<"engiz k=";
    cin>>k;                                  //файлға жазылатын жаңа мәнді енгізу
fstream FileOku("massiv.dat",ios::in|ios::out|ios::binary)  //massiv.dat файлын
                                  // екілік (бинарлық) режимде оқу-жазу үшін ашу
FileOku.seekp(8,ios::beg);    //файлдың put-көрсеткішін файлдың басынан
                                  //санағанда 8-позицияға апару
FileOku.write((char*)& k, sizeof (int)); //көрсеткіш тұрған позициядан бастап
                                  //жаңа мәнді жазу
    FileOku.close();
}
```

Программа орындалуының нәтижесі келесі түрде болады,

```
engiz k=
777
```

яғни пернетақтадан енгізілген 777 саны massiv.dat файлының басынан бастап санағандағы 8-позициясына орналасады.

8.9-жаттығу. Бұл жаттығу өзгеріске ұшыраған massiv.dat екілік файлынан, байт-деректерді позициялары бойынша оқып, экранға шығарып көрсетеді:

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
int main()
{  fstream FileOku("massiv.dat",ios::in|ios::binary); //файлды оқу үшін екілік
                                                      //режимде ашу
  int Poz=0;          //байттың позициясы, алғашқысының номері нольге тең
  while (Poz<=16)
  {
FileOku.seekp(Poz,ios::beg); //файл көрсеткішін көрсетілген позицияға апару
FileOku.read((char*)& k, sizeof (int)); //көрсетілген позициядан оқу
    cout<<k<<"\n";                //экранға шығару
    Poz=Poz+4;                    //келесі позицияны алу
```

```
    }
  FileOku.close();
   getch();
}
```

Бұл программаның орындалуының нәтижесі келесі түрде шығатын болады:

```
1
2
777
```

Бұл нәтижеден алдыңғы жаттығуда берілген **777** санының көрсетілген позицияға жазылғанын көруге болады. Сондай-ақ, мұнда да файлдан оқу файлдың басынан бастап (ios::beg) көрсетілген позициялар бойынша жүргізілгенін көруге болады.

Бақылау сұрақтары

1. Программалаудағы «ағын» ұғымы не үшін енгізілген?
2. Ағынның қандай түрлері бар?
3. Бинарлық ағын деген не?
4. Мәтіндік ағынның ерекшелігі не?
5. Стандарт ағындар қандай болады?
6. cerr, clog ағындарының қызметі қандай?
7. Буферлік ағындар мен буферлік емес ағындардың айырмашылығы неде?
8. Ағын кластарының сипаттамалары қандай файлдарда берілген?
9. Программа мен сыртқы енгізу-алу құрылғыларының арасындағы байланыс қалай орнатылады?
10. Ағын кластарының қандай түрлері бар?
11. Файл деген не?
12. Файлдар құрылу тәсілдері бойынша қалай бөлінеді?
13. Екілік (бинарлық) файл мен мәтіндік файлды пайдаланудағы негізгі айырмашылықтар қандай?
14. Файлға қатынаудың қандай түрлері бар?

15. Файлға біртіндеп қатынау қалай жүзеге асырылады?
16. Файлға еркін қатынау деген не?
17. Файлға еркін қатынау үшін файл қандай режимде ашылуы керек?
18. Файлға еркін қатынауда қандай функциялар қолданылады?
19. seekg() және seekp() функцияларының қызметі қандай?
20. Программадағы ofstream типті объектінің қызметі қандай?
21. Программадағы ifstream типті объектінің қызметі қандай?
22. Программадағы fstream.h файлының қызметі қандай?
23. Файлды ашудың қандай режимдері бар?
24. Файлды ашу режимдері не үшін қолданылады?
25. Файлдың логикалық интерфейсі деген не?
26. Программада файлдың соңын іздеуді ұйымдастыратын функция қалай жазылады?
27. Файлдың get-көрсеткіші не үшін қолданылады?
28. Файлдың put-көрсеткіші қандай мақсатта қолданылады?

Тапсырмалар

1. N компоненттен тұратын бүтін сандар файлын ұйымдастыру керек. Файлдың тақ индексті компоненттерінің қосындысын анықтап экранға шығарыңыз.

2. N компоненттен тұратын нақты сандар файлын ұйымдастыру керек. Файлдың барлық оң компоненттерін олардың квадрат түбірлерімен, ал теріс компоненттерін олардың квадраттарымен алмастырыңыз. Өңдегенге дейінгі және өңдегеннен кейінгі файл ішіндегілерін экранға шығарыңыз.

3. N компоненттен тұратын бүтін сандар файлын ұйымдастыру керек. Файлды өсуі бойынша сұрыптаңыз. Файлдағы максимал санды анықтап экранға шығарыңыз.

4. N компоненттен тұратын бүтін сандар файлын ұйымдастыру керек. Файлды өсуі бойынша сұрыптаңыз. Өңдегенге дейінгі және өңдегеннен кейінгі файл ішіндегілерін экранға шығарыңыз.

5. N компоненттен тұратын бүтін сандардың f файлын ұйымдастыру керек. Бұдан кейін g және h файлдарын құру керек. g файлына f файлынан барлық жұп сандарды, ал h файлына барлық тақ сандарды жазыңыз. Экранға f, g және h файлдарының ішіндегілерді шығарыңыз.

6. N компоненттен тұратын символдық f файлын ұйымдастыру керек. Содан кейін кері ретте орналасқан f файлының барлық компоненттері бар g файлын ұйымдастырыңыз. Файлдардың ішіндегілерді экранға шығарыңыз.

7. N компоненттен тұратын бүтін сандар файлын ұйымдастыру керек. Файлдың 10-нан бастап барлық элементтерін таңбалары қарама-қарсы 10 элементке алмастырыңыз. Өңдегенге дейінгі және өңдегеннен кейінгі файл ішіндегілерін экранға шығарыңыз.

8. N компоненттен тұратын бүтін сандар файлын ұйымдастыру керек. 3 санына еселік болатын барлық сандарды олардың екі еселенген көбейтіндісімен алмастырыңыз. Өңдегенге дейінгі және өңдегеннен кейінгі файл ішіндегілерін экранға шығарыңыз.

9. Зауыт жұмысшыларының айлық еңбекақылары жайында деректері бар файл қалыптастырыңыз. Әр құрылымда келесі өрістер болу керек: жұмысшы фамилиясы, цех атауы, еңбекақы өлшемі. Х цехы бойынша төлемнің жалпы сомасын (қосындысын), жұмыс цехының орташа табысын есептеңіз.

10. Цехта бір апта ішінде жинаушылар (сборщики) жинаған өнім саны бар файл қалыптастырыңыз. Әр құрылымда келесі өрістер болу керек: жинаушы фамилиясы, алты күндік аптада күнделікті жинаған өнім саны. Жинаушының фамилиясын және бір аптада жиналған бөлшектердің жалпы санын анықтайтын программа жазыңыз.

11. Типі «Көліктер» тақырыбына құрылған құрылым болып табылатын массив элементтерін файлға сақтаңыз. Файлдағы деректерді пайдаланып, қолданушының сұрауы бойынша көлікті іздеп тауып, оны барлық мәліметтерімен қосып экранға шығаратын программа құру керек.

12. Типі «Геометриялық фигуралар» тақырыбына құрылған құрылым болатын массив элементтерін файлға сақтаңыз. Файлдағы деректерді пайдаланып, қажет фигураның ауданы мен периметрін экранға шығаратын программа құру керек.

13. Файлдың компоненттері – типі «Геометрия. Стереометрия» тақырыбына құрылған класс болатын массив элементтері. Файлдағы деректерді пайдалана отырып қажет дененің бетінің ауданы мен көлемін экранға шығаратын программа құру керек.

14. Файлдың компоненттері – «Компьютердің құрылғылары» тақырыбы бойынша анықталған класс типті массив элементтері. Файлдағы деректерді пайдаланып ең қымбат ноутбук туралы мәліметтерді экранға шығаратын программа құрыңыз.

15. Мекеме қызметкерлерінің телефон нөмірлері класс түрінде жарияланады. Типі осы анықталған класс болатын массив элементтері файлға сақталады. Файлдағы деректерді пайдалана отырып, қызметкерлердің телефондарын іздеуді олардың фамилиясы мен инициалдары бойынша ұйымдастыратын программа жазыңыз.

16. Массив элементтерінің типі «Аэропорт» тақырыбына құрылған класс және олар файлда сақталған. Файлдағы деректерді пайдаланып, жолаушының фамилиясы, ұшу бағыты, салон түрі (мысалы, VIP, бизнес, эконом және т.б.) және құны көрсетілген билетті экранға шығаратын программа құрыңыз.

17. «Архитектуралық нысандар» тақырыбына құрылған класс, файлда сақталатын массив элементтерінің типі болып табылады. Файлдағы деректерді пайдаланып, архитектуралық нысандарды салынған уақыты бойынша сұрыптап экранға шығаратын программа құрыңыз.

18. Бір өлшемді массив элементтерінің мәндерін файлдан оқи отырып, оның жұп элементтерінің қосындысын есептеуді функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

19. Бір өлшемді массив элементтерінің мәндерін файлдан оқи отырып, оны «тікелей таңдау» әдісімен сұрыптаудың программасын жазыңыз.

20. Бір өлшемді массив элементтерінің мәндерін файлдан оқи отырып, оны «көпіршік» әдісімен сұрыптау программасын жазыңыз.

21. Өлшемі 3x5 болатын бүтін санды массив-матрица элементтерінің мәндері мәтіндік файлда берілген. Файлдағы деректерді пайдаланып, матрица жолдарындағы мәні ең үлкен элементті табуды функция түрінде жазыңыз және оны программада қолдануды көрсетіңіз.

22. Өлшемі 5x5 болатын бүтін санды массив-матрица элементтерінің мәндері мәтіндік файлда берілген. Файлдағы деректерді пайдалана отырып, матрицаның бас диагоналындағы мәні ең үлкен элементті анықтайтын программа құрыңыз.

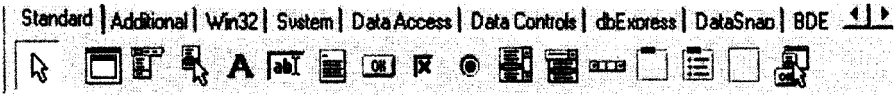
III БӨЛІМ. WINDOWS ҚОСЫМШАСЫН ПРОГРАММАЛАУ

- VCL компоненттер
- Графика
- Анықтама жүйесі
- Ерекше жағдайлар

9-тарау. VCL КОМПОНЕНТТЕРІ

9.1. Негізгі визуалдық компоненттер

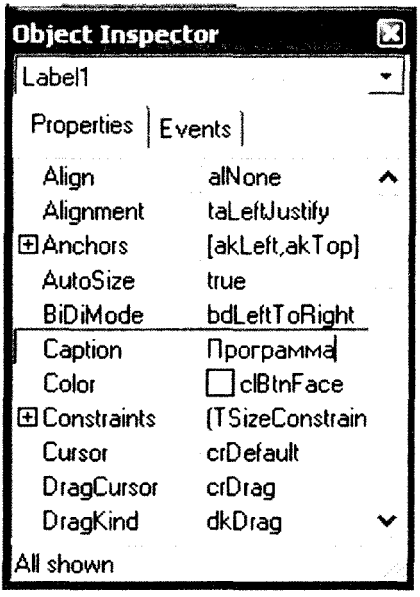
Windows жүйесінің терезелік қосымшаларын құруда қолданылатын негізгі компоненттер VCL (Visual Component Library) – визуалды компоненттер кітапханасында жинақталған. Визуалды компоненттерді қолдану программа құруды көп жеңілдетеді және жылдамдатады. Бұл компоненттер әдетте негізгі терезенің компоненттер палитрасында орналасады. Мысалы, Standart жапсырмасындағы компоненттер (9.1-сурет):



9.1-сурет. *Компоненттер палитрасы*

Компоненттердің қасиетін өзгерту екі түрлі жолмен жасалады: объектілер инспекторының (Object Inspector) терезесінде немесе программалау жолымен. Мысалы, Label1 компонентінің Caption қасиетін Object Inspector терезесінде өзгерту келесі түрде болса (9.2-сурет),программалық жолмен өзгерту үшін программа мәтінінде келесі түрде жазылуы керек:

```
Label1->Caption= "Программа";
```



9.2-сурет. *Object Inspector терезесі*

Объектінің немесе компоненттердің әдістерін шақыру келесі түрде жазылады:

Объектінің аты ->әдістің аты;

Мысалы, Memo1 компонентінің Clear әдісін шақыруды программада келесі түрде жазады:

```
Memo1->Clear();
```

Компоненттерге тән оқиғалар немесе функциялар келесі түрдегі

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{          ... ... ...  //C++ те жазылған программалық код }
```

шаблондар түрінде анықталған. Мұндай шаблон – функциялар программа мәтініне автоматты түрде бірден қойылады, шаблондарды сол

компоненттің белгісіне екі рет шерту немесе Object Inspector терезесіндегі Events жапсырмасындағы оқиғалар арқылы шақырып алуға болады. Объектілерді басқаруға арналған программалық кодтар осы функция-шаблондарға жазылады. Мысалы, Button1 компонентінің OnClick оқиғасына жазылған келесі:

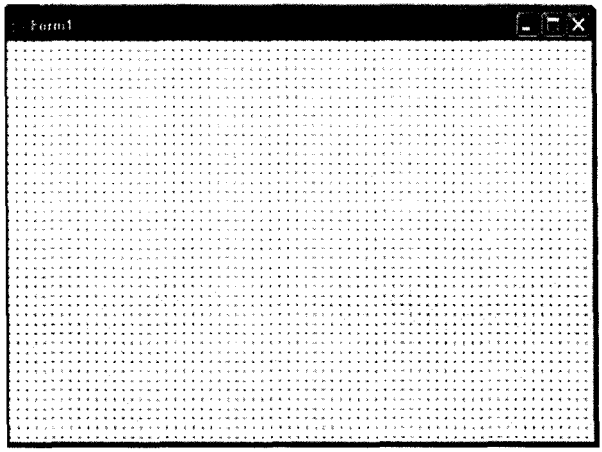
```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   Label1->Caption="Программа";
}
```

Программалық код орындалғанда форма терезесіндегі «Label1» деген жазу «Багдарлама» деген мәтінмен алмастырылатын болады.

9.1.1. **Form** *компоненті*

C++ Builder ортасында құрылатын қосымшаның орындалуы кезінде экранға шығатын визуалдық бөлімінің негізін TForm класының өкілі болып табылатын Form компоненті құрайды.

C++ Builder ортасы жүктелгенде, құрылатын жобаға (Project1) сәйкес форма терезесі (Form1) өзі автоматты түрде пайда болады (9.3-сурет). Форма терезесі – бұл Windows қосымшаларына тән тақырыптық зонасы,



9.3-сурет. *Форма терезесі*

жүйелік мәзір батырмасы, басқару батырмалары және таза жұмыстық облысы бар, өлшемі өзгеріп отыратын терезе.

Жобаға тағы да форма терезелерін қосу үшін File → New → Form командасы орындалады.

Қосымшада бірнеше форма болған жағдайда олардың кез келген біреуі (әдетте бұл Form1) негізгі, бас форманың ролін атқарады және ол программа орындалған кезде экранға бірінші болып шығады, ал негізгі форманың ролін кейін тұрған Form2, Form3, т.б. біреуі атқару керек болған жағдайда жоба программасының мәтініндегі формалардың құрылу тәртібі өзгертіледі.

Программа құруда Form компонентінің келесі қасиеттері мен әдістері жиі пайдаланылады:

TCaption типтер класына жататын Caption қасиеті форманың тақырыбын, яғни тақырыптық зонадағы мәтінді жазу үшін қолданылады;

TComponentName типіне жататын Name қасиеті Form компонентінің атын немесе идентификаторын беру үшін қолданылады, әдетте ол Form1, Form2 ... болып кете береді, егер қолданушы қажет болған жағдайда бұл атауларды өзгерте алады және программада оны сол өзі берген атпен шақырып пайдаланады, мысалы, Name қасиетінің мәні Form1-ден MainForm-ға өзгертілсе, онда оны программада пайдалануға

```
Form1→ Caption:= "Форма терезесі";
```

деп жазудың орнына,

```
MainForm →Caption:= "Форма терезесі";
```

түрінде жазады. Бұл – Name қасиеті C++ Builder ортасындағы қолданылатын барлық программалық компоненттерге тән қасиет.

Форманың Visible (типі Boolean) қасиеті C++ Builder ортасының визуалдық компоненттері үшін анықталған, егер бұл қасиеттің мәні

```
Form1→Visible = false ;
```

болса, программа орындалған кезде форма көрінбей тұрады, ал әдетте оның мәні true болып тұрады. Form компонентінің жоғарыда айтылған қасиеттерін объектілер инспекторының терезесінде өзгерту өте ыңғайлы болады және бұл терезеден оның тағы да басқа көптеген қасиеттерін көруге болады;

Программада Form компонентінің жаңа экземпляр-данасын құру үшін constructor Create(AOwner: TComponent); конструкторы қодаланылады, егер жаңа дананы құру команда түрінде талап етілетін болса, оны программа мәтінінде келесі түрде жазады:

```
Application→CreateForm(TForm1, Form1);
```

бұл программаның негізгі терезесін құру болып табылады, ал қалған терезелерін құруды, мысалы:

```
Form2=TForm2.Create(Application);
```

түрінде беруге болады.

Close функциясы форма терезесін жабуды қамтамасыз етеді, программадағы жазылуы:

```
Form1→ Close();
```

Hide функциясының қызметі – форманы экранда көрсетпей жасырып тұрады, жазылуы:

```
Form1→ Hide();
```

Show функциясы орындалғанда көрінбей тұрған форма алдыға шығып экранда көрсетіліп тұрады, жазылуы:

```
Form1→ Show();
```

Форма компонентінің TNotifyEvent класына жататын OnCreate оқиғасы форма құрылған кезде бір-ақ рет орындалады, сондықтан бұл функцияға форманың құрылуымен бірге орындалатын (мысалы, пароль енгізу сияқты) операциялар жазылады. OnCreate оқиғасына сәй-

кес функция Object Inspector терезесінің Events жапсырмасындағы OnCreate жолының терезесіне екі рет шерту арқылы шақырылады:

```
void __fastcall TForm2::FormCreate(TObject *Sender)
{ ... ... ... }
```

9.1.2. **Label, Memo, Edit** *мәтіндік компоненттері*

Терезелік қосымшаларда мәтінмен жұмыс жасауға арналған ең қарапайым деген әрекеттерді орындай алатын компоненттерге, Standart панелінің Label, Memo, Edit компоненттері жатады.

Label1 компоненті қосымша терезесінің бетіндегі көрініп тұратын мәтіндерді бейнелеу үшін қолданылады. Программада бұл мәтіндер Label компонентінің типі AnsiString болатын Caption қасиетінің мәндері ретінде беріледі. Мысалы,

```
Label. 1Caption="Бұл мәтін";
```

Әдетте Caption қасиетінің мәндерін объектілер инспекторының терезесінде беру ыңғайлырақ болады.

Label компонентінің WordWrap (типі **Bool**) қасиеті де программалауда жиі қолданылады, егер ол **true** мән қабылдаса, жолға сыймай қалған сөздер автоматты түрде келесі жолға тасымалданады.

Қосымшаларда мәтін түріндегі деректерді терезелер арқылы енгізуде Standart панелінің Edit, Memo компоненттері қолданылады.

Edit компоненті (немесе енгізу жолағы, немесе бір жолды редактор деп те атайды) мәтінді кішкентай терезе түріндегі бір жолға енгізуді немесе мәтін түріндегі нәтижені терезеге шығарып қоюды қамтамасыз етеді. Қосымшада Edit компонентінің терезесінде берілген мәтінді, курсорды оңға және солға жылжытатын бағыттауыштар көмегімен, символдарды өшіретін <Backspace> және <Delete> пернелерін пайдаланып және фрагменттерді бөліп алып жөндеуге болады. Edit компонентінің терезесіне енгізілген мәтін оның типі AnsiString болатын Text қасиетінің мәні болып табылады, яғни Edit терезесінде берілген деректер программада тек жолдық тип ретінде ғана қабылданады, ал оны әрі қарай программада сандар ретінде пайдалану үшін түрлендіру функцияла-

ры қолданылады. Терезеге енгізілген мәтінді бүтін (int) немесе нақты (**float**) типтердің біріне айналдыру үшін келесі түрлендіру функциялары қолданылады:

– StrToInt (AnsiString s) жолдық типті бүтін типке;

– StrToFloat(AnsiStrings) жолдық типті нақты типке түрлендіреді.

Керісінше, программада сандық шама түрінде есептелген нәтижелерді Edit терезесіне шығарылатын жолдық типке келтіру үшін қолданылатын түрлендіру функциялары:

– IntToStr(int Value) бүтін типті шаманы жолдық типке түрлендіреді,

– FloatToStr(Extended Value) немесе FloatToStrF(long double Value, TFloatFormat Format, int Precision, int Digits) нақты типті жолдық типке түрлендіреді, мұндағы Value – жолдық типке түрлендірілетін типі long double болатын сандық шама, Precision – түрлендіру дәлдігі, яғни түрлендірудегі цифрлардың жалпы санын білдіреді, Digits үтірден кейінгі цифрлардың санын көрсетеді, ал Format түрлендіру форматы деп аталады және ол келесі мәндерді қабылдайды. Мысал үшін n=3,141593654; k=5; m=2 деп алынса:

• ffGeneral – жалпы түрге келтіру, мысалы – 3,1416;

• ffExponent – экспоненциалдық түрге келтіру, мысалы – 3,1416E+00;

• ffFixed – жылжымайтын үтір арқылы жазылатын түрге келтіру, мысалы – 3,14;

• ffNumber – разрядтары ажыратылып жазылған түрге келтіру, мысалы – 3,1416*1000=3 141,60;

• ffCurrency – ақша бірлігімен жазылатын түрге келтіру, мысалы – 3 141,60 р.;

Edit терезесіндегі мәтіннің үлкен немесе кіші әріптермен жазылуын немесе регистрлердің ауысып отыруын оның CharCase (типі TEditCharCase) қасиеті арқылы береді, сәйкесінше ол үш түрлі мән қабылдайды:

– ecNormal – регистр өзермейді, сол күйінде тұрады;

– ecLowerCase – төменгі регистрдің әріптеріне көшеді;

– ecUpperCase – жоғарғы регистрдің бас әріптеріне ауысады.

Edit компонентінің типі Char болатын PasswordChar қасиеті осы терезеге пароль енгізу үшін қолданылады. Әдетте оның мәні #0 болып тұрады да, терезеге енгізілген мәтіннің өзі шығады. Ал оны жасырып

көрсетпеу үшін PasswordChar қасиетінің қабылдайтын мәнін басқа бір символға, мысалы, «*» немесе «?» секілді мәндерге өзгерту керек, сонда енгізілген құпия мәтіннің орнына «********» немесе «?????????» түріндегі жолдар шығады.

Программа терезесінде Edit1, Edit2, Edit3 компоненттері арқылы берілген бірнеше терезелердің бірінен екіншісіне ENTER пернесі арқылы көшіп отыру үшін оларға басқару фокусын беруді (курсорды беретін) қамтамасыз ететін SetFocus әдісін қолдануға болады, ол әдіс осы компоненттің терезесінде тұрып басылған пернеге жауап беретін OnKeyPress оқиға өңдеушісінің денесінде шақырылады. Мысалы, Edit2 компонентінің терезесінен ENTER пернесі арқылы Edit3 терезесіне көшу үшін сол Edit2-нің OnKeyPress оқиға өңдеушісіне келесі кодты жазуға болады:

```
void __fastcall TForm2::Edit2KeyPress(TObject *Sender, char &Key)
{
if(Key>='0' || Key<='9') ; // бұл 0- 9 цифрларды көрсету
if(Key==8) ; // Backspace – өшіру пернесінің ішкі коды
if(Key==13) // ENTER пернесінің ішкі коды
    Edit3->SetFocus();
}
```

Ал форма терезесіндегі барлық терезелік компоненттерде басқару фокустарының өзара ауысып отыруын ұйымдастыру үшін Form компонентінің SelectNext (TWinControl* CurControl, bool GoForward, bool CheckTabStop); әдісі қолданылады.

Memo компоненті (көп жолды редактор деп те атайды) мәтінді бірнеше жолдары бар, айналдыру жолағы болатын үлкен терезеге енгізуді немесе шығаруды қамтамасыз етеді.

Memo компонентінің терезесіне енгізілген мәтінді тұтасымен пайдалану үшін оның типі AnsiString болатын Text қасиеті, ал жеке-жеке жолдар түрінде пайдалану үшін өзіне тән қасиеттері мен әдістері бар TAnsiString класымен анықталған типке жататын Lines қасиеті қолданылады. TAnsiString класында жолды номерлеу нольден басталады.

Memo (немесе Edit) компонентімен берілген терезелерді тазалау үшін Clear әдісі қолданылады, мысалы:

Memo2->Clear(); – Memo2 терезесін толығымен тазалайды.

Add әдісі терезедегі мәтіннің соңына жаңадан берілген жолды апарып тіркейді, мысалы,

Memo3->Lines->Add ("Жаңа жол тіркеледі");

Қосымшадағы Memo терезесін мәтіндік файлдарды ашып оқу үшін немесе терезедегі мәтінді текстік файлға жазып сақтау үшін де қолдануға болады. Ол үшін оның SaveToFile (FileName) әдісі – терезедегі мәтінді файлға жазуды, ал LoadFromFile (FileName) әдісі – керісінше, файлдағы мәтінді Memo терезесіне шығаруды қамтамасыз етеді. Мысалы:

```
...
AnsiString FileName = "C:\\WINDOWS\\WIN.INI"; // файлдың аты
Form1->Memo1->Lines->LoadFromFile(FileName);
Form1->Memo1->Lines->SaveToFile(ChangeFileExt(FileName, ".BAK"));
...
```

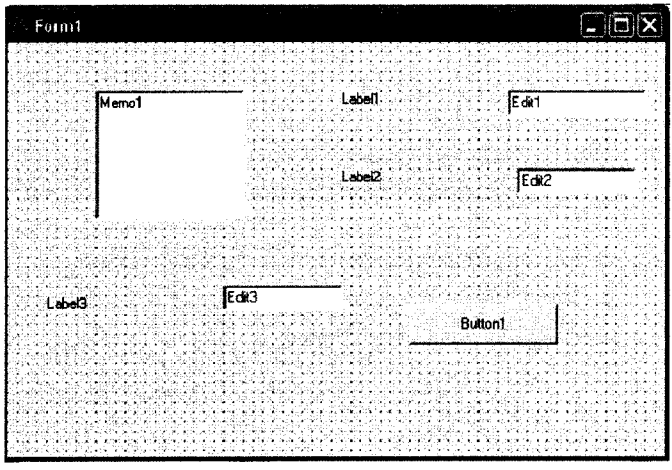
9.1.3. ***Button*** *басқарушы компоненті*

Button компоненті осы компонентке бекітілген кодқа сәйкес командаларды (нұсқауларды немесе әрекеттерді) орындайды. Бұл компонентке код жазу үшін оның форма терезесіндегі белгісіне екі рет шертеді, нәтижесінде модуль терезесі (немесе код редакторының терезесі) ашылады және ол терезеде Button компонентінің OnClick әдісіне сәйкес функцияның шаблоны пайда болады:

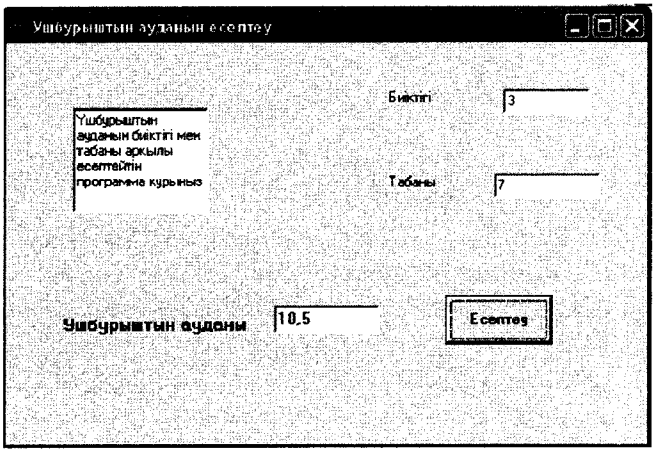
```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ... ... ...
}
```

9.1-жаттығу. Үшбұрыштың ауданын, биіктігі мен табаны бойынша есептеуді орындайтын қарапайым қосымша құру. Қосымша терезесі келесі түрде болуы қажет (9.4, а және 9.4, б-сурет).

Жаттығудың орындалуы:



а) Форма терезесі



б) Қосымша терезесі

9.4-сурет. *Үшбұрыштың ауданын есептеуді орындайтын қарапайым қосымшаның терезелері*

1. File→ New → Application командалары орындалады.

2. Жаңа құрылған жобаны сақтау үшін File→ Save All командасы орындалады. Жоба құрамына кіретін файлдардың бір бумада сақталуы қажет.

3. Форма терезесіне қажет компоненттер 9.4, а-суреттегідей орналастырылып және олардың келесі қасиеттері өзгертіледі:

Компоненттің аты	Қасиетінің аты	Мәні
Form1	Caption	Үшбұрыштың ауданын есептеу
Memo1	Lines	Үшбұрыштың ауданын биіктігі мен табаны бойынша есептейтін программа құрыңыз
Label1	Caption	Биіктік
Label2	Caption	Табан
Label3	Caption	Үшбұрыштың ауданы
Edit1	Text	
Edit2	Text	
Edit3	Text	
Button1	Caption	Есептеу

4. «Есептеу» деп аталатын батырмаға екі рет шертіліп, пайда болған терезеге келесі код жазылады:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{   int biktik, taban;
        float audan;
   biktik=StrToInt(Edit1->Text);
   taban=StrToInt(Edit2->Text);
   audan=float(biktik*taban)/2;
      Edit3->Text=FloatToStr(audan);
    }
```

9.2. Тізімдер құру. ListBox, ComboBox компоненттері

C++Builder ортасында құрылатын қосымшалардағы тізімнің көмегімен таңдау жұмысын ұйымдастыру үшін Standart панелінің ListBox, ComboBox компоненттерін қолдануға болады.

Standart панелінің ListBox, ComboBox компоненттері. ListBox компоненті қарапайым тізімдерді жасауда қолданылады (9.6-сурет).

ComboBox компоненті тізімді таңдау және енгізу терезесінің қызметін қатар атқаратын аралас тізімдер жасау үшін қолданылады (9.7-сурет).

ListBox, ComboBox компоненттерінің негізгі қасиеттері:

Style (TListBoxStyle) қарапайым тізімнің стилін анықтайды, үш түрлі мән қабылдайды, осыған байланысты тізімнің қосымшадағы түрі өзгеріп отырады.

BorderStyle (TBorderStyle) қасиетінің мәндеріне сәйкес қарапайым тізім рамкасының көріну-көрінбеуі анықталады.

DroppedDown (Bool) қасиеті аралас тізімнің қосымшада толық ашылған немесе бір жол түрінде көрініп тұруын қамтамасыз етеді.

Items (TAnsiStrings) қасиеті тізімдерді бір өлшемді жолдық массив түрінде қарастыруға мүмкіндік береді. Тізімдегі әрбір жол-жолдық массивтің элементі болып есептеледі және оларды номерлеу нольден басталады. Мысалы,

ListBox1->Items[3]="Петров С. У." // (9.6-сурет).

Columns қасиеті тізім жолдарын бағандарға бөліп орналастырғандағы бағандардың санын көрсету үшін қолданылады және типі Int, әдетте оны Object Inspector терезесінде көрсетеді.

Count (Int) – тізімдегі элементтердің (жолдардың) санын білдіреді, демек тізімдегі бірінші элементтің номері 0 болса, онда соңғы элементтің номері ListBox.Items. Count-1 болады.

ItemIndex (Int) – қасиеті тізімнен қажет элементті таңдап алуды қамтамасыз етеді. Мысалы, ListBox1->ItemIndex=2 операторы тізімдегі үшінші жолдың таңдалғанын білдіреді.

Sorted (Bool) – қасиеті true мән қабылдағанда тізім алфавит бойынша орналасады, ал false болса, кез келген ретпен тұрады.

Selected (Bool) – қасиеті тізімді программалық жолмен басқаруда программа текстінде тізімнің қандай элементі таңдалғанын көрсету үшін қолданылады. Мысалы,

```
ListBox1->Selected[1]=true;
ListBox1->Selected[3]=true;
```

тізімнің екінші және төртінші элементінің таңдалғанын білдіреді.

Add әдісі ListBox, ComboBox компоненттерінің екеуіне де ортақ, ол берілген тізімнің соңына жаңа жолды тіркеу үшін қолданылатын функция ретінде программада келесі түрде жазылады:

```
Add(const S:AnsiString);
```

мұндағы S – тіркелетін жолдың мәтіні, ал функцияның нәтижесі сол тіркелген жолдың номерін көрсету болып табылады. Мысалы,

```
ComboBox1->Items.Add('Gana gol kosildi');
```

Процедура түрінде анықталған Insert әдісінің қызметі – бұл тізімнің кез келген жеріне жаңа жолды кірістіру, программадағы жазылуы:

```
Insert(int Index; const AnsiString S);
```

мұнда S – жол, тізімге Index номермен барып кіргізіледі, ал сол жердегі бұрынғы жол және қалғандары бір позицияға төменге түсіріледі. Мысалы,

```
ComboBox1->Items.Insert(3,"№ 2 gana gol kosildi");
```

Delete процедурасы номері көрсетілген жолды тізімнен алып тастауды жүзеге асырады, жазылуы:

```
Delete(int Index);
```

Мысалы, программада ListBox1->Items.Delete(3); жолы орындалғанда тізімнің екінші жолы алынып тасталады.

Тізімді толығымен тазалау үшін Clear әдісін пайдаланады, мысалы:

```
ListBox1->Items.Clear;
```

жолы орындалғанда тізімде ештеңе қалмайды.

Берілген S – жолдың тізімде бар-жоғын және нешінші орында тұрғанын анықтау үшін IndexOf функциясын қолданады, жазылуы:

```
int IndexOf( const AnsiString S);
```

Тізімдегі жолдарды мәтіндік (.txt) файлға жазып сақтау үшін немесе берілген мәтіндік файлдағы жолдарды оқып, оны тізімге көшіру үшін сәйкесінше келесі әдістерді қолдану қарастырылған:

```
SaveToFile(const AnsiString FileName);-
```

файлға жазып сақтау, ал

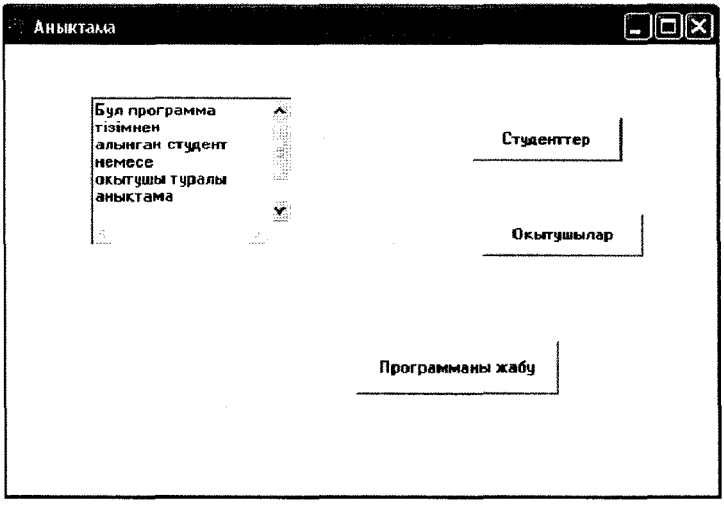
LoadFromFile (const AnsiString FileName); – файлдан оқып тізімге көшіру.

Мысалы, ComboBox1->Items.LoadFromFile("c:\Gruppa\Family.txt");

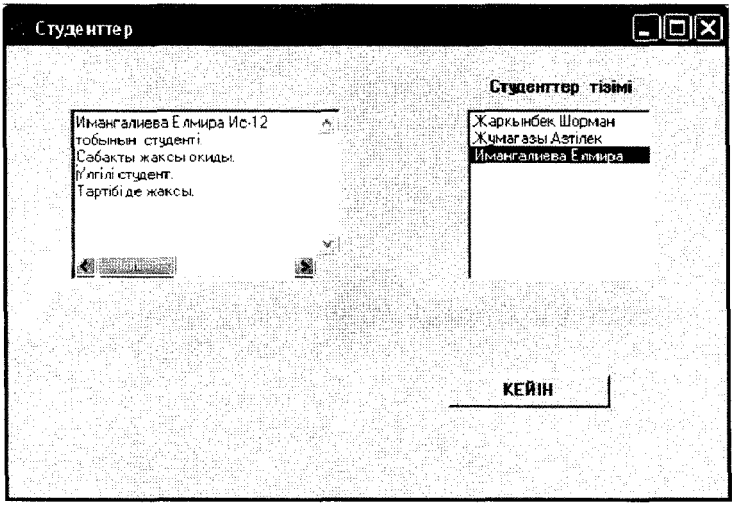
9.2-жаттығу. Студент пен оқытушы туралы ақпарат бере алатын қарапайым қосымша құру керек. Әрбір студент немесе оқытушы туралы ақпарат оның фамилиясына сәйкес аталатын *.txt файлда сақталсын. Қосымша құруда тізімдерді пайдаланыңыз.

Жаттығудың орындалуы:

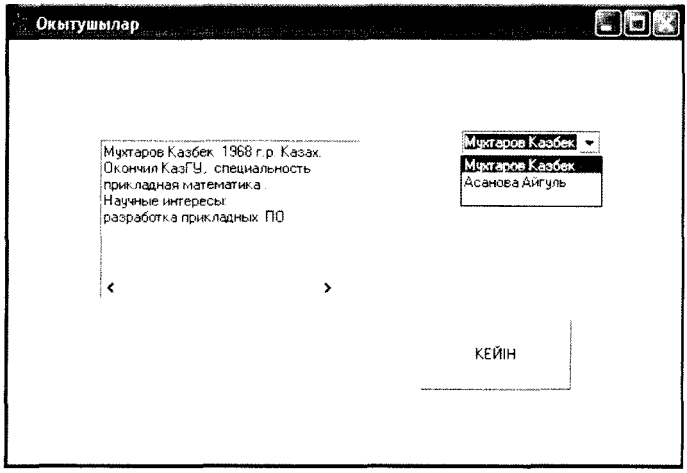
1. Қосымша үш (9.5, 9.6, 9.7-суреттер) терезеден тұрады. «Анықтама» терезесіндегі батырмалар көмегімен «Студенттер» немесе «Оқытушылар» терезелерінің біріне көшуге болады. Бұл терезелерде сәйкесінше студент немесе оқытушы туралы нақты мәлімет алынады.



9.5-сурет. *«Анықтама» терезесі*



9.6-сурет. *«Студенттер» терезесі*



9.7-сурет. *«Оқытушылар» терезесі*

2. «Анықтама» терезесі (Form1) үшін жазылған программа мәтіні:

```
#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{
}
//«Студенттер» батырмасының оқиға өңдеуші коды
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}
// «Оқытушылар» батырмасының оқиға өңдеуші коды
```

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
// «Программаны жабу» батырмасының оқиға өңдеуші коды
void __fastcall TForm1::Button3Click(TObject *Sender)
{
Form1->Close();
}
```

3. «Студенттер» терезесі (Form2) үшін жазылған программа мәтіні:

```
#include "Unit2.h"
#include "Unit1.h"
#include "Unit3.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "SHDocVw_OCX"
#pragma resource "*.dfm"
TForm2 *Form2;
//---------------------------------------------------------------------------
__fastcall TForm2::TForm2(TComponent* Owner)
    : TForm(Owner)
{
}
// «КЕЙІН» батырмасының оқиға өңдеуші коды
void __fastcall TForm2::Button1Click(TObject *Sender)
{
Form1->Show();
Form3->Hide();
Form2->Hide();
}
// Студенттер тізімі жазылған :ListBox1 компонентін шерту оқиғасының
//коды
void __fastcall TForm2::ListBox1Click(TObject *Sender)
{
  if(Form2->ListBox1->ItemIndex==0)
```

```
        Memo1->Lines->LoadFromFile  (“Жаркынбек.txt”) ;
  if (Form2->ListBox1->ItemIndex==1)
        Memo1->Lines->LoadFromFile(“Жумагазы.txt”);
  If (Form2->ListBox1->ItemIndex==2)
       Memo1->Lines->LoadFromFile(“Имангалиева.txt”);
 }
```

4. «Оқытушылар» терезесі (Form3) үшін жазылған программа мәтіні:

```
#include “Unit3.h“
#include “Unit1.h“
#include “Unit2.h“
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link “SHDocVw_OCX“
#pragma resource“*.dfm“
TForm3 *Form3;
//---------------------------------------------------------------------------
__fastcall TForm3::TForm3(TComponent* Owner)
      : TForm(Owner)
{
}
//Оқытушылар тізімі жазылған ComboBox1 компоненті  оқиғасының коды
void __fastcall TForm3::ComboBox1Change(TObject *Sender)
{
    if(ComboBox1->ItemIndex==0)
            Memo1->Lines->LoadFromFile(“Мухтаров.txt”);
   if(ComboBox1->ItemIndex==1)
           Memo1->Lines->LoadFromFile(“Асанова.txt”);
 }
// «КЕЙІН» батырмасының оқиға өңдеуші коды
void __fastcall TForm3::Button1Click(TObject *Sender)
{
Form1->Show();
Form3->Close();
Form2->Hide();
}
```

9.3. Қосымшадағы ауыстырғыштардың қызметі

C++Builder ортасында программалауда, программа ұсынатын мүмкіндіктердің бірін немесе қатарынан бірнешеуін таңдау үшін ауыстырғыштар (переключатель, флажок) қолданылады. Программада мұндай ауыстырғыштардың қызметін Standart панелінің CheckBox, RadioButton және RadioGroup компоненттері атқарады. Ауыстырғыш екі күйде ғана бола алады: «қосылған» күйде (☑ немесе ⦿) және «ажыратылған» күйде (□ немесе ○). Ауыстырғыштарды мүмкіндігіне қарай екі топқа бөледі: тәуелсіз және өзара тәуелді. Тәуелсіз ауыстырғыштардың жұмысы басқа ауыстырғыштарға байланыссыз, тәуелсіз болады.

9.3.1. **CheckBox** *компоненті*

Программада CheckBox компонентінің көмегімен жасалатын ауыстырғыш, тәуелсіз ауыстырғыштар тобына жатады. Әдетте қажет болған жағдайда бірнеше ауыстырғыштарды бір топқа біріктіріп пайдаланады, ол үшін контейнердің ролін атқаратын арнаулы CheckBox және Panel және т.б. компоненттер қолданылады.

Қосымшаларды құруда CheckBox компонентінің келесі қасиеттері қолданылады:

Checked қасиеті Bool типіне жатады және ауыстырғыштың екі күйін көрсету үшін қолданылады, яғни ауыстырғыш «қосылған» күйде болғанда true мән қабылдайды, ал «ажыратылған» болса, false мән қабылдайды. Мысалы,

```
CheckBox1-> Checked=true;
CheckBox2-> Checked=false;
```

Bool типті Enabled қасиеті ауыстырғыштың бір күйден екінші күйге ауысып отыруына рұқсат беру-бермеуді қамтамасыз етеді. Мысалы,

State қасиеті арнаулы TCheckBoxState типіне жатады және ауыстырғыштың, мүмкін болатын үш күйдің: «қосылған», «ажыратылған», «рұқсат берілмеген» бірінде болуын жүзеге асырады. Қабылдайтын мәндері:

cbUnchecked – «ажыратылған»,
cbChecked – «қосылған»,
cbGrayed – «рұқсат берілмеген».

Әдетте қосымшада ауыстырғыштың күйін өзгерту үшін тышқанның сол жақ батырмасына шертеді, яғни қандай болғанына қарамастан, әйтеуір күйі өзгеретін болса, онда CheckBox компоненті үшін міндетті түрде OnClick оқиғасы пайда болады. Бұл оқиғаны өңдеуге арналған функция, көбінесе ауыстырғыштың күйіне байланысты белгілі бір әрекеттердің немесе операторлардың орындалуын қамтамасыз ету үшін қолданылады. Мысалы,

```
void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
if (CheckBox1->Checked = = true)
Memo1->Font->Size=10;
}
```

9.3.2. **RadioButton** *компоненті*

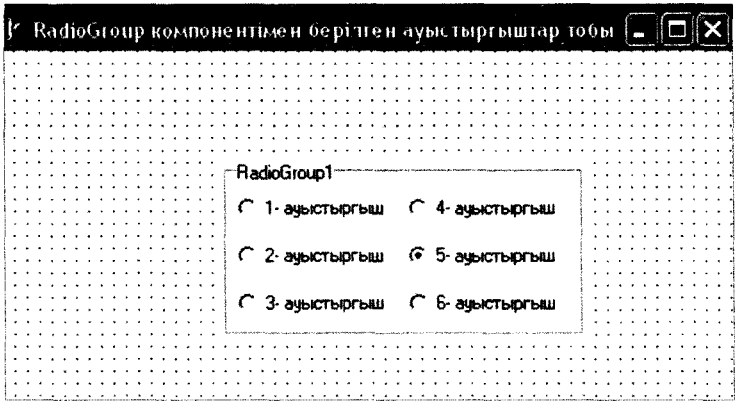
RadioButton компонеті тәуелді ауыстырғыштар қатарына жатады, яғни бір контейнерде орналасқан RadioButton түріндегі бірнеше ауыстырғыштардың біреуі «қосылған» күйде болса, онда қалғандарының барлығы бірдей автоматты түрде «ажыратылған» күйге көшеді.

RadioButton компонетінің күйін таңдау үшін Boolean типіне жататын Checked қасиетін пайдаланады.

RadioButton компонетінің, тәуелсіз ауыстырғыш CheckBox компонентінен өзгешелігі, мұнда OnClick оқиғасы ауыстырғышты «қосылған» күйге келтіру үшін бір рет қана шақырылады, ал оны алып тастау үшін қайта шерткенде бұл оқиға орындалмайды, сол себепті «қосылған» күйде тұрған тәуелді ауыстырғышты, «ажыратылған» күйге келтіру үшін сол топтағы басқа бір ауыстырғышты таңдауға тура келеді.

9.3.3. **RadioGroup** *компоненті*

Қосымша құру барысында бір топқа біріктіріліп және номерлері бойынша реттеліп тұрған ауыстырғыштар тобын пайдалануды RadioGroup компоненті жасайды. Мұнда ауыстырғыштарды топтастыру үшін контейнердің қажеті болмайды.



9.8-сурет. *RadioGroup1 компоненті қойылған форма терезесі*

Қосымшаларды құруда RadioGroup компонентінің келесі қасиеттерін жиі пайдаланады:

Columns қасиеті RadioGroup тобындағы ауыстырғыштардың орналасуындағы бағандардың санын көрсету үшін қолданылады және типі Integer, оны Object Inspector терезесінде көрсеткен ыңғайлы болады, мысалы, 9.8-суреттегі RadioGroup1 тобының ауыстырғыштары екі баған түрінде орналасқан, яғни:

```
RadioGroup1-> Columns=2;
```

Items қасиетінің типі AnsiString болады және ол топтағы ауыстырғыштардың атын беру үшін қолданылады. Мысалы, «1-ауыстырғыш», «2-ауыстырғыш» (9.8-сурет) және т.б. атаулар осы Items қасиетінің көмегімен беріледі. RadioGroup тобындағы ауыстырғыштар реттелген болғандықтан, оларды қажет болған жағдайда массив элементтері сияқты пайдалануға болады, ол үшін Items қасиеті пайдаланылады, мысалы:

```
RadioGroup1->Items[0]="1-ауыстырғыш";
```

Бүтін типті ItemIndex қасиеті топтағы ауыстырғыштарды номері бойынша таңдауды (немесе «қосылған» күйге келтіруді) ұйымдастырады, әдетте бірде-бір ауыстырғыш «қосылмаған» болса, онда

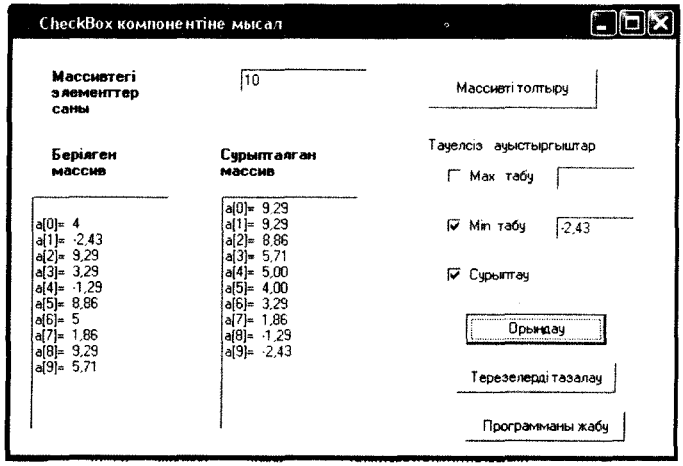
```
RadioGroup1->ItemIndex= -1;
```

болғаны, ал жоғарыда келтірілген суретте (9.8-сурет) :

```
RadioGroup1->ItemIndex= 4;
```

Сондай-ақ, RadioGroup компонентімен берілетін ауыстырғыштар тобына жаңа ауыстырғышты жол ретінде тіркеу үшін Add(), ал алып тастау үшін Delete(), толығымен тазалау үшін Clear () әдістерін қолдануға болады.

9.3-жаттығу. Кездейсоқ сандар генераторын (random () функциясын) пайдаланып құрылған массивтің max және min элементтерін табуды және массивті сұрыптауды қолданушының сұранысына байланысты бір-біріне тәуелсіз орындай алатын қосымша құрыңыз (9.9-суреттегідей).



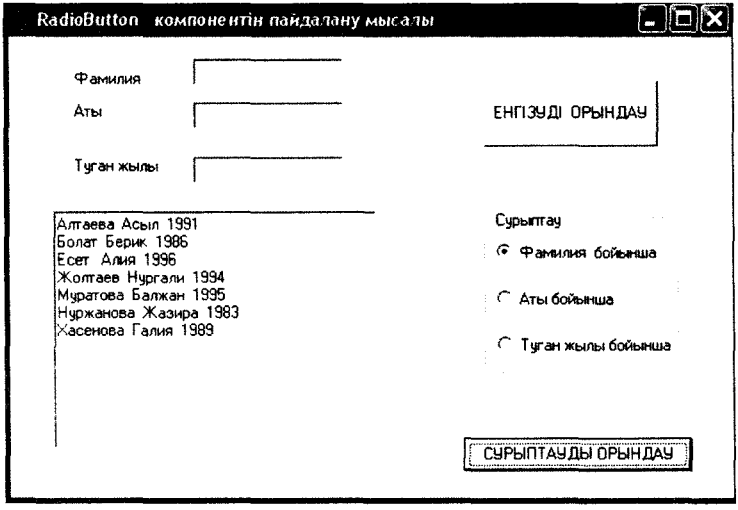
9.9-сурет. *Қосымша терезесі*

Жаттығудың орындалуы:

```
#include "Unit1.h"
#include "stdlib.h"
TForm1 *Form1;
float a[100];
int  Elem_sani=0;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{
}
//---Орындау батырмасының коды ----------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  float max=a[0];
  float  min=a[0];
  if (Edit1->Text=="")
  {ShowMessage("Massiv elementterinin sanin beriniz");
  Edit1->SetFocus();}
  if (CheckBox1->Checked == True)   //max элементті табу
{for (int  i=0;i<Elem_sani;i++) {
 if (a[i]>max) max=a[i];
 }
 Edit2->Text=FloatToStrF(max,ffFixed,5,2);
 }
 if (CheckBox2->Checked == True)   //min элементті табу
 {for (int  i=0;i<Elem_sani;i++) {
  if (a[i]<min) min=a[i];   }
  Edit3->Text=FloatToStrF(min,ffFixed,5,2);
  }
  if (CheckBox3->Checked == True)  // массивті өсу ретімен сұрыптау
  for(int  i=0;i<Elem_sani;i++)
  { max=a[i]; int  Nmax=i;
  for(int  j=i;j<Elem_sani;j++)
```

```
 if (a[j]>max) {max=a[j]; Nmax=j;}
 a[Nmax]=a[i]; a[i]=max;
Memo2->Lines->Add("a["+Int ToStr(i)+"]= "+FloatToStrF(max,ffFixed,5,2)); }
 }
//------Массивті толтыру батырмасының коды----------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
if (Edit1->Text=="")
{ShowMessage("Massiv elementterinin sanin beriniz");
 Edit1->SetFocus();}
 Elem_sani=StrToInt (Edit1->Text);
 for(int i=0; i<Elem_sani;i++)
 { a[i]= (float) random(100)/7-3;
Memo1->Lines->Add("a["+Int ToStr(i)+"]= "+FloatToStrF(a[i],ffGeneral,3,1));}
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormActivate(TObject *Sender)
{Edit1->SetFocus();}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button4Click(TObject *Sender)
{Form1->Close();}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{  Memo1->Clear();  Memo2->Clear();
  Edit1->Clear(); Edit2->Clear(); Edit3->Clear();}
//---------------------------------------------------------------------------
```

9.4-жаттығу. Студенттер туралы деректер (фамилиясы, аты және туған жылы) класс түрінде анықталған. Группадағы студенттер туралы деректерді сұрыптауды (фамилиясын немесе атын алфавит бойынша, туған жылдарын кему реті бойынша орналастыру) орындайтын программа жазыңыз. Программа терезесі 9.10-суретте көрсетілген.



9.10-сурет. *Қосымша терезесі*

Жаттығудың орындалуы:

1. Терезедегі компоненттер қасиеттерінің және оқиғаларының мәндері:\

Компоненттің аты	**Қасиеттерінің мәндері, оқиғалары**
Form1	Caption = "RadioButton компонентін пайдалану мысалы'
Label1	Caption = "Фамилия"
Label2	Caption = "Аты"
Label3	Caption = "Туған жылы"
Memo1	Lines.Strings = (" ")
Edit1	Text қасиеті тазаланады, OnKeyPress = Edit1KeyPress
Edit2	Text қасиеті тазаланады, OnKeyPress = Edit2KeyPress
Edit3	Text қасиеті тазаланады, OnKeyPress = Edit3KeyPress
Button1	Caption = "ЕНГІЗУДІ ОРЫНДАУ", OnClick = Button1Click
Button2	Caption = "СҰРЫПТАУДЫ ОРЫНДАУ", OnClick = Button2Click

RadioGroup1 Caption = "Сұрыптау"
Items.Strings =
("Фамилия бойынша"
"Аты бойынша'
"Туған жылы бойынша ")

Программалық коды келесі түрде болады:

```
#include "Unit1.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
class stud
{
  public:
      AnsiString fam, ati;
      int tugG;
 };

  stud  grup[10]; int n=0;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

// 'ЕНГІЗУДІ ОРЫНДАУ' батырмасының коды
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 grup[n].fam=Edit1->Text;
 grup[n].ati=Edit2->Text;
 grup[n].tugG=StrToInt(Edit3->Text);
 Memo1->Lines->Add(grup[n].fam+" "+
 grup[n].ati+" "+IntToStr(grup[n].tugG));
 n++;
```

```
 Edit1->Clear();Edit2->Clear();Edit3->Clear();
 Edit1->SetFocus();
 }
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1KeyPress(TObject *Sender, char &Key)
{
if(Key==13) Edit2->SetFocus();
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Edit2KeyPress(TObject *Sender, char &Key)
{
  if(Key==13) Edit3->SetFocus();
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Edit3KeyPress(TObject *Sender, char &Key)
{
if(Key==13) Button1->SetFocus();
}

//---------------------------------------------------------------------------

// 'СҰРЫПТАУДЫ ОРЫНДАУ' батырмасының коды
void __fastcall TForm1::Button2Click(TObject *Sender) {
if(RadioGroup1->ItemIndex==0) //fam boinsha syriptay bastaldi
{  Memo1->Clear();
for(char i='A';i<='Я';i++)
for (int j=0;j<n;j++)
if(grup[j].fam[1]==i)
Memo1->Lines->Add(grup[j].fam+" "+
grup[j].ati+" " +IntToStr(grup[j].tugG));
} // fam boinsha syriptay bitti

if(RadioGroup1->ItemIndex==1) //ati boinsha syriptay bastaldi
{  Memo1->Clear();
```

```
for(char i='A';i<='Я';i++)
for (int j=0;j<n;j++)
if(grup[j].ati[1]==i)
Memo1->Lines->Add(grup[j].ati+" "+
grup[j].fam+" "+IntToStr(grup[j].tugG));
} //ati boinsha syriptay bitti

if(RadioGroup1->ItemIndex==2)  //tugG boinsha syriptay bastaldi
{  Memo1->Clear();
for(int i=0;i<n;i++)
{ stud max=grup[i]; int maxI=i;
for (int j=i;j<n;j++)
if(grup[j].tugG > max.tugG)
{  max=grup[j]; maxI=j;}
Memo1->Lines->Add(max.ati+" "+
max.fam+" "+IntToStr(max.tugG));
grup[maxI]=grup[i];
grup[i]=max;
}
} //tugG boinsha syriptay bitti
}
```

9.4. Қосымшада мәзірлердің жұмысын ұйымдастыру

Барлық Windows қосымшаларында қолданушы интерфейсінің элементі ретінде ең көп тараған элемент – *мәзір*. *Мәзір* – қосымшада орындалатын командаларды атқаратын қызметтеріне немесе т.б. белгілеріне қарай бір топқа біріктіріп тиімді пайдалану үшін қолданылады. Windows қосымшаларында қолданылатын мәзірлерді келесі түрлерге бөледі:

– *жүйелік мәзір* (SystemMenu) бұл – Windows қосымшаларының барлығына тән мәзір түрі, сондықтан C++Builder-де қосымша құрғанда жүйелік мәзір форманың өзімен қоса анықталады. Жүйелік мәзірдің сурет-таңбасы, форманың тақырыптар зонасының сол жағында, ал

оның командаларына сәйкес сурет-таңбалар (_ , ☒ ,□ немесе 🗗) оң жағында орналасады және оларды форманың BorderIcons (типі TBorderIcons) қасиетіндегі biSystemMenu (типі Boolean) мәнін false өзгерту арқылы алып тастауға болады.

– қосымшадағы *негізгі мәзір* (немесе оны бас мәзір, горизонталь мәзір деп те атайды) формаға MainMenu компонентінің көмегімен қойылады. Әдетте Windows қосымшалары негізгі мәзір көмегімен басқарылады. Қосымшада негізгі мәзір біреу-ақ болады. Бұл MainMenu компоненті визуалдық емес компоненттер қатарына жатады, яғни бастапқы проектілеу кезіндегі форма терезесінде көрініп тұрғанымен, программа орындалған кездегі шығатын терезеде көрінбей тұрады.

– қосымшаның жекелеген элементтеріне арналып жасалатын *контекстік мәзір* (тышқанның оң жақ батырмасын басқанда пайда болатын мәзір) қосымшаға визуалдық емес PopupMenu компонентінің көмегімен енгізіледі. Қосымшадағы әрбір элементтің, (мысалы, Button1 немесе Button2) әрқайсысының атқаратын қызметіне қарай, командалары әр түрлі болатын бөлек-бөлек мәзірлер тағайындауға болады. Ол үшін формадағы орналасқан компоненттердің әрқайсысына сәйкес сол формаға бірнеше PopupMenu1, PopupMenu2, PopupMenu3, ...қойылады және әрбір компоненттің (мысалы, Button1-дің) PopupMenu қасиетінің мәні өзіне сәйкес контекстік мәзірге (мысалы, PopupMenu3-ке) өзгертіледі.

Қосымшада мәзірлердің командалары бір жол немесе жолдардан тұратын баған-тізім түрінде орналасады, сондықтан C++Builder-де мәзірдің командасын *пункт* деп атайды және соған сәйкес TMenuItem класы анықталған. Программада мәзірді пайдалану оның пункттері, дәлірек айтқанда пункттің қасиеттері арқылы жүзеге асырылады. Пункттің өзі әрі қарай тағыда подпункттерге немесе ішкі пункттерге бөлінуі мүмкін.

Мәзір пункттерінің негізгі қасиеттері:

Bitmap (TBitmap) – қосымшадағы мәзір пунктінің атауының сол жағында тұратын сурет-таңбаларды (мысалы, Открыть, Печать, C++Builder-дегі ▶ Run және т.б.) орналастырады. Қалыпты жағдайда мәні Nil, яғни сурет-таңба жоқ;

Break (TMenuBreak) – мәзір пункттерін бағандарға бөлуді орындайды. Үш түрлі мәнді қабылдауы мүмкін (mbNone – қалыпты жағдайы,

яғни бір жолға жазу; mbBreak – ағымдағы орыннан бастап жаңа баған құру; mbBreakBar – ағымдағы орыннан бастап сызықпен ажыратылған жаңа баған құру);

Caption (AnsiString) – мәзір пунктінің тақырыбын беру үшін қолданылады. Егер тақырып ретінде «1-пункт» деп жазатын болсақ, онда қосымшада мәзір пунктінің орнына «1-пункт» сөзі шығады. Ажыратқыш сызығы көрінеді;

Checked (Bool) – мәзір пунктін таңдау үшін қолданылады. Бастапқы мәні False, яғни ерекшеленбеген;

AutoCheck (Bool) – Checked қасиетінің мәнін автоматтық түрде керіге өзгерту;

Count (Int) – мәзірдің қарастырылып отырылған пунктіндегі ішкі пункттер санын көрсетеді;

Enabled (Bool) – пункттің актив немесе пассив күйін тағайындайды;

Visible (Bool) – пункттің экранда көріну-көрінбеуін қамтамасыз етеді. Қалыпты жағдайдағы мәні True, яғни пункт мәзірде көрініп тұрады.

Мәзір пунктіне байланысты *негізгі оқиға* – тышқанның немесе пернетақтаның көмегімен пунктті таңдау кезінде пайда болатын OnClick оқиғасы.

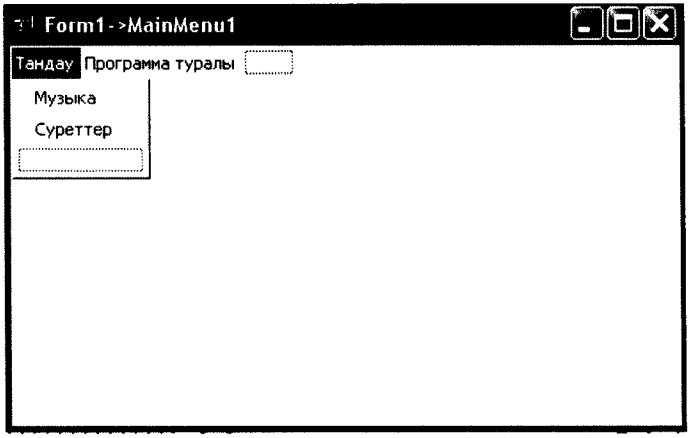
9.4.1. Мәзір конструкторын пайдалану

C++Builder ортасында формаға қойылған мәзірдің пункттерімен жұмыс жасау үшін Мәзір конструкторын (Menu Designer) қолданады. Оны шақыру үшін формада орналасқан MainMenu немесе PopupMenu компоненттеріне екі рет шерту керек, болмаса олардың Items қасиеті шақырылады (9.11-сурет).

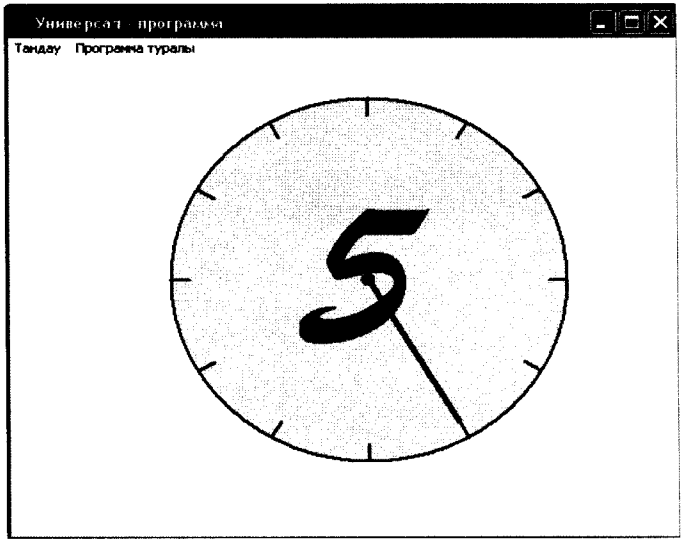
9.5-жаттығу. Музыка тыңдауды және суреттер қарауды орындай алатын әмбебап қосымша құрыңыз. Қосымша төрт терезеден тұрады: «Универсал-программа» терезесі – 9.12, а-сурет; «Музыка» терезесі – 9.12, б-сурет; «Суреттер» терезесі – 9.12, в-сурет; «Анықтама» терезесі – 9.12, г-сурет.

Программа мәзірлермен (MainMenu және PopupMenu компоненттері) жұмыс жасауды көрсететін демонстрациялық программа болып табылады. Программада музыкаларды таңдау үшін суреттер сақталған файлдарды таңдау үшін OpenDialog, OpenPictureDialog компоненттері

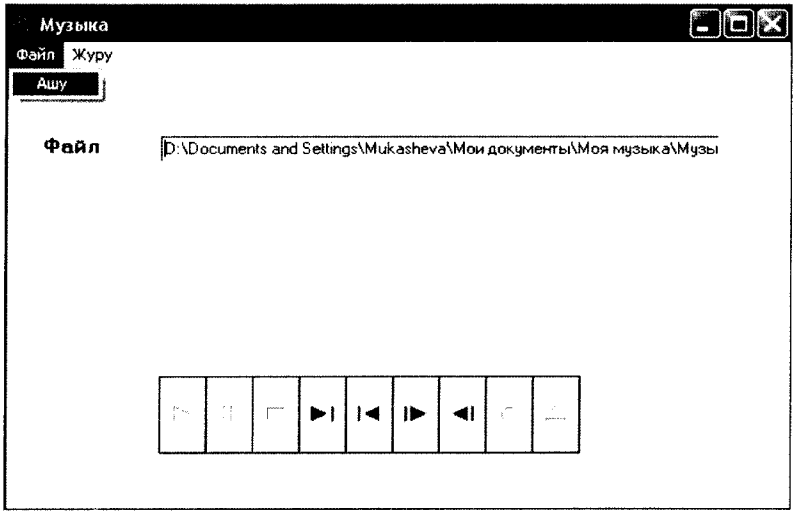
қолданылды. Программаның бастапқы «Универсал-программа» терезесінде сағат тіліне сәйкес ағымдағы уақытты көрсетіп тұратын Animate компоненті қолданылды. Музыкалық аудиофайлдарды тыңдау үшін, System саймандар панеліндегі MediaPlayer, суретті қарау үшін Image компоненттері қолданылды. Бұл компоненттердің қасиеттерін өзгерту төменде қарастырылады.



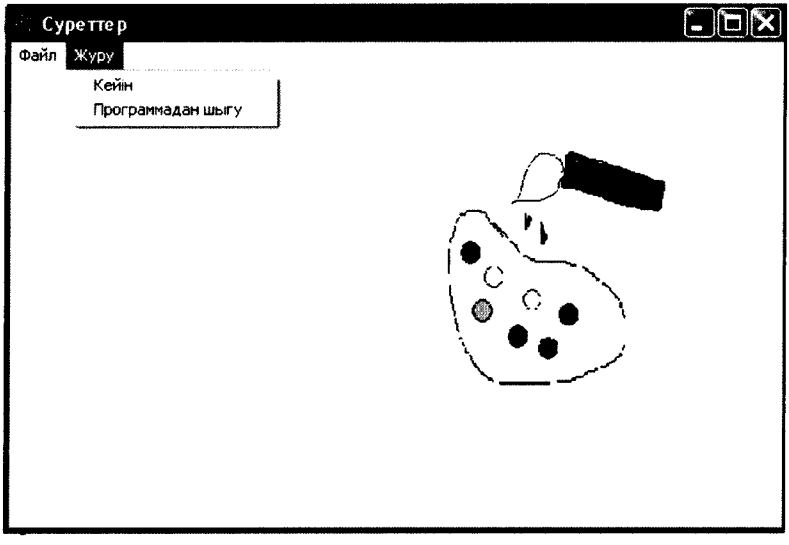
9.11-сурет. *Мәзір дизайнерінің терезесі*



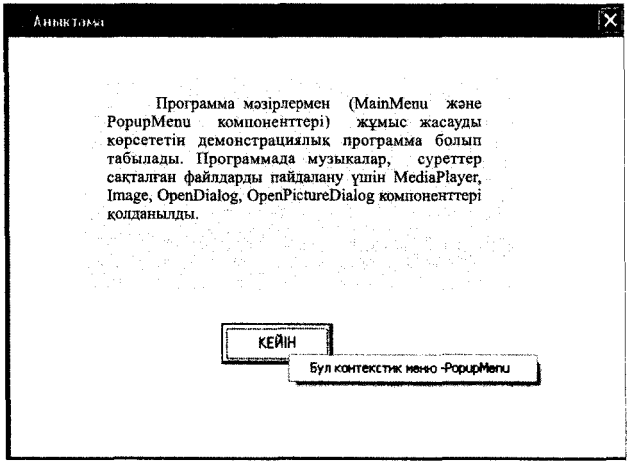
9.12, а-сурет. *«Универсал-программа» терезесі*



9.12, б-сурет. *«Музыка» терезесі*



9.12, в-сурет. *«Суреттер» терезесі*



9.12, г-сурет. *«Анықтама» терезесі*

Жаттығудың орындалуы:

1. «Универсал-программа» (Form1) терезедегі компоненттер қасиеттерінің және оқиғаларының мәндері:

Компоненттің аты	**Қасиеттерінің мәндері, оқиғалары**
Form1	Caption = "Универсал-программа "
MainMenu1	N1: Caption = "Таңдау" N2: Caption = "Музыка" OnClick = N2Click N3: Caption ="Суреттер" OnClick = N3Click N4: Caption = "Программа туралы" N5: Caption = "Анықтама" OnClick = N5Click N6: Caption = "Программадан шығу" OnClick = N6Click
Animate1	Active = True FileName = 'D:\WINDOWS\clock.avi'

2. «Универсал-программа» терезесіне сәйкес программалық код:

```
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::N2Click(TObject *Sender)
{
Form2->Show();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::N3Click(TObject *Sender)
{
Form3->Show();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::N6Click(TObject *Sender)
{
Form1->Close();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::N5Click(TObject *Sender)
{
```

```
Form4->Show();
}
//---------------------------------------------------------------------------
```

3. «Музыка» (Form2) терезедегі компоненттер қасиеттерінің және оқиғаларының мәндері:

Компоненттің аты	**Қасиеттерінің мәндері, оқиғалары**
Form2	Caption = "Музыка"
MainMenu1	N1: Caption = "Файл" N2: Caption = "Ашу" OnClick = N2Click N3: Caption = "Жүру" N4: Caption = "Алға" OnClick = N4Click N5: Caption = "Кейін" OnClick = N5Click
Label1	Caption = "Файл" Font.Height = -13 Font.Name = "MS Sans Serif" Font.Style = [fsBold]
MediaPlayer1	System жапсырмасында орналасқан
OpenDialog1	Dialogs жапсырмасында орналасқан
Edit1	Text қасиеті тазаланады

4. «Музыка» терезесіне сәйкес программалық код:

```
#include <vcl.h>
#pragma hdrstop

#include "Unit2.h"
#include "Unit1.h"
#include "Unit3.h"
```

```
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;
//---------------------------------------------------------------------------
__fastcall TForm2::TForm2(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm2::N2Click(TObject *Sender) // Файл -> Ашу пунктінің коды
{
 if (OpenDialog1->Execute())
   {  Edit1->Text=OpenDialog1->FileName;
 MediaPlayer1->FileName=OpenDialog1->FileName;
 MediaPlayer1->Open();
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm2::N4Click(TObject *Sender)
{
Form3->Show();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::N5Click(TObject *Sender)
{
Form1->Show();
}
//---------------------
```

5. «Суреттер» (Form3) терезедегі компоненттер қасиеттерінің және оқиғаларының мәндері:

Компоненттің аты	**Қасиеттерінің мәндері, оқиғалары**
Form3	Caption = "суреттер"
MainMenu1	N1: Caption = "Файл" N2: Caption = "Ашу" OnClick = N2Click N3: Caption = "Жүру" N4: Caption = "Кейін" OnClick = N4Click N5: Caption = "Программадан шығу" OnClick = N5Click
OpenPictureDialog1	Dialogs жапсырмасында орналасқан
Image1	Stretch = True

6. «Суреттер» терезесіне сәйкес программалық код:

```
#include <vcl.h>
#pragma hdrstop
 #include "Unit1.h"
 #include "Unit2.h"
#include "Unit3.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm3 *Form3;
//---------------------------------------------------------------------------
```

```
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm3::N4Click(TObject *Sender)
{
Form2->Show();
}
//---------------------------------------------------------------------------
void __fastcall TForm3::N5Click(TObject *Sender)
{
Form1->Close();
}
//---------------------------------------------------------------------------
void __fastcall TForm3::N2Click(TObject *Sender)
{
if (OpenPictureDialog1->Execute())
Image1->Picture->LoadFromFile(OpenPictureDialog1->FileName);
}
```

7. «Анықтама» (Form4) терезедегі компоненттер қасиеттерінің және оқиғаларының мәндері:

Компоненттің аты	**Қасиеттерінің мәндері, оқиғалары**
Form4	Caption="Анықтама" BorderIcons=[biSystemMenu] *biSystemMenu=true* *biMinimize=false* *biMaximize=false* biHelp=false
Label1	Caption = "Программа мәзірлермен (MainMenu және PopupMenu компоненттері) жұмыс жасауды көрсететін демонстрациялық программа болып табылады. Программада музыкалар, суреттер сақталған файлдарды пайдалану үшін MediaPlayer, Image, OpenDialog, OpenPictureDialog компоненттері қолданылды" WordWrap = True

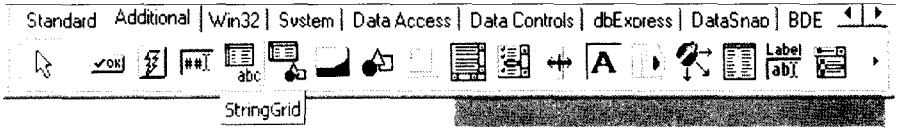
PopupMenu1	Caption = "Бұл контекстік мәзір – PopupMenu"
Button1	Caption = "КЕЙІН" PopupMenu = PopupMenu1 OnClick = Button1Click

8.«Анықтама» терезесіне сәйкес программалық код:

```
#include <vcl.h>
#pragma hdrstop
 #include "Unit1.h"
#include "Unit4.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm4 *Form4;
//---------------------------------------------------------------------------
__fastcall TForm4::TForm4(TComponent* Owner)
      : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm4::Button1Click(TObject *Sender)
{
Form1->Show();
}
```

9.5. Массив компоненттері

C++ Builder ортасында қосымша құруда деректерді кестелер түрінде беру үшін Additional жапсырмасындағы StringGrid және DrawGrid компоненттері (9.13-сурет) қолданылады.



9.13-сурет. *StringGrid компонентін таңдау*

Кесте ұяшықтарына графикалық және мәтіндік деректер орналасатын болса, онда DrawGrid компоненті қолданылады. DrawGrid компоненті арқылы берілген кесте ұяшықтарында текстік және графикалық деректерді тек көруге ғана мүмкіндік беріледі, ал ондағы дерек сақталынбайды. Кесте ұяшықтарына мәтіндік деректерді енгізіп және оны сақтауға, өңдеуге мүмкіндік беретін – бұл StringGrid компоненті.

Additional панелінің StringGrid және DrawGrid компоненттерін сәйкесінше формаға орналастырғанда, жай ғана бос ұяшықтары бар кестелер немесе торкөздер пайда болады, ал программаның орындалуы барысында ұшықтардың деректермен толтырылуын программист өзі ескеріп, жүзеге асыруы тиіс.

Кесте өлшемін (баған және жол санын) Long int типті ColCount және RowCount қасиеттері анықтайды. Қалыпты күйде олардың мәндері 5-ке тең. Баған мен жолдың номерлері нольден басталады, яғни кесте өлшемі – 6x6. Мысалы, келесі фрагмент кесте ұяшықтарын 0-ден басталып, 1-ге артып отыратын мәндермен толтырып беретін болады:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 int I, J, K;
 K = 0;
 for (I = 0; I < StringGrid1->ColCount; I++)
  for (J = 0; J < StringGrid1->RowCount; J++)
   StringGrid1->Cells[I][J] = IntToStr(++K);
}
```

Кестенің сол жақ шеткі бағандары мен алғашқы жолдарын өзгермейтін күйге келтіруге болады. Өзгермейтін баған мен жол сандарын **int** типті сәйкес FixedCols және FixedRows қасиеттер анықтайды. Қалыпты күйде олардың мәндері 1-ге тең, басқа түспен боялған, егер олардың мәндерін 0-ге өзгертсе, онда кестеде өзгермейтін баған мен жол болмайды.

Кестедегі бағандардың енін өзгерту үшін типі **int** болатын DefaultColWidth қасиеті, ал кесте жолының биіктігін өзгерту үшін DefaultRowHeight қасиеттері қолданылады. Мысалы, программада тере-

зенің өлшемдері өзгергенде кесте ұяшықтарының (баған мен жолдың) өлшемдерін өзгерту келесі түрде жазылады:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 if (DrawGrid1->DefaultColWidth > 90)
   DrawGrid1->GridLineWidth = 2;
 else
   DrawGrid1->GridLineWidth = 1;
}
```

GridLineWidth қасиетін кесте ұяшығын көмкеріп тұрған сызықтарының қалыңдығын немесе жіңішкелігін беру үшін қолданады. Әдетте оның мәні 1-ге тең болып тұрады, егер оны 3-ке өзгертсеңіз, онда кесте ұяшықтары қалың бояулы сызықпен көмкерілетін болады.

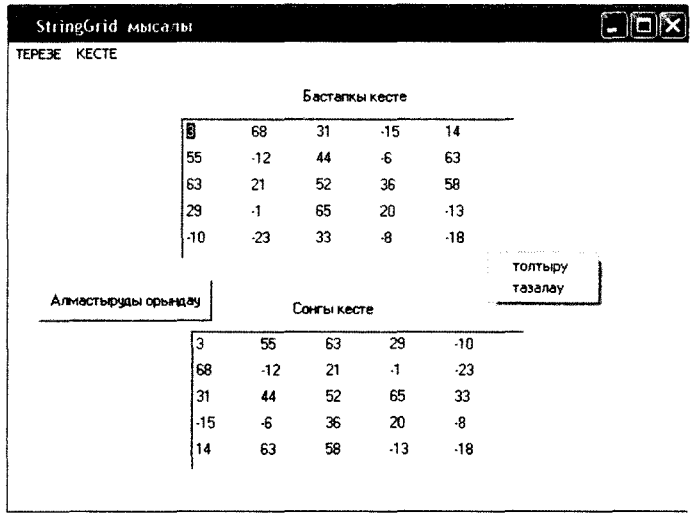
StringGrid кестенің жекелеген ұяшықтарын пайдалану үшін оған сәйкес типі Ansistring болатын Cells(ACol, ARow) екі өлшемді массиві (қасиеті) қолданылады. Бұл массив программаның орындалуы кезінде ғана мүмкін болады және ұяшықтан деректі оқуды немесе ұяшыққа деректі жазуды қамтамасыз етеді. Мұндағы ACol – баған номерін, ал ARow жол номерін көрсетеді, номерлеу нөлден басталады.

Кестелердің параметрлерін өзгерту оның Options (типі TGridOptions) қасиеті арқылы жүргізіледі. Мысалы, кесте ұяшығына пернетақтадан немесе басқаша тәсілмен дерек енгізу үшін оның Options->goEditing қасиетінің мәні **true**-ге өзгертілуі тиіс.

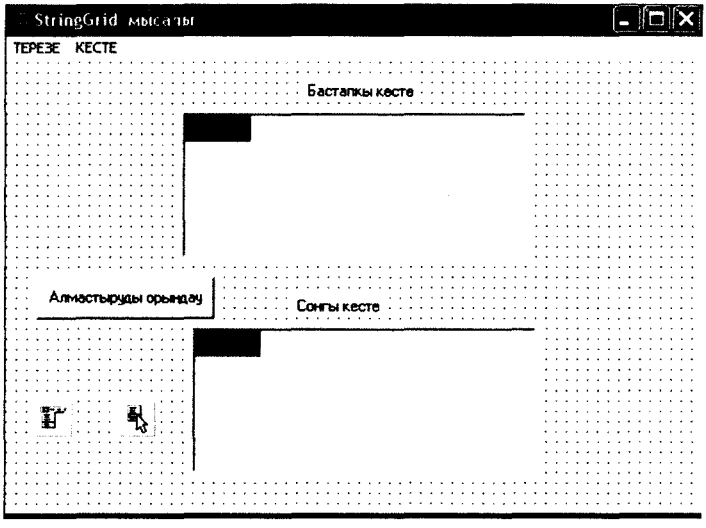
Options->goTabs қасиетінің мәні **true** болғанда, ұяшықтан ұяшыққа ауысуды Tab (Shift+Tab) пернесінің көмегімен орындауға мүмкіндік болады.

DrawGrid, StringGrid компоненттерінің қасиеттерін Object Inspector терезесінен өзгерту ыңғайлырақ болып келеді.

9.6-жаттығу. A(5,5) матрицаның бағандары мен жолдарын алмастырғанда шығатын матрицаны көрсетіңіз. Қосымша терезесі 9.14, а және 9.14, б-суреттерде көрсетілген.



9.14, а-сурет. *Қосымша терезесі*



9.14, б-сурет. *Қосымшаға сәйкес форма терезесі*

Жаттығуды орындау:

Қосымша терезесіндегі компоненттер қасиеттерінің және оқиғаларының мәндері:

Компонент	Компоненттің қасиеттері	Мәндері
Form1	Caption	"StringGrid мысалы"
Label1	Caption	"Бастапқы кесте"
Label2	Caption	"Соңғы кесте"
StringGrid1	ColCount	5 (бағандарының саны)
StringGrid2	RowCount	5 (жолдың саны)
	FixedCols	0 (бекітілген немесе қозғалмайтын баған саны)
	FixedRows	0 (бекітілген немесе қозғалмайтын жолдар саны)
	DefaultRowHeight	20 (жолдың биіктігі)
	DefaultColWidth	50 (бағанның ені)
	Height	113 (кесте сыртындағы тік төртбұрышты облыстың биіктігі)
	Width	265 (кесте сыртындағы тік төртбұрышты облыстың ені)
	Options->goEditing	True (ұяшыққа дерек енгізуге болады)
	Options->AlwaysShowEditing	True (енгізу фокусы орналасқан ұяшықты редакциялуға болады)
	Options->goTabs	True (Tab пернесін басып ауысуға болады (Shift+Tab))
	PopupMenu	PopupMenu1
MainMenu1	N1: Caption = "ТЕРЕЗЕ"	
	N3: Caption = "Жабу" OnClick = N3Click	
	N2: Caption = "КЕСТЕ"	
	N4: Caption = "Толтыру" OnClick = N4Click	

	N5:
	Caption = "Тазалау"
	OnClick = N5Click
	N6:
PopupMenu1	Caption ="толтыру"
	OnClick = N6Click
	N7:
	Caption = "тазалау'
	OnClick = N7Click
Button1	Caption = "Алмастыруды орындау"
	OnClick = Button1Click

Қосымша терезесіне сәйкес программалық код:

```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
 #include "stdlib.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{
}
//---------------------------------------------------------------------------
// «ТЕРЕЗЕ» мәзірінің «Жабу» подпунктінің коды
void __fastcall TForm1::N3Click(TObject *Sender)
{
Form1->Close();
}
// «КЕСТЕ» мәзірінің «Толтыру» подпунктінің коды
void __fastcall TForm1::N4Click(TObject *Sender)
{
for(int i=0;i<5;i++)
```

```
for(int j=0;j<5;j++)
StringGrid1->Cells[i][j]=IntToStr(i+j-2);
}
// «КЕСТЕ» мәзірінің «Тазалау» подпунктінің коды
void __fastcall TForm1::N5Click(TObject *Sender)
{
 for(int i=0;i<5;i++)
StringGrid1->Rows[i]->Clear();
}
// Кестенің контекстік мәзірінің «толтыру» подпунктінің коды
void __fastcall TForm1::N6Click(TObject *Sender)
{
 for(int i=0;i<5;i++)
for(int j=0;j<5;j++)
StringGrid1->Cells[i][j]=IntToStr(rand()%100-27);
}
// Кестенің контекстік мәзірінің «тазалау» подпунктінің коды
void __fastcall TForm1::N7Click(TObject *Sender)
{
  for(int i=0;i<5;i++)
StringGrid1->Cols[i]->Clear();
}
// «Алмастыруды орындау» батырмасының коды
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 for(int i=0;i<5;i++)
for(int j=0;j<5;j++)
StringGrid2->Cells[j][i]=StringGrid1->Cells[i][j];
}
```

Бақылау сұрақтары

1. Визуалды компоненттер кітапханасы қалай аталады?

2. Визуалды компоненттер орналасатын негізгі контейнер компонент қалай аталады?

3. Жобаға жаңа форманы қосу үшін қандай командаларды орындайды?

4. Бірнеше жолдан тұратын мәтінді енгізу үшін қандай компонент қолданылады?

5. Edit компоненттерінің бірінен екіншісіне ENTER пернесін басу арқылы көшуді қалай атайды?

6. ENTER пернесін басу арқылы көшуді орындау үшін компоненттердің қандай әдісін пайдаланады?

7. Компоненттер қасиеттерін өзгертуді қандай терезеде жасауға болады?

8. Программада тізімдер не үшін қолданылады?

9. ListBox компоненті қандай тізім жасау үшін қолданылады?

10. ComboBox компоненті қандай тізімдер жасауда қолданылады?

11. Қажет элементті немесе жолды таңдау үшін тізімнен қандай қасиеті пайдаланылады?

12. Тізімнің Sorted қасиеті не үшін қолданылады?

13. Тізімнің Count қасиеті не үшін қолданылады?

14. Тізімнің Items қасиеті не үшін қолданылады?

15. Тізімнің соңына жаңа жолды тіркеу үшін қандай әдіс қолданылады?

16. IndexOf функциясы қандай қызмет атқарады?

17. Clear әдісі не үшін қолданылады?

18. SaveToFile әдісі не үшін қажет?

19. Қосымша мәзірлерінің қандай түрлері бар?

20. Мәзір конструкторының қызметі қандай?

21. Қосымшадағы компоненттердің контекстік мәзірін құру үшін қолданылатын компонент қалай аталады?

22. Қосымшада ауыстырғыштар қандай қызмет атқарады?

23. Ауыстырғыштардың күйін тағайындайтын қасиеті қалай аталады?

24. Form компонентінің Show(); әдісі қандай қызмет атқарады?

25. Form компонентінің Hide (); әдісі қандай қызмет атқарады?

26. IntToStr(int Value); функциясының қызметі қандай?

27. Edit компонентінің CharCase қасиетінің қызметі қандай?

28. Edit компонентінің PasswordChar қасиетінің қызметі қандай?

29. Кестелермен жұмыс жасау үшін қандай визуалдық компоненттер қолданылады?

30. DrawGrid және StringGrid компоненттерінің ерекшеліктері қандай?

Тапсырмалар

9.1-тақырып бойынша

Келесі мазмұндағы есептерді шеше алатын қосымшалар құрыңыз:

1. Пирамиданың көлемін табу.
2. Үш қабырғасы бойынша үшбұрыштың ауданын есептеу.
3. Параллелепипедтің көлемін есептеу.
4. Тізбектің жалпы кедергісін есептеу (параллель қосылған).
5. Трапецияның ауданын есептеу.
6. Тізбектің жалпы кедергісін есептеу (тізбектей қосылған).
7. Екі қабырғасы және арасындағы бұрышы бойынша үшбұрыштың ауданын есептеу.
8. Бүтін сандармен арифметикалық амалдарды орындай алатын калькулятор программа жасау.
9. Тригонометриялық функцияларды және түбір табуды орындай алатын калькулятор жасау.
10. Шеңбердің ұзындығын, дөңгелектің ауданын есептеу.

9.2-тақырып бойынша

Тізімдерді және олардың қасиеттері мен әдістерін, уақиғаларын пайдалана отырып төмендегі:

1. Азық-түлік дүкенінің ассортименті туралы дерек беретін;
2. Поликлиниканың дәрігерлері туралы дерек беретін;
3. Математикалық формулалар анықтамалығының қызметін атқаратын;
4. Кітапханадағы картотека қызметін атқаратын;
5. Атына сәйкес түстерді көрсете алатын;
6. Фильмдер картотекасының қызметін атқаратын;
7. Футбол клубтары туралы анықтамалар беретін;
8. Ағылшынша-қазақша аударма сөздіктің қызметін атқаратын;
9. Информатика терминдерінің қазақша түсіндірме сөздігінің қызметін атқаратын;
10. Қазақстанның жоғары оқу орындары туралы дерек бере алатын мазмұндағы қосымша құрыңыз.

9.3-тақырып бойынша

Таңдауларды ауыстырғыштар көмегімен жүзеге асыратын қосымшалар құрыңыз:

1. Қолданушының таңдауына сәйкес тізбек мүшелерін арифметикалық, геометриялық прогрессия ережелері немесе т.б. заңдылықтар бойынша құратын және алынған тізбектің min элементін табатын және тізбек элементтерін кему реті бойынша орналастыратын программа жазыңыз.

2. Қолданушының таңдауына сәйкес «Өте жақсы», «өте жақсы және жақсы», «жақсы және қанағаттанарлық», «тек қанағаттанарлық» бағалармен оқитын студенттердің тізімін шығарып беретін программа жазыңыз.

3. Қолданушының таңдауына сәйкес фамилиясы «А»-дан басталатын қызметкерлер тізімін, орташа жалақыдан жоғары жалақы алатын қызметкерлер тізімін, орташа жалақыдан төмен жалақы алатын қызметкерлер тізімін шығарып беретін программа құрыңыз.

4. Қолданушының таңдауына сәйкес аяққиім дүкенінің ассортиментінен әйел адамның аяққиімдерін, ер адамдар аяққиімдерін, балалар аяққиімдерін олардың құнымен, шығарған фирмасымен және размерімен көрсететін программаны жазыңыз. Ең қымбат және ең арзан балалар аяққиімін де көрсететін болсын.

5. Қолданушының таңдауына сәйкес автосалондағы машиналарды маркасына, шыққан жылына және бағасына және түсіне қарай сұрыптап шығарып беретін программа жазыңыз.

6. Қолданушының таңдауына сәйкес поликлиникадағы дәрігерлердің аты-жөні бойынша, мамандығы бойынша және еңбек өтілі бойынша сұрыптап шығаратын программа құрыңыз.

7. Кітапханадағы кітаптарды, авторы, шыққан жылы, бағасы және тиражы бойынша сұрыптайтын программа құрыңыз.

8. Нақты сандардан тұратын бір өлшемді масивтегі оң және теріс элементтерді сұрыптап шығаратын және оң элементтердің ең үлкенін табатын программа құрыңыз.

9. Аэропорт анықтамалық бюросының деректеріне сәйкес рейстерді бағыттары бойынша, апта күндері бойынша және номерлері бойынша сұрыптайтын программа құрыңыз.

10. Стоматологқа келушілердің тізімінен қажет фамилияларды бас әрпі бойынша, келген күні бойынша және дәрігері бойынша сұрыптап шығарып беретін программа құрыңыз.

11. Келушінің сұранысына қарай асхана мәзірінен ақпарат беретін программа құрыңыз және оларды ретіне қарай / бірінші, екінші, үшінші, салат немесе ыстық тамақ / сұрыптап шығарыңыз.

12. Оқытушының журналындағы студенттерді фамилиялары бойынша және жеті лабораториялық жұмыстардан жинаған орташа балдарының өсу ретімен сұрыптап беретін программа құрыңыз.

13. Баспасөз киоскісінің ассортиментін журналдар, газеттер, тілі және елі бойынша сұрыптайтын программа құрыңыз.

9.4-тақырып бойынша

1. Бір өлшемді массив элементтерін клавиатурадан енгізу немесе кездейсоқ сандармен толтыру қажет. Алынған элементтердің ең үлкені мен ең кішісін анықтап және олардың орналасу ретін табуға программа құру керек. Қосымшада мәзірді пайдаланыңыз.

2. Формаға қойылған негізгі мәзірдің «Форма түсі» пункті арқылы форманы түрлі түске бояуды, «Мәзір пунктін басқару» арқылы пункттердің қасиеттерін өзгертуді және *Label* компонентінің контекстік мәзірінің пункті арқылы арқылы *Label* компонентіне мәтін жазуға программа құру қажет.

3. Берілген бір өлшемді массивтің элементтерінің қосындысын табу программасын құрыңыз /Нұсқау: массивті енгізуде MainMenu, қосынды табуда PopupMenu компонетін пайдаланыңыз/.

4. Ақша айырбастау пунктіндегі таблоның жұмысын мәзірлер арқылы программалаңыз.

5. Келушінің сұранысына қарай асхана мәзірінен ақпарат беретін программа құрыңыз.

6. Сандық массивті толтыратын және сұрыптайтын және т.б. амалдарды орындайтын программаны мәзірлер көмегімен жазыңыз.

7. Memo компонентіндегі мәтінді сақтайтын және форматтауды /шрифтіні өзгерту, түстерін өзгерту/ мәзірлер арқылы орындайтын программа құрыңыз.

8. Символдық массивті толтыратын және сұрыптайтын және т.б. амалдарды орындайтын программаны мәзірлер көмегімен жазыңыз.

9. Топ студенттері туралы анықтама беруді мәзірлер көмегімен орындайтын программа жазыңыз.

10. Формадағы барлық орналасқан элементтер туралы анықтамаларды олардың контекстік мәзірі арқылы қарау мүмкін болатын программа жазыңыз.

9.5-тақырып бойынша

Қосымша құруда осыған дейін қарастырылған мәтінмен жұмыс жасау компоненттері, тізімдер, ауыстырғыштар, мәзірлер, командалық батырмалар, диалогтік терезелер және олардың қасиеттері мен әдістері пайдаланылуы қажет.

1. Матрицаның әрбір бағанының оң элементтерінің қосындысын және олардың санын есте сақтау. Нәтижесін екі жол түрінде шығару.

2. Матрицаның әрбір жолының оң элементтерінің қосындысын және олардың санын есептеп есте сақтау. Нәтижесін екі баған түрінде баспадан шығару.

3. Матрицаның бас диагоналіндегі және одан төмен орналасқан элементтердің қосындысын және элементтер санын есептеу.

4. Матрицаның бас диагоналінен төмен орналасқан оң элементтерінің қосындысын және олардың элементтер санын есептеу.

5. Матрицаның жұп орындағы элементтерінің орнына нольдерді жазып және матрицаны баспаға шығару.

6. Матрицаның теріс элементтерінің орнына нольдерді, ал оң элементтер орнына бірлерді жазу.

7. Матрицаның әрбір жолындағы максимал және минимал элементтерді тауып, оларды сәйкесінше осы жолдың бірінші және соңғы элементтерінің орнына жазу керек. Матрицаны баспаға шығару керек.

8. Элементтері бүтін сан болатын матрицаның әрбір жолының төртке еселі элементтерінің санын және нәтижелерінің ең үлкенін табу керек.

9. Элементтері бүтін сан болатын матрицаның әрбір жолының беске еселі элементтерінің санын және нәтижелерінің ең үлкенін табу керек.

10. A(4,3) матрицаның элементтері екі-үш таңбалы немесе одан да көп таңбалы сандар болсын. Матрицаның әрбір элементін оның цифрларының қосындысымен алмастыру керек. Шыққан матрицаны көрсетіңіз.

11. A(4,4) матрицаны сағат тілінің бағытымен 90 градусқа бұрғанда шығатын матрицаны көрсетіңіз.

12. A(7,5) матрицаның элементтерін жол бойынша сұрыптап орналастырыңыз.

13. A(5,7) матрицаның элементтерін баған бойынша сұрыптап орналастырыңыз.

14. A(10) массив элементтерін үш түрлі әдіспен сұрыптаңыз: алмастыру әдісі (метод обмена), тікелей таңдау әдісі (метод прямого выбора), кірістіру әдісі бойынша сұрыптау (сортировка с вставкой) .

15. A(4,3) матрицаның элементтері екі-үш таңбалы немесе одан да көп таңбалы сандар болсын. Матрицаның әрбір элементін оның цифрларының көбейтіндісімен алмастыру керек. Шыққан матрицаны көрсетіңіз.

10-тарау. C++ BUILDER 6 ОРТАСЫНДАҒЫ ГРАФИКА

10.1. Графиктік файлдардың форматтары

C++ Builder 6 ортасында графиктік файлдардың келесі түрдегі типтерімен жұмыс жасауға болады (10.1-кесте):

10.1-кесте

C++ Builder 6. *Графиктік файл форматтары*

Файл форматтары (түрлері)	Кеңеймелері
Биттік матрица (Bitmaps)	. BMP
Пиктограммалар	. ICO
Метафайлдар (Metafiles)	. WMF
IPEG Image file	. JPG , . JPEG
EnhancidMetafiles	. EMF

Бұл файлдардың барлығында суреттер немесе кескіндер сақталады, бірақ бұлардың файлдағы сақталу тәсілдері әр түрлі болады. Мысалы, .BMP; .ICO кескіндер файлға биттік матрица түрінде сақталады. Бұл биттік матрицалар кескіндегі әрбір пиксельдің (нүктенің) түсін көрсетеді. Бұл биттік матрицада сақталған кескінді кез келген компьютер өзінің мүмкіндігіне қарай көрсете алады. Келесі .ICO кеңеймесімен берілген файлда пиктограммалар (значоктар) сақталады. Пиктограммалардың өлшемін өзгерту қажет емес. Метафайлдарда кескіннің биттері сақталмайды, керісінше сол кескінді салу әдістері сақталады. Яғни, олар – сурет салу командалары.

10.1.1. Графиктік объектілер кластары

C++ Builder 6 ортасында графиктік кескінді сақтауға арналған келесі кластар қолданылады:

– TBitMap, биттік матрица түрінде сақталатын кескін объектілері үшін анықталған;

– TIcon, пиктограммалар түрінде сақталатын кескін объектілері үшін анықталған;

– TMetafile, кескінді салу әдістері сақталатын объектілер үшін анықталған.

Бұл кластардың барлығы үшін түпкі класс TGraphic класы болып табылады. TPicture класы – өзінде, жоғарыда көрсетілген үш кластың барлық мүмкіндіктерін қамти алады. Сондықтан TPicture класының, графиктік объектілері үшін төмендегідей,

```
TPicture->TIcon;
TPicture->TMetafile;
TPicture->TBitMap;
```

қасиеттерді шақырып пайдалануға болады.

10.1.2. Суретті қарау және суретті салу

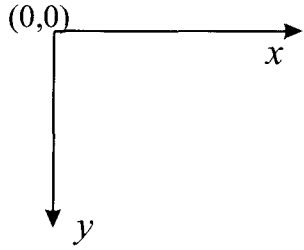
Image компоненті бұл Additational жапсырмасында орналасқан. Бұл компонентті пайдаланып жоғарыда айтылған графиктік файлдардың бірінде сақталған кескінді, программада көруге болады. Негізгі қасиеттері:

Қасиеттің аты	**Қызметі және қабылдайтын мәндері**
Picture(TPicture)	Бұл қасиет файлдағы кескінді Image терезесіне жүктейді. Image терезесінде суретті көруге ғана емес, сақтауға да болады.
AvtoSize (bool)	Егер бұл қасиет **true** мәнге ие болса, онда image компонентінің өлшемдері ол көрсететін кескін өлшеміне қарай келтіріледі.
Stretch (bool)	Егер бұл қасиет **true** мәнге ие болса, керісінше кескін немесе сурет өлшемдері image компонентінің өлшемдеріне келтіріледі. Бұл қасиет пиктограммаларға әсер етпейді, себебі пиктограммалардың өлшемдері өзгермейді.
Center (bool)	Егер бұл қасиет **true** мәнге ие болса, онда бұл кескін image компонентінің ортасында орналасады.
Transparent (bool)	Егер бұл қасиет **true** мәнге ие болса, онда кескін көрінбей тұрады (түссіз болып кетеді), бұл әсіресе бір кескіннің үстіне екіншісін беттестіргенде жиі қолданылады. Бұл қасиет тек биттік матрицаларға ғана жүреді.

Image компонентінің терезесіне кескінді жүктеу үшін немесе сақтау үшін сәйкесінше OpenPictureDialog және SavePictureDialog стандарт диалогтік терезелері қолданылады.

C++ Builder ортасында графикамен жұмыс істеу үшін кескінді орналастыратын орын немесе бет керек. C++ Builder -де ол үшін Canvas *(канва, холст – кенеп)* қасиеті анықталған. Мұндай қасиет Image, Form, PaintBox, ListBox, StringGrid, BitMap, т.б. бар.

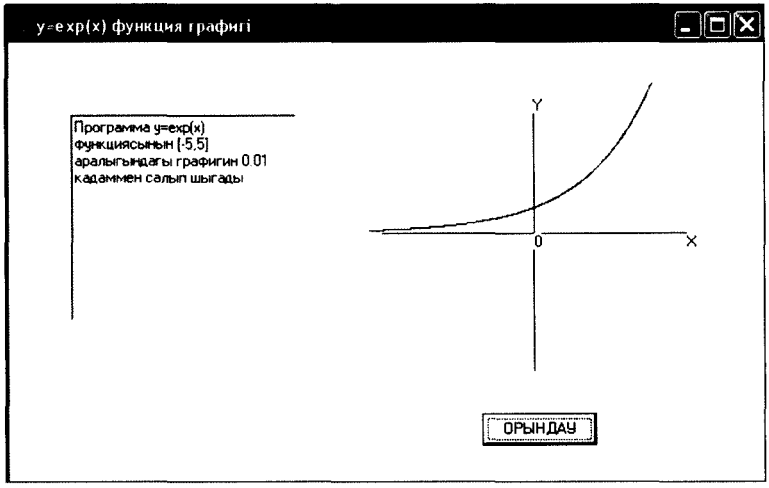
Канваның өзінің қасиеттері және әдістері бар. Канва нүктелерден тұрады, бұл нүктелердің (x,y) координаталары болады (10.1-сурет) Канваның (0,0) нүктесі оның сол жақ төбесінде орналасады.



10.1-сурет. *Канва нүктелерінің координаталары*

Бұл (x,y) координаталары пиксельмен есептеледі. Пиксельдің ең негізгі қасиеті – оның түсі. Канвада сурет салу үшін оның Pixels қасиеті қолданылады. Бұл екі өлшемді массив түрінде анықталады және ол пиксельдің түсін береді, яғни типі Pixels[X] [Y] : clColor;

10.1-жаттығу. f(x)=exp(0.5*x) функциясының графигін [-5,5] аралығында h=0.01 қадаммен салу керек. Қосымша терезесі 10.2-суретте көрсетілген.



10.2-сурет. *f(x)=exp(0.5*x) функцияның графигі*

Жаттығудың орындалуы:

1. Қосымша терезесіндегі компоненттер қасиеттерінің және оқиғаларының мәндері:

Компонент	**Компоненттің қасиеттері**	**Мәндері**
Form1	Caption	"y(x)=exp(x) функция графигі"
Memo1	Text	"Программа y=exp(x) функциясының [-5,5] аралығындағы графигін 0.01 қадаммен салып шығады"
Image1		
Button1	Caption	"ОРЫНДАУ"
	OnClick	Button1Click

Қосымшаның программалық коды:

```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include "math.h"
```

```
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
   float f(float x)  //функцияның мәндерін есептеу
   {
    float f=exp(0.5*x);
    return f;
   }
   void GrafSizy()    // график сызатын функция басталды
   {
     float x1,x2,y1,y2,x0,y0;
     float x,mx,my;
  //график салынатын облысты даярлау
  int Ibik=Form1->Image1->ClientHeight;          //image1-дің биіктігі
  int Iyz=Form1->Image1->Width;                  //image1-дің ұзындығы
//Y осьті сызу, ол (xY,yY) нүктеден басталып, (xY, cY) нүктеде біттеді;
   float xY=Iyz/2;
   float yY=10;
   float cY=Ibik-10;
   Form1->Image1->Canvas->MoveTo(xY,yY);
   Form1->Image1->Canvas->LineTo(xY,cY);
// X осьті сызу  ол (xX,yX)  нүктеден басталып, (cX, yX ) нүктеде біттеді;
   float xX=10;
   float yX=Ibik/2;
   float cX=Iyz-10;
   Form1->Image1->Canvas->MoveTo(xX,yX);
   Form1->Image1->Canvas->LineTo(cX,yX);

//бастапқы мәндерді анықтау
     x1=-5;  x2=5; y1=f(x1);   y2=f(x2);
//масштабты тағайындау
     mx=Iyz/(x2-x1); my=Ibik/(y2-y1);
//координата бас нүктеге (xY,yX) бару
   x0=xY;   y0=yX;
```

```
    Form1->Image1->Canvas->MoveTo(x0,y0);
  //график сызу басталды
   x=x1;
   while (x<x2)
        { //нүктені салу
  Form1->Image1->Canvas->Pixels[x0+mx*x][y0-my*f(x)]=clBlack;
  x=x+0.01;      //келесі мәнді алу
      }
   //Канваға мәтіндерді жазу
   Form1->Image1->Canvas->TextOutA(x0,y0,'0');
   Form1->Image1->Canvas->TextOutA(xY,yY,"Y");
   Form1->Image1->Canvas->TextOutA(cX,yX,"X");
    }   // график сызатын функция аяқталды
  //---------------------------------------------------------------------------
  __fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
  {
  }
  //---------------------------------------------------------------------------
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
      GrafSizy(); //графикті салатын  GrafSizy() функциясын шақыру
  }
  //---------------------------------------------------------------------------
  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
  Memo1->Text="Программа  y=exp(x)  функциясының  [-5,5]  аралығындағы  графигін 0.01 қадаммен салып шығады";
  }
```

Суретті салуға немесе қарауға Image компонентінен басқа Form немесе PaintBox компоненттерін де қолдануға болады. Оларда да Canvas қасиеті бар. Формада сурет салғанда оның OnPaint оқиғасы қолданылады.

10.2. Канвада сурет салу құралдары

Pen (қалам, қарындаш, қылқалам) канваның қасиеті түрінде анықталған, яғни канваның Pen(TPen) қасиеті бар. TPen класс ретінде анықталған оның өзінің қасиеттері мен әдістері бар. Негізгі қасиеттеріне мыналар жатады:

– Color – қалам түсі;

– Width – сызықтың қалыңдығы (қалыпты жағдайда 1 пиксельге тең);

– Style – қалам стилін білдіреді, оның мәндері келесі түрде болуы мүмкін:

PS Solid (тұтас сызық) _____

PS Dash (пунктир) – – – – – – –

PS Dot (нүкте) . . . . . .

PS DashDot (нүкте-пунктир) ._._._._

Канваның келесі қасиеттерін пайдаланып суреттер салуға болады:

PenPos(TPoint) – бұл қасиет канваның қаламының бастапқы позициясын анықтайды, яғни қаламның қай жерде тұрғанының координатасын көрсетеді, мысалы:

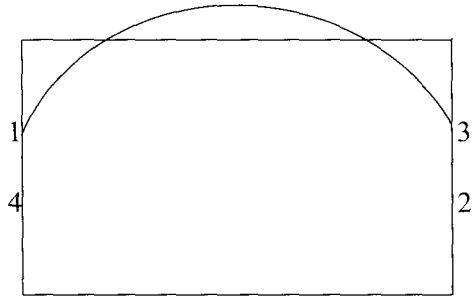
K=Image1->Canvas-> PenPos ;

MoveTo(x,y) – қасиеті қаламды координаталары (x,y) нүктесіне апарып қояды.

LineTo(x1,y1) – бұл берілген нүктеден, координаталары (x1,y1) болатын нүктеге дейінгі кесіндіні сызып береді.

Arc(x1,y1,x2,y2,x3,y3,x4,y4); – функциясы канвада төбелерінің координаталары (x1,y1), (x2,y2), (x3,y3), (x4,y4) болатын төртбұрышпен шектелген облыста эллипстің доғасын сызады. Мұндағы x1, y1, x2, y2, x3, y3, x4, y4 шамалардың типтері **int** болады. Мысалы, келесі программалық код орындалуының нәтижесінде доға сызылуы керек (10.3-сурет):

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{
  TRect R = GetClientRect();   // Ағымдағы терезеден тік төртбұрыштың
                               // координаталарын алу.
  Canvas->Arc(R.Left, R.Top, R.Right, R.Bottom, R.Right, R.Top,
                                                R.Left,  R.Top); }
```



10.3-сурет. *Доға сызу*

Chord(x1, y1, x2, y2, x3, y3, x4, y4); – бұл төбелері берілген төртбұрышта эллипстің жартылай қиылған бөлігін салып шығады.

Ellipse(x1, y1, x2, y2); – бұл диагоналінің координаталары (x1,y1) және (x2,y2) тіктөртбұрыш ішінде эллипс сызады. Егер төртбұрыш квадрат болса, дөңгелек болады.

Pie(x1, y1, x2, y2, x3, y3, x4, y4); – бұл диагоналі (x1,y1) және (x2,y2) арқылы берілген төртбұрыш ішінде эллипс секторын сызады. Сектордың қабырғалары эллипстің центрінен және x3,y3 нүктелері арқылы өтеді.

PolyBeizer(const TPoint* Points, const int Points_Size); – бұл функция түрліше сызықтарды салу үшін қолданылады. (Мұнда қисық немесе сынық сызық біріншіден төртінші нүктеге қарай өседі, қалғандары да сондай).

Rectangle(x1, y1, x2, y2); – бұл сол жақ жоғарғы төбесі (x1, y1), ал оң жақ төменгі төбесі (x2, y2) нүктелері болатын төртбұыш сызады. Бұл Brush пен Pen-нің ағымдағы қасиетін қолданады.

Draw(x,y; TGraphic*); – бұл графиктік объектіні канваға шығарады. График объектісінің орнында .BMP, .ICO немесе метафайл болуы мүмкін.

TextOut(x,y; const AnsiString Text); – бұл (x,y) нүктесінен бастап текст тұрақтысында сақталған мәтінді канваға шығарады.

FillRect(const TRect Rect); – канвада Rect төртбұрышын бояйды. Бояу үшін қылқаламның түсі және стилі қолданылды.

FloodFill(x,y; TColor Color; TFillStyle FillStyle); – (x,y) нүктесінен бастап немесе x,y нүктесі жататын тұйық облысты Color түске FillStyle стильмен немесе өрнекпен /узор/ бояйды.

Brush – бұл канваның негізгі қасиеттерінің бірі болып табылады. Канвадағы тұйық фигураларды бояйды. Канваның өзінің фонын көрсетеді. Brush объектісінің негізгі қасиеттері:

Color – түсін береді;

Style – бояу стилін тағайындайды, оның мәндері, BSSolid – тұтас бояйды, BSHorizontal – горизонталь сызықтармен бояйды.

10.3. Анимация

Анимация – бұл кескінді немесе суретті кадрлар арқылы қозғалысқа келтіру. C++ Builder-де Windows-тың дайын анимацияларын қолдануға болады немесе салынған суреттің бөліктерін координаталары арқылы қозғалтып жылжытуға болады.

Программалауда кескінді қозғалысқа келтірудің бірнеше тәсілдерін қолданады:

– Timer компоненттерін пайдалану. Мұнда уақытқа байланысты кескінді жылжытуға болады. Мұнда кескін алдын ала процедура түрінде анықталып алынады. Уақытқа байланысты кескіннің қозғалатын бөлігінің координаталары өзгертіледі;

– кескіннің түсін Canvas-тың түсімен сәйкестендіру. Мұнда сурет салушы қаламның түсін өзгертіп отырады;

– Animate компоненті бар. Win32-де орналасқан. Бұл компонент Windows-тағы стандарт видеоклиптерді формада көрсету үшін қолданылады. Бұл видеоклиптердің кеңеймесі .AVI. Стандарт видеоклиптерінің Shell 32.dll кітапханасы болады.

Animate компонентінің негізгі қасиеттері мен оқиғаларына мыналар жатады:

File Name – видеода сақталған AVI файлдың атын береді.

Common AVI – Windows-тағы стандарт мультипликациялық немесе анимациялық файлдарды белгілі бір мәндер түрінде беру үшін қолданылады.

Repetitons бұл нольден өзгеше болу керек. Ол клиптің қайталануының санын береді.

Active қасиеті – true мәнге ие болса, клип орындалады, false болған жағдайда орындалмайды.

Play әдісі бұл әдіс клипті орындауға жібереді.

Негізгі оқиғаларына OnClose, OnOpen, OnStart, OnClick және т.б. оқиғалар жатады.

Animate компонентін пайдаланып, анимация жасау мысалы, 9.4-тақырыптағы 9.5-жаттығуда қарастырылып кетті.

Бақылау сұрақтары

1. C++ Builder 6 ортасында графиктік файлдардың қандай форматтарымен жұмыс жасауға болады?
2. bmp; . ico кеңеймелі файлдарда кескіндер қалай сақталады?
3. TbitMap, TIcon, TMetafile кластары үшін түпкі класс қандай болады?
4. Қандай компоненттерге сурет салуға болады?
5. Компоненттің Canvas қасиетінің қызметі не?
6. Канваның Pen(TPen) қасиеті суретті қалай салады?
7. Pen(TPen) қаламда сызықтың стилін тағайындайтын қасиет қалай аталады?
8. Анимация жасау үшін қандай тәсілдер қолданылады?

Тапсырмалар

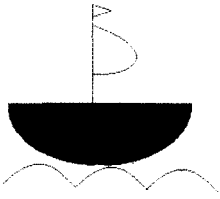
1) Функцияның графиктерін салу программаларын құрыңыз:

1. $y=a*x^2+b*x$
2. $y=x^3+a*x^2+b*x$
3. $y=a*\sin(x)+b$
4. $y=-a*x^2+b*x$
5. $y=a*x^5$

6. y=a*cos(x-1)+|x|
7. y=a/x
8. y=(x+a)/(x-b)
9. y=a+b*x+c*x^2
10. y=b/x-c/x^2
11. y=a*cos ($x^{2)}$
12. y=x^3
13. y=a*sin(x)
14. y=-a*x^2
15. y=1/x^2+2
16. y=a*cos(x-1)
17. y=a*|x|
18. y=1/(x+2)
19. y=(x+1)/(x-1)
20. y=1/(a+b*x)
21. y=(a-b)/x
22. y=1/cos (x+2)
23. y=1/(x+1)
24. y=1/sin(x)
25. y= $x^{1/2}$

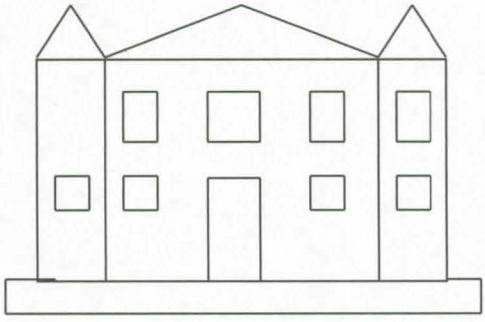
2) Суреттерді салу және оларды қозғалту программаларын құрыңыз:

«кеме»

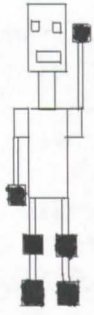


«Аяз ата»

«қорған»



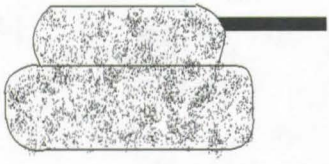
«робот»



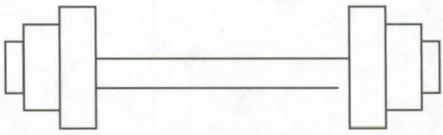
тышқан

«нысана»

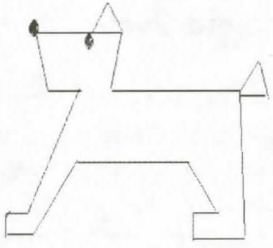
«танк»



«штанга»

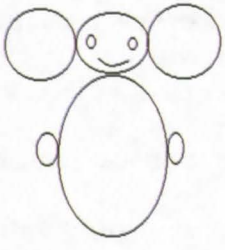


«саққұлақ»



«қоян»

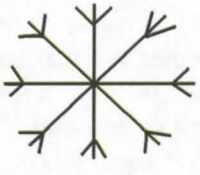
«чебурашка»



«жұлдызша»

«Күн планеталарының жүйесі»

«қар»



«кемпірқосақ»

11-тарау. ҚОСЫМШАНЫҢ АНЫҚТАМАЛЫҚ ЖҮЙЕСІ

11.1. Анықтамалық жүйені құру

Ереже бойынша толық жасақталған программалық жабдықтың өзінің анықтама жүйесі болады. Анықтама жүйесінің қызметі – бұл қолданушыға сол программа туралы және ондағы деректер, компоненттер, функциялар және т.б. туралы деректер беру. Әдетте программада бұл анықтама жүйесі F1 пернесін басу немесе программаның негізгі мәзіріндегі Help /немесе «сұрақ белгісі»- ? түрінде, немесе «справка» немесе т.б./ командасы арқылы шақырылады.

Қарапайым қосымшалардың анықтамалық жүйесі бірнеше тақырыпты қамтитын бір ғана мәтіндік файлдан тұруы мүмкін, ал күрделі қосымшалар үшін мұндай файлдар бірнеше болуы мүмкін.

C++Builder-де жасалатын қосымшаның анықтамалық жүйесін құру келесі төрт кезеңдерден тұрады:

1. Басқару элементтерінің (форма және ондағы компоненттер) контекст номерлерін тағайындау;
2. Мәтіндік редакторды (мысалы, Word-ты) пайдаланып анықтамалық жүйенің мәтіндік файлын жасау;
3. Арнаулы компилятор көмегімен анықтама файлын жасау;
4. Анықтама файлын қосымша жобасына (проектісіне) апарып қосу.

11.1.1. Контекстік номерлерді тағайындау

Форма және онда орналасқан компоненттер анықтамалық жүйеге сәйкес контекстік номер ала алады, ол үшін компоненттің HelpContext қасиетіне анықтама файлындағы сәйкесінше тақырыптың номері беріледі. Қосымша орындалған кезде компонентті белгілеп алып F1 пернесін басса, онда анықтама терезесінде сол компонентке қатысты анықтаманы көре алатын боласыз, егер оған сәйкес бөлім немесе тақырып тағайындалмаған болса, сәйкесінше хабарлама шығады.

11.1.2. Анықтамалық жүйенің мәтіндік файлы

Анықтамалық жүйенің мәтіндік файлы кез келген мәтіндік редакторда, бірақ *.rtf форматта жасалады. Мұны жасау үшін Microsoft

Word процессорын да пайдалануға болады. Мәтіндік файл жеке тақырыптарды білдіретін бөлімдерден (немесе парақтардан) тұрады. Әрбір бөлім үшін келесі атрибуттар тағайындалады (Microsoft Word-тағы символ-таңбалары):

– бөлімнің тақырыбы (заголовок немесе $) – бұл бөлімнің атауы, ұзындығы 128 символдан аспайтын мәтін, мысалы, Massiv, Programma tyrali;

– бөлімнің контексті (#) – бұл бөлімді идентификациялау үшін берілетін мәтін, ол символдардан, сандардан тұруы мүмкін, мысалы T_1, ID_11;

– бөлімнің негізгі (ключевой) сөздерінің тізімі (K) анықтамадан сөздерді іздеу үшін беріледі, егер тізімде бірнеше сөз берілетін болса, онда олар «;» арқылы ажыратылады;

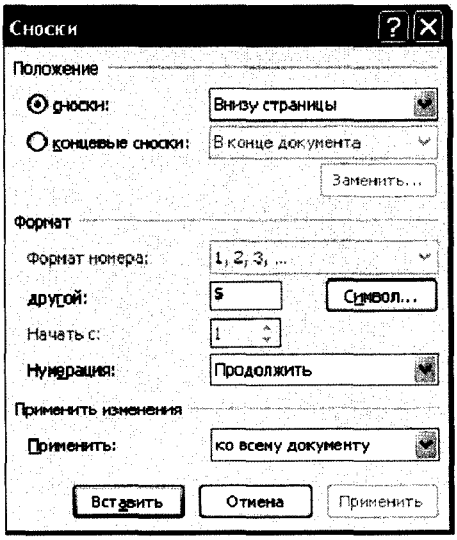
– бөлімнің реттік номері (+), анықтаманы біртіндеп қарау кезіндегі пайда болу ретін білдіреді;

– макрокоманда (!) – бұл бөлімді көрсету кезінде орындалатын командалар;

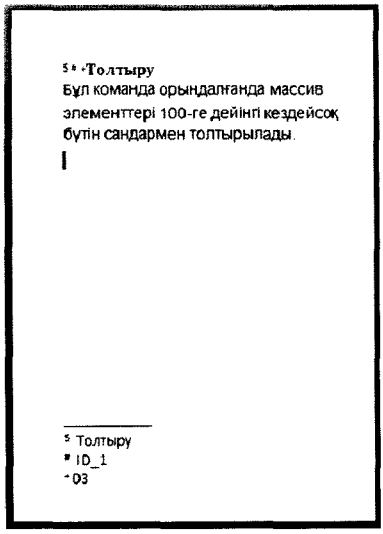
– компиляцияның тегі (*) – бұл бөлімді анықтамалық жүйеге енгізу-енгізбеуді тағайындайды.

Microsoft Word-та анықтама мәтіндік файлындағы бөлімдердің атрибуттары сноскалар түрінде беріледі, бөлімге сносканы қою үшін курсорды бөлімнің тақырыбының алдына (немесе соңына) апарып, Вставка->Сноска (Word 2003) немесе Ссылки->Вставить сноску (Word 2007) командаларын орындайды. Пайда болған (11.1-сурет) терезедегі «Формат» жапсырмасындағы «другой» тұсына қажет таңбаны, мысалы «$» апарып қоясыз, сонда нәтижесінде парақ екіге бөлінеді, парақтың жоғарысында тақырып және оған сәйкес мәтіндер қалады, ал парақтың төменгі жағында мәтінді сноскадан бөліп, ажыратып тұратын көлденең сызық және оның астында сносканың мәтіндері түріндегі атрибуттар жазылады, мысалы, негізгі атрибуттары тағайындалған бөлім келесі түрде болуы мүмкін (11.2-сурет).

Microsoft Word-та жасалған *.rtf форматтағы мәтіндік файлды анықтамалық жүйенің *.hlp файлына айналдыру үшін арнаулы компиляторлар қолданылады. Мұндай компиляторлар көптеп саналады, мысалы, Instant Help, Microsoft Help Workshop және т.б. C++Builder құрамындағы HELP/TOOLS каталогында Microsoft Help Workshop



11.1-сурет. *«Сноски» терезесі*



11.2-сурет. *Негізгі атрибуттары тағайындалған «Толтыру» бөлімі*

компиляторының hcrtf.exe файлы орналасқан, оның көмегімен Microsoft Word-та жасалған *.rtf форматтағы мәтіндік файл негізінде *.hlp анықтамалық жүйенің файлын алады (11.1-жаттығуда қаралады).

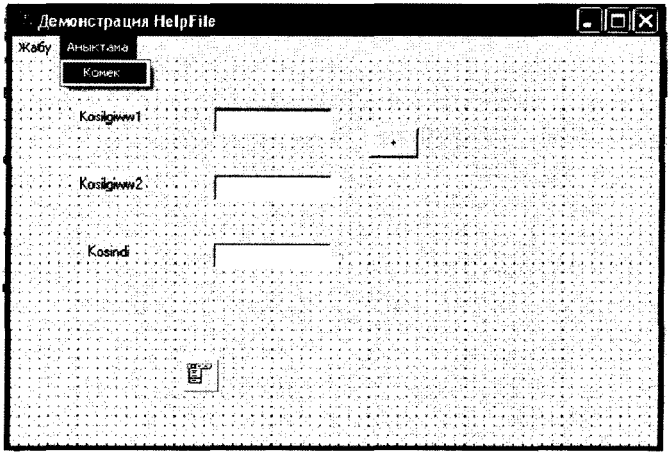
Анықтама файлын қосымша жобасына апарып қосу үшін C++Builder ортасында құрылып жатқан жобаның Project->Options->Application командалары орындалып, ол терезедегі Browse батырмасын пайдаланып, қажет *.hlp файлына баратын маршрут көрсетіледі. Содан соң анықтаманы F1 батырмасы көмегімен шақыру үшін келесі код жазылады:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{ HelpContext=1; }
```

Анықтаманы мәзірден шақыру үшін сәйкесінше мәзір пунктіне келесі код жазылады:

```
void __fastcall TForm1::N8Click(TObject *Sender)
{Application->HelpCommand(HELP_CONTENTS,0);}
```

11.1-жаттығу. Екі санды қосуды орындайтын қарапайым қосымшаның анықтамалық жүйесін жасау (11.3-сурет).



11.3-сурет. *Анықтама файлын пайдаланатын қосымша терезесі*

Жаттығудың орындалуы:

Жаттығуды орындау жоғарыда айтылған төрт кезең бойынша жүргізіледі.

1-кезең

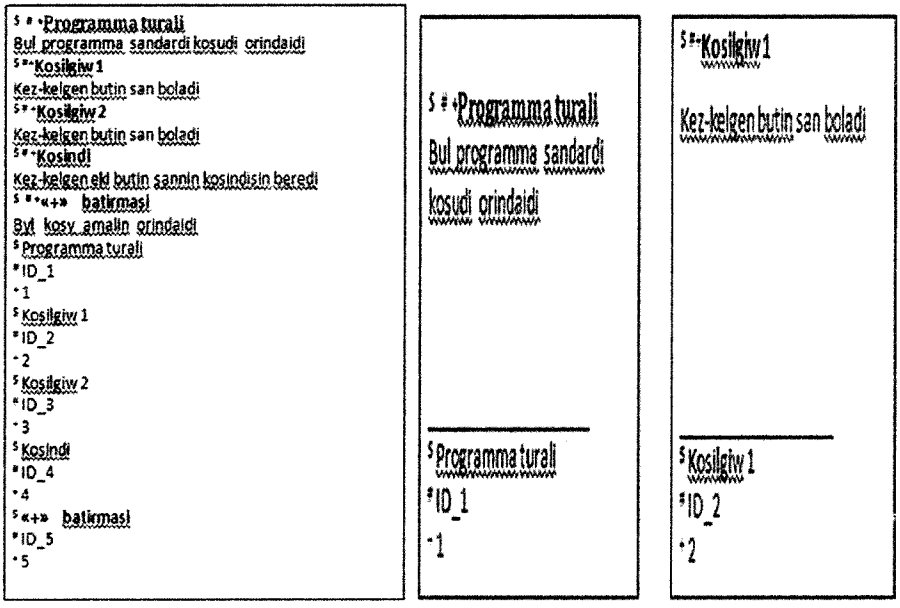
C++Builder ортасында екі санды қосуды орындайтын қарапайым қосымша құрылады, қосымша терезесі келесі түрде болады (11.3-сурет) және қосымша жеке бумаға сақталып қойылады. Суреттегіге сәйкес форма тақырыбы, Label1, Edit1 және Button1 компоненттерінің сәйкес қасиеттері өзгертіледі. Жоғарыда құрылған ProbHelp.rtf файлындағы бөлімдердің номеріне сәйкес Edit1 компоненттің контекст номеріне, яғни HelpContext қасиетіне 2 мәні берледі, себебі оған сәйкес анықтама бөлімі Kosilgiw1-дің реттік номері 2-ге тең, тура сол сияқты Edit2 мен Edit3-тің HelpContext қасиеті сәйкесінше 3 пен 4 болады да, Button1 компонентінің HelpContext қасиеті өзіне сәйкес бөлімнің реттік номері 5-номерді алады.

2-кезең

Бұл кезең Microsoft Word редакторын пайдаланып, бастапқы *.rtf форматтағы анықтаманың мәтіндік файлын құруды қамтиды.

11.4-сурет. *ProbHelp.rtf файлының бастапқы терезесі*

Программадағы элементтерге сәйкес оның жалпы көрінісі келесі түрде болады (11.4, 11.5-суреттер) және бұл файлды, қосымша үшін ашылған буманың ішінен MyHelp деген жаңа бума ашылады да, ол сонда ProbHelp.rtf деген атпен сақталады. Бұл құжаттың терезесінде 11.4-суреттегідей мәтіндерді теру керек (анықтама терезесінде шығатын мәтін өзгеріп кетпес үшін ағылшын әріптері пайдаланылды) және мұнда ең алғашқы бөлім «Programma turali» Word-тағы Заголовок1 стилімен теріледі, ал қалған бөлімдердің тақырыптарын жай «қалың бояулы» шрифімен жасаса жеткілікті. Мәтін теріліп болған соң әрбір бөлімге сәйкес «сноскалар» қойылады (11.5-сурет, сол жақтағы терезе). Келесі қадамда курсорды бөлімдерде орналасқан жай мәтіндер мен келесі бөлім тақырыбының арасына қойып, Вставка->Разрыв страницы командасы орындалады, сонда 5 бөлімге сәйкес 5 терезе шығады (11.5-сурет, ортада және оң жақтағы терезелер). Файл *.rtf форматта сақталуы тиіс.



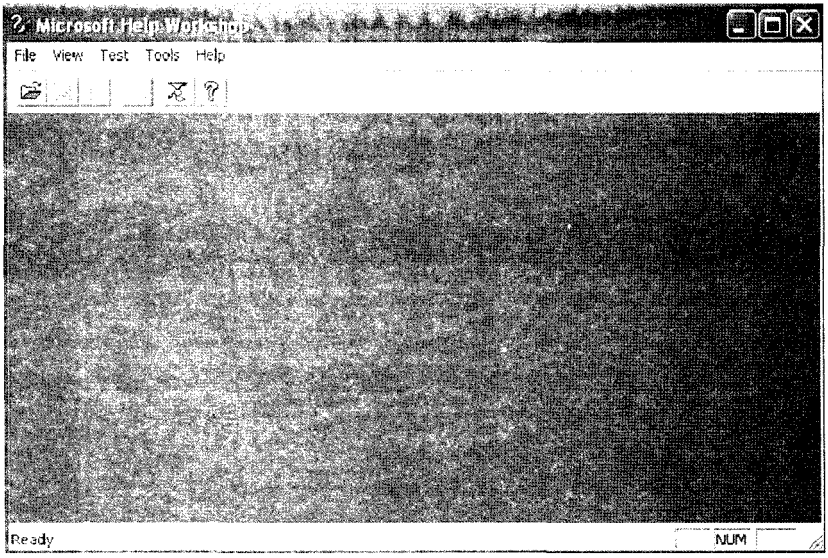
11.5-сурет. *ProbHelp.rtf файлының «сноска» қойылғаннан кейінгі /сол жақта/ және «разрыв страницы» қойылғаннан кейінгі /ортада, оң жақта/ терезелері*

3-кезең

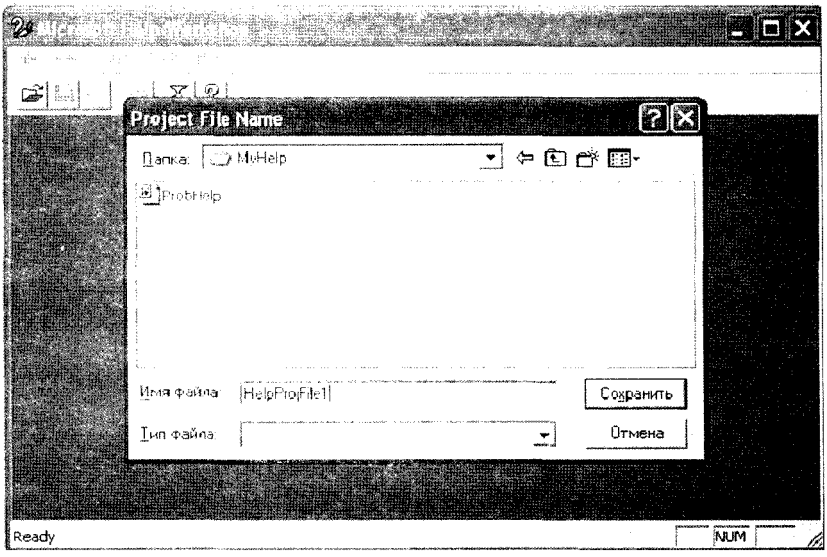
Microsoft Word-та жасалған MyHelp/ProbHelp.rtf форматтағы мәтіндік файлды анықтамалық жүйенің *.hlp файлына айналдыру үшін арнаулы Microsoft Help Workshop компиляторын ашу керек, ол үшін Programm Files/Borland/CBuilder6/ HELP/TOOLS каталогында Microsoft Help Workshop компиляторының hcrtf.exe файлы іске қосылады, сонда келесі терезе пайда болады (11.6-сурет).

Келесі қадамда File->New командасы орындалады, пайда болған терезеден Help Project таңдалынады.

Пайда болған Project File Name терезесінде, жоғарыдағы даярланған ProbHelp.rtf форматтағы мәтіндік файл сақталған MyHelp бумасын ашып алу керек болады, сонан соң барып Имя файла тұсындағы терезеге жоба файлының атын, мысалы, HelpProjFile1 теріп жазылады (11.7-сурет.)

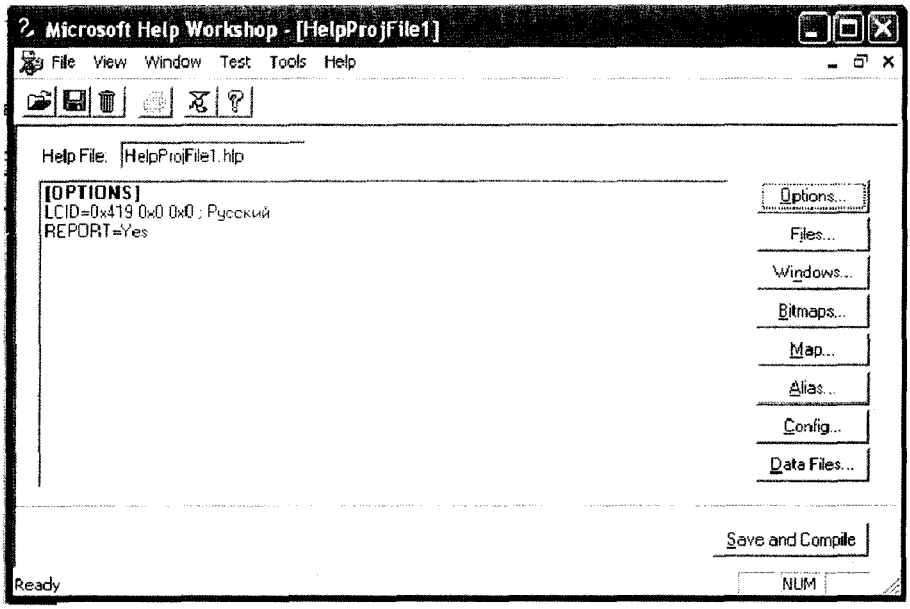


11.6-сурет. *Microsoft Help Workshop компиляторының бастапқы терезесі*



11.7-сурет. *Project File Name терезесінде проект файлының атын беру*

«Сохранить» батырмасы басылғаннан кейін пайда болған терезеде HelpProjFile1.hlp файлының параметрлерін тағайындау басталады (11.8-сурет).

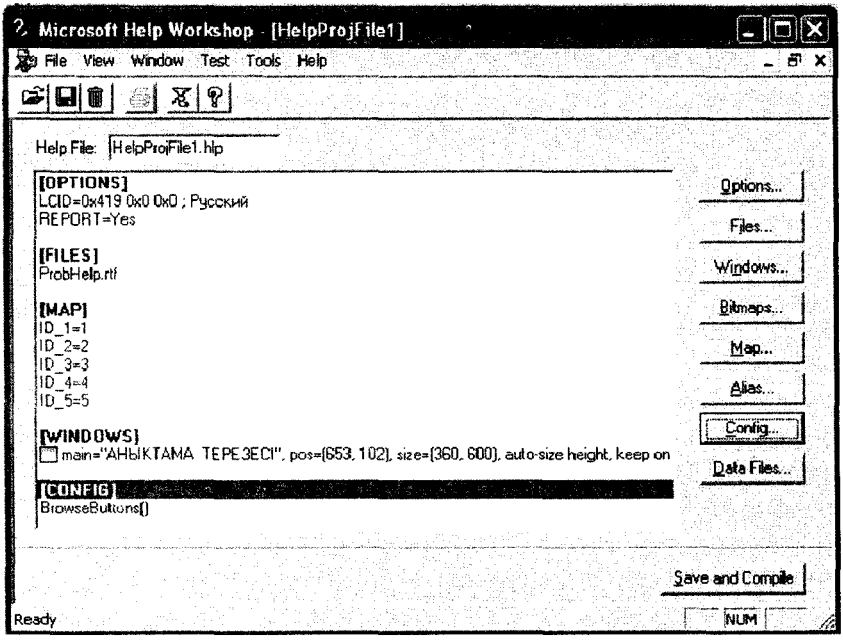


11.8-сурет. ***HelpProjFile1.hlp*** *файлы парметрлерін тағайындау терезесінің бастапқы түрі*

Бұл терезеде келесі параметрлер өзгертіледі:

– Files параметрі бұл алдыңғы даярланған ProbHelp.rtf файлына баратын маршрутты беру үшін қолданылады. Ол үшін терезедегі сол Files батырмасына шертеді, сонда Topic Files терезесі шығады, сол терезедегі Add батырмасына шертіледі, ProbHelp.rtf файлы сақталған MyHelp бумасы ашылуы керек, сол бумадағы жалғыз файл ProbHelp файлын белгілеп, «Открыть» командасы орындалады, нәтижесінде 8-суреттегі терезеде келесі жол, Files параметрі пайда болады (11.9-сурет);

– Window параметрі, бұл программадағы анықтама шығатын терезенің атын беру үшін қолданылады, бұл параметрді тағайындау үшін Window батырмасына шертіледі, пайда болған Create a Window



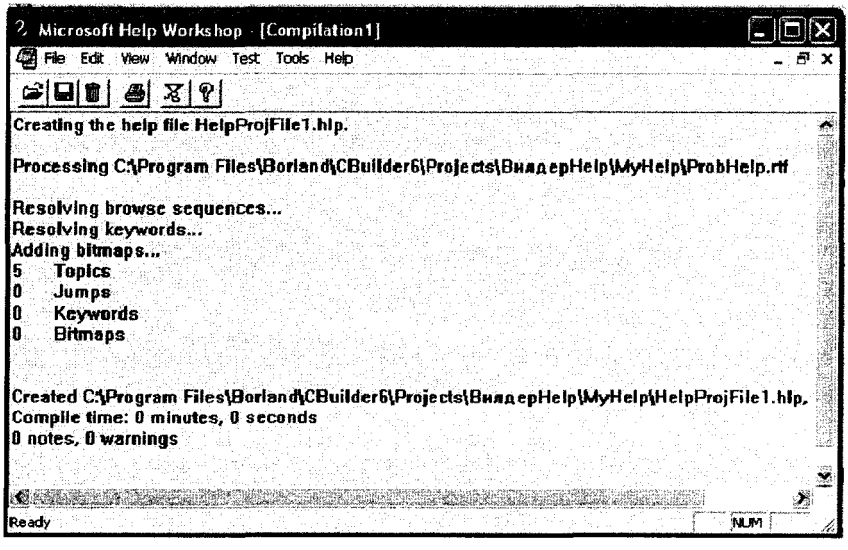
11. 9-сурет. *Анықтама файлы жобасының параметрлерін тағайындау терезесі*

терезесіндегі курсор тұрған жерге main деп терезенің типі жазылады, ОК басылғаннан соң келесі терезеде, қасиеттерін беретін Window Properties терезесіндегі Title bar text тұсына терезенің атын, мысалы, «АНЫҚТАМА ТЕРЕЗЕСІ» деп жазады, нәтижесінде 11.9-терезеде Window парметрінің жолы пайда болады;

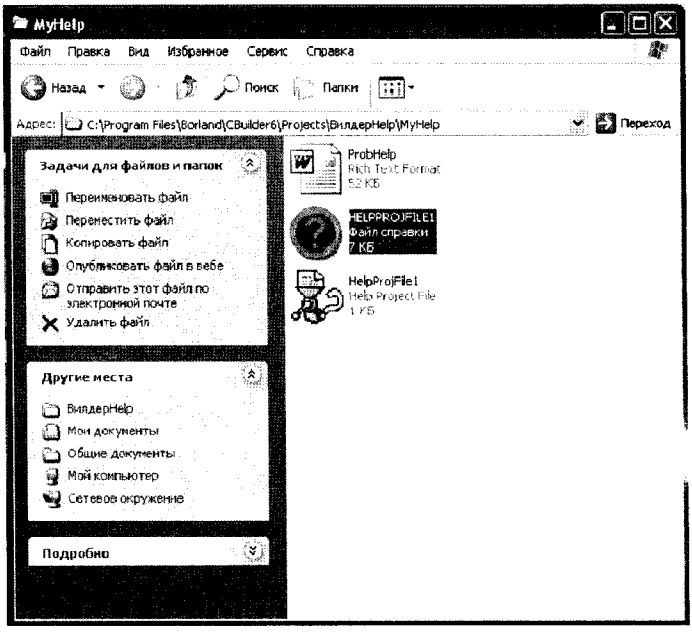
– Map параметрінің терезесінде ProbHelp.rtf анықтама файлындағы әрбір тақырыпшаға «#» таңбасы арқылы берілген идентификациялық атаудың (ID_1, ID_2, ..., ID_5) оның «+» арқылы берілген реттік номерлеріне (1,2, ... , 5) сәйкестігі тағайындалады, ол үшін терезедегі Map батырмасына шертіп, пайда болған терезедегі Add командасы орындалады, пайда болған келесі Add Map Entry терезеде Topic ID параметріне ID_1, ID_2, ... , ID_5 идентификациялық атауларды, ал Mapped Numeric Value парметріне сәйкес 1, 2, ..., 5 номерлерді бір-бірлеп жазып және олардың әркайсысы үшін ОК орындалып отырады, нәтижесінде 11.9-терезедегі Map параметрі пайда болады;

Config параметрінің қызметі анықтама файлының конфигурациясына өзгерістер енгізу, мысалы, анықтама файлында >> немесе << батырмаларын пайдаланып алдыға не кейінге оралуды жүзеге асыру үшін ол файлға BrowseButtons() функция – макросын қосу керек болады, ол үшін терезедегі Config батырмасына шертіп, пайда болған Configuration Macros терезесінде Add командасы орындалады, сонан соң шығатын Add Macro терезеде BrowseButtons() деп жазып, OK басылады, нәтижесінде 11.9-терезеде Config параметрі шығады;

Соңғы қадам – бұл параметрлері тағайындалған анықтама файлының жобасын компиляцияға жіберу (11.10-сурет). Ол үшін терезедегі Save and Compile батырмасына шертеді, егер жобада қате болмай бәрі дұрыс болса, онда нәтижесінде 11.10-суреттегі терезе шығады және де MyHelp бума тексерілетін болса, онда тағы да екі файлдың, HELPPROJFILE1.hlp анықтама файлының және HelpProjFile1.hpj жоба файлының пайда болғанын көруге болады (11.11-сурет). Егер бастапқы ProbHelp.rtf файлына өзгерістер жасалатын болса, онда жоба файлын ашып, жоба терезесіндегі Save and Compile батырмасына шертіп қайтадан компиляция жасауға болады.



1. 10-сурет. *Анықтама файлы жобасын компиляциялау нәтижесін көрсететін терезе*



11.11-сурет. *MyHelp бумада пайда болған HELPPROJFILE1.hlp анықтама файлы және HelpProjFile1.hpj жоба файлы*

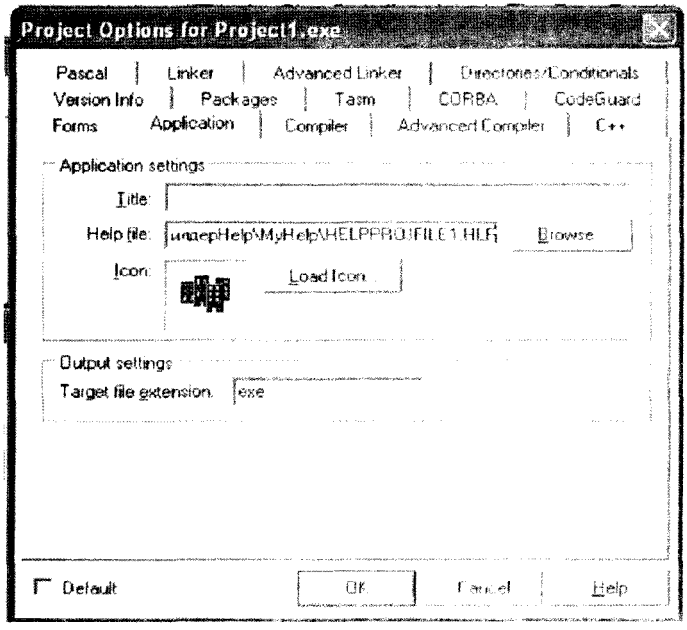
4-кезең

Бұл кезеңде дайындалған HELPPROJFILE1.hlp анықтама файлын программа-қосымшаға тіркеу және оны түрлі жолдармен шақыруды ұйымдастыру орындалады.

4.1. Анықтама файлын қосымшаға тіркеу үшін қосымша терезесінде Project->Options командасы орындалады, пайда болған терезеден Application бөлімі таңдалады, 11.12-суреттегі терезе шығады, осы терезедегі Browse батырмасының көмегімен MyHelp бумадағы HELPPROJFILE1.hlp файлы тауып алынады да, тіркеледі;

4.2. Басқару фокусы форма терезесіндегі компоненттердің бірінде тұрғанда F1 батырмасы арқылы, сол компонентке сәйкес анықтаманы алуды ұйымдастыру үшін формаға сәйкес *.cpp файлға келесі код жазылады:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
                                    { HelpContext=1; }
```

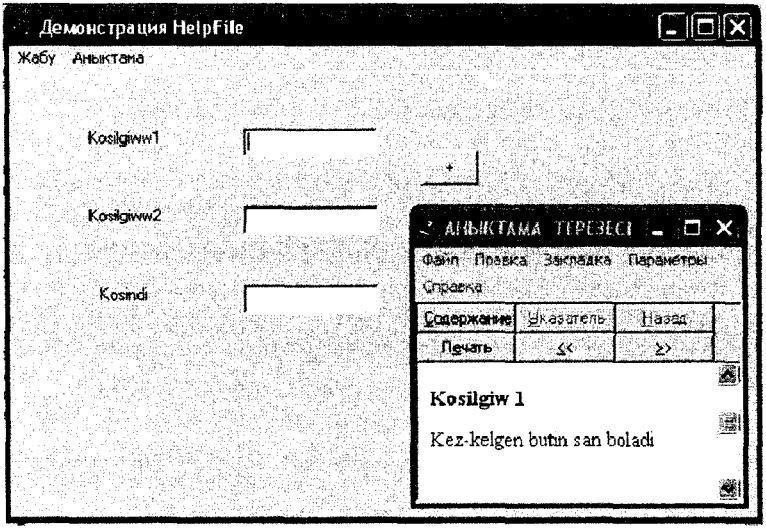


11.12-сурет. *Анықтама файлын қосымшаға тіркеу терезесі*

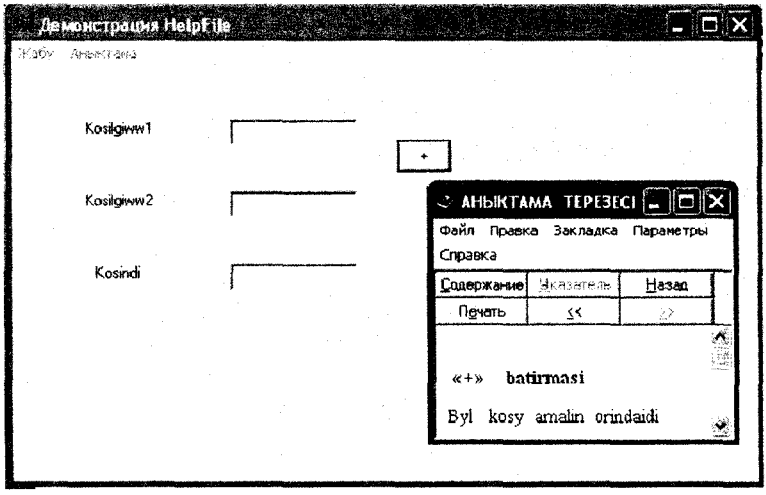
4.3. Анықтама файлын программа терезесіндегі мәзір арқылы шақыру үшін мәзір пунктін шерту оқиғасына (мысалы, Анықтама-> Көмек немесе N3 пунктінің оқиғасына) келесі код жазылады:

```
void __fastcall TForm1::N3Click(TObject *Sender)
{ Application->HelpCommand(HELP_CONTENTS,0); }
```

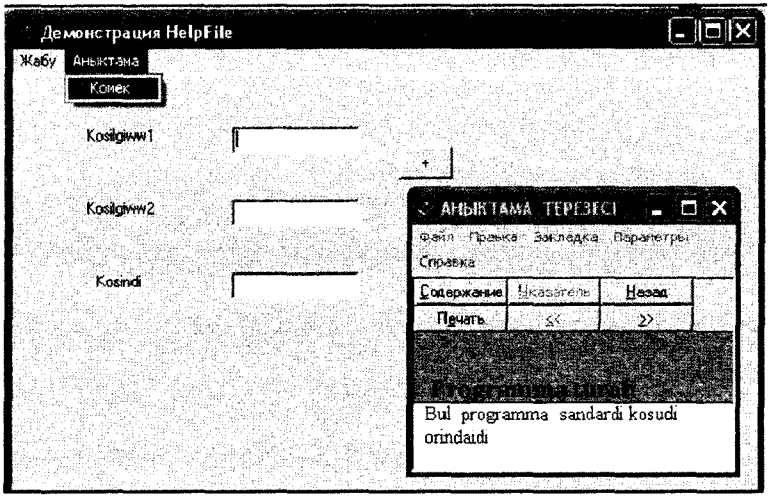
Қосымшаны орындауға жіберіп тексеріп көруге болады, егер фокус Kosilgiw1 терезесінде тұрғанда F1 пернесін басылса, онда соған сәйкес анықтама бөлімін көруге болады (11.13, а-сурет), ал егер фокус «+» батырмасында тұрса, оның өзіне сәйкес бөлім шығады (11.13,б-сурет). Сол сияқты Анықтама->Көмек командаларын орындаса, онда анықтаманы басынан бастап қарауға болады, алдыға және кейінге жылжу үшін анықтама терезесіндегі >> немесе << батырмалары пайдаланылады (11.13, в-сурет).



11.13, а-сурет. *Kosilgiw1 үшін анықтаманы шақыруды демонстрациялау*



11.13, б-сурет. *«+» батырмасы үшін анықтаманы шақыруды демонстрациялау*



11.13-сурет. *в) Анықтама->Көмек командалары арқылы анықтаманы шақыру*

Бақылау сұрақтары

1. Қосымшаның анықтамалық жүйесін құрудың не қажеті бар?
2. Қосымшаның анықтамалық жүйесінің атқаратын қызметтері қандай?
3. Қосымша анықтамалық жүйесін құрудың қандай тәсілдері бар?
4. Қосымша терезесіндегі визуалды компоненттер үшін олар туралы анықтамалар алу мүмкін бе?
5. Қосымшада **F1** пернесін басудың нәтижесі не болады?
6. Қосымшаның анықтамалық жүйесін құру қандай кезеңдерден тұрады?
7. Компоненттің *HelpContext* қасиетін қандай қызметі орындайды?
8. Анықтамалық жүйенің мәтіндік файлы қандай форматта сақталады?
9. Анықтама файлын қосымша жобасына апарып қосу қайсы терезеде орындалады?

Тапсырмалар

Келесі мазмұндағы программалардың анықтамалық жүйесін құрыңыз:

1. Пирамиданың көлемін табу.
2. Үш қабырғасы бойынша үшбұрыштың ауданын есептеу.
3. Параллелипипедтің көлемін есептеу.
4. Тізбектің жалпы кедергісін есептеу (параллель қосылған).
5. Трапецияның ауданын есептеу.
6. Тізбектің жалпы кедергісін есептеу (тізбектей қосылған).
7. Екі қабырғасы және арасындағы бұрышы бойынша үшбұрыштың ауданын есептеу.
8. Бүтін сандармен арифметикалық амалдарды орындай алатын калькулятор программа жасау.
9. Тригонометриялық функцияларды және түбір табуды орындай алатын калькулятор жасау.
10. Цилиндр көлемін табу.
11. Шеңбердің ұзындығын, дөңгелектің ауданын есептеу.

12-тарау. ПРОГРАММАДАҒЫ ЕРЕКШЕ ЖАҒДАЙЛАР

12.1. Қателер және ерекше жағдайлар

Программалаудағы «ерекше жағдайлар» түсінігі қосымшаның орындалуы кезіндегі пайда болатын қателермен байланысты қарастырылады. Программист өзінің құрған қосымшасы орындалған кезде болуы мүмкін қателерді анықтап, ол қателер бола қалған жағдайда программа қалай жұмыс жасауы керек екенін алдын ала қамтамасыз етуі тиіс. Жалпы программалау кезінде жіберілетін қателерді келесі топтарға бөледі:

– *синтаксистік қателерге* программа мәтінін теру кезінде операторлардың қате жазылуы, операторларды айыру белгілерінің қойылмауы, программа соңының көрсетілмеуі және т.б. жатады. Әдетте синтаксистік қателерді анықтау компилятордың қызметіне жатады, яғни программа синтаксистік қатесі жөнделмейінше компиляциядан өтпейді.

– *логикалық қателер*, есеп алгоритмінің дұрыс құрылмауынан болады. Логикалық қатесі бар программалар түсініксіз жұмыс жасайды, мысалы, цикл алгоритмінде циклден шығу шарты дұрыс құрылмаған болса, онда программа ешбір тоқтамастан қайталанып, нәтиже бермей жұмыс жасауы мүмкін, сол сияқты есептеу алгоритмдерінде көбейтіндінің бастапқы мәнін нольге тең деп алғанда, нәтижеде үнемі ноль шығуы мүмкін және т.б. Мұндай қателерді программаны тестілеу, яғни әр түрлі мәндер үшін орындап көру арқылы табады.

– *динамикалық қателер* – бұл программаның орындалуы кезінде пайда болып, оның орындалу тәртібінің бұзылуына немесе нәтижесіз тоқтап қалуына әкеліп соқтыратын қателер. Динамикалық қателерді немесе «орындау уақыты кезіндегі қателер» («ошибка времени выполнения», Runtime errors) деп те атайды. Динамикалық қателерге, мысалы, есептеу кезінде бөлшек бөлімінің нольге тең болуы, түбір астында теріс сан кездесіп қалуы, жады ресурстарының жетпей қалуы, программада көрсетілген маршрут бойынша файлдың табылмай қалуы, принтерде қағаздың бітіп қалуы және т.б. көптеген нәрселер жатады. Міне, қосымшалардағы осындай динамикалық қателерге байланысты бола-

тын жағдайларды *«ерекше жағдайлар»* деп атап және олармен жұмыс жасау үшін программалау тілдерінде *«ерекше жағдайларды өңдеу»* түсінігі енгізілген.

12.2. Ерекше жағдайлар класы

C++ Builder ортасындағы ерекше жағдайлардың барлығы жеке объектілер ретінде қарастырылады және бұл объектілер үшін базалық класс TObject-ден тарайтын TException класы болады, программада бұл кластың сипаттамасы SysUtils стандарт модулінде беріледі.

Егер программада ерекше жағдайлар үшін жаңа кластар құрылатын болса, онда олар TException класының өкілдері ретінде анықталады және бұл кластардың аттарын Е әрпінен бастап жазу қабылданған. Мысалы, EDivByZero, EAbort, EOutOfMemory және т.б.

Программалауда жиі қолданылатын ерекше жағдайлар кластарына келесілер жатады:

– EAbort – программаның орындалуын, ерекше жағдайлардың жалпы өңдеушісін шақырмай-ақ, жай ғана үзетін ерекше жағдайлар класы.

– EintError – бүтін сандарға амалдар қолдану кезінде пайда болатын қателерді өңдеуге арналған базалық класс және бұл кластан тарайтын келесі тума кластар қолданылады:

– EDivByZero – бүтін санды нольге бөлу;

– ERangeError – бүтін типті айнымалыға берілген диапазонға жатпайтын мәнді меншіктеу;

– EIntOverflow – бүтін типті айнымалыларға амалдар қолданғанда шамадан тыс мәндердің шығып кетуі.

– EMathError – жылжымалы үтір арқылы берілетін нақты сандарға амалдар қолдану кезінде пайда болатын қателерді өңдеуге арналған базалық класс және бұл кластан тарайтын келесі тума кластар қолданылады:

– EInvalidOp – нақты сандарға амалдар қодану кезінде пайда болатын қателер класы;

– EZeroDivide – нақты санды нольге бөлу;

– EOverFlow – нақты типті айнымалыға жады диапазонына сыймайтын жатпайтын мәнді меншіктеу;

– EUnderflow – нақты сандарға амалдар қолданудың мағынасы жоқтығын білдіріп, нәтижесінде нольге тең мән болатынын білдіретін жағдайлар класы;

– EInOutError – кез келген типтегі файлдан оқуға немесе жазуға байланысты қателер класы;

– EinvalidPointer – көрсеткіштік типтегі айнымалыларға мүмкін емес амалдар қолдану;

– EConvertError – StrToInt немесе StrToFloat және т.б. функцияларды пайдаланып типтерді түрлендіргенде кететін қателерді өңдеу класы;

– EFCreateError – файл құруда жіберілетін қателер;

– EFOpenError – файлды ашу кезінде болатын қателер;

– EResNotFound – көрсетілген файлда ресурстың болмауы;

– EPrinter – баспаға шығаруда болатын қателер;

– EMenuError – менюлер құруда жіберілетін қателер;

– EinvalidGraphicOperation – графиктік объектілермен жұмыс жасауда кететін қателер.

– TException класынан тарайтын жоғарыда аталған тума кластар үшін анықталған қасиеттер:

– Message қасиеті (типі AnsiString) ерекше жағдайдың немесе қатенің сипаттамасын сұхбат терезіне шығару үшін қолданылады, жариялануы:

```
_property AnsiString Message = { read=FMessage,
                                 write=FMessage};
```

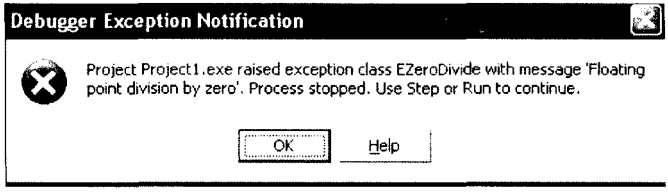
HelpContext қасиеті (типі THelpContext) анықтама жүйесіндегі объект түрінде анықталған қатеге сәйкес бөлімнің номерін немесе идентификатор номерін көрсету үшін қолданылады, жариялануы:

```
_property int HelpContext = {  read =FHelpContext,
                              write=FHelpContext,
                              nodefault};
```

12.3. Ерекше жағдайларды өңдеу

Қосымшаларда ерекше жағдайларды (**ЕЖ – ерекше жағдай**) өңдеу екі түрлі жолмен жүзеге асырылады: ерекше жағдайларды *жалпы өңдеуші* (глобальный обработчик) және *дербес өңдеуші* (локальный обработчик). Қосымшадағы ерекше жағдайларды жалпы өңдеушінің жұмысын қамтамасыз ету Application объектісінің қызметіне жатады, Application объектісі жүйе тарапынан ерекше жағдайдың пайда болғаны жөнінде хабар алысымен TExceptionEvent типіне жататын OnException оқиғасын генерациялайды да, осы пайда болған оқиғаның өңдеушісі, қосымшадағы ерекше жағдайларды жалпы өңдеушінің қызметін атқарады. Әдетте қосымшалардағы ерекше жағдайларды жалпы өңдеуші – бұл Application объектісінің әдісі болып есептелетін HandleException(System::TObject* Sender); функциясы. Бұл функция арқылы шақырылатын қосымшаның ShowException(System::TObject* ExceptObject, void* ExceptAddr); әдісі, пайда болған ерекше жағдайдың сипаттамасын экранға, сұхбат терезесіне шығарады. Мысалы, қосымшаның орындалуы кезінде «нольге бөлу немесе бөлшектің бөлімі нольге тең» болған ерекше жағдайларда жалпы өңдеуші бұл жөнінде қолданушыға хабарлайды (12.1-сурет).

C++ Builder ортасында программаның орындалуы кезінде пайда болған динамикалық қателер жалпы өңдеушінің көмегімен автоматты түрде ерекше жағдайлар кластарының біріне жататын объектілерге түрлендіріледі, ал бұл объектілер өз кезегінде программаның орындалуын тоқтатып, сол қатеге сәйкес хабарлама мәтінін экранға шығарады.



12.1-сурет. *Ерекше жағдайды жалпы өңдеуші қызметінің мысалы*

Ерекше жағдай өңделгеннен кейін, соған сәйкес құрылған ерекше жағдайлар кластарының біріне жататын объектілер өздігінен жойылып кетеді. Егер пргораммада кездесетін ерекше жағдайдың дербес

өңдеушісі болмаса, онда ол программаның жалпы өңдеушісі көмегімен өңделеді.

Программада дербес өңдеушіні қолданудың жолдары:

1. **try ... catch** блогында ерекше жағдайды өңдеу;
2. **try ... __finally** блогын пайдаланып, жадыны «қоқыстан» тазалау.

12.3.1. **try ... catch** *операторы*

Ерекше жағдайларды өңдеуде программистің дербес өңдеушілерді қолдануы үшін **try ... catch** операторы пайдаланылады.

Жалпы жазылуы:

```
try
  { // «қорғалған блоктың» басы
     throw 1-өрнек;   // бұл оператор ерекше жағдайды оны
                                        //өңдеушіге лақтырады.
     throw 2-өрнек;
      ...
     throw n-өрнек;
  } // «қорғалған блоктың» соңы
catch (1-өрнектің типіне сәйкес айнымалының типі)
   {
     1-өрнекке сәйкес ЕЖ өңдеушінің коды
   }
catch (2-өрнектің типіне сәйкес айнымалының типі)
  {
     2-өрнекке сәйкес ЕЖ өңдеушінің коды
   }

...
catch (n-өрнектің типіне сәйкес айнымалының типі)
  {
    n-өрнекке сәйкес ЕЖ өңдеушінің коды
   }
```

мұндағы **try** *(«орындап көру, байқап көру» – пробовать, попытаться, деген мағынаны білдіреді)* сөзімен басталып, {...} фигуралы жақшаға алынған программа мәтінінің фрагменттерін *қорғалған блоктар* немесе *уақытша блоктар* деп атайды, яғни ерекше жағдай туғызуы мүмкін программа коды осы қорғалған блокта орналасуы керек.

Келесі **throw** («тастай салу, лақтырып жіберу» – бросать, кидать деген мағынаны білдіреді) операторының қызметі, бұл программиске ерекше жағдайды өзінің қажетіне қарай жасауына, генерациялауға (немесе лақтыруына) мүмкіндік береді, егер **throw** операторы қолданылмаса, программадағы ЕЖ-ды C++ -тің атқарушы жүйесі немесе стандарт функциялар генерациялайды. Бұл **throw** операторынан кейін тұрған өрнектің типі пайда болатын ЕЖ-дың типін анықтайды. **throw** операторын параметрсіз де қолдануға болады. ЕЖ пайда болғаннан кейін ол блоктың жұмысы тоқтайды да, ЕЖ-дың типіне сәйкес оны өңдеуші код ізделінеді.

Өңдеуші код **catch** қызметші сөзінен басталады. Содан кейін жай жақшаның ішінде ЕЖ типі көрсетіледі, егер жай жақшаның ішінде көп нүкте (...) тұрса, онда ол кез келген ЕЖ -ды өңдей алады, мысалы:

```
catch(int i)
   {
            // int типті ЕЖ өңдеуші код
   }
catch(char* str)
   {
        // char* типті ЕЖ өңдеуші код
   }
catch (...)
  {
            //кез келген типті ЕЖ өңдеуші код
   }
```

12.3.2. **try..._ finally** *операторы*

try ..._finally операторының (немесе қорғалған блогының) программадағы жазылуы:

```
try
{
        //қорғалған блок
 }
_finally
{
        // ерекше жағдайлар туса да орындалатын код
 }
```

try..._finally операторының **try ... catch** операторынан айырмашылығы, **finally** бөліміндегі код ерекше жағдайлар туса да орындала беретіндігімен ерекшеленеді.

12.1-жаттығу. Консолдық қосымшада қарапайым ерекше жағдайды өңдеуді орындап көру.

Жаттығуды орындау:

Программалық код келесі түрде болады:

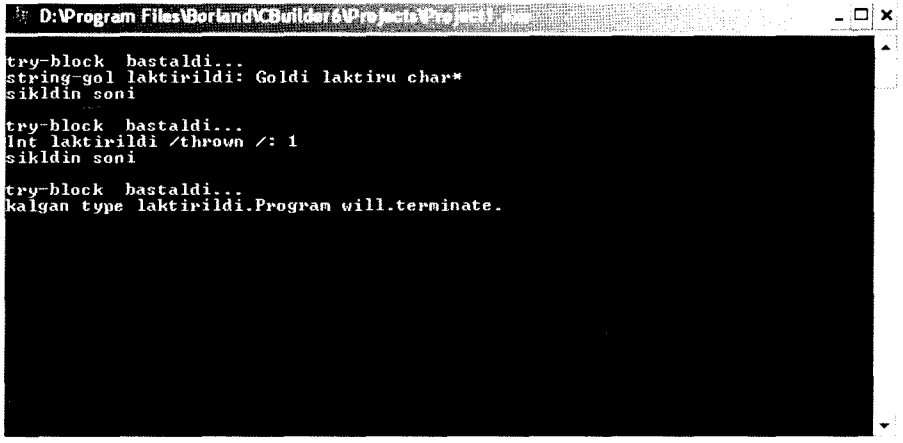
```
#include <vcl.h>
#include <iostream.h>
#include <condefs.h>
#pragma hdrstop
//---------------------------------------------------------------------------
#pragma argsused
int main()
{
   double d = 1.0;
for (int i=0; i<4; i++)
{
try
```

```
{
cout << "\n"<<"try-block  bastaldi..." <<"\n";
switch (i)
{
case 0:
throw "goldi laktiru "; // char* типті  жолды лақтыру
case 1:
throw 5; // int.-бүтінді лақтыру.
default:
throw d; //  double-ді лақтыру.
}
// Келесі оператор ЕЖ байланысты орындалмай
cout<< "Byl try-block-ka bailanisti orindalmaidi" << "\n";
} //  try-блоктың соңы
catch(int i)
{// int типті ЕЖ өңдеуші код
cout << "Int laktirildi /thrown /: " << 1 << "\n";
}
catch(char* str)
{ //char*  типті ЕЖ өңдеуші код
cout << "string-gol laktirildi: " << str << "\n";
}
catch (...) { //Кез келген ЕЖ өңдеуші код
cout << "kalgan type laktirildi."<< "Program will.terminate." <<"\n";
cin.ignore () ;
return -1; // Программадан шығу
}
cout<< "sikldin soni"<< "\n";
} // циклдің соңы
cout << "progr soni" << "\n"; // Бұл операторлар орындалмайды, себебі
                              //үшінші catch (...) программаны аяқтап тастайды
return 0;
}
```

Бұл мысалда цикл төрт рет орындалуы керек. Цикл ішінде **i**-цикл параметрінің мәндеріне сәйкес **int**, **char*** және **double** типті ЕЖ

шақыратын қорғалған блок орналасқан. **i=0** болғанда **throw** операторы **char*** типті жолды лақтырады, ал оны **char*** типті екінші тұрған өңдеуші ұстап алады да, соған сәйкес жазуды шығарады, себебі кодта содан басқа ештеңе жазылмаған, тура сол сиякты i=1 болғанда, қорғалған блоктағы **throw** операторы **int** типті ЕЖ лақтырады, оны бірінші тұрған **catch(int** i) өңдейді. i=2 болғанда **double** типті ЕЖ пайда болады, ал оған сәйкес **catch** қарастырылмаған, бірақ оны кез келген жағдай үшін орындалатын үшінші **catch** (...) ұстайды және ондағы кодтағы cin.ignore () және return -1; командалары программадан шығуды немесе аяқтауды орындайды, яғни цикл төртінші рет i=3 үшін орындалып үлгермейді.

Программа орындалуының нәтижесі (12.2-сурет):



12.2-сурет. *Консолдық қосымшада қарапайым ерекше жағдайды өңдеу*

12.2-жаттығу. Windows-қа арналған қарапайым қосымшадағы «бөлу», «логарифмдеу» амалдарына байланысты туындайтын ерекше жағдайларды екі түрлі жолмен, яғни жалпы өңдеушінің және дербес өңдеушінің көмегімен орындап көру.

Жаттығуды орындау:

Алдымен Windows-ға арналған қарапайым қосымша құрылады. Мысалы, бұл қосымша мәндері программа терезесінде енгізілетін a,b,c

сандарына сәйкес b/a және log(c) өрнектерінің мәндерін есептейтін болсын. Құрылатын программада b/a және log(c) өрнектерін есептеу кезінде туындайтын ерекше жағдайлар үшін ортақ бір өңдеуші тағайындалатын болады. Бұл программаның коды келесі түрде болады:

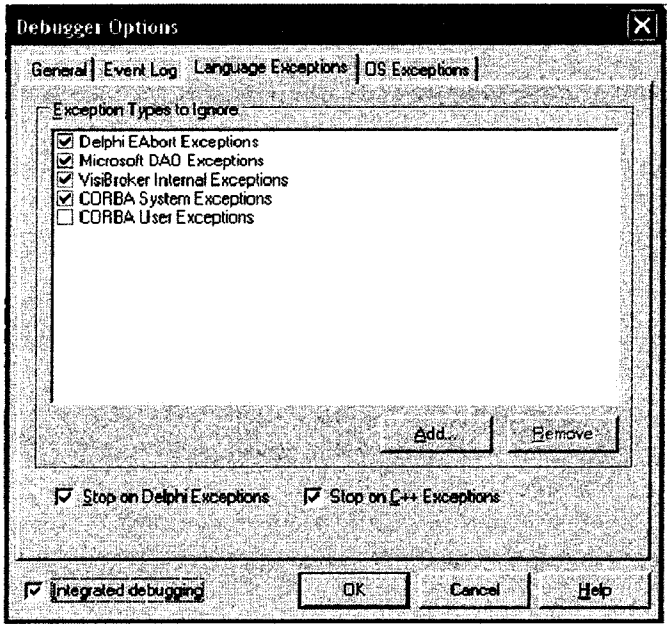
```
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{  float a,b,c;
float d,x1,x2;
a=StrToInt(Edit1->Text);   //a-ның мәнін енгізу
b=StrToInt(Edit2->Text); //b-ның мәнін енгізу
c=StrToInt(Edit3->Text);  //c-ның мәнін енгізу
try                              //қорғалған блок басталды
{ d=b/a;
x1=log(c);
x2= d;
Edit4->Text=FloatToStrF(x1,ffGeneral,5,2);
Edit5->Text=FloatToStrF(x2,ffGeneral,5,2);
}
catch(...)   // барлық ерекше жағдайлар үшін ортақ өңдеуші
{
MessageDlg("try-блокта кате бар",mtError, TMsgDlgButtons() << mbOK, 0);
}
Edit6->Text="Мен орындалдым";  // catch(...)- блоктан кейін
                                  // орындалатын оператор
}
```

Бұл қосымшаны екі түрлі жағдайда орындап көру керек.

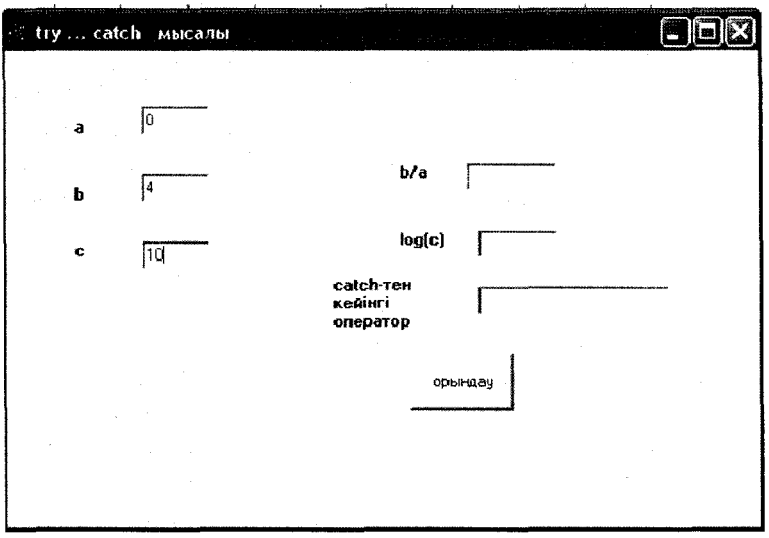
1-жағдай.

Жалпы өңдеушінің қызметі қарастырылатын болады, сондықтан бұл кезде C++ Builder ортасының жалпы өңдеушісі іске қосылған болуы керек немесе Debugger Options терезесіндегі Stop on Delphi Exceptions және Stop On C++ Exceptions немесе Integrated debugging опциялары іске қосылып тұруы қажет (12.3-сурет).

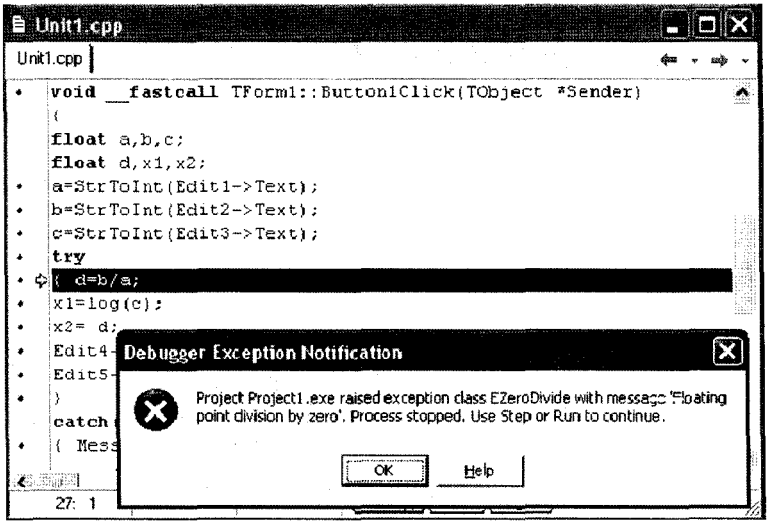
Келесі қадамда программа терезесінен a, b, c сандарының немесе айнымалыларының мәндері ретінде, сәйкесінше 0, 4, 10 сандары енгізілетін болады (12.4-сурет). Әрі қарай «орындау» батырмасына шерткен кезде **try-** қорғалған блокта d=b/a; операторы үшін «санды нольге бөлуге болмайды» деген қағидаға байланысты ерекше жағдай туады. Бұл жағдайда ерекше жағдайды өңдеуді жалпы өңдеуші өзі атқаратын болады. Жалпы өңдеушінің аталған ерекше жағдайға сәйкес хабарламасы әдеттегідей Debugger Exception Notification терезесіне шығарылатын болады (12.5-сурет).



12.3-сурет. *Жалпы өңдеушінің опцияларын іске қосу*



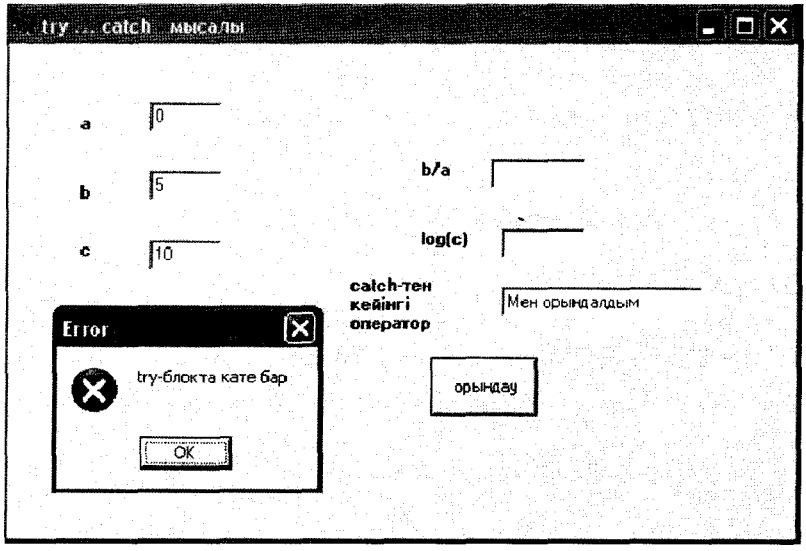
12.4-сурет. *a, b, c айнымалыларына мәндер беру*



12.5-сурет. *Жалпы өңдеушінің ерекше жағдайға сәйкес хабарламасы*

Дербес өңдеушінің жұмысын көру үшін жалпы өңдеушіні уақытша тоқтата тұру керек болады, ол үшін Tools -> Debugger Options командаларын орындап, пайда болған терезеде Language Exceptions жапсырмасының Stop on Delphi Exceptions және Stop On C++ Exceptions немесе Integrated debugging опцияларының қызметтері тоқтатылады (12.3-сурет).

Келесі қадамда тура бірінші жағдайдағы секілді программа терезесінен a,b,c айнымалыларының мәндері беріледі. «Орындау» батырмасына шерткеннен кейін, **try-** қорғалған блоктағы d=b/a; операторы үшін туған ерекше жағдайды өңдеуді, бұл жолы программистің өзі тағайындаған **catch(...)** блоктағы өңдеушісі атқаратын болады. Дербес өңдеушінің қызметінің нәтижесінде экранға «try-блокта қате бар» деген хабарлама шығады және өңдеушіден кейін тұрған оператордың орындалғанын терезелердің біріндегі «Мен орындалдым» деген мәтіннен көруге болады (12.6-сурет).



12.6-сурет. *Дербес өңдеушінің ерекше жағдайға сәйкес хабарламасы*

Қосымшаның орындалуы кезінде пайда болатын қателерді алдын ала болжау, күрделі есептердің қатарына жататын болғандықтан, программаның қате кетуі мүмкін негізгі бөлігін «қорғалған блокқа орналастыру» тәсілін программистердің кеңінен қолданатынын байқауға болады.

Қосымшаның орындалуы кезінде пайда болуы мүмкін қателердің әрқайсысына сәйкес, қатені өңдеушінің болуы программаның тиімділігін және құндылығын арттыратын болады.

Жалпы, қосымшаны жобалау және жүзеге асыру барысында қосымша орындалған кезде пайда болуы мүмкін қателердің барлығы алдын ала ескеріліп, ол қателер бола қалған жағдайдағы программаның қалай жұмыс жасау керектігі, яғни әрбір қатеге сәйкес өңдеушінің қызметтері де жан-жақты қарастырылуы тиіс.

Бақылау сұрақтары

1. Қосымшада қателердің қандай түрлері кездеседі?
2. Динамикалық қате деген не?
3. Логикалық қателер қандай болады?
4. Қосымшаның орындалуы кезінде пайда болатын қателер қалай аталады?
5. Ерекше жағдай деген не?
6. Ерекше жағдайды өңдеудің қандай тәсілдері бар?
7. **try...__ finally** операторының **try ... catch** операторынан айырмашылығы неде?
8. Ең жоғарғы ерекше жағдай класы қалай аталады?
9. Қорғалған блоктың атқаратын қызметі қандай?
10. Қорғалған блоктағы **throw** операторының атқаратын қызметі қандай?
11. «Нақты санды нольге бөлуге» сәйкес ерекше жағдай класы қалай аталады?
12. EmathError ерекше жағдайлар класын қандай қателерді өңдеу үшін қолданады?
13. EFCreateError ерекше жағдайлар класын қандай қателерді өңдеу үшін қолданады?

14. FOpenError ерекше жағдайлар класын қандай қателерді өңдеу үшін қолданады?

Тапсырмалар

Келесі мазмұндағы есептерді шешуге арналған қосымшалардағы ерекше жағдайларды анықтап, оны екі жолмен өңдеуді (жалпы және дербес өңдеушілер көмегімен) қарастырыңыз:

1. Пирамиданың көлемін табу;
2. Үш қабырғасы бойынша үшбұрыштың ауданын есептеу;
3. Параллелипипедтің көлемін есептеу;
4. Параллель қосылған тізбектің жалпы кедергісін есептеу;
5. Трапецияның ауданын есептеу;
6. Шеңбердің ұзындығын табу;
7. Тізбектей қосылған тізбектің жалпы кедергісін есептеу;
8. Дөңгелектің ауданын есептеу;
9. Квадрат теңдеуді шешу;
10. Бүтінсанды массив элементтерінің арифметикалық ортасын табу;
11. Бүтінсанды массив элементтерінің геометриялық ортасын табу;
12. Матрицаларды өзара көбейту.

Негізгі мәзір және оның командалары

Негізгі мәзірдің **File** бөлімінде келесі командалар орналасады:

1-кесте

***File** бөлімінің командалары*

File командалары	Қызметтері
New	Жаңа жобаны ашу, мұның өзі жобаның түріне байланысты әрі қарай бөлінеді: Application-Windows қосымшасының жаңа жобасын құру; CLX Application-Linux платформалы қосымшаның жаңа жобасын құру; Data Module – деректер қорымен жұмыс жасайтын компоненттерді біріктіретін жаңа модульді құру; Form – жаңа форманы құру, Frame – жаңа фрейм құру; Unit – жаңа модуль құру; Other – өзгеше жобалар құру.
Open	Жобаға қатысатын файлдарды сұхбат терезесінде таңдап ашуды жүзеге асырады.
Open Project	Сұхбат терезесінде тек қана *.bpr немесе * .bpk кеңеймелі жоба файлдарын ашуды орындайды.
Reopen	Соңғы жұмыс жасалған жобалардың жоба файлдары мен модульдерінің файлдарын тізімнен таңдап ашуға мүмкіндік береді.
Save	Ағымдағы жобаны сақтау, егер жоба бірінші рет сақталатын болса, онда жоба файлы мен модуль файлының аттары сұралады.
Save As	Ағымдағы жобаның модулін немесе модуль файлын (*.cpp) жаңа атпен қайтадан сақтау.
Save Project As	Ағымдағы жобаның жоба файлын (*.bpr) жаңа атпен қайта сақтау
Save All	Ағымдағы жобаның жоба файлын және модулін қатарынан сақтау.
Close	Ағымдағы жоба файлын, модулін және форманы жабу.
Close All	Барлық ашылған файлдарды жабу.

Include Unit Hdr	Ағымдағы модульге немесе модуль файлына #include директивасы арқылы Use Unit сұхбат терезесінен басқа модульдерді қосуды орындайды, мұны пайдалану үшін жобадағы модуль файлдарының саны 2-ден кем болмауы қажет.
Print	Ағымдағы файлды баспаға шығару.
Exit	C++Builder ортасының жұмысын аяқтап, жабу.

Негізгі мәзірдің **Edit** бөлімінде келесі командалар орналасады:

2-кесте

***Edit** бөлімінің командалары*

Edit командалары	Қызметтері
Undo/Undelete	Соңғы жасалған әрекетті қайтару немесе алып тастау
Redo	Қайтарылған немесе алынып тасталған әрекетті қайтадан қалпына келтіру
Cut	Белгіленген орынды (мәтін немесе объект) қиып алып, уақытша буферге (clipboard) апару
Copy	Белгіленген орынның көшірмесін уақытша буферге апару
Paste	Уақытша буфердегі көшірмелерді курсормен көрсетілген жерге қою, егер уақытша буферде көшірмелер жоқ, бос болса, бұл команда пассив болады
Delete	Белгіленген орынды жою
Select All	Формадағы орналасқан барлық компоненттерді бірден белгілеу
Align to Grid	Белгіленген компоненттерді торкөздің ең жақын нүктесіне туралайды
Bring to Front	Формадағы бірнеше компоненттер беттесіп орналасқанда, белгіленген компонентті алдыға шығару
Send to Back	Формадағы бірнеше компоненттер беттесіп орналасқанда, белгіленген компонентті артына қарай жіберу

Align	Белгіленген компоненттің сұхбат терезесінде берілетін параметрлер бойынша вертикаль және горизонталь түрде орнын ауыстыру
Size	Белгіленген компоненттің сұхбат терезесінде берілетін параметрлері бойынша өлшемдерін өзгерту
Scale	Формадағы барлық компоненттердің өлшемдерін проценттік шкала бойынша өзгерту
Tab Order	Формадағы компоненттерге Tab пернесімен көшудің ретін анықтау
Creation Order	Формадағы визуалдық емес компоненттерге Tab пернесімен көшудің ретін анықтау
Flip Children	Формадағы оң және сол жақтағы орналасқан компоненттердің орындарын ауыстыру
Lock Controls	Формадағы барлық компоненттердің өз орындарын сақтап қалуын қамтамасыз етеді, яғни оларды жылжыту мүмкін емес
CORBA Refresh	Барлық орындалатын CORBA кластарын алдын ала құрылған IDL файлға түрлендіреді
Use CORBA Object	Сервермен жұмыс жасау үшін байланысқан CORBA объектінің кодтарын CORBA клиенттік қосымшасына қосу

Негізгі мәзірдің **Search** бөлімінің командалары және олардың қызметтері:

3-кесте

***Search** бөлімінің командалары*

Search командалары	**Қызметтері**
Find	Берілген мәтінді код редакторының терезесіндегі мәтіннен іздеп тауып береді
Find in Files	Берілген мәтін кездесетін файлдардың тізімін код редакторының төменгі жағындағы терезеге шығарып береді
Replace	Берілген мәтінді тауып, оны басқа бір жаңа мәтінмен ауыстырады
Search Again	Соңғы іздеген мәтінді қайтадан іздейді

Incremental Search	Курсорды іздеген мәтінге апарады
Go to Line Number	Курсорды номері көрсетілген жолға апарып қояды
Go to Address	Берілген адреске бару

Негізгі мәзірдің **View** бөлімінің командалары және олардың қызметтері:

4-кесте

***View** бөлімінің командалары*

View командалары	Қызметтері
Project Manager	Project Manager терезесін экранға шығару
Object Inspector	Object Inspector терезесін экранға шығару
Object TreeView	Object TreeView терезесін экранға шығару
To-Do List	To-Do тізімді ағымдағы жобамен байланыстыру
Alignment Palette	Align палитрасының терезесін экранға шығару
ClassExplorer	ClassExplorer терезесін экранға шығару
Component List	Components сұхбат терезесін экранға шығару
Window List	Ашық тұрған Windows терезелерінің тізімін шығарады
Debug Windows	Debugger ішкі мәзірлерін экранға шығару (Окно отладки)
Desktops	Экрандағы жеке беттерді сақтау, қарау, алып тастау
Toggle Form/Unit	Форма және модуль терезелерін бері қарай шығару
Units	View Unit сұхбат терезесін экранға шығару, бұл терезеде жобадағы барлық модульдердің тізімі шығады
Forms	View Form сұхбат терезесін экранға шығару, бұл терезеде жобадағы барлық формалардың тізімі шығады
Type Library	Type Library редакторының терезесін экранға шығару
New Edit Window	Жаңа код редакторын ашу
Toolbars	Тізімнен таңдалған саймандар панелін көрсету немесе жасыру

Project бөлімінің командалары

5-кесте

***Project** бөлімінің командалары*

Project командалары	Қызметтері
Add to Project	Файлды жоба құрамына кіргізу
Remove from Project	Файлды жоба құрамынан алып тастау
Import Type Library	Жобаға кітапханалардан типтерді импорттау
Add to Repository	Жобаны объектілер қоймасына (Object Repository) қосу
View Source	Жоба файлының мәтінін код редакторының терезесіне шығару
Edit Option Source	Ағымдағы жобаның * .bpr файлын код редакторының терезесіне шығару
Export Makefile	Жоба файлына (*.bpr) қосымша makefile (*.mak) файлын құру
Add New Project	New Items сұхбат терезесінен таңдалынған объектілерді жобаға қосу, бұл терезеде объектілер қоймасында сақталған жобаларды да таңдауға болады
Add Existing Project	Open Project сұхбат терезесінен таңдалған жоба файлын (*.bpr) жоба менеджеріне қосу
Compile Unit	Қандайда болмасын бір өзгерістер жасалған модульдерді (Unit) компиляциялау
Make project	Жоба құрамындағы соңғы өзгерістер жасалған файлдарды ғана жылдам компиляциялайды (ускоренная сборка проекта)
Build project	Жоба құрамындағы файлдарда өзгеріс болған сайын олардың барлығын толығымен компиляциялайды (полная сборка проекта)
Information for project	Жоба туралы деректерді экранға шығару
Make All Projects	Жоба тобына (ProjectGroup) кіретін барлық жобалардың барлығын бірден жылдам компиляциялау
Build All Projects	Жоба тобына (ProjectGroup) кіретін барлық жобаларды барлығын бірден толық компиляциялау
Options	Project Options сұхбат терезесінде жобаның опцияларын тағайындау, мұнда жобаның версиясы туралы деректерді, таңбасын, анықтама файлын және т.б. беруге болады

Run бөліміндегі командалар:

6-кесте

***Run** бөлімінің командалары*

Run командалары	Қызметтері
Run	Қосымшаны компиляциялау және орындау немесе программаны бірден орындау
Attach to Process	Орындалып тұрған процестердің тізімі берілген терезені шығару
Parameters	Қосымшаның арнайы параметрлерін беретін терезені шақыру
Register ActiveX Server	Жоба ActiveX жоба болғанда, оны Windows регистріне тіркеу (зарегистрировать)
Unregister ActiveX Server	Жоба ActiveX жоба болғанда, оны Windows регистрінен шығару (отменить регистрацию)
Install MTS Objects	Жобадағы MTS пакеттің MTS объектілерін инсталляция жасау
Install COM+ Objects	Жоба COM+ қосымша болғанда, объектілерді инсталляциялау үшін қолданылады
Step Over	Программаны әрбір жолы бойынша қадамдап орындау (F8 пернесі басылып отырады), мұнда бір подпрограмма бір қадам ретінде орындалады
Trace Into	Программаны қадамдап орындау, мұнда подпрограммалардың әрбір жолы жеке, бір қадам ретінде орындалады
Trace To Next Source Line	Программадағы келесі орындалатын жол алдындағысының коды тексеріліп болғанша тоқтап тұрады
Run To Cursor	Программа код редакторының терезесіндегі курсор тұрған жерге дейін ғана орындалады
Run Until Return	Бұл программа уақытша тоқтап тұрған кезде (Program Pause) функцияларды орындай алады
Show Execution Point	Орындалу нүктесіндегі курсордың позициясын анықтайды
Program Pause	Орындалып тұрған программаны уақытша тоқтату
Program Reset	Программаның орындалуын аяқтап, уақытша жадыны босатады
Inspect	Енгізілетін жолды тексеру үшін терезе ашады

Evaluate/Modify	Evaluate/Modify сұхбат терезесінде немесе кодта көрсетілген символдар туралы дерек береді
Add Watch	Watch Properties бақылау терезесін шақырады
Add Breakpoint	Программаның қай жерге дейін орындалып тұрғанын көрсетеді

Component бөлімінің командалары:

7-кесте

***Component** бөлімінің командалары*

Component командалары	**Қызметтері**
New Component	Жаңа компонентті Component Expert арқылы іздеу
Install Component	Компонентті немесе жаңа пакетті орнату
Import ActiveX Control	ActiveX кітапханасынан жобаға импорттау
Create Component Template	Құрылатын компонентке (мысалы, Component Wizard көмегімен) жаңа атауды, палитра парағын және таңбасын беруді орындайтын терезені шақырады
Install Packages	Жобаға қажет арнайы пакеттерді орнату
Configure Palette	Компоненттер палитрасының конфигурациясын тағайындайтын сұхбат терезесін шақыру

Database бөлімінің командалары:

8-кесте

***Database** бөлімінің командалары*

Database командалары	**Қызметтері**
Explore	SQL Explore программасын шақыру
SQL Monitor	SQL Monitor программасын шақыру
Form Wizard	Деректер қорының кестесімен жұмыс жасайтын Form Wizard шеберін шақыру

Tools бөлімінің командалары келесі кестеде орналасқан:

9-кесте

***Tools** бөлімінің командалары*

Tools командалары	Қызметтері
Environment Options	Программистің өз қажетіне қарай орта конфигурациясын, мысалы, компоненттер палитрасын, форма терезесінің дизайнын және т.б. опцияларды өзерту үшін Environment Options сұхбат терезесін шақыру
Editor Options	Код редакторының конфигурациясын өзгерту үшін Editor Properties сұхбат терезесін шақыру
Debugger Options	Отладка жасаудың оццияларын тағайындайтын Debugger Options сұхбат терезесін шақыру
Repository	Объектілер тізімін көрсететін Object Repository сұхбат терезесін шақыру
Build Tools	Компиляциялау утилиттерін (мысалы, CCompiler) таңдауды немесе өзгертуді орындайтын Build Tools сұхбат терезесін шақыру
External Editor	Web парақ редакторын қосады
Web App Debugger	Web қосымшамен Web сервер қосымшаның отладкасын тестілеу
VisiBroker SmartAgent	VisiBroker SmartAgent программасын жүктеу немесе тоқтату
IDL Repository	IDL Repository сұхбат терезесін шақыру
Regenerate CORBA IDL Files	Regenerate CORBA IDL Files сұхбат терезесін шақыру
Configure Tools	Tools мәзірін өзгерту үшін Tools Options сұхбат терезесін шақыру
Database Desktop	Paradox, dBASE, және SQL форматтағы деректер қорын құру, қарау, сұрыптау, өзгерту, сұраныс жасау және т.б. орындайтын Database Desktop программасын шақыру. Ол Program Files\Common Files\Borland Shared\Database Desktop маршрутымен орналасады
Visual C++ Project Conversion Utility	Microsoft Visual C++ жобаларының файлдарын C++Builder жобаларының файлдарына айналдыру

CodeGuard Configuration	CodeGuard-ты пайдаланып, отладка орындалу уақытында арнайы опцияларды тағайындауға болады
Image Editor	Қосымша үшін ресурс файлдарын, таңбаларды (icon) және т.б. құратын және өзгертетін Image Editor редакторын шақыру
Package Collection Editor	Сайман коллекция пакеттерін қарау және өзгерту мен басқа файлдарды пакетке қосуды, алып тастауды орындай алады
XML Mapper	Жалпы XML құжаттар мен деректер жиынтығын арасындағы байланысты орнату құралдарының терезесін экранға шығару

Негізгі мәзірдің **Windows** бөлімінде сол жобадағы ашық тұрған терезелердің тізімдері орналасады, тізімнен таңдау арқылы қалаған терезеге ауысуға болады.

Help бөлімінің командалары:

10-кесте

***Help** бөлімінің командалары*

Help командалары	**Қызметтері**
C++Builder Help	C++Builder ортасының анықтама жүйесін шақыру
C++Builder Tools	C++Builder ортасының анықтама жүйесін шақыру
Borland Home Page	Web браузер терезесінде Borland's World Wide Web сайтын ашу
Borland Community Page	Web браузерде Borland's Web сайтының арнаулы парағынан жаңа толықтыруларды, мақалаларды, мысалдардың кодтарын қарау.
C++Builder Home Page	Web браузерде өзіңіздің C++Builder Web парағыңызды қарау
C++Builder Developer Support	C++ Builder үшін ақпараттар беретін, техникалық қолдау көрсететін және соңғы версияларын жүктеуге мүмкіндік беретін жасаушы-компанияның веб-сайтына тікелей қатынау
C++Builder Direct	C++Builder және Borland туралы жаңалықтарды C++Builder Web парағында автоматты түрде хабарлап, тікелей байланыстыру

Customize	Анықтама файлдарының орналасу, қосу және алып тастау конфигурацияларын орындайтын OpenHelp программасын шақыру
About	C++Builder версиясы туралы деректерді экранға шығару

Терминдердің қазақша-орысша сөздігі

А

ағымдағы позиция	текущая позиция
адрес бойынша мәнді алу амалы	операция доступа по адресу
адресті алу амалы	операция взятия адреса
айнымалының әсер ету аймағы	область действия переменной
айнымалының көріну аймағы	область видимости переменной
айнымалының өмір сүру уақыты	время существования переменной
атау аймағы	пространство имен

Ә

әдеттегідей, қалыпты жағдайда	по умолчанию

Б

буынсөз	подстрока

Г

глобальдық уақытпен берілген статикалық класс жадылары	статический класс памяти с глобальным временем жизни

Д

дерек-мүшелер	данные-члены
деректерді еркін пайдалану	произвольный доступ к данным
деректердің динамикалық құрылымы	динамическая структура данных
деректің жадыдағы ішкі берілуі	внутреннее представление данных в памяти

Е

ерекше жағдайлар	исключительные ситуации
ерекше жағдайларды өңдеу	обработка исключительных ситуации

Ж

жай қабылдаушылық	одиночное или простое наследование
жалпы өңдеуші	глобальный обработчик

жапсырма	вкладка
жергілікті көру аймағы	локальная область видимости
жылжымалы мәтін	всплывающая подсказка
жылжымалы үтірмен берілетін нақты сандар	числа с плавающей точкой
К	
кең ауқымды көру аймағы	глобальная область видимости
кезек	очередь
көп қабылдаушылық	множественное наследование
көрсеткіш	указатель
кіріктірілген орта	интегрированная среда
Қ	
қабылдаушылық механизмі	механизм наследования
қайыра жүктеу механизмі	механизм перегрузки
қалып күй қатары	строка состояния
қасиет-	свойства
қорғалған блоктар	защищенные блоки
қосымша	приложение
қызметші сөз	служебное слово, ключевое слово
Л	
локальдық уақытпен берілген автоматты класс жадылары	автоматический класс памяти с локальным временем жизни
О	
оқиға	событие
Ө	
өріс	поле
П	
параметрленген кластар	параметризованне классы
С	
сілтеме	ссылка

Т

таза виртуал әдіс	чистый виртуальный метод
тақырыптық файл	заголовочный файл
типтерді келтіру	приведение типов
тоқтатушы әдіс	перекрывающий метод
тума класс	производный класс, дочерний класс
тума объект	объект- потомок
түпкі класс	базовый класс, родительский класс
түпкі объект	объект-родитель
тізім	список

Ү

үйлесімді функциялар	дружественные функции

Ф

файлға біртіндеп қатынау	последовательный доступ файлу
файлға еркін қатынау	произвольный доступ файлу
функция-мүшелер	функции-члены
функцияны қайыра жүктеу	перегрузка функции
функцияның көру аймағы	область видимости функции

І

ішкі берілу кодтары	внутренние коды представления

Әдебиеттер

1. Архангельский А.Я., Тагин М.А. Программирование в C++ Builder 6 и 2006. – М.: ООО «Бином-Пресс», 2007 г. – 1184 с.: ил.

2. Бадд Т. Объектно-ориентированное программирование в действии. – СПб.: Питер, 1997. – 464 с.

3. Буч Г. Объектно-ориентированный анализ и проектирование с примерами на C++. – М.: БИНОМ, 1998. – 560 с.

4. Вирт Н. Алгоритмы+структуры данных = программы / Пер. с англ. – М.: Мир, 1985. – 406 с.

5. Дейтел Х., Дейтел П. Как программировать на Си. – М.: Бином, 2000. – 1088 с.

6. Культин Н.Б. C++ Builder в задачах и примерах. – Петербург, 2005. – 336 с: ил.

7. М.Ө. Мұқашева. Турбо Паскаль тілінде программалау негіздері. – Актөбе, 2003. – 104 б.

8. М.Ө. Мұқашева, К.К. Заурбекова, Программалау /Pascal, Delphi/. Оқу құралы. – Астана: Л.Н. Гумилев атындағы ЕҰУ, 2007 ж. – 235 б.

9. М.Ө. Мұқашева. «Объектіге бағдарланған программалаудың негізгі ұғымдарын оқыту ерекшеліктері» // Қ.А. Ясауи атындағы ХҚТУ Хабаршысы. – №1. – 2007 ж. – 185-192-бб.

10. М.Ө. Мұқашева. «C++ Builder 6 қосымшасының анықтама жүйесін құру» // «Проблемы дифференциальных уравнений, анализа и алгебры» V халықаралық конференция материалдары. – Ақтөбе, 2009 ж., 510-514-бб.

11. Мукашева М.У. «Математические основы решения обучающих задач по программированию», Вестник ЕНУ им. Л. Н. Гумилева. – №4. – 2006 г. – 42-46 с.

12. Подбельский В. В. Язык Си++: Учебное пособие. – М.: Финансы и статистика,1996. – 560 с.

13. Лафоре Р. Объектно-ориентированное программирование в C++. Классика Computer Science. 4-е изд. – СПб.: Питер, 2003. – 928 с.

14. C/C++. Программирование на языке высокого уровня /Т.А. Павловская. – СПб.: Питер, 2003. – 461 с.: ил.

15. Страуструп Б. Язык программирования C++. – СПб.: БИНОМ, 1999. – 991 с.

16. Шилдт Герберт. C++. Руководство для начинающих, 2-издание: Пер. с анг. – М.: Издательский дом «Вильямс», 2005. – 672 с. :ил.

«Жоғары білім» сериясы

Мұқашева Манаргүл Өмірзаққызы

ПРОГРАММАЛАУ
/ C++ Builder 6 /

Оқулық

Редакторы Гүлден Оспанова
Техникалық редакторы Эльмира Заманбек
Көркемдеуші редактор Жеңіс Қазанқапов
Корректоры Назгүл Бимағанбетова
Компьютерде беттеген Эльмира Заманбек

Басуға 03.02.13 қол қойылды.
Пішімі 60x84 $^{1}/_{16}$. Қағазы офсеттік. Офсеттік басылыс.
Шартты баспа табағы 26,51.
Тапсырыс №0038. Таралымы 1000 дана.

«Фолиант» баспасы.
010000, Астана қаласы, Ш. Айманов көшесі, 13

«Фолиант» баспасының баспаханасында басылды