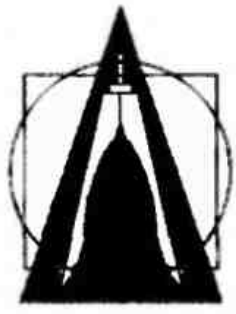
КӘСІПТІК БІЛІМ



А. Шайқұлова, С. Аманжолова,
Н. Асқарова

БАҒДАРЛАМАЛЫҚ ЖАСАҚТАМАНЫҢ ҚАЗІРГІ ЗАМАНҒЫ ЖАБДЫҚТАРЫ

Оқулық

2-басылым

Қазақстан Республикасы Білім және ғылым министрлігі
техникалық және кәсіптік білім беру
ұйымдарына ұсынады

«Фолиант» баспасы
Астана-2010

УДК 004 (075.32)
ББК 32.973 я 722
Ш 18

Пікір жазғандар:
Қайранов М.Ж. – техника ғылымдарының кандидаты, доцент;
Ермеков Н.Т. – физика-математика ғылымдарының кандидаты, доцент

Ш 18 **Шайқұлова А., Аманжолова С., Асқарова Н. Бағдарламалық жасақтаманың қазіргі заманғы жабдықтары:** *Оқулық*. 2-басылым. – Астана: Фолиант, 2010. – 216 б.

ISBN 978-601-292-118-2

Кітапта жүйелік және инструментальды бағдарламалық жасақтамалардың теориялық негіздері мен әдістері туралы, жүйелік жоба құрылымы туралы мағлұмат беріледі.

Оқулыққа Delphi-дің кеңінен таралған визуальды бағдарламалау ортасында қарастырылған жұмыс енгізілген.

Оқулық орта кәсіптік білім беру орындарының оқытушылары мен оқушыларына ұсынылады.

УДК 004 (075.32)
ББК 32.973 я 722

ISBN 978-601-292-118-2

КІРІСПЕ

Оқулықтың мақсаты оқушыларды қазіргі заманғы бағдарламалық жасақтама жабдықтарының ішінде кеңінен қолданысқа ие болған Delphi ортасымен және Delphi-дегі Object Pascal бағдарламалау тілінің мүмкіндіктерімен таныстыру.

Кітапта жүйелік және инструментальды бағдарламалық жасақтамалардың теориялық негіздері мен әдістері туралы, жүйелік жоба құрылымы туралы мағлұмат беріледі. Сондай-ақ, бағдарламалық өнімдердің жіктелінуі және олардың жүйелік көрсетілуі, жүйелік талдау әдістері, бағдарламалық жасақтамаларды құрылымдық жобалаудың теориялық негіздері мен әдістері қарастырылады.

Бұл жинаққа Delphi-дің кеңінен таралған визуальды бағдарламалау ортасында қарастырылған жұмыс енгізілген. Delphi негізінде қосымшаларды жылдам құру концепциясы (RAD – Rapid Application Development) жатқандықтан, жинақта Windows-та жұмыс істейтін қосымшаларды құруға арналған визуальды компоненттерді, формаларды, оқиғаларды, қасиеттерді пайдалануға көп көңіл бөлінген. Сонымен қатар жиі қолданылатын визуальды компоненттерді пайдаланып, Delphi ортасында Паскаль тілінің операторлары көмегімен бағдарламалауды үйрену мәселелері қамтылған.

Оқулықта берілген жұмыстарды орындау барысында оқушылар Windows 9x, Windows NT-де бағдарламалаудың жаңа мүмкіндіктерімен танысып, динамикалық массивтер, файлдар, параллельді ағындармен жұмыс істеу дағдыларын қалыптастырады. Әрбір жұмыстың сипаттамасында параметрлері мен қолданылатын іс-әре-

кеттері толық сипатталған тілдің операторлары келтірілген.

Оқулық Орта кәсіптік білім беретін оқу орындарындағы қазақ бөлімдерінің оқушылары үшін бағдарламалық жабдықтарды жобалауда көмекші құрал болмақ.

I ТАРАУ

DELPHI ЖӘНЕ ОНЫҢ НЕГІЗГІ ЖҰМЫС ПРИНЦИПТЕРІ

1.1. Borland Delphi

Жаңа бағдарламалау жүйесінің негізгі өзгерістері ретінде төмендегілерді атап көрсетуге болады.

• Object Pascal жаңа бағдарламалау тілі, Borland Pascal тілінің бұрынғы нұсқасының өңделген түрі болып табылады.

• RAD (Rapid Application Development) өңдеу технологиясына бағытталған бағдарламалау ортасының компонентті модулі.

Object Pascal бағдарламалау тілі өңдеу құралдарының нарығында C++, Java сияқты объектілік-бағдарланған тілдермен қатар пайда болды. Borland компаниясы объектілік-бағдарланған тілдерді дамытып, Object Pascal тілін құруда өз тәжірибелерінің жетістігін көрсете білді. Жаңа тіл синтаксис жағынан да, ұсынылатын мүмкіндіктер жағынан да сәтті шықты. Бұл тіл объектілік-бағдарланған бағдарламалаудың (объектно-ориентированное программирование) барлық негізгі механизмдерін қамтыды.

Borland Delphi бағдарламалау жүйесі Windows ОЖ-де орындалатын нәтижелеуші бағдарламаларды құруға арналған.

Borland Delphi бағдарламалау жүйесінің және оның компоненттік үлгісінің негізін VCL (Visual component library) кіхапханасы құрайды. Бұл кітапханада опера-

циялық жүйе интерфейсі мен басқаруының негізгі орындары компонент түрінде жүзеге асырылған. Сондай-ақ, мұның құрамына «клиент-сервер» құрылымындағы нәтижелеуші бағдарламаларды өңдеу үшін Borland Delphi құрамындағы BDE (Borland database engine) тәсілі еніп отыр. Енді ол нәтижелеуші бағдарламалар VCL кітапханаларының көмегімен ДҚ серверлер диапазонына шығу мүмкіндігін қамтамасыз етеді. BDE көмегімен нәтижелеуші бағдарламалар Microsoft SQL, Server, Interbase, Sybase, Oracle және тағы басқа типті ДҚ серверлермен өзара жұмыс жасайды.

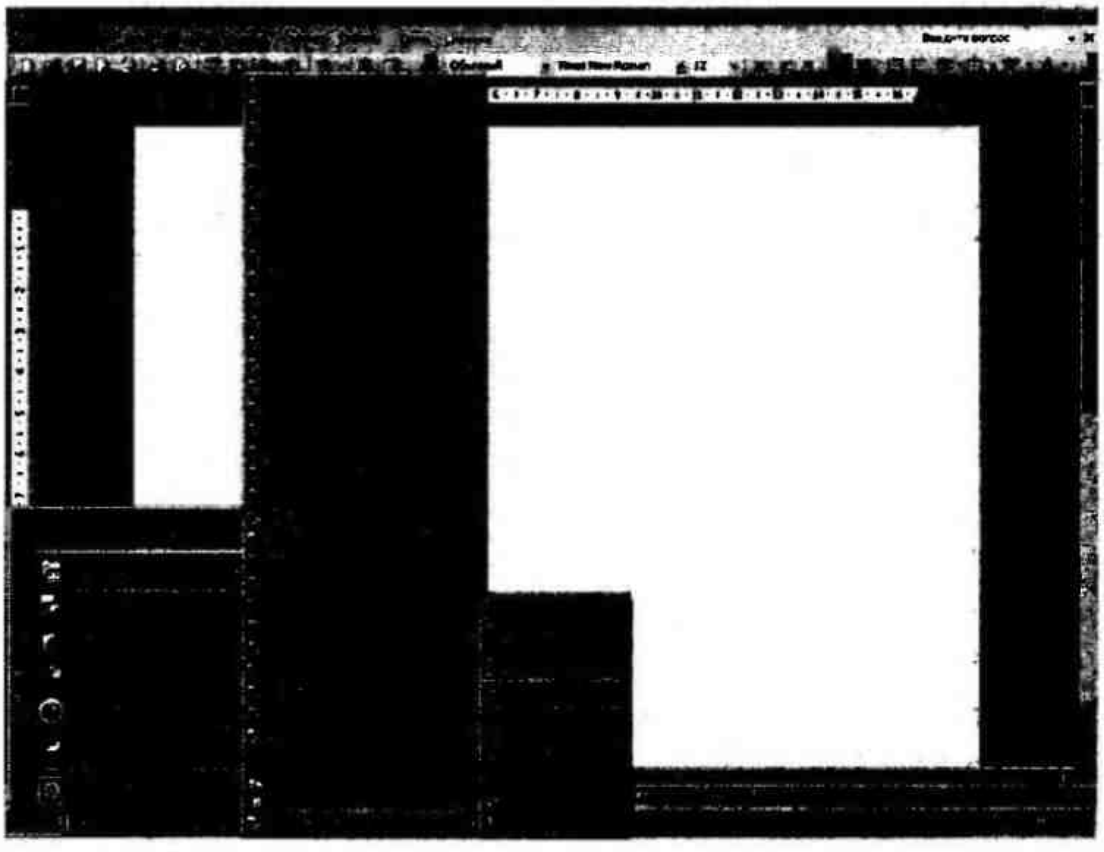
Borland Delphi бағдарламалау жүйесі бірнеше рет жаңарып, қолданушыларға бірнеше жаңартылған нобайларды ұсынды. Жүйенің соңғы нобайында (4,5 нобайлары) қосымшаның үш деңгейлі құрылымдық нәтижелеуші бағдарламаларды өңдеу тәсілдерінің жинағы қамтылған. Borland Delphi бағдарламалау жүйесі берілген құрылымның серверлік, сондай-ақ клиенттік бөлімін өзгертуге мүмкіндік береді. Онда COM/DCOM технологияларын (Mic.Win типті ОЖ көп тараған), сонымен бірге COBCA технологиясын (тек қосымшаның клиенттік бөлімін өңдегенде ғана) қолдануға болады.

Delphi-ге өте қарапайым (1-сурет) түрде Borland Delphi 7 менюінен Delphi 7 командасын таңдау арқылы кіруге болады.

1.2. Delphi-мен жұмысты бастау

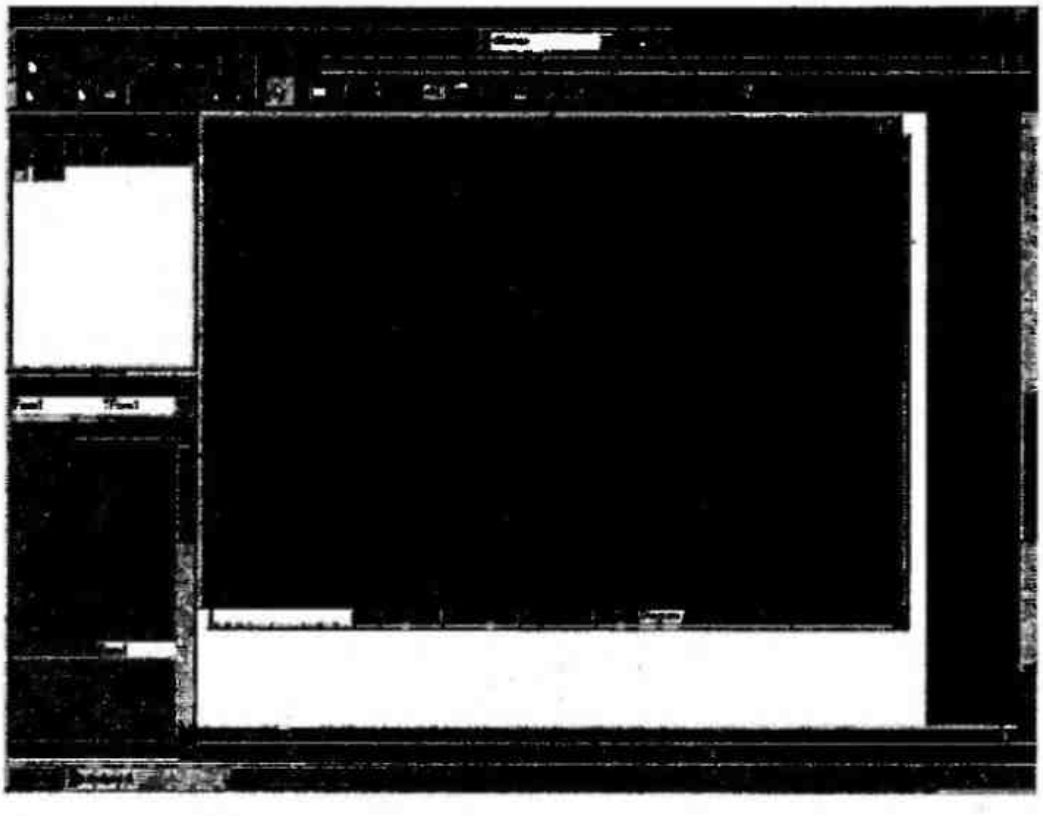
Экрандағы Delphi көрінісі өзгешелеу (2-сурет). Экранда бір терезенің орнына 5 терезе пайда болады:

- Басты терезе **(Delphi 6)**;
- Стартты форма терезесі **(Form 1)**;
- Объектілер қасиеттері редакторының терезесі **(Object Inspector** (объектілер инспекторы);
- Объектілер тізімін қарау терезесі **(Object TreeView** объектілер бұтағын құру);
- Код редакторы терезесі **(Unit 1. pas)**.



1-сурет. Delphi-ді жүктеу

Код редакторының терезесі стартты форма терезесімен толық жабылып тұр.



2-сурет. Экрандағы Delphi көрінісі

Басты терезеде (3-сурет) меню командалары, инструменттер панелі және компоненттер палитрасы орналасқан.

3-сурет. Басты терезе

Ескерту: Бағдарламалық жасақтаманың жүйелік бағдарламалық жасақтама және қолданбалы бағдарламалық жасақтама болып бөлінетіні белгілі. Жүйелік бағдарламалық жасақтамаға – барлық операциялық жүйелер кірсе, ал қалған бағдарламалар қолданбалы бағдарламалық жасақтамаға жатады. Қолданбалы бағдарламалар қысқаша қосымшалар деп аталады.

Object Inspector терезесі – объектілер қасиеттерінің мәндерін редакциялауға арналған. Визуальды жобалау терминологиясында объект дегеніміз, бұл – сұхбаттық терезе және басқару элементтері (енгізу/шығару өрісі, командалық батырмалар, ауыстырып-қосқыштар және т.б).

Объект қасиеттері – объект түрін сипаттайтын, орналасу жағдайларын көрсететін сипаттамалар.

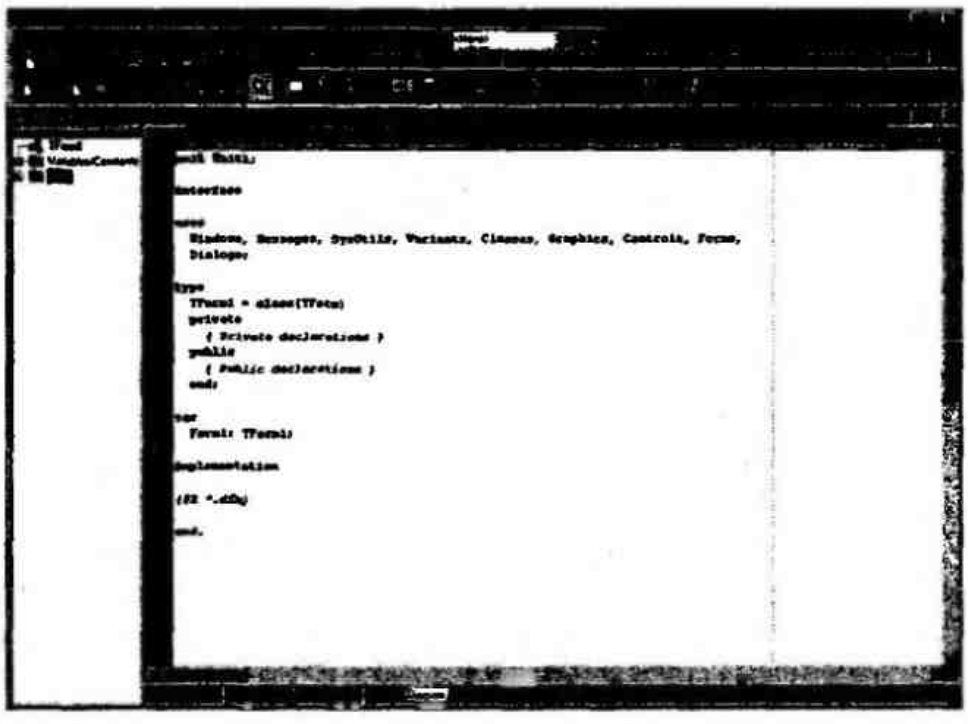
Мысалы, **width** – форма енінің өлшемі;

Heiqht – форма биіктігінің өлшемі;

Top және **left** – форманың экранға орналасуы.

Caption – тақырып аты.

4-сурет. Объект қасиеттері редакторының терезесі



5-сурет. Код редакторының терезесі

Код редакторы терезесінде (5-сурет) форма терезесін бір шетке ысырып қойып, бағдарлама мәтінін жаза беруге болады. Жаңа жобамен жұмысты бастауда код редакторы терезесі Delphi бағдарламасының үлгісін (шаблонын) қабылдайды.

1.3. Бірінші жоба

Delphi мүмкіндіктерін және визуальды жобалау технологиясын көрсету үшін спортшының белгілі бір аралықты жүгіріп өту жылдамдығын есептейтін бағдарлама құрастырамыз.

6-сурет. Жүгіру жылдамдығын есептеу бағдарламасының терезесі

Жаңа бағдарламамен жұмысты бастамас бұрын Delphi-ді жүктеңіз. Егер сіз ол ортада жұмыс істеп отырған болсаңыз және сізде басқа жоба таңдалған болса, онда **File** (файл) менюінен **New/Application** (құру/қосымша) командасын таңдаңыз.

1.3.1.Форма

Delphi-де жаңа жобамен жұмыс жасау стартты форманы құрудан басталады.

Бағдарламаны өңдеу кезеңіндегі сұхбаттық терезе **форма** деп аталады.

Форманың және оның компоненттерінің қасиеттік мәндерін өзгерту және қарау үшін **Object Inspector** терезесі қолданылады.

Форма қасиеттері:

Name – форма аты.

Caption – тақырып мәтіні.

Width – форма ені.

Height – форма биіктігі.

Top – форманың жоғарғы шекарасынан экранның жоғары шекарасына дейінгі арақашықтық.

Left – форманың сол жақ шекарасынан экранның сол жақ шекарасына дейінгі қашықтық.

BorderStyle – шекара түрі (қарапайым, жіңішке болуы да, шекараның болмауы да мүмкін).

BorderIcons – терезені басқару батырмасы.

Icon – сұхбаттық терезенің бас жағындағы жүйелік менюді шығару батырмасын белгілейтін белгі.

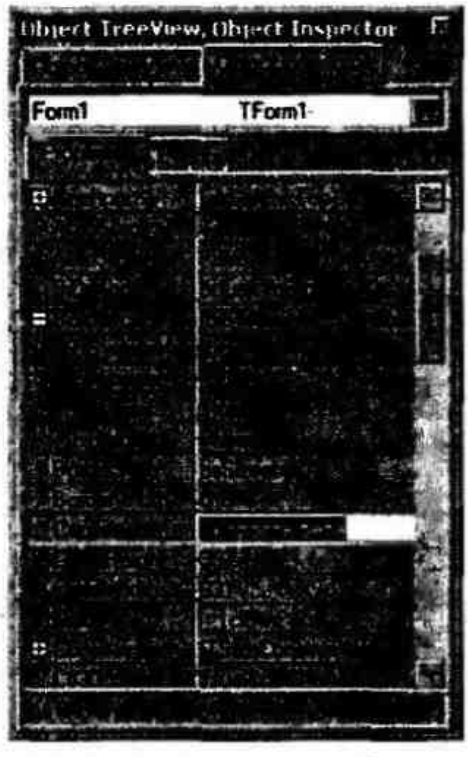
Color – өң түсі.

Font – қаріп.

Форма көлемін меңзерді (курсорды) қойып, тінтуірді (мышь) жылжыту арқылы өзгертуге де болады.

Форма құру үшін, алдымен қасиет мәнін **Caption (тақырып)** өзгертуге тура келеді. Біздің мысалымызда **Form1** мәтінін «жүгіру жылдамдығына» ауыстыру қажет. Ол үшін **Object Inspector** терезесіндегі **Caption** жолына тінтуір батырмасын әкеп шерту керек, нәтижесін-

де 7-ші суреттегідей меңзер сызығы пайда болып, «жүгіру жылдамдығына» мәтін енгізуге және қасиет мәндерін анықтауға болады. Бұл қасиеттерге 250 және 330 мәндерін меншіктейміз.



7-сурет. Caption қасиет мәнінің өзгеруі

Ескертпе! Экрандағы форманың орналасу тәртібі мен өлшемі және тағы да басқа басқару элементтері өлшемдері пиксель, яғни экранда нүкте түрінде көрсетіледі.

Форма – бұл қарапайым терезе. Сондықтан оның көлемін басқа терезелердей тінтуір батырмасын басу арқылы шекарасынан ұстап алып жылжытып немесе орынын ауыстырып өзгерте аламыз. Өзгертіп болысымен автоматты түрде **Height** және **Width** қасиеттері мәндерінің өзгеріп шыға келгенін байқаймыз. Олар форманың орнатылған өлшеміне сәйкес болмақ. Бағдарламаны жүктегеннен кейінгі экрандағы сұхбаттық терезенің орналасуы форманың өңделгеннен кейінгі жағдайына сәйкес келеді, бұл **Top** мәнінің қасиетімен (экранның жоғарғы шекарасынан шегініс) және **Left** (экранның сол жақ шекарасынан шегініс) қасиетімен анықталады. Бұл қасиеттердің мәндерін сонымен қатар форма терезесін тінтуірдің көмегімен ауыстыру арқылы беруге де болады.

Кейбір қасиеттерді таңдағанда, мысалы: **BorderStyle**, қасиеттің ағымдағы мәнінің оң жағында тізімді ашатын белгі пайда болады. Мұндай қасиеттер мәндерін тізімнен таңдау жолымен беруге болады (8-сурет). Кейбір қасиеттері күрделі болады, яғни олардың қасиеттері басқа мәндер жиынтығының қасиеттерімен анықталады. Күрделі қасиет мәндерінің аттарының алдында «+» белгісі тұрады, бұл белгіні басқан кезде қасиеттерді анықтайтын тізім ашылады (9-сурет).

Мысалы: **BorderIcons** қасиеті бағдарламамен жұмыс кезінде терезелерді қандай пернелердің басқаратынын анықтайды. Егер **biMaximize** қасиетіне **False** мәнін меншіктесек, онда бағдарлама жұмысы кезінде «ашу» (жаю) батырмасы терезе басында көрінбейтін болады.

8-сурет. Тізімнен таңдау қасиеттер әрекеттері мәндерін орнату

9-сурет. BorderIcons қасиеттер инспекторынан күрделі тізімді ашу

Кейбір қасиеттер мәндерінің қатарынан үш нүктелі командалық батырмаларды көруге болады. Бұл деген сөз, қасиеттер мәндерін беру үшін қосымша сұхбаттық терезелерді пайдалануға болады деген ұғым.

1-кестеде өңделетін бағдарламаға сәйкес өзгертуге болатын форма қасиеттері берілген. Қалған қасиеттер өзгеріссіз қалдырылған және кестеде келтірілмеген.

1-кесте

Қасиеті	Мәні
Caption	Жүгіру жылдамдығы
Height	250
Width	330
BorderStyle	BsSingle
BorderIcons. biMinimize	False
Border Icons. biMaximize	False
Font.Size	10

Кестеде келтірілген кейбір қасиеттердің атауларында нүкте қойылған, бұл нақтылау қасиеттерінің мәнін беруді білдіреді. Басты формаға сәйкес қасиеттер мәндері орнатылып болғаннан кейін, ол мынадай түрге ие болады (10-сурет).

10-сурет. Қасиет мәнін анықтағаннан кейінгі форма көрінісі

1.3.2. Компоненттер

Жүгіру жылдамдығын есептеу бағдарламасына қолданушы мынадай бастапқы деректерді енгізу керек: арақашықтық ұзындығы және уақыт. Мұндай бағдарламаларда деректер кілтжиыннан редакциялау өрісіне енгізіледі: сондықтан да **Edit** редакциялау өрісінің компонентін формаға қою қажет.

Көп пайдаланылатын компоненттер **Standart** салымында орналасқан.

Формаға компонент қосу үшін компоненттер палитрасынан сол компонентті таңдап, оның пиктограммасында тінтуірдің сол жақ пернесін басамыз. Әрмен қарай тінтуір көрсеткішін (указатель мыши) форманың сол жақ жоғарғы бұрышына орналастырып, тағы бір рет шерту керек. Нәтижесінде формада стандартты өлшемді компонент пайда болады.

Компонент өлшемін формаға қосылу барысында беруімізге болады. Ол үш палитрадан компонентті таңдап алғаннан соң тінтуір көрсеткішін форма компонентінің сол жақ жоғарғы бұрышына басып, тінтуір батырмасынан қолды алмастан компоненттің оң жақ төменгі бұрышындағы нүктеге апарып батырманы жіберу қажет. Сонда формада керекті өлшемді компонент пайда болады.

Delphi-дегі әрбір компонентке ат беріледі, ол өз кезегінде компонент атынан және оның реттік нөмірінен тұрады (компонент аты мен қатар санынан тұрады). Мысалы, егер формаға екі **Edit** компонентін қосса, олардың аттары **Edit1** және **Edit2** болып белгіленеді. Қолданушы **Name** қасиетінің мәнін өзгерту арқылы компонент атын өзгерте алады. Қарапайым бағдарламаларда компоненттің аттары өзгертілмейді.

11-сурет. Edit-тің екі компоненті қосылған кейінгі форма түрі

11-суретте **Edit** редакциялау өрісінің екі компоненті (бастапқы деректерді енгізуге арналған) қосылғаннан кейінгі форма түрі берілген. Компоненттердің бірі ерек-

ше белгімен белгіленген. Белгіленген компонент қасиеті **Object Inspector** терезесінде көрсетіледі. Басқа компонент қасиетін көргіңіз келсе, тінтуірдің сол жақ батырмасын компонентке апарып шерту керек. Компонент-тер аттарын **Object TreeView** терезесінде немесе **Object Inspector** терезесінен таңдауға болады.

2-кесте

Edit – редакциялау өрісі компонентінің негізгі қасиеттері көрсетілген кесте

Name	Компонент аты. Бағдарламада компонентке және оның қасиеттеріне, жекелеген жағдайларда редакциялау өрісінде енгізілген мәтіндерге қатынас құру үшін қолданылады.
Text	Енгізу және редакциялау өрісіндегі мәтін.
Left	Компоненттің сол жақтағы шекарасынан форманың сол жақтағы шекарасына дейінгі аралық.
Top	Компоненттің жоғарғы шекарасынан форманың жоғарғы шекарасына дейінгі арақашықтық.
Height	Өріс биіктігі.
Width	Өріс ені.
Font	Енгізілген мәтін қарпі.
ParentFont	Егер қасиеттер мәні True болса, Font қасиеті өзгергенде Font компонентінің қасиеттік мәні автоматты түрде өзгереді.

Компоненттің орналасуын және өлшемін тінтуір арқылы өзгертуге болады.

Компоненттің жайын өзгерту үшін, тінтуір көрсеткішін оның көрінісіне орнату керек, тінтуірдің сол жақ батырмасын басып тұрып, компоненттің контурын форманың қажетті нүктесіне орнатуға болады. Сол кезде **Left** және **Top** қасиеттерінің мәндері (компоненттің жоғарғы сол жақ бұрышының координатасы) көрсетіледі.

Компоненттің өлшемін өзгерту үшін, оны белгілеп алу керек, тінтуірді маркердің үстіне апарып, сол жақ батырмасын басу арқылы компонент шекарасын өзгертуге болады. Нәтижесінде **Height** және **Width** қасиеттерінің ағымдағы мәндері көрсетіледі.

Object Inspector арқылы компоненттің қасиетін және форма қасиетін өзгертуге болады. **Object Inspector** терезесінде керекті компоненттің қасиеті пайда болуы үшін,

бұл компонентті белгілеп алу керек (тінтуірді оның бейнесіне әкеліп басу қажет). Сондай-ақ, **Object Inspector** терезесінің жоғарғы жағындағы құлап ашылатын тізімнен (Object Inspector) немесе **Object TreeView** терезесінен компонентті таңдауға болады.

12-сурет. Компонент жағдайын өзгерткендегі Left және Top қасиеттерінің ағымдық мәндер көрінісі

Компонент өлшемдерін өзгерткенде **Height** және **Width** қасиеттерінің ағымдық мәні өзгеріп шығады.

13-сурет. Object TreeView терезесінен компонент таңдау

Edit1 және **Edit2** қасиеттерінің мағыналары 3-кестеде көрсетілген.

Edit1 компоненті – арақашықтық ұзындығын енгізуге арналған.

Edit2 компоненті – уақытты енгізу үшін белгіленген. Екі компоненттің де **Text** қасиетінің мәні бос жол болып қалады.

3-кесте

Name	Edit1	Edit1
Text		
Top	56	88
Left	128	128
Height	21	21
Width	121	121

Бағдарламаның терезесінде редакциялау өрісінен басқа, бағдарлама туралы және енгізу өрісінің қызметі туралы қысқаша мәлімет болу керек. Формаға мәтінді шығару үшін ***шығару өрісі*** пайдаланылады. Мәтінді шығаратын өріс – бұл **Label** компоненті. **Label** компонентінің белгісі **Standart** салымында орналасқан. **Label** компоненті формаға редакциялау өрісі сияқты қосылады.

Өңделіп жатқан формаға **Label** -дің төрт компонентін қосу қажет.

Бірінші өріс – ақпараттық хабар шығаруға арналған, екінші және үшінші өрістер – енгізу өрістерінің қызметтері туралы мәлімет шығаруға арналған, төртінші өріс – жылдамдық туралы нәтижені шығару үшін белгіленген.

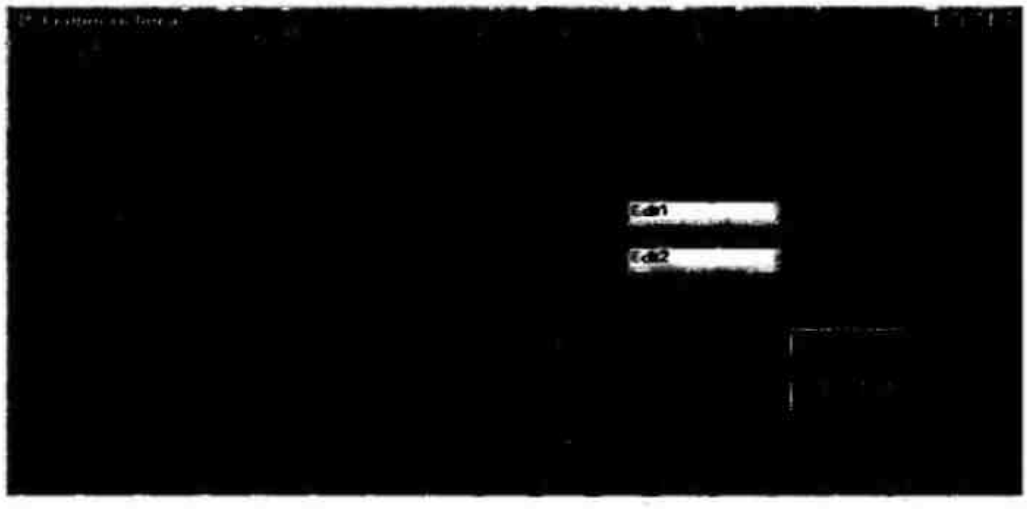
4-кесте

Label компонентінің қасиеттері

Қасиет	Түсінік
1	2
Name	Компонент аты. Бағдарламада компонентке және оның қасиеттеріне қатынас құру үшін белгіленген.
Caption	Көрсетілетін мәтін.
Font	Мәтінді көрсетуге арналған қаріп.

1	2
ParentFont	Егер қасиет мәні True болса, онда мәтін форма үшін орнатылған қаріппен шығарылады.
AutoSize	Өріс өлшемінің оның құрамындағылармен анықталу белгісі.
Left	Шығару өрісінің сол жақ шекарасынан форманың сол жақ шекарасына дейінгі арақашықтық.
Top	Шығару өрісінің үстіңгі шекарасынан форманың үстіңгі шекарасына дейінгі арақашықтық.
Height	Шығару өрісінің биіктігі.
Width	Шығару өрісінің ені.
WordWrap	Жолға сыймаған сөздер келесі жолға автоматты түрде көшеді.

AutoSize және **WordWrap** қасиеттеріне назар аударыңыз. Егер де шығару өрісі бірнеше қатарды қамтитын болса, осы қасиеттерді пайдаланған жөн. **Label** компонентін формаға қосқаннан кейін **AutoSize** қасиетінің мағынасы *True* мәніне теңеледі, яғни өріс өлшемі **Caption** қасиетінің мәнін өзгерткен кезде автоматты түрде өзгереді деген сөз. Егер де шығару өрісіндегі мәтін бірнеше жолды қамтысын десеңіз, онда **Label** компонентіне қосылғаннан кейін **AutoSize** қасиетіне *False* мәнін, **WordWrap** қасиетіне – *True* мәнін беру керек. Сонан кейін **Width** және **Height** қасиеттерінің мағыналарын өзгертуге қажетті өлшемді беру қажет. Осы әрекеттерден кейін ғана **Caption** қасиетіне өріске шығатын мәтіндерді енгізуге болады.



14-сурет. Мәтін шығару өрісі қосылғаннан кейінгі форма түрі

5-кесте

Қасиеті	Компонент			
Name	Label1	Label2	Label3	Label4
AutoSize	False	True	True	False
WordWrap	True	False	False	True
Caption	Бағдарлама спортшы жүгіріп өткен арақашықтық жылдамдығын есептейді.	Арақашықтық (метрмен)	Уақыт (мин, секунд)	
Top	8	56	88	120
Left	8	8	8	8
Height	33	13	13	41
Width	209	102	832	273

Форма құру кезінде – формаға **Есептеу (Вычислить)** және **Аяқтау (Завершить)** командалық екі батырмаларын қосуымыз керек. **Button** командалық компоненті формаға басқа компоненттер секілді қосыла бастайды. **Button** компонентінің белгісі **Standart** салымында орналасқан.

6-кесте

Button компонентінің қасиеттері

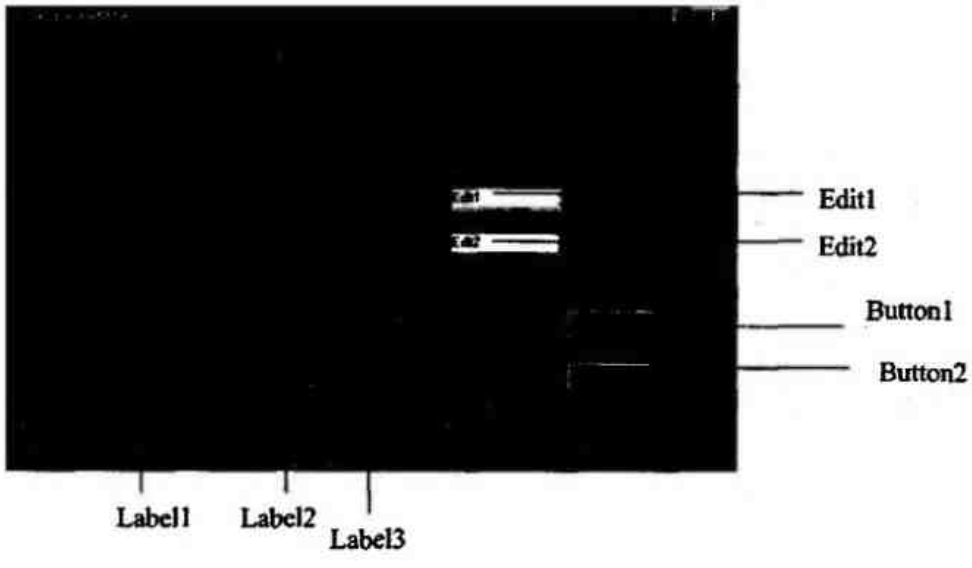
Қасиет	Баяндалуы
Name	Компонент аты. Бағдарламада компонентке және оның қасиеттеріне қол жеткізу үшін қолданылады.
Caption	Батырмадағы мәтін
Enabled	Қасиетінің мәні True болса пернені қолдануға болады, ал қасиет мәні False болса қолдануға болмайды.
Left	Батырманың сол жақ шекарасынан форманың сол жақ шекарасына дейінгі арақашықтық.
Top	Батырманың жоғарғы шекарасынан форманың жоғарғы шекарасына дейінгі арақашықтық.
Height	Батырма биіктігі
Width	Батырма ені

Формаға екі командалық батырма қосылғаннан кейін 7-ші кестедегі қасиет мәндерінің сәйкестігін анықтауымыз керек.

7-кесте

Name	Button1	Button2
Caption	Есептеу	Аяқтау
Top	176	176
Left	16	112
Height	25	25
Width	75	75

Соңғы форма түрі 15-суретте келтірілген.



15-сурет. Жүгіру жылдамдығы бағдарламасының формасы

Форма қосымшасын құруды аяқтаған соң бағдарлама мәтінін жазуға кірісеміз. Алдымен бағдарламалау үшін өте қажет түсініктерді талдауымыз керек:

– оқиға;

– оқиғаны өңдеу процедурасы.

1.3.3. Оқиға және оқиғаны өңдеу процедурасы

Құрылған форманың түрі қосымшаның қалай жұмыс істейтіндігін көрсетеді. Қолданушы редакциялау өрісіне

(поля) бастапқы мәліметтерді енгізу керек және сол есептің шығу жолына байланысты **Есептеу** (Вычислить) батырмасын басу керек. Осы көрсетілген командалық батырманы шерту – Windows-та *Оқиға* (событие) деп аталады.

Оқиға (Event) – бағдарлама жұмыс істеп жатқан уақыт аралығында орындалады.

Delphi-де әрбір оқиғаға арнайы ат беріледі.

Мысалы, тінтуір (мышь) батырмасын бір рет шерту – *Onclick* оқиғасы, ал тінтуір батырмасын екі рет шерту *OnDbClick* оқиғасы деп аталады.

8-кесте

Windows-тің кейбір оқиғалары

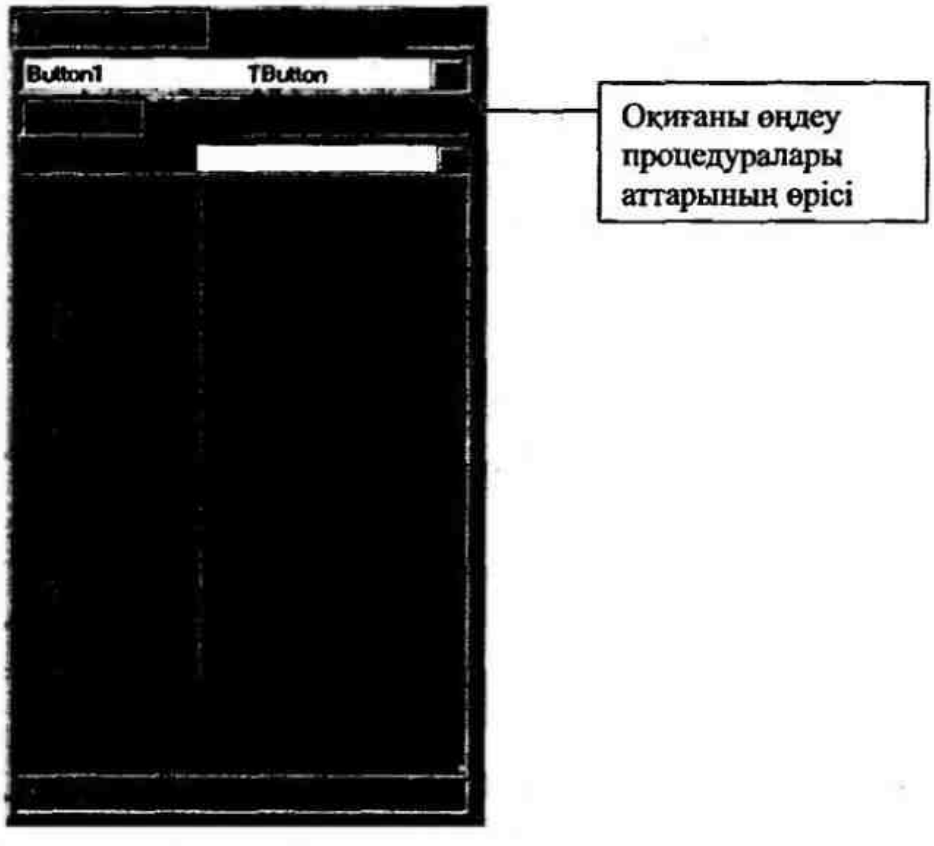
Оқиға	Жүзеге асырылады
OnClick	Тінтуір батырмасын басқан кезде
OnDblClick	Тінтуір батырмасын екі рет басқан кезде
OnMouseDown	Тінтуір батырмасын басқанда
OnMouseUp	Тінтуір батырмасын босатқанда
OnMouseMove	Тінтуір батырмасының орынын ауыстырғанда
OnKeyPress	Кілтжиын пернесін басқанда
OnKeyDown	Кілтжиын пернесін басуда. OnKeyDown және OnKeyPress кезектесіп қайталанатын оқиғалар.
OnKeyUp	Кілтжиын пернесін босатқанда
OnCreate	Объектіні құру кезінде (форма, басқару элементі). Осы оқиғаны өңдеу процедурасы айнымалыларды анықтау үшін, дайындық әрекеттерін орындау үшін қолданылады.
OnPaint	Бағдарлама жұмысының басында экранда терезе пайда болған кезде.
OnEnter	Фокустың басқару элементімен қабылдағанда
OnExit	Фокустың басқару элементімен жоғалтқанда

Delphi-де оқиғаға деген реакция *оқиғаны өңдеу процедурасы* сияқты жүзеге асырылады. Осылайша пайдаланушының әрекеттеріне сай бағдарлама кейбір қызметтерді атқаруы керек, ол үшін бағдарлама жасаушы

оқиғаға сәйкес өңдеу процедурасын жазуы қажет. Оқиғаны өңдеудің көпшілік бөлігін компонент атқаратынын ескеруіміз керек. Сондықтан бағдарлама жасаушы оқиғаны өңдеу процедурасын тек оқиға реакциясы стандарттан ерекше болса немесе анықталмаған жағдайда ғана өңдеуі керек. Мысалы, есептің шарты бойынша **Edit** өрісіне енгізілетін символдарға шек қойылмаса, онда **OnKeyPress** оқиғаны өңдеу процедурасын жазудың қажеті жоқ, өйткені бағдарлама жұмысы кезінде оқиғаны стандартты өңдеу процедурасы пайдаланылады.

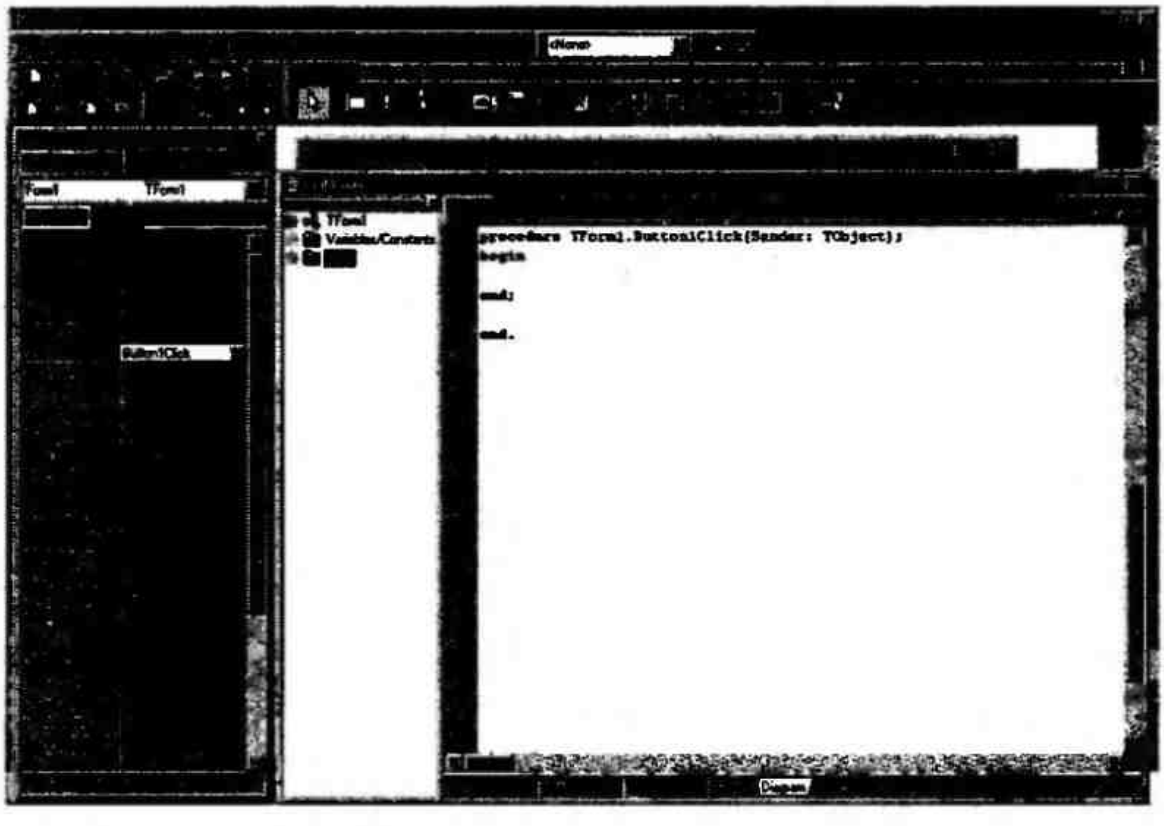
Оқиғаны өңдеу процедурасын құруға кірісу үшін, алдымен **Object Inspector** терезесінен компонент және осы терезеден **Events (оқиға)** салымын таңдау керек.

Events салымының сол жақ бағанасында (16-сурет) таңдалған компонент (объект) қабылдайтын оқиға аттары көрсетілген. Егер оқиғаны өңдеу процедурасы анықталса, онда оқиға атының жанында процедура аты оң жақ бағанада көрсетіледі.



16-сурет. Events салымына компонент қабылдай алатын оқиғалар тізімі жазылған

Оқиғаны өңдеу функциясын құру үшін тінтуір батырмасын оқиғаға сәйкес өңдеу процедурасы атына екі рет шерту керек. Нәтижесінде код редакторы терезесі ашылып, онда оқиғаны өңдеу функциясының шаблоны қосылады. Ал **Object Inspector** терезесіндегі оқиға атының жанында оны өңдеу функциясының аты пайда болады. Delphi оқиғаны өңдеу функциясына 2 бөліктен тұратын ат беріледі. Атаудың бірінші бөлігі объектіні (компонентті) құрайтын формаға теңестіріледі. Екінші бөлігі объект немесе оқиға атына теңестіріледі. Біздің мысалда форма аты – **Form1**, командалық батырма аты – **Button1**, ал оқиға аты – **Click** болып беріледі.



17-сурет. Delphi-де генерацияланған оқиғаны өңдеу процедурасына мысал

Код редакторы терезесіндегі *begin* және *end* сөздерінің арасына оқиғаны өңдеу функциясын жүзеге асыруға қажетті нұсқауларды жазуға болады. Төмендегі бағдарламада **Есептеу** командалық батырмасы үшін **OnClick** оқиғаны өңдеу функциясының мәтіні келтірілген. Бағдарламаның қалай көрсетілгеніне назар аударыңыздар.

Код редакторы терезесінде басты сөздердің жалпы көрінісі – қалың қаріппен, түсініктемелер – курсивті қаріппен берілген. Бағдарламалау ережелеріне сай бағдарлама нұсқаулары шегініс арқылы бөлек-бөлек жіктеліп жазылған. Бұл әдіс бағдарламаны түсінуге ыңғайлы болмақ.

```
//Есептеу пернесін басу.

Procedure TForm1. Button1Click (Sender: Tobject);
Var
dist: integer; // арақашықтық, метр
 t: real; // уақыт, бөлшек сан

min: integer; // уақыт, минут
 sek: integer; // уақыт, секунд

 v: real; // жылдамдық

begin
// енгізу өрісінен бастапқы мәндерді алу
dist:=StrToInt (Edit1. Text);
t:=StrToFloat (Edit2. Text);

// алдын-ала түрлендіру
min:=Trunc(t); // минут саны – бұл t санының бүтін бөлігі
sek:=Trunc(t*100) mod 100; // секунд саны – t санының бөлшек бөлігі

//есептеу
v:=(dist/1000) / (min*60+sek) / 3600);

// нәтиже шығару
label4. Caption:= 'Арақашықтық: '+ Edit1.Text + 'м' +
#13 +
'Уақыт: ' + IntToStr(min)+' мин ' +
 IntToStr(sek) + ' сек '+ #13 +
```

```
'Жылдамдық: '+ FloatToStrF(v,ffFixed,4,2)+
' км/сағ';
end;
```

Button1Click функциясы жылдамдықты есептеп, нәтижесін **label4** өрісіне шығарады. Бастапқы мәндер **Edit1** және **Edit2** редакциялау өрісінен енгізіледі. *Text* қасиеті бағдарламаның жұмысы кезінде қолданушы енгізетін символдар қатарын қамтиды. Бағдарлама дұрыс жұмыс істеу үшін бұл қатарда тек сандар болуы керек. Қатар мәндері түгел сандарға ауысу үшін бағдарламада *StrToInt* және StrToFloat функциялары қолданылады. Бұл функциялардың атқаратын қызметі қатарды санға түрлендіру болып табылады. *StrToInt* функциясы осы функцияға параметрлер ретінде берілген қатардағы символдарды тексереді (*Edit1.Text* – **Edit1** өрісінің мәні), егер барлық символдар дұрыс болса, онда сәйкес санды қайтарады. Бұл сан dist айнымалысына жазылады. Дәл осылайша StrToFloat функциясы жұмыс жасайды. Ол **Edit2** өрісінің құрамындағыларына сәйкес бөлшек санды қайтарады. Бұл сан *t* айнымалысына келіп жазылады.

Dist және *t* айнымалыларына бастапқы мәндер жазылғаннан кейін дайындық жұмыстары және есептеулер жүргізіледі.

Алдымен *Trunc* функциясын қолдану арқылы санның бөлшек бөлігі «лақтырылып тасталады», *t* айнымалысының бүтін бөлігі белгіленеді, бұл – минут саны. *Trunc(t*100) mod 100* өрнегінің мәні секундтар саны болмақ. Бұл өрнек былайша есептеледі.

Бірінші *t* саны 100-ге көбейтіледі. *t* -айнымалысын 100-ге көбейткендегі шыққан нәтиженің бүтін бөлігі *Trunc* функциясы арқылы қайтарылады. Осылайша алынған сан 100-ге модуль бойынша бөлінеді. Модуль бойынша бөлу нәтижесі – осы бөлуден қалған қалдық болмақ.

Барлық деректер дайын болған соң есептеу басталады.

Жылдамдық *км/сағатпен* көрсетілетіндіктен, *метр* мен *минут* арқылы берілетін уақыт пен арақашықтық өлшемдері *километрге* және *сағатқа* түрлендіріледі.

Есептелген жылдамдық мәні *Caption* қасиетіне мәндерді меншіктеу жолымен **label4** өрісінде шығарылатын болады. Қатардағы сандарды түрлендіру үшін *IntToStr* және *FloatToStr* функциясы қолданылады. Аяқтау батырмасын басу нәтижесінде бағдарлама жұмысты тоқтатады, ол үшін экраннан бағдарламаның басты терезесін жауып, алып тастау керек. Ол **Close** көмегімен жүзеге асырылады. Аяқтау батырмасы үшін **onClick** оқиғасын өңдеу төмендегі бағдарламада берілген:

```
// Аяқтау батырмасын басу
Procedure TForm1.Button2Click (Sender: Tobject);
begin
Form1.Close // бағдарламаның басты терезесін жабу
End;
```

1.4. Код редакторы

Код редакторы бағдарламалау тіліндегі кілттік сөздерді (Procedure, var, begin, end, if және тағы басқалар) қалың қаріппен белгілейді. Бұл бағдарлама құрылымын түсінікті түрде беру үшін қолданылады. Кілттік сөздермен қатар, код редакторы түсініктемелерді курсивті қаріппен белгілеуге де мүмкіндік береді.

1.4.1. Түсініктеме жүйесі

Мәтінді теру кезінде редактор процедуралар мен функциялар және объектілердің қасиеттері мен тәсілдері жөнінде анықтамалық ақпараттар шығарып береді. Мысалы, егер біз редактор терезесінде **ShowMessage** деп жазсақ, онда экран бетінде **ShowMessage** функциясының параметрлері көрсетілген терезе шығады. Параметрлердің бірі қалың қаріппен белгіленеді, міне, осылай редактор бағдарламашыға қандай параметрді енгізу керектігін білдіретін түсінік береді. Параметрлер мен үтірді енгізгеннен соң көмекші терезеде келесі параметр

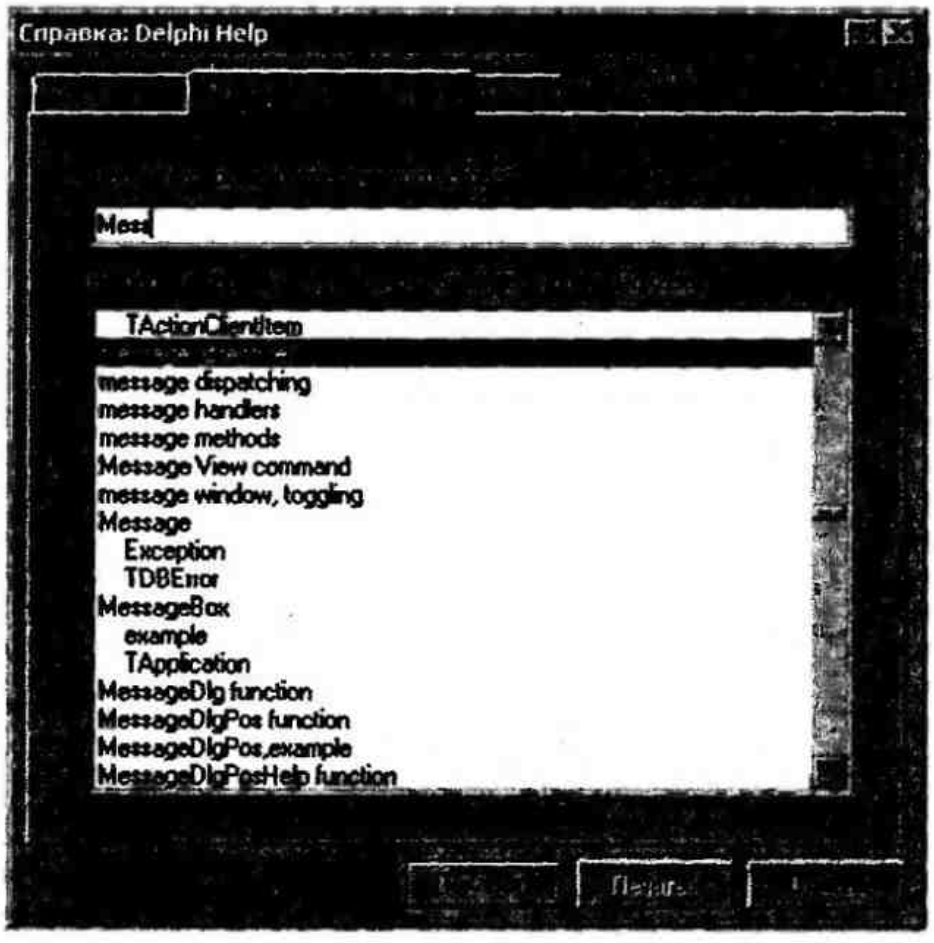
белгіленіп тұрады. Осылайша барлық параметрлер толық көрсетіліп болғанға дейін осы әрекеттер орындала бермек. Код редакторы объектілері үшін әдістер мен қасиеттер тізімі шығарылады. Бағдарламашы объект (компонент) аты мен нүктені жазып болысымен, көмекші терезеде осы объектінің әдістері мен қасиеттер тізімі пайда болады. Керекті элементке өту үшін меңзердің (курсордың) орнын ауыстыру қажет немесе кілтжиыннан *қасиет* немесе әдістің алдыңғы бірнеше әріптерін терсе жеткілікті. Тізімнің керек элементі таңдалып **<Enter>** пернесін басқан соң ғана бағдарлама мәтініне керек қасиет немесе әдістер қосылады. Көмекші терезенің бағдарлама мәтінін дайындауда көп көмегі бар. Егер бағдарлама жазған уақытта көмекші терезе пайда болмаса, онда бағдарлама жасаушы қателесті (процедура немесе функция аты дұрыс жазылмаған) деген сөз.

1.4.2. Анықтама жүйесі

Бағдарламаны теру барысында тілдің құрылымы жөнінде немесе функциялар туралы анықтамалар алуға болады. Ол үшін код редакторы терезесінде анықтама алу керек сөзді теріп, сонан соң **«F1»** пернесін басу қажет.

Ақпараттық анықтаманы **Help** менюінен **Delphi Help** командасын таңдау арқылы алуға болады.

Бұл кезде экранда анықтамалық жүйенің стандартты терезесі пайда болады (18-сурет). Терезедегі «тақырыптық нұсқағыш» салымында анықтама алумен байланысты тақырыпты анықтайтын кілттік сөзді енгізіңіз. Кілттік сөз ретінде функцияның, процедуралардың, қасиеттердің немесе әдістер аттарының алғашқы бірнеше әрпін енгізсе жеткілікті.



18-сурет. Кілттік сөз бойынша анықтамалық ақпарат іздеу

1.5. Жоба құрылымы

Delphi жобасы – бағдарламалық бірлік жиынын (модульдерді) қамтиды.

Басты деп аталатын модульдердің бірі – бағдарламаның орындалуы басталатын нұсқауды қамтиды. Осы нұсқаудан бағдарламаның орындалуы басталады.

Қосымшаның басты модулі толықтай Delphi-де қалыптасады.

Басты модуль файлының кеңейтілуі .dpr болып белгіленеді. Қосымшаның басты модулінің мәтінін көру үшін **Project** менюінен **View Source** командасын таңдау қажет. Төмендегі бағдарлама жүгіру жылдамдығын есептеуге арналған.

Program *vrun;*
Uses

```
Forms,
Vrun1 in 'vrun1.pas' {Form1};

{$R*.res}

begin
Application. Initialize;
Application. CreateForm (TForm1, Form1);
Application.Run;
End.
```

Басты модуль **program** сөзінен басталады, онан соң жоба атына сәйкес болатын бағдарлама атын беру қажет.

Жоба аты жобаны сақтау кезінде беріледі де, ол компилятор арқылы құрылатын бағдарламаны – *орындамалық файл* атын анықтайды. Әрмен қарай **Uses** сөзінен кейін қолданылатын модульдер аты жазылады: **Forms** кітапханалық модулі және vrun1.pas формасының модулі.

Түсінікке ұқсас (**$R*.RES**) қатары – бұл директива, компиляторға ресурстар файлын қосады. Ресурстар файлы қосымшалар ресурстарынан тұрады: пиктограммалар, меңзерлер, биттік образдар және тағы басқалар. Жұлдызша белгісі – ресурстар файлының аты жоба файлымен бірдей екенін білдіреді, бірақ кеңейтілуі *.res*.

Ресурстар файлы мәтіндік файл емес, сондықтан оны мәтін редакторының көмегімен қарауға мүмкіндік жоқ. Ресурстар файлымен жұмыс үшін арнайы бағдарламалар қолданылады, мысалы, **Resourse Workshop**. Сонымен қатар Delphi құрамына кіретін **Image Editor** утилитін де пайдалануға болады, оған қатынас **Tools** менюіндегі **Image Editor** командасын таңдау арқылы ұйымдастырылады.

Басты модульдің орындалатын бөлігі *begin* және *end* нұсқауларының арасында орналасқан.

Орындамалық бөлім нұсқауы қосымшаны инициализациялауды және стартты терезені экранға шығаруды қамтамасыз етеді.

Басты модульмен қатар, әрбір бағдарлама өзінде ең болмағанда бір форма модулін қамтиды.

Онда қосымшаның стандартты формасына түсінік пен оның жұмысын қолдайтын процедуралар сақталады.

Delphi-дегі әрбір формаға өзіндік модуль сәйкес келеді.

Төменде жүгіріс жылдамдығын есептеу бағдарламасы модулінің мәтіні берілген.

```
Unit vrun1;
Interface

Uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;

Type
TForm1=class (TForm)
Edit1: TEdit;
Edit2: TEdit;
Label1:Tlabel;
Label2: Tlabel;
Label3: Tlabel;
Label4: Tlabel;
Button1: Tbutton;
Button2: Tbutton;
Procedure Button1Click (Sender: Tobject);
Procedure Button2Click (Sender: Tobject);

Private
   {Private declarations}
public
   {Public declarations}
end;

var

Form1: Tform1;
```

```
Implementation

{$R*.dfm}

//есептеу батырмасын басу
Procedure Tform1.Button1Click (Sender: Tobject);
var

dist: integer; //арақашықтық, метр
t: real; //уақыт, бөлшек, сандар түрінде

min: integer; //уақыт, минут
sek: integer; //уақыт, секунд
v:real; //жылдамдық

begin
 //енгізу өрісінен бастапқы деректерді алу
dist:= StrToInt(Edit1.Text);
t:= StrToFloat(Edit2.Text);

 //алдын ала түрлендіру
min:= Trunc(t); //минут саны – t санының бүтін бөлігі
sek:=Trunc(1*100) mod 100; // секунд саны – t санының бөлшек бөлігі

// есептеу
v:=(dist/1000) / (min*60+sek) / 3600;

// нәтиже шығару
Label4.Caption:= 'Дистанция' + Edit1. Text+ ' м' + #13 +
'Уақыт: ' + IntToStr(min) + ' мин ' +
 IntToStr(sek)+ ' сек' + #13 +
'Жылдамдық: ' + FloatToStrF(v,ffFixed,4,2) +
'км/сағ';
end;

//аяқтау батырмасын басу
procedure TForm1.Button2Click (Sender: Tobject);
```

```
begin
Form1.Close;

end;
end.
```

Модуль *unit* сөзінен басталады да, одан кейін модуль аты беріледі.

Осы ат жоғарыдағы бағдарламада мәтіні келтірілген, қосымшаның басты модуліне қатысты uses нұсқауында қолданылатын модульдер тізімінде ескеріледі.

Модуль мынандай бөлімдерден тұрады:

– интерфейс;

– жүзеге асыру;

– инициализациялау;

Интерфейс бөлімі (**Interface** сөзінен басталады) бағдарламаның басқа модульдері үшін модульдің қандай бөлімінің ыңғайлы екендігін компиляторға хабарлайды. Осы бөлімде (Uses сөзінен кейін) берілген модульмен қолданылатын кітапханалық модульдер берілген. Сонымен қатар бұл жерде **type** сөзінен кейін Delphi арқылы құрылған форманың баяндалуы орналасады.

Жүзеге асыру бөлімі **implementation** сөзімен ашылады да, форма жұмысын қолдайтын локальді айнымалылар, процедуралар мен функциялардың баяндалуларын қамтиды.

Жүзеге асыру бөлімі {$R *.DFM} директивасынан басталады. Бұл орындалатын файлды генерациялау процесінде форманы баяндауды компиляторға көрсететін болады. Форманы баяндау .dfm кеңейтілуі бар файлда орналасқан. Ол файлдың аты модуль атымен сәйкес.

{$R *.DFM} директивасынан кейін формалар мен оның компоненттері үшін оқиғаларды өңдеу процедуралары жазылады. Осы жерге бағдарлама жасаушы басқа процедуралар мен функцияларды қоса алады.

Инициализациялау бөлімі модуль айнымалыларын инициализациялауға мүмкіндік береді. Инициализациялау бөлімінің нұсқаулары жүзеге асыру бөлімінен кейін

(барлық процедуралар мен функцияларды баяндау) *begin* және *end* кілттік сөздерінің арасына жазылады. Егер инициализациялау бөлімі бірде-бір нұсқауды қамтымайтын болса, онда *begin* сөзі көрсетілмейді.

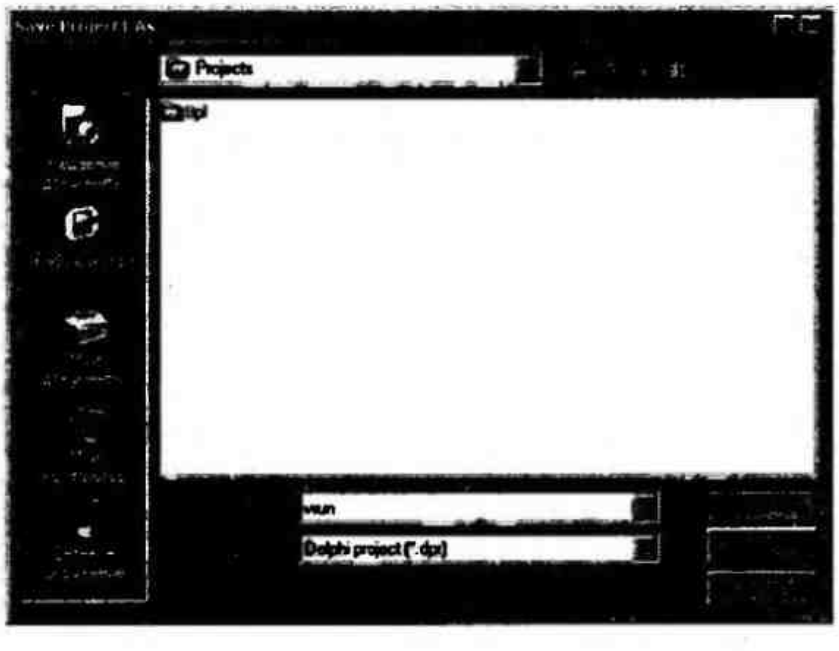
Жобаны сақтау

Жоба – бұл файлдар жиыны, бұл арқылы компилятор бағдарламаның орындалатын (*exe*-файлдар) файлын құрады. Жоба мынадай файлдардан тұрады:

– жобаны баяндау файлынан (*dpr*- файл)
– басты модуль файлынан (*pas*- файл)
– ресурстар файлынан (*res*-файл)
– форманы баяндау файлынан (*dfm*-файл)
– функцияны баяндау файлынан (*cpp*-файл) және т.б.

Жобаны сақтау үшін **File** менюінен **Save Project As** командасын таңдау қажет.

Егер жоба әлі сақталмаған болса, Delphi онда алдымен модульді (код редакторы терезесінің құрамындағыларды) сақтауды ұсынады, сондықтан экранда **Save Unit1 As** терезесі пайда болады. Осы терезеде жоба файлдарына арналған папканы таңдап, модуль атын енгізу қажет.



20-сурет. Модульді сақтау және жобаны сақтау

Сохранить (Сақтау) батырмасын басқаннан кейін жоба файлының атын енгізу қажет терезе пайда болады. Модуль файлдарының аттары (*pas* -файл) және жоба файлдары (*dpr*-файл) аттарының әр түрлі екендігіне назар аударыңыз. Компилятормен генерацияланатын файл аты жоба атымен сәйкес болады.

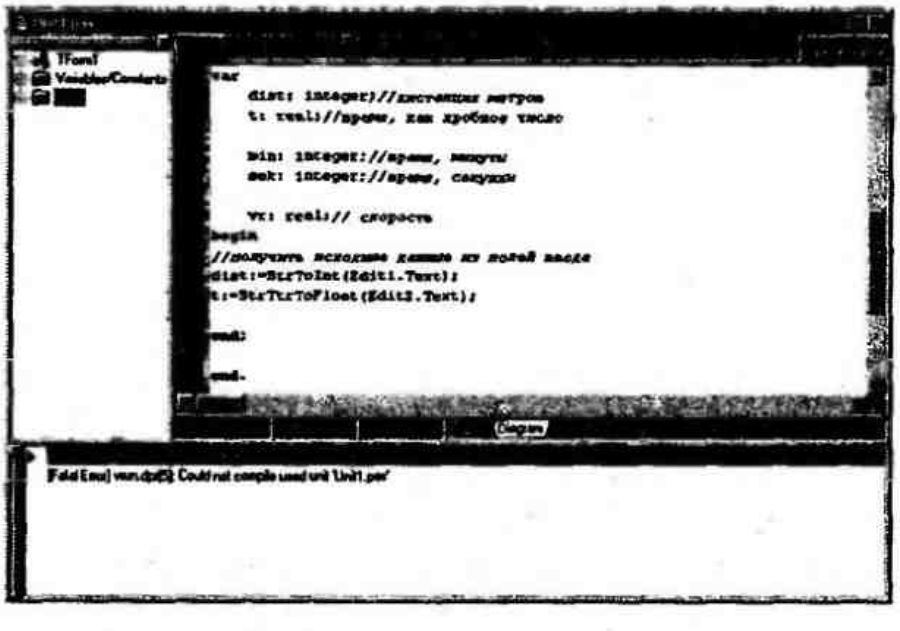
1.6. Компиляция

Компиляция – бастапқы бағдарламаны орындалатын бағдарламаға түрлендіру процесі. Компиляция екі кезеңнен тұрады. Бірінші кезеңде бағдарлама мәтініндегі қатенің жоқтығы тексеріледі, ал екінші кезеңде – орындалатын бағдарлама (**exe**-файл) генерацияланады.

Оқиғаны өңдеу функциясының мәтінін енгізгеннен кейін және жобаны сақтағаннан кейін **Project** менюінен **Compile** командасын таңдап, компиляцияны орындау қажет.

Компиляция процесі және оның нәтижесі **Compiling** сұхбаттық терезесінде көрініс табады. Осы терезеге компилятор қателердің (*Errors*), ескертпелердің (*Warning*) және кеңестердің (*Hints*) санын шығарып береді.

Қателер, ескертпелер мен түсініктер туралы хабардың өзі код редакторы терезесінің төменгі жағында орналасқан (21- сурет).



21-сурет. Қате табылғаны жөніндегі компилятор хабары

1.6.1. Қателер

Компилятор орындалатын бағдарламаны бастапқы бағдарлама мәтінінде синтаксистік қате болмаған жағдайда генерациялайды. Көп жағдайда бағдарламаның терілген мәтінінде қате кетіп отырады. Бағдарлама жасаушы бұл қателерді түзету керек.

Қатені қамтитын үзіндіге ауысу үшін меңзерді қате туралы хабар шығарған қатарға орнатып және контекстік менюден **Edit source** командасын таңдау қажет.

Қателерді жөндеу процесі итерациялық сипатқа ие. Алдымен баяндалмаған айнымалылар тексеріледі. Бағдарламада қолданысқа ие болған айнымалылардың барлығы айнымалыларды баяндау бөлімінде баяндалып кету керек. Кезекті түзетулер мен өзгерістерді бағдарламаға енгізгеннен кейін, компиляцияны қайта орындауға жіберу қажет. Қатені түзету кезінде қате жіберген қатарды ғана емес, сонымен қатар оның алдындағы қатарды да тексеру қажет.

Төмендегі кестеде жиі кездесетін қателер берілген.

9-кесте

Жиі кездесетін қателер

Компилятордың хабары	Ықтимал себептер
Undeclared identifier (Баяндалмаған идентификатор)	а) *Var* бөлімінде баяндалмаған айнымалылар қолданылады. б) Айнымалы атын жазуда кеткен қателер. Мысалы, *Summa* айнымалысы баяндалған, ал бағдарлама мәтінінде *Suma:=0* болып белгіленген. в) Нұсқаулар атын жазудағы қате. Мысалы, *repeat* орнына *repet* жазылса.
Unterminated string (аяқталмаған қатар)	Қатарлық тұрақтыларды жазу кезіндегі қате. Мысалы, аяқтаушы тырнақша қойылмай кетсе.
Incompatible types...and... (Бірікпейтін типтер)	Нұсқаудағы өрнек типтері алынған мәндер типіне сәйкес келмесе.
Missing operator or semicolon (Оператор жоқ немесе нүктелі үтір қойылмай кеткен)	Нұсқаудан кейін нүктелі үтір қойылмаса.

Егер компилятор көп қате тапса, онда барлық хабарды қарап шығып, қатені түзетіп, қайтадан компиляциялау керек.

Егер бағдарламада синтаксистік қате болмаса, компилятор бағдарламаның орындалатын файлын құрады. Орындалатын файл аты да жоба файлының атына ұқсас келеді, оның кеңейтілуі – *.exe*.

Delphi орындалатын файлды жоба файлы орналасқан каталогқа қояды.

1.6.2. Ескертпелер мен түсініктемелер

Бағдарламада қате болып табылмайтын, бірақ күмән туғызатын жағдай байқалған кезде, компилятор түсініктеме (*Hints*) мен ескертпелер (*Warnings*) береді. Мысалы, жиі кездесетін ескертпе, ол – баяндалып кеткен айнымалылардың қолданылмай қалуы:

Variable... is declared but never used in...

Шын мәнінде, егер ол айнымалыны қолданбасақ, оны баяндаудың қажеті не?

Төмендегі (10 кестеде) кестеде компилятор жиі шығаратын ескертпелер берілген.

10-кесте

Компилятор жиі шығаратын ескертпелер

Ескертпе	Ықтимал себептер
Variable... is declared but never used in...	Айнымалы қолданылмайды
Variable ... might not have been initialized. (аталынбаған айнымалылардың қолданылуы ықтимал)	Бастапқы мәнді айнымалыға меншіктейтін нұсқау бағдарламада қамтылмаған.

1.7. Бағдарламаны жүктеу

Бағдарлама мәтінін енгізгеннен кейін, оны Delphi ортасынан тікелей, яғни, әзірлеу ортасымен жұмысты аяқтамастан орындауға жіберу қажет. Ол үшін **Run** меню пунктінен **Run** командасын таңдау қажет немесе инструменттер панеліндегі сәйкес батырманы басса жеткілікті.

Орындау уақытындағы қате

Қосымшаның орындалуы барысында түрлі қателердің туындауы мүмкін. Бұл қателер *Орындау* уақытындағы қателер (**run-time errors** немесе **exception**) деп аталады. Көп жағдайда мұндай қателердің туындауына бастапқы деректердің дұрыс енгізілмеуі себеп болады.

Мысалы, жүгіру жылдамдығын есептеу бағдарламасындағы *Уақыт* өрісіне 3.20 мәнін енгізсек, яғни бүтін бөлік пен бөлшек бөліктің арасын нүктемен ажыратсақ, онда **Есептеу** батырмасын басқан кезде экранда бұл жөніндегі қатемен байланысты терезе пайда болады.

22-сурет. Орындау уақытындағы қате хабары (бағдарлама Windows ОЖ-нен жіберілген)

Қатенің туындау себебі мынада. Бағдарлама мәтінінде санның бөлшек бөлігі бүтін бөлігінен нүкте арқылы бөлінеді. Қолданушы редакциялау өрісіне бастапқы деректерді енгізген кезде санның бүтін және бөлшек бөліктерінің арасын нүктемен немесе үтірмен айырып көрсете алады. Бұл екі символдан қайсысын қолдануға болатындығы Windows ОЖ-ң жұмысқа бапталуына байланысты.

Егер Windows-тың жұмысқа бапталуында санның бүтін бөлігі мен бөлшек бөлігінің үтірмен ажыратылатыны көрсетілсе (Ресейде бұл стандартты орнатым), ал қолданушы бағдарламамен жұмыс кезінде редакциялау өрісіне, мысалы 3.20 қатарын енгізсе, онда

```
t=StrToFloat (Edit2.Text)
```

нұсқауы орындалған кезде, Ресейлік стандарт бойынша бұл мәннің енгізілуінде қате кеткенін білдіретін хабар шығады.

23-сурет. Қатенің пайда болғандығы жөніндегі хабарға мысал

ОК батырмасын басқаннан кейін код редакторы терезесінде қате көрсетілген нұсқауды қамтитын бағдарлама қатары белгіленіп көрсетіледі. Бағдарлама жасаушы бағдарламаның орындалуын одан әрі жалғастыра беруіне болады (**Run** менюінен **Step Over** командасын таңдау қажет) немесе бағдарламаның орындалуын үзіп тастай алады (ол үшін **Run** менюінен **Program Reset** командасын таңдау қажет).

1.8. Өзгерістер енгізу

Жүгіру жылдамдығы бағдарламасын бірнеше рет жұмыс орындауға жібергеннен кейін бағдарламаға өзгерту енгізу қажеттілігі туындайды.

Мысалы, арақашықтықты енгізіп, **<Enter>** пернесін басқаннан кейін меңзер *Уақыт* өрісіне ауысса дұрыс болар еді делік. Сонымен қатар *Арақашықтық* және *Уақыт* өрістерінде қолданушы тек қана цифрлар енгізетін мүмкіндік жасау керек болсын.

Бағдарламаға өзгерістер енгізу үшін Delphi-ді жұмысқа жүктеп, сәйкес жобаны ашу қажет. Ол үшін **File** менюінен **Open Project** командасын таңдаған жөн. Сонымен қатар **File** менюіндегі **Reopen** командасын қолданса да болады. **Reopen** командасын таңдаған кезде бағдарламашының соңғы уақыттарда жұмыс жасаған жобаларының тізімі ашылады.

Төменде **Edit1** және **Edit2** компоненттері үшін **OnKeyPress** оқиғаны өңдеу процедурасы қосылған Жүгіру жылдамдығы бағдарламасы берілген. Бағдарламаға оқиғаны өңдеу процедурасын қосу үшін **Object**

Inspector терезесінде процедураны құратын компонентті, сонан соң **Events** салымынан оқиғаны таңдап, процедура атына меңзерді қойғаннан кейін тінтуір батырмасын екі рет басу қажет. Delphi оқиға өңдеу процедурасының үлгісін (шаблонын) қалыптастырады. Осы әрекеттерден кейін өңдеу процедурасын жүзеге асыратын нұсқауларды енгізуге болады.

```
Unit vrun1;
interfase

uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;

Type
TForm1=class(TForm)
Edit1: TEdit;
Edit2: TEdit;
Label1:TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Button1: TButton;
Button2: TButton;
procedure Button1Click (Sender: TObject);
procedure Button2Click (Sender: TObject);
procedure Edit1KeyPress (Sender: TObject; var Key: Char);

private
   {Private declarations}
public
   {Public declarations}
end;

var
Form1:TForm1;
implementation
{$R *.dfm}
//Есептеу батырмасын басу
```

```
procedure TForm1.Button1Click (Sender: TObject);
var

dist: integer; //арақашықтық, метр
t: real; //уақыт, бөлшек, сан түрінде
min: integer; //уақыт, минут
sek: integer; //уақыт, секунд

v: real; //жылдамдық
begin
 //енгізу өрісінен бастапқы деректерді алу
dist:= StrToInt(Edit.Text);
t:= StrToFloat(Edit2.Text);

//алдын ала түрлендіру
min:= Trunc(t); // минуттар саны – t санының бүтін бөлігі
sek:=Trunc(t*100) mod 100; // секундтар саны – t санының бөлшек бөлігі

//Есептеу
v:=(dist/1000) / ((min*60 + sek) / 3600);

//нәтиже шығару
Label4.Caption:= 'Арақашықтық ' + Edit1.Text + 'м'
+ #13 +
 'Уақыт:' + IntToStr(min) + 'мин' +
IntToStr(sek) + 'сек' + #13 +
'Жылдамдық:' + FloatToStrF(v,ffFixed,4,2) +
'км/сағ';
end;

//Аяқтау батырмасын басу
procedure TForm1.Button2Click (Sender: TObject);
begin
 Form1.Close;
end;
//Арақашықтық өрісіндегі пернені басу
```

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var
Key: Char);

begin
//Key – символ, басылған пернеге сәйкес.
//Егер символға қатынас құру мүмкіндігі болмаса,
// онда процедура оны 0 кодты символмен алмастырады,
//нәтижесінде редакциялау өрісінде символ пайда болмайды
//және қолданушы кейбір пернелер жұмысын бағдарлама
//қабылдамайды деген ойда қалалары сөзсіз.

case Key of
'0' .. '9': ;// цифр
#8; : ;// <Back Space> пернесі
#13; : Edit2. SetFocus ;//<Enter> пернесі

//Басқа символдар – тыйым салынған
else Key:=Chr(0); //Символдар көрсетпеу
end;

end;

end.
```

Өзгерістерді енгізіп болғаннан кейін жобаны сақтау керек. Ол үшін **File** менюінен **Save All** командасын таңдау қажет.

1.9. Қосымшаны соңғы рет жұмысқа баптау

Бағдарлама түзетілгеннен кейін оның жұмысқа дайындығын соңғы рет тексеріп шығып, бағдарламаға ат беру қажет. Сонымен қатар осы бағдарлама файлына сәйкес қосымшаның орындамалық файлын белгілейтін белгішені тандап алу керек. Бұл файлды арнайы папкада немесе жұмыс үстелінде сақтауға болады.

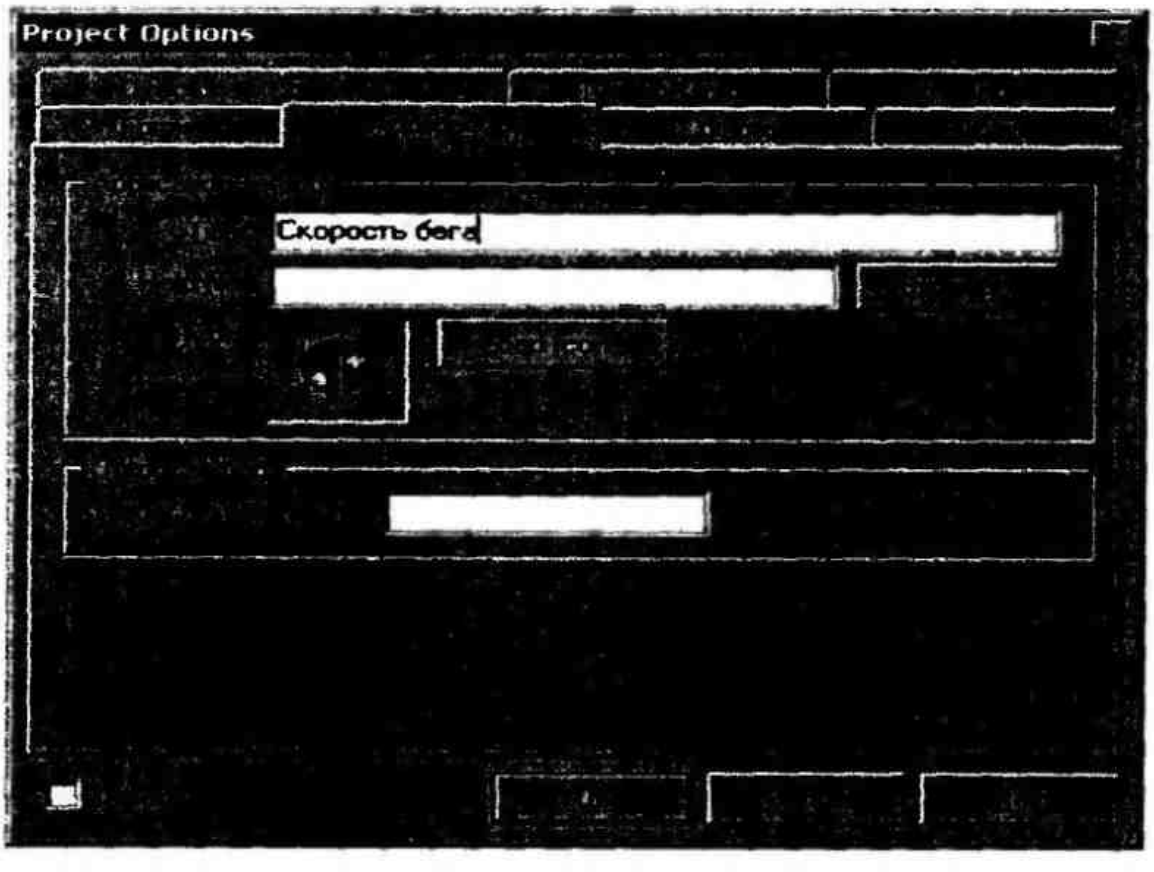
Қосымшаны жұмысқа лайықтау – **Project** менюінен **Options** командасын таңдау нәтижесінде пайда болатын, **Project Options** сұхбаттық терезесінің **Application** салымында орындалады.

Title өрісінде қосымшаның аты енгізіледі.

Қосымшаға стандартты белгішелерден ерекше белгіше беру үшін **Load Icon** батырмасын басу қажет. Әрі қарай ыңғайлы белгішені табу үшін папкаларды қараудың стандартты терезесі (белгіше **.ico** кеңейтілуі бар файлдарда сақталады) қолданылады.

1.9.1. Қосымшалар үшін ерекше белгілер құру

Delphi құрамына **Image Editor** (Көріністер редакторы) бағдарламасы кіреді. Оның көмегімен өзіңіз құрған қосымшаға ерекше белгі қоюыңызға болады.



24-сурет. Application салымын қолдана отырып белгіше орнатуға және бағдарламаға ат беруге болады

Image Editor бағдарламасы **Tools** менюінен сәйкес команданы таңдау арқылы жұмысқа қосылады.

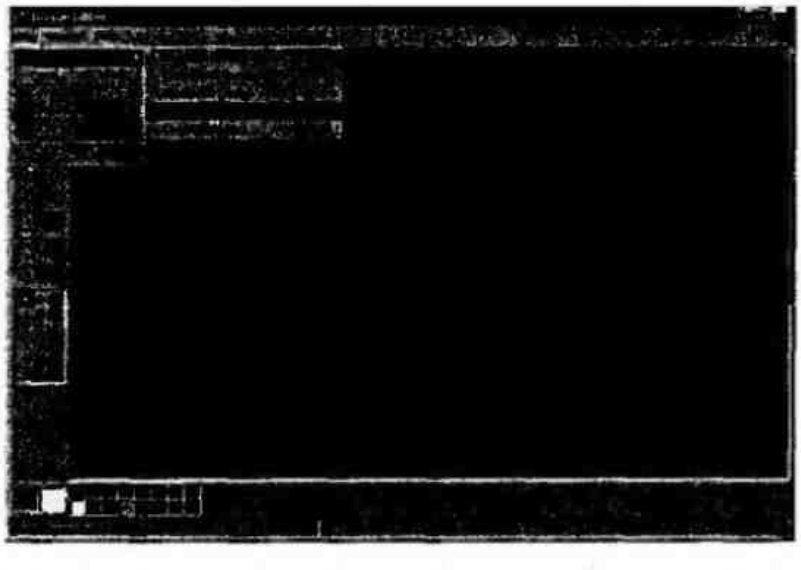
Жаңа белгішені құру жөніндегі жұмысты бастау үшін **File** менюінен **New** командасын таңдап, ал пайда болған тізімнен **Icon File** опциясын алу қажет (25-сурет).

Құрылатын файл типін таңдағаннан кейін **Icon Properties** терезесі ашылады да, онда құрылатын белгішеге сәйкес сипаттамалар таңдалынады: *Size* (өлшемі) – 32x32 (Windows ОЖ-гі белгішелердің стандартты өлшемі) және *Colors* (палитра) – 16 түсті. **ОК** батырмасын басудың нәтижесінде **Icon1.ico** терезесі ашылады. Онда стандартты құралдар мен палитраны пайдалана отырып қажетті белгішенің суретін салуға болады.

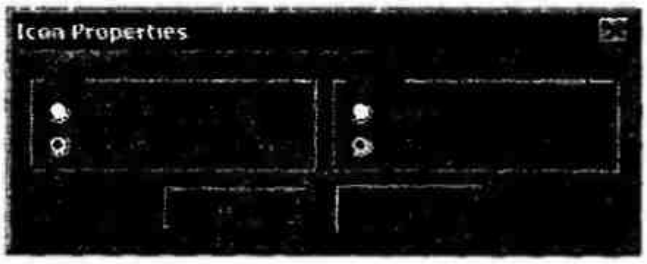
Image Editor-дегі сурет салу процесінің қарапайым графикалық редактордағы картина құру процесінен ешқандай айырмашылығы жоқ. Мысалы, **Microsoft Paint**. Дегенмен де атап өтерлік жақтары да жоқ емес. Бейнелеу өрісі алғашқы кезде (transparent) мөлдір түспен боялған. Егер белгішені осы фонда салатын болса, оны шығару кезінде бейне өрісінің белгіше салынған бөлігі фонын қабылдайды.

Картиналар құруда қате сызылған немесе боялған элементтерді мөлдір түспен бояу арқылы өшіруге (жоюға) болады.

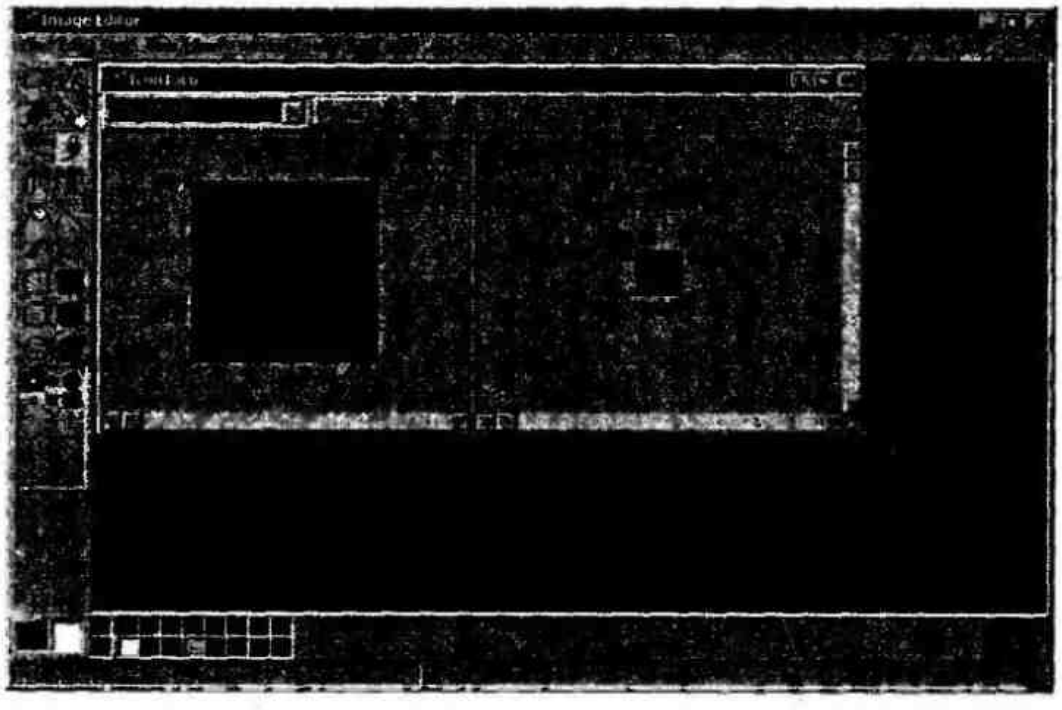
Мөлдір түстен басқа, палитрада инверсті түстер де бар.



25-сурет. Жаңа белгімен жұмысты бастау



26-сурет. Белгіше параметрлерін таңдау



27-сурет. Белгіше салу терезесі

Құрылған белгіні сақтау үшін **File** менюінен **Save** командасы таңдалады.

1.9.2. Қосымшаны басқа компьютерге көшіру

Тек қана стандартты компоненттерді қамтитын және бір ғана .exe-файлдан тұратын шағын бағдарламаны басқа компьютерге дискетті пайдалана отырып, қолмен көшіруге болады. Басқа компьютердегі мұндай бағдарламаны жұмысқа жүктер кезде ешқандай қиындықтың туындамасы сөзсіз.

Кітапханаларды, драйверлерді және басқа да бағдарламалық компоненттерді (мысалы, **ActiveX** типті бақы-

лау элементтері) басқа компьютерге қолмен көшіру барысында қиындықтар туындайды. Мұндай бағдарламалар үшін орнату дискетін құрып алған дұрыс. Осы мақсатта Delphi орнату комплектісі құрамына кіретін **InstallShield Express** дестесі (пакеті) қолданылады.

Бақылау сұрақтары:

1. Delphi-дегі стартты форма терезесі дегенді қалай түсінесіз?
2. Жүйелік бағдарламалық жасақтаманың қолданбалы бағдарламалық жасақтамадан айырмашылығы қандай?
3. RAD (rapid application development) тәсілі дегенді қалай түсінесіз?
4. Жаңа жобамен жұмыс істеуге арналған стартты форма қалай құрылады?
5. Standart салымындағы компоненттердің қызметі қандай?
6. Object Inspector терезесінің қызметі?
7. Оқиға, өріс, қасиет дегендерге түсінік беріңіз.
8. Модуль қандай бөлімдерден тұрады, әр бөлімнің қызметі?
9. Open Project командасы қай кезде қолданылады?
10. InstallShield Express дестесі (пакеті) не үшін керек?

II ТАРАУ

БАҒДАРЛАМАЛАУ НЕГІЗДЕРІ

2.1. Бағдарлама

Жазылған бағдарламаны қолданғанда, пайдаланушы адам компьютерге бастапқы деректерді кілтжиынның көмегімен енгізеді, ал компьютер нәтижені экранға, баспа құрылғысына немесе файлға шығарып береді. Шын мәнінде бастапқы деректерді нәтижеге түрлендіру әрекетін компьютер процессоры (белгілі бір алгоритм бойынша) орындайды. Бұл алгоритм арнайы тілде жазылған, оны бағдарлама деп атаймыз.

Бағдарламаны өңдеу кезеңдері

«Бағдарлама жазу» ұғымы белгілі бір әрекеттерді орындау командаларын қағаз бетінде немесе мәтіндік редактор көмегімен жазумен түсіндіріледі.

Бағдарламалау – бұл мынадай қадамдардан тұратын бағдарламаны құру (өңдеу) процесі:

1. Бағдарламаға қойылатын талапты анықтау.
2. Алгоритм әзірлеу.
3. Командаларды жазу (кодтау).
4. Түзету.
5. Тесттен өткізу.

Бағдарламаға қойылатын талапты анықтау

Бағдарламаға қойылатын талапты анықтау – бұл бастапқы мәліметтерді баяндау және нәтижеге қойылатын талаптарды қалыптастырудағы маңызды кезең болмақ.

Егер бағдарлама Windows ОЖ-де жұмыс істейтін болса, онда міндетті түрде сұхбаттық терезе құрылады. Осы сұхбаттық терезе арқылы қолданушы мен бағдарлама арасындағы қатынас қамтамасыз етіледі.

Алгоритм құру

Алгоритм құру кезеңінде нәтиже алу үшін орындауға қажетті әрекеттерді анықтау керек. Егер тапсырма бірнеше жолмен (тәсілмен) шешілетін болса, онда әрине, шешу алгоритмінің түрлі нұсқалары қолданылады. Олай болса бағдарлама жасаушы осы нұсқалардың ішінен ең тиімдісін белгілі бір өлшем бойынша (мысалы, алгоритмді шешу жылдамдығы) таңдауы тиіс. Алгоритмді құру кезеңінің нәтижесінде алгоритмге сөзбен нақтырақ түсініктеме беріледі немесе оның блок-сызбасы құрылады.

Кодтау

Бағдарламаға қойылатын талап анықталып, шешім алгоритмі құрылғаннан кейін, алгоритм бағдарламалаудың таңдалған тілінде жазылады. Нәтижесінде бастапқы бағдарлама алынбақ.

Түзету

Түзету – бұл қателерді іздеу және оларды жою процесі. Бағдарламадағы қателерді екі топқа жіктеуге болады: синтаксистік (мәтіндегі қателер) және алгоритмдік қателер. Синтаксистік қателерді оңай тауып, жөндеуге болады. Алгоритмдік қателерді аңғару қиынырақ. Егер бағдарлама бір немесе бірнеше кіріс деректерін енгізген кезде дұрыс жұмыс жасайтын болса, онда түзету жұмысын аяқталды деп есептеуге болады.

Тесттен өткізу

Сіздің бағдарламаңызды басқа қолданушылардың пайдаланатынына көзіңіз жететін болса, онда тесттен

өткізу кезеңінің маңыздылығы ерекше. Бұл кезеңде кіріс деректерінің көп мөлшерінде бағдарлама қалай жұмыс жасайды, дұрыс емес деректер енгізгенде бағдарлама жұмысында қандай өзгерістерді байқауға болады, міне, осыларды тексеру қажет.

2.2. Алгоритм және бағдарлама

Бағдарламаны құрудың бастапқы кезеңінде бағдарлама жасаушы тапсырманы шешу әрекеттерін, яғни алгоритмін құрып алуы қажет. Алгоритм дегеніміз – бастапқы деректерден нәтижеге дейінгі процестерді анықтайтын нақты түсінік.

Алгоритм төмендегідей үш қасиетті қамтыса, тапсырманы шешудегі койылымды дұрыс деп бағалауға болады:

– бірмәнділік;
– жалпыламалық;
– нәтижелілік.

Алгоритмнің бірмәнділігі деп әрекетті орындау ережелерінің тұтастығын және орындалу тәртібін түсінуге болады.

Алгоритмнің жалпыламалылығы – тапсырмалар кластарын шешудегі белгілі бір шекте бастапқы деректердің мәндері өзгерген кезде де бағдарлама шешімін табуда кедергінің туындамауы.

Алгоритмнің нәтижелілігі – алгоритмнің орындалуы барысында анық нәтиже алуға ұмтылушылықпен анықталмақ.

Мысалы, $ax^2 + bx + c = 0$ түрінде берілген квадрат теңдеуді шешу қажет. Бұл есепті шешу үшін мынадай қадамдарды құрамыз:

1. $b^2 - 4ac$ формуласы бойынша дискриминантты есептеу:

2. Егер дискриминант мәні нөлден үлкен немесе оған тең болса, онда теңдеу түбірін табамыз: $x1=\frac{-b-\sqrt{d}}{2a}$;

$x2=\frac{-b+\sqrt{d}}{2a}$; мұндағы d – дискриминанттың бірінші қадамындағы есептелген мән.

3. Егер, дискриминант мәні нөлден кіші болса, онда теңдеудің шешімі жоқ.

Бұл келтірілген түсініктер жоғарыда берілген алгоритмнің барлық қасиеттерін қамтиды:

– бірмәнділік (түсіндірмеде теңдеу коэффициенттерінің қалай белгіленетіндіктері көрсетіліп, дискриминант мәнін және теңдеу түбірін есептеудің формулалары келтірілген);

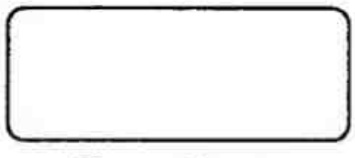
– жалпыламалылығы (түсіндірмеде коэффициенттердің нақты мәндері емес, коэффициенттер белгіленулері қолданылған формулалар келтірілген);

– нәтижелілік (түсіндірме бойынша анықталған қадаммен орындалған бағдарламадан дұрыс нәтиже алынады).

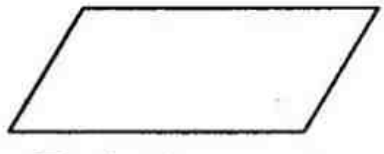
Алгоритмді баяндауда «коэффициент» және «теңдеу түбірі» деген сияқты жалпылама түсініктер қолданылады. Тапсырманы шешуде бұл түсініктер нақтыланады.

Абстрактылы квадрат теңдеу түбірінің мәнін табудың қажеті жоқ, тек нақты теңдеуді шешу қажет, яғни теңдеу коэффициенттерін берсе жеткілікті.

Тапсырманы шешу алгоритмін сөзбен баяндауға да, графиктік түрде блок-сызба көмегімен көрсетуге де болады. Алгоритмді блок-сызба түрінде көрсету кезінде арнайы символдар қолданылады.

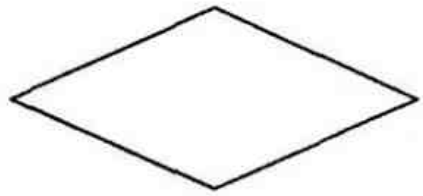


Басы/Аяғы



Енгізу/шығару

Өңдеу



Таңдау

28-сурет. Алгоритмді блок-сызба түрінде көрсету үшін қолданылатын негізгі символдар

Алгоритмді блок-сызба түрінде көрсету бағдарлама жасаушыға орындалатын әрекеттердің реттілігін түсінуге мүмкіндік береді.

Delphi-де бағдарламалау кезінде тапсырманы шешу алгоритмі оқиғаларды өңдеу процедуралары алгоритмдерінің жиынтығын қамтиды. Сұхбаттық терезені құрғаннан кейін және оқиғаларды өңдеу алгоритмін қалыптастырып алғаннан кейін бағдарламаны жазуға кірісеміз.

Төменде *сатып алу құнына* бағдарлама мәтіні берілген.

```
unit pokupka_1;

interface

uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;

type
TForm1 = class(TForm)
Edit1: TEdit;
Edit2: TEdit;
Label1: TLabel;
Label2: TLabel;
Button1: TButton;
Label3: TLabel;
```

```
    procedure Button1Click (Sender: TObject);
    procedure Edit2KeyPress (Sender: TObject; var Key:
Char);
    procedure Edit1KeyPress (Sender: TObject; var Key:
Char);

    private
       {Private declarations}
    public
       { Public declarations}
    end;

    var
    Form1: TForm1;

    Implementation

    {$R *.dfm}

    //ішкі бағдарлама
    procedure Summa;
    var

    cena: real; //бағасы
    kol: integer; //саны
    s: real; /сома

    mes: string [255] // хабар

    begin

    cena:= StrToFloat (Form1.Edit1.Text);
    kol:= StrToInt(Form1.Edit2.Text);
    s:=cena * kol;
    if s > 500 then

    begin
    s:= s * 0.9;
```

```
    mes:= '10% -ға түсіру' + #13;
    end;
    mes:= mes+ 'Сатып алу құны: ' +
    FloatToStrF(s, ffFixed,4,2)+ 'руб.';

    end;

    //Құны батырмасын басу

procedure TForm1.Button1Click (Sender: TObject);
    begin
     summa; // Сатып алу сомасын есептеу
    end;

    //Саны өрісінде пернені басу

    procedure TForm1. Edit2KeyPress (Sender: TObject; var
Key: Char);

    begin
    case Key of
     '0' .. '9', #8: ; // цифрлар және <BackSpace> пернесі
    #13: Summa; // сатып алу құнын есептеу
    else Key:= Chr(0); // Символды көрсетпеу
    end;

    end;

    //Бағасы өрісіндегі пернені басу
    procedure TForm1.Edit1KeyPress (Sender: TObject; var
Key: Char);
    begin

    case Key of

    '0' .. '9', #8 : ; // цифрлар және <BackSpace> пернесі
    #13: Form1.Edit2.SetFocus; // <Enter> пернесі
    '.', ',':
```

```
begin
if Key = '.'
then Key:= ',';

if Pos (' , ', Edit1.Text) < > 0
then Key:= Chr(0);
end;

else //Басқа символдарға тыйым салынған

Key:= Chr(0);
end;

end;

end.
```

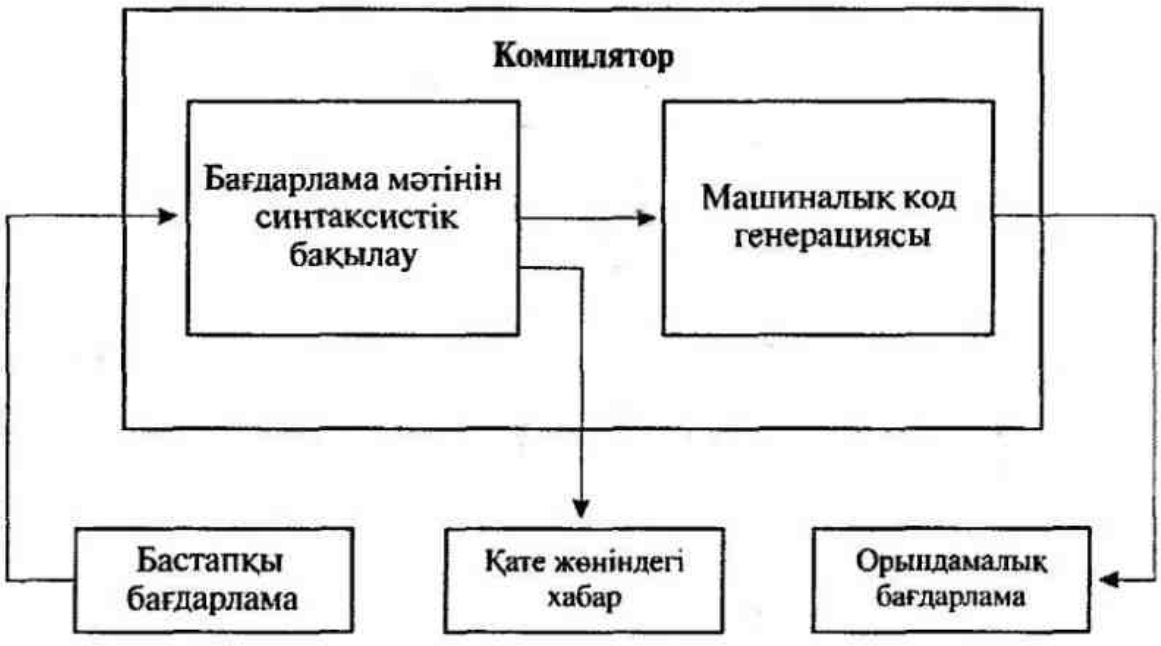
2.3. Компиляция

Бағдарламалау тілінің нұсқауы түрінде берілген бағдарлама бастапқы бағдарлама деп аталады. Ол адамға түсінікті нұсқаулардан құрылғанмен де, бұл нұсқаулар компьютер процессорына түсініксіз. Процессор бастапқы бағдарлама нұсқауларына сәйкес жұмысты орындау үшін, бастапқы бағдарлама машиналық тілге – процессор командалары тіліне аударылуы қажет. Бастапқы бағдарламаны машиналық кодқа түрлендіруді **компилятор** деп аталатын арнайы бағдарлама жүзеге асырады.

Компилятор екі міндетті тізбектей орындайды:

1. Бастапқы бағдарлама мәтініндегі синтаксистік қатенің жоқтығын тексереді.

2. Орындалатын бағдарлама – машиналық кодты құрады (генерациялайды).



29-сурет. Компилятор жұмысының сызбасы

Орындалатын бағдарламаны генерациялау бастапқы бағдарлама мәтінінде синтаксистік қателер болмаған жағдайда ғана жүзеге асырылатынын атап өткен жөн.

Компилятор арқылы машиналық кодты генерациялау бағдарлама мәтінінде синтаксистік қатенің жоқтығын көрсетеді. Тесттен өткізу арқылы, алынған нәтижені талдау арқылы бағдарламаның дұрыстығына көз жеткізуге болады.

Бақылау сұрақтары:

1. Бағдарлама (программа) дегеніміз не?
2. Бағдарламаны өңдеу кезеңдері.
3. Бағдарламаға қандай талаптар қойылуы мүмкін?
4. Бағдарламаны тесттен өткізу дегенді қалай түсінесіз?
5. Алгоритмнің қасиеттері.
6. Компиляция процесін қалай түсіндіресіз?
7. Бағдарламаны генерациялау дегенді қалай түсіндірер едіңіз?

III ТАРАУ

OBJECT PASCAL БАҒДАРЛАМАЛАУ ТІЛІ

3.1. Бағдарламалау тілінде қолданылатын негізгі типтер

Object Pascal тілінде жазылған бағдарлама оператор деп аталатын нұсқаулар жиынын қамтиды.

Нұсқаулар бір-бірінен нүктелі үтір арқылы ажыратылады.

Әрбір нұсқау сөздерден (идентификатордан) тұрады.

Идентификатор мыналарды белгілейді:

– тіл нұсқауын (:=, if, while, procedure);

– ішкі бағдарламаны (процедураны немесе функцияны);

– айнымалыны (жады ұяшығын);

– тұрақтыны;

– арифметикалық (+, –, *, /) немесе логикалық (and, or, not) операцияларды;

– бағдарлама тарауының басын және аяғын және т.б.

3.1.1. Деректер типтері

Бағдарлама түрлі деректер типтерімен операциялар орындайды: бүтін және бөлшек сандармен, символдармен, символдар қатарымен, логикалық шамалармен. Кез келген бағдарламалау тілдерінде қандай деректер енгізілмесін, оның міндетті түрде машиналық команда түріндегі типтерді белгілеу белгісі болады.

Бүтін тип

Object Pascal-да бүтін сандарды баяндаудың жеті типі бар: Shortint, Smallint, Longint, Int64, Byte, Word,

Longword. Бұл типтер жөнінде төмендегі кестеде толық берілген:

11-кесте

Типі	Диапазон	Форматы
Shortint	–128..127	8 бит
Smallint	–32768..32767	16 бит
Longint	–2147483648..2147483647	32 бит
Int64	-2^{63}.. $-2^{63}-1$	64 бит
Byte	0..255	8 бит, таңбасыз
Word	0..65535	8 бит, таңбасыз
Longword	0..4294967295	8 бит, таңбасыз

Object Pascal үшін әмбебап бүтін тип – Integer (эквиваленті Longint).

Нақты тип

Object Pascal-да нақты типтің алты түрі бар: Real48, Single, Double, Extended, Comp, Currency. Типтер мәндері компьютер жадысында деректерді сақтаумен байланысты диапазондарымен, цифрлар мәндері және байттар санымен ерекшеленеді.

Төмендегі кестеде нақты типтердің түрлері және олардың диапазондары берілген.

12-кесте

Типі	Диапазоны	Мәндік цифрлар	Байт
Real48	$2.9*10^{-39}$.. $1.7*10^{38}$	11–12	6
Single	$1.5*10^{-45}$.. $3.4*10^{38}$	7–8	4
Double	$5.0*10^{-324}$.. $1.7*10^{308}$	15–16	8
Extended	$3.6*10^{-4951}$.. $1.1*10^{4932}$	19–20	10
Comp	$-2^{63}+1$ .. $2^{63}-1$	19–20	8
Currency	–922337203685477.5808 .. 922337203685477.5807	19–20	8

Object Pascal үшін әмбебап нақты тип – Real (эквиваленті Real48).

Символдық типтер

Object Pascal екі символдық типті қолдайды: AnsiChar, WideChar. AnsiChar – ANSI-кодировкадағы символдар, 0-ден 255 дейінгі диапазондағы сандарға сәйкес. WideChar типі – Unicode кодтауында символдарды орналастыруға мүмкіндік береді (0-ден 65535-ке дейінгі диапазонда жатыр), бірақ оны Delphi-де сирек қолданады. Unicode кодтауында алғашқы 256 символ ANSI – кодтаудың символдарына сәйкес келеді, бірақ деректердің типтерінің өзі тікелей сәйкес келмейді.

Символдарды сақтау және өңдеу үшін AnsiChar, WideChar типті айнымалылар қолданылады.

Object Pascal үшін әмбебап символдық тип – Char (эквиваленті AnsiChar).

Қатарлық тип

Қатарлар мынадай типтермен беріледі:
- ShortString;
- LongString;
- WideString;

ShortString типті айнымалыға жады статикалық түрде бөлінеді, яғни бұл типтегі қатар символдарының саны 255-тен артпайды. Ал LongString және WideString типті айнымалыға жады динамикалық түрде бөлінеді, яғни бағдарлама жұмысы кезінде мұндай қатар ұзындығы шектелмеген. Осы типтермен қатар универсалды String типі де қолданылады.

String → ShortString типіне эквивалентті.

Символдық немесе қатар айнымалылары айнымалыларды баяндау тарауында айтылады.

Аты: String;

немесе

Аты: String (Ұзындығы);

Мұндағы;

– Аты – қатарлық типті айнымалы;

– String – қатарлық типті білдіретін кілттік сөз.

– ұзындығы – қатардың барынша ұзындығын анықтайтын бүтін типті тұрақты.

Мыс: name String [30];

Егер қатарлық типті айнымалы ұзындығы көрсетілмесе, онда ұзындық автоматты түрде 255 символға тең болып қалыптасады.

Бағдарлама мәтінінде символдар тізбегі (қатар) дара тырнақшаға алынады.

Мыс:

Paro1:= "құпия сөз";

немесе

Paro1:= "2001 "; бұл дұрыс емес;

компиляциялау кезінде бұл мәтіннің типін дұрыс көрсету керектігі жөнінде хабар шығады, 'char and inreger'.

Логикалық тип

Логикалық шама **(True)** ақиқат немесе **(False)** жалған мәндерінің бірін ғана қабылдайды. Object Pascal-да логикалық шамаларды **Boolean** типі арқылы баяндауға болады.

3.1.2. Айнымалылар

Айнымалы – бағдарлама әрекет жасайтын деректердің орналасқан жады аумағы.

Бағдарлама деректерді басқарғанда немесе деректермен түрлі әрекеттер орындағанда, ол жады ұяшықтарының құрамындағылармен, яғни *айнымалылармен* әрекеттер орындайды.

Айнымалылардың аты болады. Оларды ақпарат элементтерін сақтауға қабілетті құрылғымен салыстыруға болады, мыс: *санмен*. Бағдарламаның орындалуы барысында бұл құрылғы құрамындағыларының өзгеруі мүмкін. Жалпы айнымалыға мынадай анықтама беруге болады:

Айнымалы бұл компьютер *жадысының аумағы* (ұяшығы), бағдарламада қолданылатын мәндерге немесе өрнектерге міндетті түрде айнымалылар меншіктелуі керек.

Айнымалы ретінде қолданылады:

- Латын алфавитінің әріптері;
- Цифрлар;
- Кейбір арнайы символдар;

Айнымалы атауындағы бірінші символ әріп болуы керек. Айнымалы атында «пробелді (бос орынды)» қолдануға болмайды.

Мынаған назар аудару керек: Object Pascal компиляторы бас әріппен кіші әріпті (айнымалы атындағы) ажырата алмайды, сондықтан да SUMMA, Summa, summa аттары бір ғана айнымалыны белгілейді.

Жалпы айнымалы аты оның белгіленуімен байланысты болу керек. Мыс: $ax^2 + bx + c = 0$

Бұған мынадай айнымалыларды меншіктеген жөн:

$a, b, c, x1$ және $x2$.

Тағы бір мысал, егер бағдарламада сатып алу сомасын, төмендетілген түсіріп сату бағасын сақтауға арналған айнымалы болатын болса, онда бұл айнымалыларға мынадай аттар меншіктеген дұрыс:

TotalSumm және *Discount* немесе *ObSumma* және *Skidka*

Object Pascal-да әрбір айнымалы оны пайдаланбастан бұрын баяндалып кету керек. Баяндаудың көмегімен айнымалының бар болу фактысы орнатылып қана қоймай, сонымен қатар оның типі беріледі, оның мүмкін болатын мәндерінің диапазоны көрсетіледі.

Жалпы түрде айнымалыны баяндау нұсқауы былай көрсетіледі:

Аты: типі;

Мұндағы:

Аты – айнымалының аты.

Типі – деректер типтері.

Мыс:

```
a: Real;
b:Real;
i:integer;
```

Бағдарлама мәтінінде әр айнымалының баяндалуы жеке қатарда жазылуы қажет. Егер бағдарламада бір типке жататын бірнеше айнымалы берілсе, онда оны былай көрсетуге болады:

```
a, b, x: Real;
x1, x2: Real;
```

3.1.3. Тұрақтылар

Object Pascal-да тұрақтылардың екі түрі бар:

1. Қарапайым.
2. Атаулы.

Қарапайым тұрақты – бүтін немесе бөлшек сандар, символдар қатары немесе жекелеген символдар, логикалық мәндер.

Бүтін сандар қарапайым түрде жазылады. Бөлшек сандар, санның бүтін және бөлшек бөлігінің арасы нүктемен ажырату арқылы жазылады. Егер тұрақты теріс болса, алдына «минус» таңбасы қойылады.

Төменде сандық тұрақтыларға мысалдар берілген:

Мыс:

```
123
0.0
–524.03
0
```

Бөлшек сандарды жылжымалы үтірлі сандар түрінде де көрсетуге болады.

Төмендегі кестеде қарапайым, алгебралық және жылжымалы үтірлі формада сандардың көрсетілуі берілген.

13-кесте

Сан	Алгебралық форма	Жылжымалы үтірлі форма
1000000	$1*10^6$	1.0000000000E+06
–123.452	$-1.23452*10^2$	–1.2345200000 E +02
0.0056712	$5.6712*10^{-3}$	5.6712000000E–03

Қатарлық және символдық тұрақтылар

Қатарлық және символдық тұрақтылар тырнақшаға алынып жазылады. Төменде осы тұрақтыларға мысалдар берілген:

'Object Pascal программалау тілі'
'Delphi 6'
'2.4'
'Д'

Осы мысалдағы '2.4' тұрақтысына назар аударыңыз. Бұл мысал символдық тұрақты болып қалыптасқан, 2.4 санын емес, "екі бүтін оннан төрт" деген символды білдіреді.

Логикалық тұрақтылар

Логикалық өрнектер нәтижесі ақиқат немесе жалған болады. Ақиқатқа **True** тұрақтысы, жалған мәнге **False** – тұрақтысы сәйкес болмақ.

Атаулы тұрақтылар

Атаулы тұрақтылар дегеніміз – бағдарламада тұрақтының орнына қолдануға болатын идентификатор (атау).

Атаулы тұрақтылар да, айнымалылар сияқты қолданылмастан бұрын міндетті түрде баяндалып кетуі керек. Жалпы түрде атаулы тұрақтыларды баяндау нұсқасы былайша беріледі.

Тұрақты = мәні;

мұндағы:

тұрақты – тұрақты аты;

мәні – тұрақты мәні.

Төменде атаулы тұрақтыларды баяндауға мысал берілген:

Bound=10;

Title = "Жүгіріс жылдамдығы";

pi = 3.1415926;

Атаулы тұрақтыны баяндағаннан кейін, бағдарламада тұрақтының орнына оның атын қолдануға болады. Тұрақтылар типтері олардың түрлерімен анықталады, мысалы:

– 125 – бүтін типті тұрақты;
– 0,0 – нақты типті тұрақты;
– "Орындау" – қатарлық типті тұрақты;
– "\ " – символдық тұрақты.

3.2. Тілдегі негізгі функциялар және операторлар

3.2.1. Меншіктеу операторы

Меншіктеу операторы негізгі есептеу операторы болып табылады. Егер бағдарламада есептеу әрекетін орындау керек болса, онда міндетті түрде меншіктеу операторын пайдалану қажет.

Меншіктеу операторының орындалуы нәтижесінде айнымалы мәні өзгеріп, оған мән меншіктеледі.

Жалпы түрде меншіктеу операторы былай жазылады:

Аты:= өрнек;

Мұндағы.

Аты – меншіктеу нәтижесінде мәні өзгеруі мүмкін айнымалы аты;

:= – меншіктеу операторының символы.

Өрнек – мәні айнымалыға меншіктелетін өрнек. Меншіктеу операторы символының сол жағында орналасқан.

Мысалы:

*Summa:= Cena * Kol;*

Skidka: = 10;

Found:= False;

Өрнек

Өрнек операндалардан және операторлардан тұрады. Операторлар операндалар арасына орналасады да, операндалармен орындалатын әрекеттерді белгілейді. Өрнек операндалары ретінде мыналарды қолдануға болады:

– айнымалыларды;

– тұрақтыларды;

– функцияларды немесе басқа өрнектерді.

14-кесте

Object Pascal-дағы негізгі алгебралық операторлар

Операторлар	Әрекеттер
+	Қосу
-	Азайту
*	Көбейту
/	Бөлу
DIV	Бүтін санды бөлу (бүтін бөлік алынады)
MOD	Бөлуден кейінгі қалдықты есептеу

DIV, MOD – операторларынан басқа операторларды жазғанда, операндалар мен операторлар арасына пробел қоймауға болады.

DIV операторы бір санды басқа санға бөлу нәтижесінен санның бүтін бөлігін алуға мүмкіндік берсе, MOD операторы бөлгеннен кейінгі қалдықты шығарып береді.

Мысалы:
5 DIV 2=2;
5 MOD 2=1.

Өрнектердің тұрақтыларды немесе айнымалыларды қамтыған түрде берілетіні анық.

Мысалы:
123
0.001
i
A + B/C
***Summa* * 0.75**
(B1 + B3 + B3)/3
***Cena MOD* 100**

Өрнек мәнін есептеген кезде, операторлар приоритеттеріне назар аударған жөн.

*, /, DIV, MOD – приоритеттері жоғарғы операторлар,

+, –, – приоритеттері төмен операторлар болып есептеледі.

Операторларды есептеу барысында приоритеті жоғары операторлар бірінші орындалып, приоритеті төмен операторлар екінші кезекте орындалу мүмкіндігіне ие болады.

Егер операторлардың приоритеттері бірдей өрнек кездессе, онда операторлар сол жақтан басталып орындалады.

Мысалы:

(r1 + r2 + r3) / (r1*r2*r3) – жақшаға алынған өрнек 1 операнда болып саналады.

Жалпы, жақшадағы өрнектің приоритеті жоғары болмақ, сонымен қатар жақшалардың санына, олардың дұрыс ашылып, жабылғанына ерекше көңіл бөлген жөн.

Бағдарламада жақшалардың қате қойылуына байланысты синтаксистік қателер жиі кездесіп отырады.

Өрнек типі

Өрнек түрі өрнекке кіретін операндалар типтерімен анықталады және орындалатын операцияға байланысты. Егер операцияға қатысатын екі операнда да бүтін типті болса, онда өрнек нәтижесі де бүтін болмақ. Егер операцияға қатысатын операндалардың біреуі бөлшек типті болатын болса, онда нәтиже де бөлшек сан болмақ.

Өрнек типін анықтау өте маңызды. Бағдарламада айнымалы типтері айнымалыларды баяндау нұсқауында беріледі.

Мыс:

0, 1 және –512 сандары – бүтін типтер (integer)
1.0, 0.0 және 3, 2E–05 сандары – нақты типтер (real)

Төменде операндалар типіне және операторлар түрлеріне байланысты өрнек типін анықтау ережелері берілген.

15-кесте

Оператор	Операндалар типтері	Өрнек типі
*, +, –	Егер ең болмағанда 1 операнда типі = real болса.	Real
*, +, –	Егер 2 операнда = integer	Integer
/	Real немесе integer	әрдайым Real
DIV, MOD	Әрдайым integer	әрдайым integer

Меншіктеу нұсқауының орындалуы

Меншіктеу нұсқауы былайша орындалады:

– алдымен меншіктеу нұсқауы символының оң жағындағы өрнек мәні есептеледі;

– сонан соң есептелген мән меншіктеу нұсқауының сол жағындағы айнымалыға меншіктеледі.

Мысалы:

i:= 0; – i айнымалысының мәні нөлге тең болады.

a:= b + c; a айнымалысының мәні b мен c айнымалылары мәндерінің қосындысына тең.

j:= j + 1; j айнымалысының мәні бірге ұлғаяды.

Тағы бір ескеретін жағдай *real* типті айнымалыларды *real* немесе *integer* типті өрнек мәндерімен сәйкестендіруге болады, *integer* тек қана *integer* типті өрнек мәнімен сәйкестендіріледі.

Мысалы, i және n айнымалылары – Integer типті, ал d айнымалысы – real типті болса, онда

I:= n /10;

I:=1.0 нұсқауы қате болып табылады,

ал, d:= i+1;

нұсқауы дұрыс болмақ.

Компиляция кезінде өрнек типі мен айнымалы типтерінің сәйкестіктері тексеріледі. Егер өрнек типі айнымалы типіне сәйкес болмаса, онда компилятор осы әрекетке байланысты туындайтын қатені көрсетіп береді:

Incompatible types ... and ...,

мұндағы көп нүктенің орнында өрнек және айнымалы типтері көрсетіледі. Мысалы, n айнымалысы бүтін типті болса, онда n:= m/2 нұсқауы дұрыс емес, сондықтан да компиляция кезінде мынадай хабар шығады:

Incompatible types 'Integer' and 'Extended'.

3.2.2. Стандартты функциялар

Жиі кездесетін есептеулер мен түрлендірулерді орындау үшін Object Pascal-да көптеген стандартты функциялар қарастырылған.

Функция мәні оның атымен байланысты. Сондықтан да функцияны меншіктеу нұсқауының операндасы ретінде қолдануға болады.

Мысалы:

k:=Sqrt(n).

мұндағы Sqrt – квадрат түбірді есептеу функциясы болып табылады. Функция мән типімен және параметрлер типімен сипатталады.

Функция мәні меншіктелетін айнымалы типі функция типіне сәйкес болу қажет.

Арифметикалық функциялар

Төмендегі кестеде түрлі есептеулерді орындауға мүмкіндік беретін арифметикалық функциялар тізбегі келтірілген.

16-кесте

Функция	Мәні
Abs (n)	n-абсолют мәні
Sqrt (n)	n-квадрат түбір
Sgr (n)	n-шаршы
Sin (n)	Синус n
Cos (n)	Косинус n
Arctan (n)	Арктангенс n
Exp (n)	Экспонента n
Ln (n)	n натурал логарифм
Rundom (n)	0 ден n-1-ге дейінгі диапазондағы кездейсоқ бүтін сандар

Типтерді түрлендіру функциялары

Типтерді түрлендіру функциялары ақпаратты енгізу/шығаруды қамтамасыз ететін нұсқауларда жиі қолданылады. Мысалы, сұхбаттық терезенің шығыс өрісіне *real* типті мәнді шығару үшін, алдымен берілген айнымалының мәні болып табылатын санды алу керек. Мұны *FloatToStr* функциясының көмегімен жүзеге асыруға мүмкіндік бар. Бұл функцияның атқаратын қызметі нақты типті өрнекті қатар түрінде көрсету.

Төмендегі кестеде жиі қолданылатын түрлендіру функциялары берілген.

17-кесте

Функция	Мәні
Chr (n)	Коды n – ге тең символ
IntToStr (n)	n бүтін санын көрсету
FloatToStr (n)	Нақты n мәнінің кескіні
FloatToStr(n,f,k,m)	Нақты типті n мәнінің кескіні. Функцияны шығарған кезде көрсетіледі: f – формат (кескіндеу тәсілі); k – дәлдік (қажетті цифрлардың жалпы саны); m – ондық нүктеден кейінгі цифрлар саны
StrToInt (s)	s – қатарды бейнелейтін бүтін
StrToFloat (s)	s – қатарды бейнелейтін нақты тип
Round (n)	n-ге жақын жуықталып алынған бүтін
Trunc (n)	n – бүтін бөлік
Frac (n)	Нақты n типтің бөлшек бөлігі
Int (n)	Нақты n типтің бүтін бөлігі

Функцияны қолдану

Функциялар әдетте өрнек операндалары ретінде қолданылады. Бұл жағдайда функция параметрлері айнымалы, тұрақты немесе өрнек болуы мүмкін. Төменде стандартты және түрлендіру функциялары қолданылған мысалдар берілген.

```
n:= Round( (x2-x1)/dx);
x1:= (-b+Sqrt(d)) / (2*a);
m:= Random(10);
cena:=StrToInt(Edit1.Text);
Edit2.Text:= IntToStr( 100);
mes:= 'x1=' +FloatToStr( x1);
```

Деректерді енгізу

Әдетте қарапайым бағдарлама бастапқы деректерді енгізу терезесінен немесе редакциялау өрісінен (Edit компоненті) алады.

Енгізу терезесі – экранда **InputBox** функциясын шақыру нәтижесінде пайда болатын стандартты сұхбаттық терезе. **InputBox** функциясының мәні – қолданушы енгізетін қатар болып табылады. Жалпы түрде деректерді енгізу нұсқауы былай беріледі:

Айнымалы := InputBox (Тақырып, Түсінік, Мәні);

мұндағы:

Айнымалы – қатарлық типті айнымалы, мәні қолданушыдан алынатын болады;

Тақырып – енгізу терезесінің аты.

Түсінік – хабарды түсіндіретін мәтін.

Мән – енгізу өрісінде пайда болатын мән.

Мысалы:

S:= InputBox ("Фунт-килограмм", "фунттағы салмақты енгізіңіз", "0");

30-сурет. Енгізу терезесіне мысал

Егер бағдарламаның жұмысы кезінде бағдарлама жасаушы қатарды енгізіп, **OK** батырмасын тінтуір арқылы басатын болса, онда **InputBox** функциясының мәні енгізілген қатар болмақ. Егер қолданушы **Cancel** батырмасын басса, онда **InputBox** функциясының мәні функцияға *Мән параметрі* ретінде берілген қатар болмақ. Бұл мысалда **InputBox** функциясының мәні қатарлық (string) тип. Егер бағдарламада сандық мән алыну керек болса, онда енгізілген қатар санға түрлендірілуі керек. Онда мынадай функция қосылады:

```
S:=InputBox ('фунт-килограмм', 'фунттағы салмағын
енгізу', ' ');
Funt:=StrToFloat(s);
```

Редакциялау өрісінен енгізу

Редакциялау өрісі – **Edit** компонентімен байланысты. Редакциялау өрісінен деректерді енгізу **Text** қасиетіне қатынас құрумен жүзеге асады.

Нәтижелерді шығару

Бағдарлама нәтижелерді хабарлау терезесіне немесе сұхбаттық терезенің шығару өрісіне береді.

Хабарлау терезесіне шығару

Хабарлау терезесі пайдаланушылардың назарын аудару үшін қолданылады. Хабарлау терезесінің көмегімен бастапқы деректердің енгізуде кеткен қателерін аңғаруға немесе операцияларды орындауды тұжырымдауда берілген сұраныстарды бекітуге болады, мысалы, *файлды жою*.

Хабарлау терезесін экранға шығару үшін **ShowMessage** процедурасы немесе **MessageDlg** функциясы қолданылмақ.

ShowMessage процедурасын қолданғанда экранға мәтіні бар тереземен қатар, **OK** батырмасы шығады. Жалпы түрде **ShowMessage** процедурасын шақыру нұсқауы мына түрде беріледі:

```
ShowMessage (Хабар);
```

мұндағы *Хабар* – ол терезеге шығатын мәтін.

Төмендегі суретте **ShowMessage** ("фунттағы салмақты енгізіңіз."); нұсқауының орындалу нәтижесінде алынған хабардың терезесі көрсетілген:

31-сурет. Хабар терезесіне мысал

Терезе басындағы хабар **ShowMessage** процедурасымен шақырылады, бұл өз кезегінде **Project Option** терезесіндегі **Application** салымында беріледі.

MessageDlg функциясы әмбебаптығымен ерекшеленеді. Бұл функция хабар терезесіне стандартты белгішелердің бірін орналастыруға мүмкіндік береді. Мысалы, «Назар аударыңыз» белгішесі, сондай-ақ командалық батырмалардың санын және типін беріп, қолданушыға батырмалардың қайсысын басқанын анықтауға мүмкіндік береді.

```
r:=MessageDlg ('Файл'+FName+'жойылады.',

mtWarning, [mbOK, mbCancel], 0);
```

Төмендегі суретте осы нұсқаудың орындалу нәтижесі берілген.

32-сурет. Хабарлау терезесіне мысал

MessageDlg функциясының мәні – сан, ол мәнді тексере отырып, командалық батырманы таңдау арқылы сұхбаттың аяқталғанын анықтауға болады.

MessageDlg функциясына қатынас құру, жалпы түрде былай беріледі:

Таңдау:= MessageDlg(Хабар, Тип, Батырмалар, Анықтама Контексті),

Мұндағы:

Хабар – хабар мәтіні;

Тип – хабар типі.

Хабарлар мынадай түрлерге бөлінеді:

– ақпараттық;

– ескертуші;

– сыншылық қателер жөніндегі хабар.

Хабардың әр типіне сәйкес белгілері бар. Хабар типтері аталған тұрақтылармен беріледі. Осыған сәйкес деректер төмендегі кестеде берілген.

18-кесте

Тұрақтылар	Хабарлар түрі	Белгішелер
mtWarning	Назар аударыңыз	
mtError	Қате	
mtInformation	Ақпарат	
mtConfirmation	Тұжырымдау	
mtCustom	Қарапайым	Белгішесі жоқ

Батырмалар – хабар терезесінде бейнеленетін батырмалар тізімі. Тізім үтір арқылы бөлінген бірнеше аталған тұрақтылардан тұруы мүмкін.

19-кесте

Тұрақты	Батырма	Тұрақты	Батырма
mbYes	Yes	mbAbort	Abort
mbNo	No	mbRetry	Retry
mbOK	OK	mbIgnore	Ignore
mbCancel	Cancel	mbAll	All
mbHelp	Help		

Тізім толығымен квадрат жақшаға алынып жазылады.

Мысалы, хабарлау терезесінде **OK** және **Cancel** батырмаларын шығару үшін, батырмалар тізімін былай жазу қажет:

[mbOK, mbCancel]

Бұлардан басқа мынадай тұрақтылар да қолданылады: mbOKCancel, mbYesNoCancel және mbAbort Retry Ignore.

Бұл тұрақтылар сұхбаттық терезеде жиі қолданылатын командалық батырмалар үйлесімділігін анықтайды.

Контексті анықтама – қолданушы <F1> батырмасын басқан кезде экранда пайда болатын анықтамалық жүйенің бөлімін анықтайтын параметр.

Егер анықтаманы шығару қарастырылмаған болса, онда Контексті Анықтама параметрінің мәні 0-ге тең.

MessageDlg функциясы қайтаратын мәндер (қолданушының басқан командасына байланысты) тізімі төмендегі кестеде берілген:

20-кесте

MessageDlg функциясының мәні	Сұхбат мына батырманы басумен аяқталады
MrAbort	Abort
MrYes	Yes
MrOK	OK
MrRetry	Retry
MrNo	No
MrCancel	Cancel
MrIgnore	Ignore
MrAll	All

Сұхбаттық терезе өрісіне шығару

Ақпаратты шығаруға арналған сұхбаттық терезенің бір бөлігі – **шығару өрісі (Label** компоненті) немесе таңбалар (меткалар) өрісі деп аталады.

Шығару өрісінің құрамы **Caption** қасиетінің мәнімен анықталады. **Caption** қасиеті мәндерін қосымшаның формасын өңдеуде де, сонымен қатар бағдарламаның жұмыс сәтінде де өзгертуге болады. Мәні – символдық тип. Сондықтан да бағдарламаның жұмысы кезінде тамғалар өрісінде сандық мән шығару үшін санды қатарға түрлендіру қажет болады. Түрлендіруде FloatToStr немесе IntToStr функциялары қолданылады.

Осы айтылғандарға сәйкес мынадай мысалды көрсетуге болады:

```
Label2.Caption:=FloatToStr(kg)+ " кг";
```

Бағдарлама нұсқауларының жазбасы

Бағдарламадағы нұсқаулар басқа нұқсаулардан немесе символдардан нүктелі үтір арқылы ажыратылып жазылады. Әр нұсқаудың соңына «;» символын қойып кету қажет. Бір қатарға бірнеше нұсқауды жазуға болады, дегенмен де әр нұсқауды әр қатарға шегініс арқылы жазған жөн.

Кейбір нұсқауларды (**if, case, repeat, while** және т.б.) бірнеше қатарға шегініс арқылы жазу қабылданған.

Төменде осы айтылғандарға мысал келтірелік:

```
if d >= 0
        then
        begin
x1 :=(-b+sqrt(d)) / (2*a);
x2 :=(-b-sqrt(d)) / (2*a);
ShowMessage ('x1=' +FloatToStr(x1)+
 'x2='+FloatToStr(x2));
        end;
```

```
else
ShowMessage ('теңдеудің түбірі жоқ. ');
```

Бұл бағдарламаны шегініссіз бір қатарға былайша жазып көрелік:

Мыс:

```
if d >= 0

then
begin
x1:=(-b+sqrt (d)) / (2*a);
x2:=(-b-sqrt(d)) / (2*a);
ShowMessage ('x1='+FloatToStr(x1)+
+ 'x2='+FloatToStr(x2));
end;
else ShowMessage ('теңдеудің түбірі жоқ.');
```

Дегенмен де бірінші нұсқаны түсіну екіншіге қарағанда жеңілірек.

Ұзын өрнекті бірнеше қатарға бөліп жазуға болады. Айнымалыларды, сандық және қатарлық тұрақтыларды, сонымен қатар құрамдас операторларды (мысалы, меншіктеу операторы) бөліп жазуға болмайды.

Мысалы:

```
St:='теңдеу түбірлері'+ #13
+'x1=', FloatToStr(x1)+ #13
+'x2=', FloatToStr(x2));
```

Компилятор жұмысқа қосылған кезде бос қалдырылған орындар мен шегіністерді елемейді. Арифметикалық және логикалық өрнектерді, параметрлер тізбектерін жазған кезде бос аралық (пробел) қалдырып отыру талап етілмейді. Дегенмен де бос аралықты қолданып жазылған бағдарламаны түсінудің жеңіл екеніне көзіміз жеткен тәрізді. Төмендегі меншіктеу нұсқауының жазылуын салыстыралық:

```
x1:=(-b+sqrt (d)) / (2*a);
```

және

```
x1 := (-b + sqrt (d)) / (2 * a);
```

өрине екінші нұсқа көрнекі де, түсінікті.

Бағдарлама жұмысының логикасын түсінуді жеңілдету үшін, бағдарлама мәтініне түсініктемені қосқан жөн. Түсініктеме фигуралы жақшаға алынып жазылады. Егер түсініктеме бір қатарға жеке жазылатын болса немесе нұсқаудан кейін орналасса, онда түсініктеменің алдына екі (//) қисық сызық қойылады.

Төмендегі мысал осыған дәлел болмақ:

```
var
{Тендеу коэффициенттері}
a: real; //белгісіздің екінші дәрежесінде
b: real; //белгісіздің бірінші дәрежесінде
c: real; //белгісіздің 0-ші дәрежесінде

{Тендеу түбірлері}

x1, x2:real;
```

Бағдарламалау стилі

Жұмыс кезінде бағдарламалауды жаңа бастап жүрген бағдарламашы өзі құрып отырған бағдарламаның мағынасын түсіне білуі керек.

Жұмыс тиімді болу үшін бағдарлама оқуға жеңіл, шешілуге тиісті тапсырманың құрылымына және алгоритміне сәйкес болуы керек. Ол үшін бағдарламалау стилінің ережелеріне сүйенген жөн.

Бағдарламалау стилі – бағдарламалау процесінде қолданылатын ережелер жиыны.

Әрине, тәжірибелі бағдарламашы жақсы стильді ережелерге сүйенуі керек.

Жақсы стиль дегеніміз:

– түсініктемелерді қолдану;
– айнымалыларға, процедураларға және функцияларға мағыналы аттар беру;
– шегіністерді пайдалану;
– бос қатарларды қолдану.

Бағдарламалаудың жақсы стилін қолданғанда мәтінді теру кезеңіндегі қателердің пайда болу ықтималдылығы едәуір азаяды және түзету немесе өзгеріс енгізу процестері жеңілдейді.

Бағдарламалау стилі бағдарлама мәтінін жазу ережесіне арналған деп бір жақты түсінік беру, әрине қате пікір болар еді.

Жақсы стиль, сонымен қатар бағдарлама жұмысында да ескерілуі қажет.

Осы ұстанымдар негізінде құрылған бағдарлама әрі сенімді, әрі қолданушыға түсінікті болмақ.

Сенімділік – бағдарлама бастапқы деректерді бақылайды, қандай да бір себептермен орындалмай қалған операциялардың орындалу нәтижелерін тексеріп отырады.

Ыңғайлылық – тиімді жобаланған сұхбаттық терезелердің, анықтамалық жүйелердің қызметтерімен толықтырылмақ.

3.2.3. Тіл операторлары

Тіл операторлары есепті шешу үшін орындалуы қажет қандайда бір алгоритмдердің іс-әрекеттерін анықтайды. Бағдарлама немесе ішкі бағдарлама құрылымын осындай операторлар тізбегі ретінде қарастыруға болады. Бірінен кейін бірі орындалатын бағдарлама операторы нүктелі үтірмен бөлінеді. Паскаль тілінің барлық операторларын екі топқа бөлуге болады, қарапайым және құрылымдық.

Қарапайым операторлар

Өзінің құрамында басқа операторлары болмайтын операторлар қарапайым операторлар болып табылады. Оларға:

– меншіктеу операторы;

– GOTO шартсыз өту операторы;

– бос операторлар жатады.

Меншіктеу операторы

Меншіктеу операторы былайша орындалады:

– Меншіктеу операторы символының оң жағындағы өрнек мәні бірінші есептелінеді.

– Есептелген мән айнымалыға жазылады, айнымалы аты меншіктеу операторы символының оң жағына орналасқан.

Аты: = өрнек,

Мұндағы

Аты : = айнымалы (өзгеруі мүмкін).

Бұл операция үшін «:=» меншіктеу таңбасы пайдаланылып, оның сол жағында мән меншіктелінетін айнымалы немесе функция аты жазылады, ал оң жағында өрнек жазылады.

Айнымалы мен функцияларға файл типінен басқа кез келген типті мәндерді меншіктеуге болады. Өрнек типі мен айнымалы (немесе функция) типі меншіктеу үшін бір-бірімен сәйкес болуы керек.

GOTO шартсыз өту операторы. Белгілерді қолдану

GOTO операторы операторлар орындалуының тізбекті ретін өзгертуге және көрсетілген оператордан бастап бағдарламаны орындауға ауысуға мүмкіндік береді. Ауысу орындалатын оператор белгіленуі керек. Осы белгі GOTO операторында көрсетілуі қажет.

Object Pascal-да пайдаланатын белгілер екі типті болуы мүмкін:

– *0-ден 9999-ға дейінгі бүтін сандар;*

– *жай атаулар.*

Барлық пайдаланатын белгілер, **Label** тіркелген сөзімен басталатын белгілерді баяндау бөлімінде көрсетілуі қажет.

Мысалы:

Label, 1,2, Label 1;

Бір белгімен тек бір операторды белгілеуге болады. Белгі, белгіленген оператордан қос нүкте арқылы бөлінеді.

Құрылымдық операторлар

Құрамында басқа операторлар қамтылатын операторлар құрылымдық болып табылады. Оларға:

– құрамдас оператор;
– **IF** шартты операторы;
– **CASE** таңдау операторы;
– **REPEAT** цикл операторы;
– **WHILE** цикл операторы;
– **FOR** цикл операторы;
– **WITH** жазбалар операторы жатады.

Құрамдас оператор

Begin және **end** операторлық жақшаларға алынған, тізбекті түрде орындалатын операторлар жиынтығы құрамдас оператор болып табылады;

Begin

<1-ші оператор>;
<2-ші оператор >;
.....
<N-ші оператор >;
end;

Тілдің құрылымын жасау ережелеріне сәйкес бірнеше іс-әрекетті орындау талап етілсе, сонымен қатар тек бір операторды ғана орындауға рұқсат етілген жағдайда осындай оператор қолданылады.

Келешекте бір операторды пайдалануға болады деп көрсетілген жерлерде құрамдас операторларды пайдалануға болады.

Бағдарлама денесінің өзі де **Begin** және **End** операторлық жақшаларға алынғандықтан, құрамдас оператор деп есептеуге болады.

Шарт

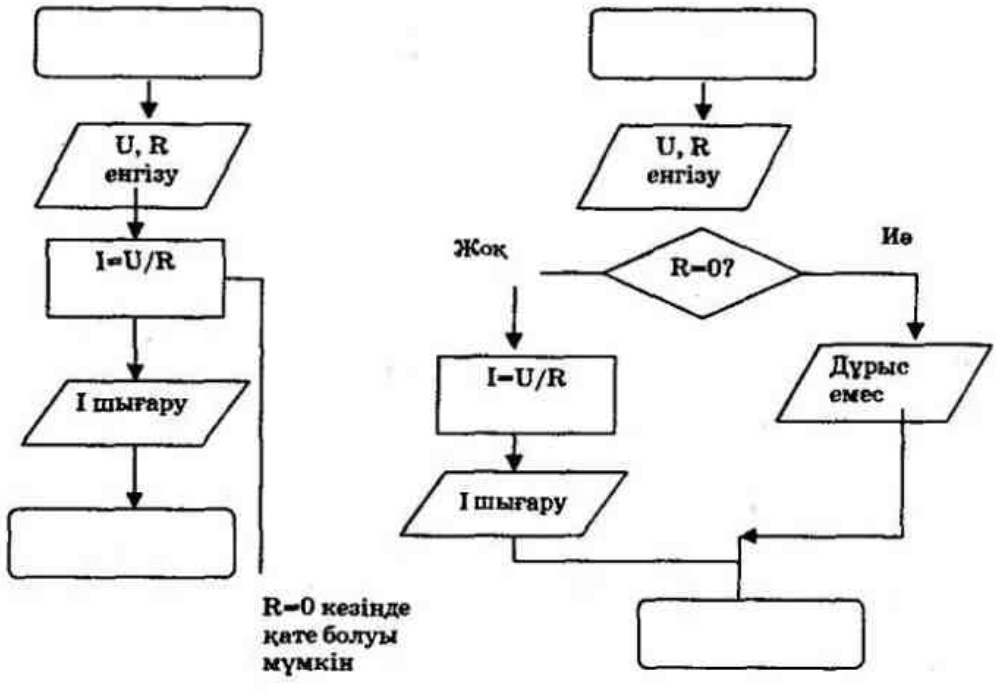
Күнделікті өмірде шарт сұрақ түрінде қалыптасады, оған әрине **Иә** немесе **Жоқ** жауаптарының бірі берілері сөзсіз. Мысалы:

Кедергі шамасы 0-ге тең бе?

Жауап дұрыс па?

Сатып алу бағасы 300 теңге болады ма?

Бағдарламадағы шарт – логикалық типті өрнек **(Boolean)**, өз кезегінде ақиқат **(True)** немесе жалған **(False)** екі мәнінің бірін қабылдайды.



33-сурет. Бір тапсырманы шешудің екі нұсқасы

Қарапайым шарт екі операндтан және салыстыру операторынан тұрады. Жалпы түрде ол былай жазылады.

Оп1 Оператор Оп2

Мұндағы *Оп1* және *Оп2* – шарт операндтары. Операнд ретінде айнымалыларды, тұрақтыларды, функцияларды немесе өрнектерді қолдануға болады.

Оператор – салыстыру операторы.
Turbo Pascal-да алты салыстыру операторы бар:

21-кесте

Оператор	Сипаттамасы	Салыстыру нәтижесі
>	Үлкен	Оп1>Оп2=True, қарама-қарсы жағдайда False
<	Кіші	Оп1<Оп2=True, қарама-қарсы жағдайда False
=	Тең	Оп1=Оп2=True, қарама-қарсы жағдайда False
<>	Тең емес	Оп1≠Оп2=True, қарама-қарсы жағдайда False
>=	Үлкен немесе тең	Оп1>=Оп2=True, қарама-қарсы жағдайда False
<=	Кіші немесе тең	Оп1<=Оп2=True, қарама-қарсы жағдайда False

Төменде шартқа мысал берілген:

Summa <= 1000 (операнд = айнымалы не тұрақты)
Score >= HBound
Sim = Chr(13)

Бірінші мысалда шарт операндтары айнымалы және тұрақты болып табылады. Бұл шарттың мәні *Summa* айнымалысының мәніне тәуелді. Егер осы шарт қанағаттандырылатын болса, онда шарт дұрыс, яғни *True* мәнін қайтарады. Егер берілген шарт қанағаттандырылмаса, яғни *Summa* айнымалысы *1000* санынан үлкен немесе оған тең болса, онда жалған (False) шығарылмақ.

Екінші мысалда операнд ретінде айнымалы қолданылады.

Үшінші мысалда операнд ретінде функция қолданылады. Бұл шарттың мәні мынадай жағдайда ақиқат болады: егер Sim айнымалысында 13-ке тең <Enter> пернесінің символдық коды бар болса. Қарама-қарсы жағдайда жалған мәнін бермек.

Қарапайым шарттардан логикалық **and** (және), **or** (немесе), **not** (терістеу) операторларының көмегімен күрделі шарттар қалыптастыруға болады.

Күрделі шарттар былай жазылады:

Шарт1 оператор шарт2

Мұндағы *шарт1* және *шарт2* – жай шарттар (логикалық типті өрнек); Оператор – **AND** немесе **OR** операторы.

22-кесте

Логикалық типті операндтарға логикалық операторлардың қолданылуы

On1	On2	On1 and On2	On1 or On2	Not On1
F	F	F	F	T
F	T	F	T	T
T	F	F	T	F
T	T	T	T	F

Заттың құнын арзандатып сату жексенбі (7) күні болады және сатып алушы 100 теңгеден көп ақша жұмсайды. Бұл шарт бағдарламада былай көрсетіледі:

Мыс: *(Summa > 100) and (Day =7)*

Шартты өзгертелік, заттың бағасын түсіріп сату кез келген күні болмақ, егер сатып алатын заттың құны 500 теңгеден асса:

((Summa >100) and (Day =7)) or (Summa > 500)

Таңдау

Тармақталу алгоритмінде бағдарламаның кезекті қадамы **if** және **case** нұсқауларының көмегімен жүзеге асырылады. **If** операторы мүмкін болатын екі оператордың бірін таңдауға мүмкіндік берсе, **case** оператор бірнеше операторлардан біреуін таңдауға мүмкіндік береді.

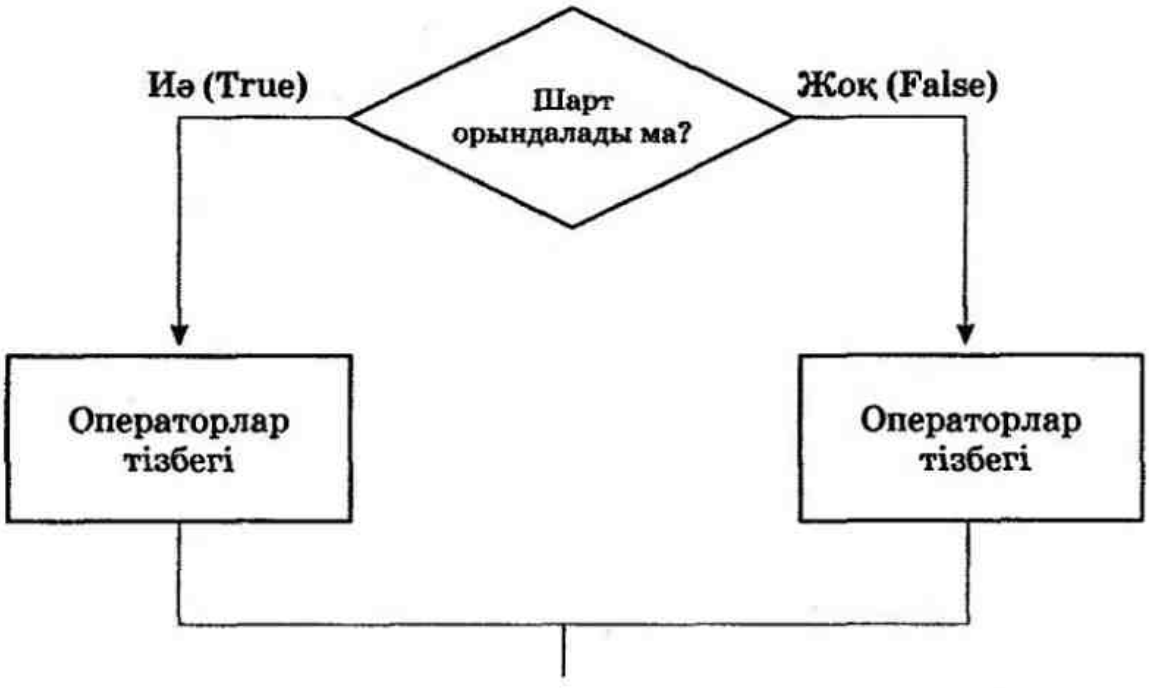
IF функциясы

Бағдарламаның жұмысы барысында екі нұсқаның біреуін таңдауға мүмкіндік беріледі. Таңдау шарттың орындалуына байланысты жүзеге асырылады. Жалпы түрде **IF** шарты былай жазылады:

```
if шарт
    then
        begin
        //нұсқау
    end
  else
begin
 //егер шарт жалған болса
 //орындауға тиісті нұсқаулар
end;
```

if операторы былай орындалады:

1. Шарт мәні есептеледі.
2. Егер шарт шынайы болса, **then**-нен кейінгі оператор орындалады.
3. Егер шарт жалған болса, **else** сөзінен кейінгі оператор орындалады.



34-сурет. **if – then – else** операторының алгоритмі

Оператордың екі нұсқасы бар:

If S then A else B; (толық айрық)

және

If S then A (қысқартылған айрық)

мұнда *S* – ақиқаттылығы тексерілетін, логикалық типті шаманы қалыптастыратын қандай да бір өрнек; *A* – *S* өрнегі ақиқат болған жағдайда орындалатын оператор; *B* – *S* өрнегі жалған болған жағдайда орындалатын оператор.

Шартты оператордың екінші нұсқасын, өрнек жалған болса, ешқандай іс-әрекет орындалмайды.

Операторды пайдалануға мысалдар:

```
If X <0 then X:=-У;

If X <1.5 then X:=-У
Else X:=1.5;
```

CASE таңдау операторы

Осы оператор көмегімен нұсқалардың кез келгенін таңдауға болады. Бұл операторды If операторынан айырмашылығы, бірнеше нұсқадан таңдалады.

Object Pascal-да бұл операторды құрылымы:

Case S of
C1:statement1;
...
CN: statementN;
Else statement;
end;

Мұнда *S* – мәні есептелінетін, кез келген реттік типті өрнек (таңдау параметрлері);

C1...CN – S өрнегінің мәні салыстырылатын тұрақтылар. Бұл тұрақтылардың ішінде бірдей тұрақтылар болмауы қажет; *statement1,..., statementN – S* өрнегінің мәні тұрақтыға сәйкес келетін оператор орындалады; *statement-S* өрнегінің мәні *C1,...,CN* тұрақтыларының бірде-бірімен сәйкес келмеген жағдайда орындалатын оператор.

else операторы бұтағының болуы міндетті емес. Егер ол болмаса және де *S* өрнегінің мәні бірде-бір тұрақтымен сәйкес келмесе, оператор бос деп қаралады.

Егер бірнеше тұрақтылар үшін бір операторлың орындалуы қажет болса, оларды үтір арқылы, бір операторға жазуға болады.

Мысал:

Case of

0,2,4,6,8:label1.caption:= 'Жұп сан';

1,3,5,7,9:label1.caption:= 'Тақ сан';
10..100:label1.caption:= 'Оннан жүзге дейінгі сандар';

Else

label1.caption:= 'Теріс сан немесе жүзден артық';

end;

Case операторы былайша орындалады:

Алдымен селектр – өрнек мәні есептеледі.

Селектр – өрнек мәні тұрақтылар тізіміндегі тұрақтымен біртіндеп салыстырылады.

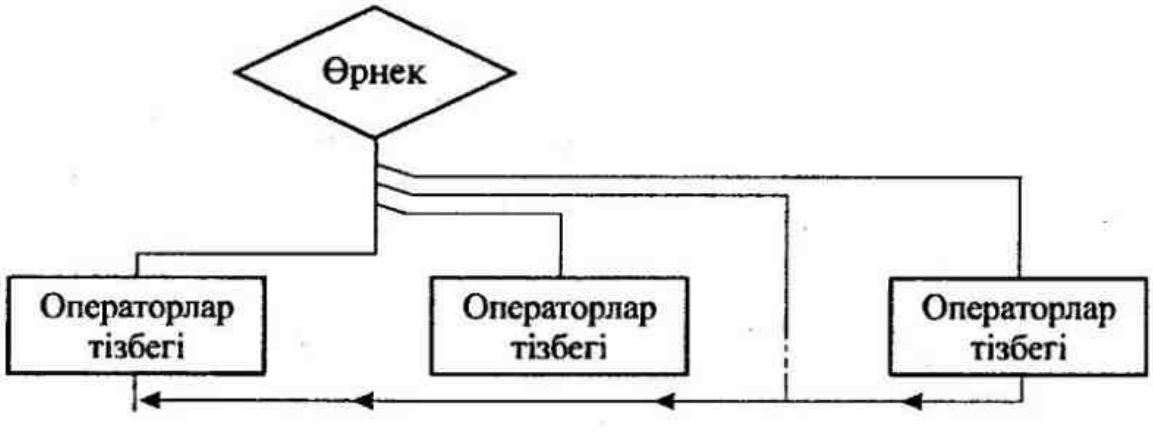
Егер өрнек мәні тізімдегі тұрақтымен сәйкес болса, онда осы тізімге сәйкес операторлар тобы орындалады. Осы кезде **Case** операторының орындалуы аяқталады.

4. Егер селектр – өрнек мәні барлық тізімдегі бірде-бір тұрақтымен сәйкестендірілмесе, онда **else**-тен кейінгі бағдарлама операторлары орындалмақ.

Case операторының синтаксисі **else**-ті және онан кейінгі орындалатын операторларды жазуды қажет етпейді.

Бұл жағдайда, егер өрнек мәні тізімдегі бірде-бір тұрақтымен сәйкес келмесе, онда **Case** -тен кейінгі операторлар орындалады.

Төмендегі суретте Case операторының жұмыс жасау принципі берілген.



35-сурет. Case операторының жұмыс жасау алгоритмі

Мысал: **case** n_day **of**
1,2,3,4,5: day:='Жұмыс күні.';
6: day:= 'Сенбі!';
7: day:= 'Жексенбі!';
end;

case n_day **of**
1..5: day:= 'Жұмыс күні.';
6: day:= 'Сенбі!';
7: day:= 'Жексенбі!';
end;
case n_day case
6: day:= 'Сенбі!';
7: day:= 'Жексенбі';
else day:= 'Жұмыс күні.';
end;

Циклдар

Көптеген тапсырмалардың шешімін табу алгоритмі циклдық болып келеді, яғни нәтижеге жету үшін кейбір әрекеттер тізбегін бірнеше рет орындауға тура келеді.

Бірнеше рет орындалатын операциялар жиыны **циклдық** деп аталады, ал сол операциялар жиыны **цикл** деп аталады.

Бағдарламада цикл **for, while, repeat** операторларының көмегімен жүзеге асырылады.

FOR цикл операторы

Мынадай тапсырманы қарастырайық.

$y=5*2-7$ функциясының -1, -0.5, 0, 0.5 және 1 нүктелеріндегі мәнін есептеу бағдарламасын жазу керек. Қойылған тапсырманың шешімін табуды қамтамасыз ететін процедура мынадай көрініске ие:

procedure TForm1.Button1Click(Sender: TObject);

```
var

y: real; // функция мәні
x: real; // функция аргументі
dx: real; // аргумент кеңейтуі (приращение)
st: string; // кестені бейнелеу

begin
st:= ' ';
x:= -1;
dx:= 0.5;
```

$$\begin{cases} y := 5*x*x-7; \\ st := st + FloatToStr(x)+'\ '+FloatToStr(y)+chr(13); \\ x := x+dx; \end{cases}$$

$$\begin{cases} y := 5*x*x-7; \\ st := st + FloatToStr(x)+'\quad '+FloatToStr(y)+chr(13); \\ x := x+dx; \end{cases}$$

$$\begin{cases} y := 5*x*x-7; \\ st := st + FloatToStr(x)+'\ '+FloatToStr(y)+chr(13); \\ x := x+dx; \end{cases}$$

$$\begin{cases} y := 5*x*x-7; \\ st := st + FloatToStr(x)+'\ '+FloatToStr(y)+chr(13); \\ x := x+dx; \end{cases}$$

$$\begin{cases} y := 5*x*x-7; \\ st := st + FloatToStr(x)+'\ '+FloatToStr(y)+chr(13); \\ x := x+dx; \end{cases}$$

```
Label1.Caption:=st;
End;
```

Мұндағы фигуралы жақшаға алынған операторлар тобы бағдарламада бірнеше рет қайталанған.

For операторын қолданып бұл процедураны былай жазуға болады.

```
procedure TForm1.Button1Click(Sender: TObject);
var
y: real; // функция мәні
x: real; // функция аргументі
dx: real; // аргумент кеңейтуі
st: string; // кестені бейнелеу
i: integer: // цикл санағыш

begin
st:= ' ';
x:= -1;
dx:= 0.5;

for i:=1 to 5 do
begin
y:=5*x*x-7;
st:=st+FloatToStr(x)+ ' '+FloatToStr(y)+chr(13);
x:=x+dx;
end;
Label1.Caption:= st;
end;
```

Шығарылатын кестедегі санды ұлғайту үшін *For i:=1 to 5 do* қатарындағы 5-ті 10-ға айырбастаса жеткілікті.

Жалпы түрде **for** операторы былай жазылады:

For санағыш:=бастапқы_мән *to* соңғы_мән *do*

```
Begin
 //операторлар
```

End.

Мұндағы:

• Санағыш – цикл операторының қайталау санын санаушы – айнымалы.

• Бастапқы_мән – цикл санағышының бастапқы мәнін анықтайтын өрнек;

• Соңғы_мән – цикл санағышының соңғы мәнін анықтайтын өрнек.

Санағыш, бастапқы мән, соңғы мән – мәндері бүтін типті болу керек.

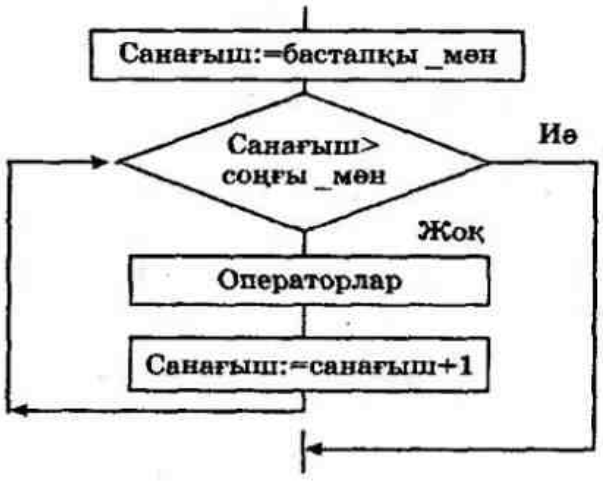
Цикл нұсқауының қайталау санын мына формула бойынша есептеуге болады;

(ақырғы_мән – бастапқы_мән+1)

For цикл операторы бір операторды алдын-ала берілген белгілі сан бойынша бірнеше рет орындауды ұйымдастырады. Операторлың екі нұсқасы бар:

For param:=Start to Finish do statement;
For param:= Finish downto Start do statement;

Мұндағы, *param* – реттік типті айнымалы болып табылатын цикл параметрі; Start – параметрдің бастапқы мәні; Finish – цикл параметрінің соңғы мәнін анықтайтын өрнек; statement – цикл денесі болып табылатын, орындалатын оператор. Start және Finish бір типті болуы керек.



36-сурет. For операторының алгоритмі

While цикл операторы

While циклі кейбір әрекеттерді бірнеше рет қайталау барысында қолданылады, қайталау саны бағдарламаны өңдеу кезінде белгісіз болып, тек бағдарлама жұмысының барысында ғана анықталуы мүмкін.

While операторының жазылу түрі:

While шарт do
//бірнеше рет орындалатын оператор

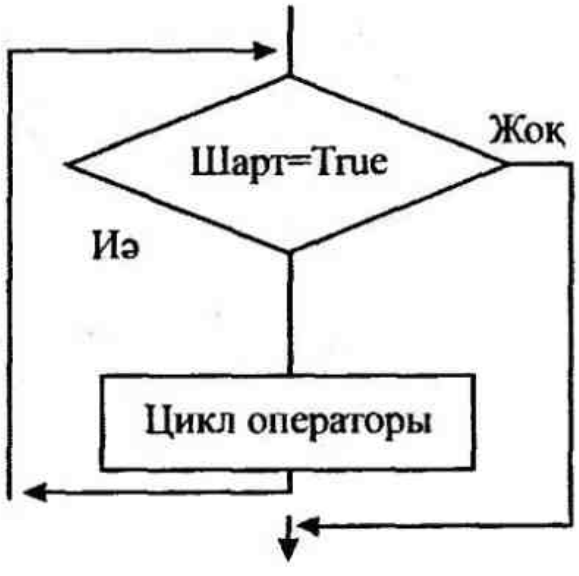
end;

Мұндағы, *шарт* – цикл операторының орындалу шартын анықтайтын логикалық типті өрнек.

While операторы былайша орындалады:

1. Шарт өрнегінің мәні есептеледі.
2. Егер Шарт=**False** болса, онда өрнек орындалмайды. **While** операторының орындалуы аяқталды.

3. Егер Шарт = **True** болса, онда шарттан кейін орналасқан өрнек орындалады, **begin** мен **end** аралығындағы цикл денесінің операторы орындалады деген сөз.



37-сурет. WHILE операторының алгоритмі

Егер логикалық өрнек жалған болса, циклдан шығу орындалады. Логикалық өрнектің ақиқаттылығы цикл-дың басында тексерілетіндіктен, цикл денесі бірде-бір рет орындалмауы мүмкін.

Оператордың құрылымы келесі түрде болады:

While S do
Statement;

Мұндағы *S* – әрбір итерацияның басында ақиқатты-лығы тексерілетін логикалық өрнек;

Statement – орындалатын оператор.

Мысалы:

```
unit pi_;

interface

uses
Windows, Messages, SysUtils, Classes, Graphics, Con-
trols, Forms, Dialogs, StdCtrls;

Type
TForm1=class(TForm)
Edit1: TEdit;   //Есептеу дәлдігі
Button1: TButton; //Есептеу батырмасы
Label1: TLabel;
Label2: TLabel; //Нәтижені шығару өрісі

procedure Button1Click (Sender: TObject);
private
   {Private declarations}
public
   { public declarations}
end;
```

```
var
Form1:TForm1;
implementation

{$R *.DFM}

procedure TForm1. Button1Click(Sender: TObject);

var

pi: real; //ПИ есептелетін мәні
t:real;   // есептеу дәлдігі
n: integer;      //қатар мүшесінің нөмірі
elem: real;      //қатар мүшесінің мәні

begin
pi:= 0;
n:= 1;
t:= StrToFloat (edit1.text);

elem:=1; //циклды бастау үшін

while elem >= t do

begin
elem:= 1 / (2*n - 1);
if n MOD 2 = 0
then pi:= pi-elem;
else pi:= pi+elem;

n:= n+1;
end;

pi:=pi * 4;
label1.Caption:= ' ПИ тең '+ FloatToStr(pi) + #13+
'Қосындылардың '+IntToStr(n) + 'қатар мүшелері.';

end;
end.
```

REPEAT цикл операторы

Қайталау операторы. Қайталау саны бағдарламаның жұмыс уақытында белгілі болады.

Жалпы түрде Repeat нұсқауы мынадай түрде жазылады:

Repeat
//нұсқаулар
Until шарт

Мұндағы *шарт* – логикалық типті, циклдың аяқталуын анықтау шарты болмақ.

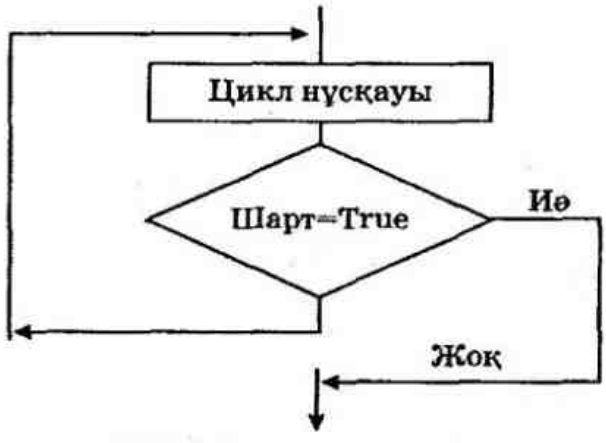
Орындалу тәртібі:

1. Алдымен **Repeat** және **Until** арасындағы цикл денесінің операторлары орындалады.

2. Сонан соң шарт тексеріледі. Егер шарт жалған болса (шарт өрнегінің мәні **False** мәнін қамтиды), онда цикл денесінің операторлары тағы да бір рет орындалады.

3. Егер шарт ақиқат болса (шарт өрнегінің мәні **True** мәнін қамтиды), онда циклды орындау тоқтатылады.

Осылайша **Repeat** және **Until** арасында орналасқан операторлар шарт жалған болғанға дейін орындала береді.



38-сурет. REPEAT операторына сәйкес алгоритм

REPEAT цикл операторы – операторлардың кез келген санынан тұратын, қайталау саны алдын-ала белгісіз болатын циклдың орындалуын ұйымдастырады. Циклдың денесі кем дегенде бір рет орындалады. Цикл денесінің 1 рет орындалуын итерация деп атайды. Логикалық типті шаманы қалыптастыратын өрнек ақиқат болғанда циклдан шығу орындалады.

Оператордың құрылымы:

Repeat
statement1;
statement 2;
.....
statementN;
Until S;

мұндағы *statement1*, *statement2* және *statementN* – цикл денесін құрайтын орындалатын операторлар; *S* – әрбір итерацияның соңында ақиқаттылығы тексерілген логикалық өрнек.

Цикл операторларында Break және Continue стандартты процедураларды қолдану

Object Pascal-да **REPEAT**, **WHILE** және **FOR** циклдарында **Break** және **Continue** стандартты процедураларды пайдалануға болады. **Break** процедурасы шығу шартының орындалуын күтпей, циклдан шығуға мүмкіндік береді.

Continue процедурасы алдыңғы итерация аяқталмаса да, циклдың жаңа итерациясын бастауға мүмкіндік береді.

3.3. Массивтер

Бір өлшемді массив

Массив – ортақ атпен аталатын, бір типті айнымалылар жиынын көрсететін деректер құрылымы.

Массивті баяндаудың екі түрі бар:
1. **Type** арқылы енгізу.
2. Айнымалылар тарауында тікелей баяндау.

Синтаксисі:

Аты: array [төм инд.. .жоғ инд] of типі;

Мұндағы: *Array* – Object Pascal -дың қордағы сөзі, массивті баяндау үшін қолданылады.

Мыс:
temp: array [1..31] of real;
kofe: array [0..2] of integer;
name: array [1..30] of string [25];

Массив элементтері өзгеріссіз тұрақты мән болатын болса, сол мәнді тұрақтылар **(const)** тарауында беріп кеткен дұрыс.

Мыс:

Const N=18;
Var A:array [1..N] of integer;
.....

Массивтермен орындалатын операциялар

Массивтермен жұмыс кезіндегі типтік операцияларға мыналар жатады:

- Массивті шығару;
- Массивті енгізу;
- Массивтің **max** және **min** элементтерін іздеу;
- Массивтің берілген элементін іздеу;
- Массивті сұрыптау (өсу немесе кему реті бойынша).

Массивтің барлық элементтерін шығаруда **for** операторын пайдаланған ыңғайлы.

Мысалы:
For i:=1 to n do
<оператор>;

StringGrid компонентін қолдану

Массивті енгізу үшін **StringGrid** компонентін пайдаланған ыңғайлы. **StringGrid** компонент белгісі **Additional** салымында орналасқан.

StringGrid компоненті ұяшықтары символдар қатарын қамтитын кестеден тұрады.

Төмендегі кестеде StringGrid кейбір компонентінің қасиеттері берілген.

23-кесте

Қасиеттері	Анықтайды
1	2
Name	Компонент аты. Бағдарламада компонент қасиеттеріне қол жеткізу үшін қолданылады.
ColCount	Кесте бағандарының саны
RowCount	Кесте қатарының саны
Cells	Кестеге сәйкес екі өлшемді массив. Col баған нөмірі және Row қатар нөмірі бойынша қиылыста орналасқан кесте ұяшығы Cells[Col, Row] элементімен анықталады.
FixedCols	Кесте бағандарының саны (сол жағында бекітілген)
FixedRows	Кесте қатарының саны (жоғарғы жағында бекітілген)
Options.goEditing	Кесте ұяшықтары құрамындағыларды редакциялау мүмкіндігінің белгісі. True – редакциялауға рұқсат, False – тыйым салынған.
Options.goTab	Меңзерді кестенің келесі ұяшығына көшіру үшін <Tab> пернесін қолдануға рұқсат (True) немесе рұқсат емес (False).
Options.GoAlwaysShow Editor	Редакциялау режимінде компонентті табу белгісі. Егер қасиет мәні False болса, онда ұяшықта меңзердің пайда болуы үшін, мәтінді теріп бастау керек, <F2> пернесін басу керек немесе тінтуірді басу керек.
DefaultColWidth	Кесте бағандарының ені
DefaultRowHeight	Кесте қатарының биіктігі
GridLineWidth	Кесте ұяшығын шектеуші сызық ені
Left	Кесте өрісінің сол жақ шекарасынан форманың сол жақ шекарасына дейінгі аралық

1	2
Top	Кесте өрісінің жоғарғы шекарасынан форманың жоғарғы шекарасына дейінгі аралық
Height	Кесте өрісінің биіктігі
Width	Кесте өрісінің ені
Font	Кесте ұяшығындағылардың сипаттамаларын көрсету қарпі
ParentFont	Форма қаріптерінің сипаттамаларын иемдену белгісі

Memo компонентін қолдану

Кейбір жағдайда массивті енгізу үшін **Memo** компоненті қолданылады.

Memo компоненті бірнеше қатарлардан тұратын мәтінді енгізуге ыңғайлы. Сондықтан да символдық массивті енгізуде тиімді болмақ.

Memo компонентін формаға енгізу үшін, **Standart** салымынан осы компоненттің белгісін таңдаймыз.

Memo компонентінің қасиеттері:

1. Name – компонент аты.

2. Text – Memo өрісіндегі мәтін.

3. Line – Memo өрісіндегі мәтін. Қатарлар жиыны ретінде қарастырылады. Қатарға қатынас нөмірі бойынша құрылады.

4. Lines.Count – Memo өрісіндегі мәтіндік қатарлар саны.

5. Left – Өрістің сол жақ шекарасынан форманың сол жақ шекарасына дейінгі аралық (сол жақтағы).

6. Top – Өрістің жоғарғы шекарасынан форманың жоғарғы шекарасына дейінгі аралық.

7. Height – өріс биіктігі.

8. Width – өріс ені.

9. Font – енгізілген мәтін қарпі.

Memo компонентін қолданған кезде массивті енгізу үшін массивтің әрбір элементінің мәнін жеке қатарда енгізу қажет, сонан соң **Enter** пернесін басқан жөн.

Мыс: {1↵ 2↵ 0↵ 3↵

Memo өрісіндегі мәтінге **Lines** қасиетінің көмегімен қатынас құруға болады. Ол үшін қажетті қатардың нөмірін квадрат жақшаға алып жазу қажет.

Lines [i-1]

Символдық массивті енгізу процедурасының негізгі циклы:

For i:=1 to SIZE do
a[i]:=Memo1. Lines [i];

Мұндағы *SIZE* – массив өлшемін анықтайтын атауы тұрақты.

a – массив.

Memo1 – **Memo** компонентінің аты. **Lines** – **Memo** компонентінің қасиеті.

Көп өлшемді массив

Екі өлшемді массивті баяндаудың нұсқауы:

Аты: ***array[*** *төменгі_шекара1 .. жоғарғы_шекара1, төменгі_шекара2 .. жоғарғы_шекара2]* ***of*** *типі.*

Мұндағы Аты – массив аты,

Array – массивті білдіретін кілттік сөз,

Төменгі_шекара1,2 және *Жоғарғы_шекара1,2* – бүтін типті тұрақтылар,

Тип – массив элементтерінің типтері.

Екі өлшемді массивтегі элементтер санын мына формуламен есептеуге болады:

(ЖШ1-ТШ1+1)(ЖШ2-ТШ2+1);* Мұндағы:

ЖШ1, ЖШ2 – 1-ші және 2-ші индексті жоғарғы шекара:

ТШ1, ТШ2 – 1-ші және 2-ші индексті төменгі шекара.

Екі өлшемді массивтерді енгізуде екі айнымалы қолданылады да, ол мынадай түрде беріледі:

Var a: array[1..n,1..n] of integer;

Массивтің барлық элементтерін шығаруда **For** нұсқауын пайдаланған ыңғайлы.

Мысалы:

For i:=1 to n do
For j:=1 to n do
<оператор>;

3.4. Ішкі бағдарламалар

Ішкі бағдарлама дегеніміз – логикалық тұжырымдалған және арнайы түрде безендірілген операторлар топтары. Ішкі бағдарламаны бағдарламаның әр жерінен бірнеше мәрте (қажетіне қарай) шақыруға болады. Ішкі бағдарламаны қолдану бағдарламаның құрылымдылығын жақсартып, оның көлемін кішірейтуге мүмкіндік береді.

Құрылымы бойынша ішкі бағдарлама қарапайым бағдарламаға ұқсас, онда да бағдарлама тақырыбы және блоктар қамтылған, дегенмен ішкі бағдарлама блоктарында модульдерді қамту бөлімі болмайды. Сонымен қатар, ішкі бағдарламаның тақырыпшасы өзінің жазылуы бойынша қарапайым бағдарламаның тақырыпшасынан ерекшеленеді.

Ішкі бағдарламамен жұмыс екі кезеңге бөлінеді:

– ішкі бағдарлама сипаттамасы;
– ішкі бағдарламаны шақыру.

Кез келген ішкі бағдарлама алдын ала сипатталады, сонан соң ғана оны шақыруға болады. Ішкі бағдарламаны сипаттаған кезде оның аты, параметрлер тізімі және ішкі бағдарламамен орындалатын әрекеттер анықталады.

Шақыру кезінде ішкі бағдарлама аты және жұмыс үшін ішкі бағдарламаға берілетін аргументтер тізімі (нақты параметрлер) көрсетіледі. Delphi түрлі модульдерінде көптеген стандартты ішкі бағдарламалар бар, оларды алдын ала сипаттаусыз шақыруға болады. Сондай-ақ, бағдарламашы өзінің ішкі бағдарламасын құра алады. Оны *қолданушы* ішкі бағдарлама деп атайды.

Ішкі бағдарламалар *процедуралар* және *функциялар* болып екіге бөлінеді. Олардың ортасында ортақ қасиеттер көп. Айырмашылығы, функция өз жұмысының нәтижесі ретінде өз атымен кейбір мәндерді қайтара алады, операнд ретінде өрнекті қолдана алады. Ішкі бағдарламамен әрекеттесу *басқару* бойынша және *деректер* бойынша жүзеге асырылады. *Басқару* бойынша әрекеттесу – басқаруды бағдарламадан ішкі бағдарламаға беру және бағдарламаға қайтып оралуды қамтамасыз ету.

Деректер бойынша әрекеттесу – белгілі бір әрекеттер орындауға қатысатын деректерді ішкі бағдарламаға берумен байланысты. Әрекеттесудің бұл тәсілі мынадай тәсілдермен жүзеге асырылады:

– файлдарды пайдалану арқылы;
– ғаламдық желі айнымалылардың көмегімен;
– параметрлер көмегімен.

Соңғы тәсіл жиі қолданылады. Параметрлер мен аргументтерге түсінік берелік: *Параметрлер* (формалды параметрлер) ішкі бағдарламаның элементі болып табылады, ішкі бағдарламамен орындалатын операцияны сипаттау кезінде қолданылады.

Аргументтер (нақты аргументтер) шақырушы бағдарламаның элементтері болып табылады. Ішкі бағдарламаны шақырған кезде олар формалды параметрлерді ауыстырады. Бұл кезде параметрлер мен аргументтер типтері, сандары жағынан сәйкестіктері тексеріледі. Параметрлер мен аргументтер аттарында айырмашы-

лықтар болуы мүмкін, бірақ олардың сандары мен орналасу тәртіптері сәйкес болу керек Ал, параметрлер мен оларға сәйкес аргументтердің типтері үйлескен болуы керек.

Ішкі бағдарламамен жұмысты тоқтату үшін *Exit* процедурасын қолдануға болады. Ол ішкі бағдарлама операторларының орындалуын үзеді де, шақырушы бағдарламаға басқару мүмкіндігін қайтарады. Ішкі бағдарламаларды тек бағдарламадан шақырып қана қоймай, оны басқа ішкі бағдарламалардан да шақыруға болады.

3.4.1 Процедуралар

Процедураның баяндалуы оның тақырыпшасынан және блоктан тұрады. Тақырыпшасында *procedure* кілттік сөзі жазылады, дөңгелек жақшада қажетті параметрлер (параметрлердің әрқайсысының типтері көрсетіледі) тізімі беріледі. Форматы мынадай:

Procedure <Аты> [(формалды параметрлер)];

Процедураға қатынас құру үшін процедураны *шақыру операторы* қолданылады. Мысал ретінде *Button1* батырмасын басу оқиғасын өңдеу процедурасын қарастырайық, онда өз кезегінде *DecodeDate* және *ChangeSt.* екі процедурасы шақырылады.

```
Procedure TForm1. Button1Click(Sender: TObject);

//ChangeStr қолданушы процедурасының баяндалуы;

Procedure ChangeStr (var Source: string; const char1,
char2: char);
  Label 10;
  Var n: integer;
  Begin
  10:
  n: =pos (char1, Source);
```

```
if n > 0 then begin
 Source [n] : =char2;
 Goto 10;
 End;
 End;

Var str1: string;
 Year, Month, Day: word;
Begin
// DecodeDate процедурасын шақыру
DecodeDate (Now, Year, Month, Day);
Str1: =Edit1. Text;
// ChangeStr қолданушы процедурасын шақыру
ChangeStr (str1, '1', '*');
Edit1. Text: =str1;
End;
```

DecodeDate процедурасы мерзімді жекелеген құрамдас бөліктерге (жыл, ай, күн) жіктеу қызметін атқарады, ол алдын ала баяндаусыз қолданылуы мүмкін, себебі ол **SysUtils** модулінде баяндалып кеткен.

ChangeStr процедурасы **Source** қатарындағы **char1** параметрі беретін символдарды **char2** параметрімен берілетін символдарға ауыстырады.

ChangeStr процедурасын шақыру **str1** жолындағы 1 символын * символына ауыстырады.

3.4.2. Функциялар

Функция баяндалуы оның тақырыпшасынан және блогынан тұрады. Тақырыпшасы *Function* кілттік сөзін қамтиды, функция аты беріледі, сондай-ақ, дөңгелек жақшаға алынған формалды параметрлердің міндетті емес тізімі және функция қайтаратын мән беріледі. Форматы мынадай:

Function <Аты> [(Формалды параметрлер)]: <Нәтиже типі>;

Қайтаратын тип кез келген тип болуы мүмкін (файлдық типтен басқа).

Блок – шектеулі блок, құрылымы бойынша процедура блогына ұқсас. Функция денесінде ең болмағанда бір меншіктеу операторы болу керек, сол жағында функция аты тұрады. Сол оператор функция қайтаратын мәнді анықтайды. Егер мұндай оператор бірнешеу болса, онда функция нәтижесі соңғы орындалған меншіктеу операторының мәні болмақ. Функцияны шақыру оның аты мен дөңгелек жақшаға алынған аргументтерін қоса шақырумен жүзеге асырылады. Аргументтері болмауы да мүмкін. Аргументтер жұп-жұбымен функция тақырыпшасында көрсетілген параметрлерге сәйкес болу керек. Процедурадан айырмашылығы, функция аты өрнекке операнд ретінде кіруі керек.

Мысал ретінде **Button1** батырмасын басу оқиғасын өңдеу процедурасын қарастыралық, онда екі функция шақырылады: **Length** және **ChangeStr2.**

```
Procedure TForm1. Button1Click(Sender: Tobject) ;
// ChangeStr2 функциясының баяндалуы
Function ChangeStr2 (Source: string; const char1, char2:
char) : string;
Label 10;
Var n: integer;
Begin
Result: =Source;
10:
n: =pos (char1, Result);
if n > 0 then begin
    Result [n]: =char2;
    Goto 10;
    End;
End;

Var str1: string;
 n: integer;
begin
```

```
str1: =Edit1. Text;
 // ChangeStr2 функциясын шақыру
Str1: =ChangeStr2 (str1, '1', '*') ;
Edit1.Text: =str1;
// Length функциясын шақыру
n: =Length (str1);
end;
```

Length функциясы қатар ұзындығын қайтарады және алдын ала баяндаусыз қолданылуы мүмкін, себебі ол **System** модулінде қамтылған. **ChangeStr2** функциясы алдыңғы мысалдағы **ChangeStr** процедурасының қызметін атқарады. Функцияны шақыру меншіктеу операторында қолданылады.

3.5. Файлдар

Файл дегеніміз – атаулы деректер құрылымы, бір типті деректер элементтерінің тізбегі. Ол тізбектегі элементтер саны шексіз. Кез келген бағдарлама деректерінің құрылымы сияқты (айнымалы, массив), файл да айнымалыларды баяндау бөлімінде баяндалып кету керек. Файлды баяндаған кезде файл элементтерінің типі көрсетіліп кетеді.

Файлды баяндау жалпы түрде былай беріледі:

Аты: **file of** Элементтер типі.

Мысалы:

```
f1: file of char;      //символдар файлы
f2: file of real;      //нақты сандар файлы
f3: file of integer;   //бүтін сандар файлы
```

Компоненттері символдық типті деректер болып табылатын файлдар *символдық* немесе *мәтіндік* деп аталады. Мәтіндік файлдың баяндалуы жалпы түрде былай көрсетіледі:

Аты: TextFile;

Мұндағы:

Аты – файлдық айнымалы аты;

TextFile – типтің белгіленуі.

Object Pascal тілінде мәтіндік файлды баяндау үшін **Text** сөзі қызмет етеді. Delphi-дегі көптеген компонент-тер біраттас қасиеттерге ие, сондықтан мәтіндік файл-ды баяндаған кезде осы баяндағышты дара күйінде қол-дануға болмайды: Мәтіндік файл **TextFile** немесе **System.Text** ретінде анықталады (System модулінен Text баяндауышы деп түсіндіріледі).

Файлдармен тікелей жұмыс жасау үшін көптеген ком-поненттер (объектілер) сәйкес әдістерді ұсынады, мы-салы, **LoadFromFile** (const FileName: String) – файлдан жүктеу немесе **SaveToFile** (const FileName: String) – файл-да сақтау. Мұндай әдістерде файлдық айнымалылар ке-рек емес және **FileName** параметрінде тек файл аты көрсетіледі де қояды.

Файлмен бірнеше жұмысты қарастыралық.

Шығару үшін файл ашу

Файлға шығармастан бұрын оны ашып алу керек. Егер шығыс файлын қалыптастыратын бағдарлама бұрын қолданылған болса, онда ол файл ретінде бір жерде сақ-талып тұруы мүмкін. Сондықтан да бағдарлама жасау-шы алдымен ескі файлмен не істейтінін анықтап алу керек (оны жаңасымен ауыстыру керек пе, жоқ па). Ескі нұсқаны пайдалану тәсілі файлды ашу уақытында анық-талады.

Файлды оған деректер жазу мақсатында ашудың бірнеше режимі бар:

– қайта жазу (бұрыннан бардың үстіне жаңа файл жазу немесе жаңа файл құру);

– бұрыннан бар файлға қосу.

Жаңа файл құру режимінде немесе бұрынғыны ауыстыру режимінде файлды ашу үшін **Rewrite(f)** процедурасын шақыру керек, мұндағы f – TextFile типті файлдық айнымалы.

Бұрыннан бар файлға қосу режимінде файлды ашу үшін **Append(f)** процедурасы шақырылады, мұндағы f – TextFile типті файлдық айнымалы.

Мысал: **Жаңа файл құру немесе бұрыннан бар файлды ауыстыру**

```
procedure TForm1.Button1Click (Sender: TObject);
Var
f: TextFile;              //файл
fName: String[80];  //файл аты
i: integer;

begin
fName := Edit1.Text;
AssignFile (f, fName);

Rewrite(f);               //қайта жазу үшін ашу
// файлға жазу
for i:=0 to Memo1.Lines.Count do //қатарлар нөлден бастап нөмірленеді
writeln(f, Memo1.Lines [i] );
CloseFile (f);  //файлды жабу

MessageDlg ('Деректер файлға жазылды', mtInformation, [mbOk], 0);
end;
```

Бұрыннан бар файлға қосу

```
procedure TForm1.Button2Click(Sender: TObject);
Var

f: TextFile;              //файл
```

```
fName: String [80]; //файл аты
i: integr;
begin
fName := Edit1.Text;
AssignFile (f, fName);

Append(f);                    //қосу үшін ашу
//файлға жазу

for i := 0 to Memo1.Lines.Count do //қатарлар нөлден
бастап нөмірленеді
Writeln(f, Memo1.Lines [i] );
CloseFile (f);          //файлды жабу
MessageDlg('Деректер файлға қосылды', mtInfor-
mation, [mbOk], 0);
end.
```

3.6. Модульдер

Жоғарыда құрылымы қарастырылып кеткен бағдарламалардан басқа, тіл құралдары модульдер құруға да мүмкіндік береді. Бағдарламадан айырмашылығы, **модуль** автономды түрде орындалуға жіберілмейді. Сондай-ақ, бағдарламада және басқа модульдерде пайдалануға болатын айнымалылар және ішкі бағдарламалар сияқты элементтерді қамтиды. Модуль жабдықтарын қолдану үшін оны міндетті түрде қосу қажет, ол үшін сол модульдің атын *uses* бөлімінде көрсетіп кету керек. Мысал ретінде System SysUtils модульдерін айтып кетуге болады. Бұл модульдер стандартты ішкі бағдарламалардың үлкен көлемін қамтиды. Еске сала кетейік, қосымшаның әрбір формасы үшін жеке модуль құрылады.

Компилятор модульді оның тақырыпшасы бойынша танып біледі.

Модуль тақырыпшадан тұрады, ондағы **unit** кілттік сөзінен кейін модуль аты және төрт бөлім беріледі: интерфейс (*Interface*), жүзеге асыру (*Implementation*), инициализациялау (*Initialization*), деинициализациялау (*Finalization*).

Модульдің құрылымы мынадай:

Unit <Модуль аты>;

// Интерфейс бөлімі
Interface
Uses <Модульдер тізімі>;
Const <Тұрақтылар тізімі>;
Type <Типтерді баяндау>;
Var <Айнымалыларды баяндау>;
<Процедуралар тақырыпшалары>;
<Функциялар тақырыпшалары>;

// Жүзеге асыру бөлімі
Implementation
Uses <Модульдер тізімі>;
Const <Тұрақтылар тізімі>;
Type <Типтерді баяндау>;
Var <Айнымалыларды баяндау>;
<Процедураларды сипаттау>;
<Функцияларды сипаттау>;

//Инициализациялау бөлімі
Initialization
<Операторлар>;

// Деинициализациялау бөлімі
Finalization
<Операторлар>;
End.

Интерфейс бөлімінде идентификаторларға сипаттамалар орналасады, бұл осы модульді қолданатын (аттары *uses* бөлімінде көрсетілген модульдер) барлық модульдер мен бағдарламаларға қол жеткізерлік болу керек. Интерфейс бөлімінде типтер, тұрақтылар, айнымалылар және ішкі бағдарламалар баяндалады. Ішкі бағдарламалар үшін тек тақырыпшалары ғана көрсетілетін болады.

Басқа қолданылатын модульдер *uses* тізімінде көрсетіледі. Интерфейс бөлісі *Interface* кілттік сөзінен басталады.

Жүзеге асыру бөлімінде ішкі бағдарлама коды орналасады, тақырыпшалары интерфейс бөлімінде келтірілген. Ішкі бағдарламалардың орналасу тәртібі олардың интерфейс бөліміндегі тақырыпшаларының орналасу тәртібімен сәйкес болмауы мүмкін. Сонымен қатар, тақырыпшада тек ішкі бағдарлама аты ғана көрсетілуі мүмкін, себебі функцияның параметрлер тізімі мен нәтиже типі алдын ала көрсетіліп кеткен болатын. Жүзеге асыру бөлімінде сондай-ақ, типтерді көрсетуге, тұрақтылар мен айнымалыларды, ішкі бағдарламаларды (тек осы модульде қолданылатын және одан тыс жерде көрінбейтін) сипаттап кетуге болады.

Интерфейс бөлімі *Implementation* сөзімен басталады.

Инициализациялау бөлімінде бағдарлама жұмысының басында орындалатын операторлар өрындалады (берілген модульмен байланысты). Модульдерді инициализациялау бөлімі бағдарламаның uses бөліміндегі тізімде қалай берілсе, сол тәртіппен орындалады. Инициализациялау бөлімі *Initialization* сөзімен басталады.

Бағдарламада инициализациялау бөлімін қолданған кезде, міндетті түрде деинициализация бөлімін де қолдану керек болады. Бұл бөлім *Finalization* сөзімен басталады және міндетті бөлім болып табылмайды. Бұл бөлімде бағдарламаны аяқтау кезінде орындалатын операторлар орналасады. Модульдерді деинициализациялау бөлімі бағдарламаның *uses* бөліміндегі тізімде қалай берілсе, сол тізімге кері тәртіппен орындалады.

3.7. Кластар және объектілер

Object Pascal тіліндегі кластар – объектілерді сипаттау үшін қолданылатын деректердің арнайы типтері. Қандай да бір болмасын класты қамтитын объект осы кластың *экземпляры* немесе осы типтің *айнымалысы* болып табылады. Класс жазбаның ерекше типі болып табылады, өз құрамында өрістер, қасиеттер және әдістер

сияқты элементтерді (мүшелерді) қамтиды. Класс өрісі жазба өрісіне ұқсас және объект туралы ақпаратты сақтауға арналған. *Әдістер* дегеніміз – өрістерді өңдеуге арналған процедуралар мен функциялар. *Қасиеттер* өрістер мен әдістер аралығында жүретін жуықталған орын. Бір жағынан, қасиетті өріс ретінде қолдануға болатын болса, мысалы, меншіктеу операторының көмегімен оларға мән меншіктеу арқылы; екінші жағынан класс ішінде қасиеттер мәндеріне қатынас класс әдістерімен орындалады.

Класты сипаттау мынадай құрылым түрінде көрсетіледі:

```
Type <Класс аты> = class (< Класс-аналық аты>)
Private
<Жекелеген сипаттама>;
protected
<Қорғалған сипаттама>;
public
<Жалпы қол жеткізерлік сипаттама>;
published
<Жарияланған сипаттама>;
End.
```

Келтірілген құрылымдағы сипаттама – *қасиеттерді, әдістерді* және *оқиғаларды* хабарлау болып табылады.

Класты баяндауға мысал:

```
Type
TColorCircle = class(TCircle);
FLeft,
FTop,
FRight,
FBottom: Integer;
Color: TColor;
End;
```

Мұндағы *TColorCircle* класы *TCircle* негізгі класс негізінде құрылады. Негізгі класпен салыстырғанда, жаңа

класс қосымша *Integer* типті төрт өрісті және *TColor* типті бір өрісті қамтиды.

Егер негізгі класс ретінде TObject класы қолданылса (барлық кластар үшін базалық класс болып табылады), онда оның атын **Class** сөзінен кейін көрсетпесе де болады. Олай болса, баяндаудың бірінші жолы мынадай түрге ие болады:

```
Type TNewClass = class
```

Кластың әр түрлі элементтері үшін қатынас құрудың түрлі құқықтарын орнатуға болады, ол үшін кластың баяндалуында арнайы қол жеткізу белгілерімен белгіленген жеке бөлімдер қолданылады.

Private және *protected* бөлімдері қорғалған сипаттамаларды қамтиды (өздері орналасқан модуль ішінде қол жеткізуге мүмкіндік бар). *Protected* бөліміндегі сипаттаманы аталған модульден тыс жерде туынды кластар үшін қолдануға да болады.

Public бөлімі бағдарламаның кез келген орнында (класқа қол жеткізерлік жерде) көрінетін жалпы қол жетерлік сипаттаманы қамтиды. *Published* бөлімі жарияланған баяндауды қамтиды, жалпы қол жетерлік баяндауға типтер туралы (Run – Time Type Information, RTTI) динамикалық ақпаратты қосуға мүмкіндік береді. Осы ақпарат бойынша қосымшаны орындау кезінде объекті элементінің қандай да бір класқа жататынын тексеру жүзеге асырылады.

Published бөлімінің тағы бір қызметі, қосымшаларды құрастырған кезде объектілер қасиеттеріне қатынасты қамтамасыз етеді.

Объектілер инспекторында жарияланған болып табылатын қасиеттер көрініс табады. Егер *published* спецификаторы көрсетілмесе, онда ол «бастапқы» қалыпта деп есептеледі де, класс аты көрсетілген жолда орналасқан кез келген баяндау жарияланған болып саналады.

Объектілер класс экземплярлары ретінде бағдарламада **var** бөлімінде қарапайым айнымалы сияқты баяндалады.

Мысалы:

```
Var
CCircle1: TColorCircle;
CircleA: TCircle;
```

Объектінің нақты элементіне қатынас құру үшін (өріске, қасиетке немесе өдіске) объект аты және элемент аты (нүктемен ажыратылған) көрсетіледі, яғни элемент аты құрамдас болып табылады.

Объект өрісіне қатынасу мысал:

```
Var
CCircle1: TColorCircle;
Begin

....

CCircle1.FLeft:=5;
CCircle1.FTop:=1;
....

End.
```

Өріс

Класс *өрісі* – класта қамтылатын деректер. Өріс қарапайым айнымалы ретінде баяндалады және кез келген типке жатуы мүмкін.

Өрісті баяндауға мысал:

```
Type TNewClass = class(TObject)
Private
FCode: integer;
FSign: char;
FNote: string;
End;
```

Мұндағы **TNewClass** жаңа класы **Tobject** базалық кластың негізінде құрылады және қосымша үш жаңа өрісті қамтиды: Fcode, FSign және FNote (сәйкесінше бүтін сандық, символдық және қатарлық типі) қабылданған келісім бойынша, өріс аттары f. префиксінен басталады (ағылшынша Field – өріс сөзінен).

Жаңа кластар құрған кезде класс-туындылар аналық кластың барлық өрістерін иемденеді, бұл өрістерді жою немесе қайта анықтау мүмкін емес, бірақ жаңаларын қосуға болады. Осылайша, түпкі кластан туынды кластар иерархиясы неғұрлым алысырақ болса, соғұрлым ондағы өрістер саны да көп болмақ. Өрістер мәндерін өзгерту әдістер мен объектілер қасиеттерінің көмегімен орындалады.

Қасиеттер

Қасиеттер – өріске қатынас құру механизмін жүзеге асырады. Әр қасиеттің өз өрісі бар, онда қасиет мәні қамтылады және осы өріске қатынасты қамтамасыз ететін екі тәсіл бар.

Қасиетті баяндау **property** сөзінен басталады және қасиет типтері мен сәйкес өріс бір-бірімен үйлесім табу керек.

Read және **write** кілттік сөздер қасиет баяндауының ішіндегі қордағы сөздер болып табылады және класс әдістерін көрсету үшін қызмет етеді. Олардың көмегімен қасиетпен байланысты өріс мәндерін оқуға немесе осы өріске жаңа мән жазуға болады.

Қасиеттерді баяндауға мысал:

```
type TNewClass = class(TObject)
 Private
FCode: integer;
FSign: char;
FNote: string;
```

```
published
property Code: integer read FCode write FCode;
property Sign: char read FSign write FSign;
property Note: string read FNote write FNote;
End.
```

FCode, FSign және *FNote* өрістеріне қатынас үшін (қорғалған бөлімде сипатталған және басқа кластарға қол жеткізу мүмкіндігін бермейтін). *Code, Sign, Note* қасиеттері қолданылады.

Әдістер

Әдіс – класс элементі болып табылатын ішкі бағдарлама (процедура немесе функция). Әдістің баяндалуы модульдің қарапайым ішкі бағдарламасының баяндалуына ұқсас. Әдіс тақырыпшасы класс баяндалуында орналасады, ал әдістің коды жүзеге асыру бөлімінде орын алады. Жүзеге асыру бөліміндегі әдіс аты құрамдас болып табылады және өзіне класс типін қамтиды.

Мысалы, **Button1Click** әдісінің баяндалуы мынадай көрініс табады:

```
Interface
...
type
TForm1 = class(TForm)
Button1: TButton;
Procedure Button1Click(Sender: TObject);
end;
...
implementation
...
procedure TForm1.Button1Click(Sender: TObject);
begin
Close;
end;
```

Класта баяндалған әдіс түрлі тәсілдермен шақырылуы мүмкін, бұл шақырылатын әдістің түріне байланысты. Әдіс түрі модификатормен анықталады, ол класс баяндалуындағы әдіс тақырыпшасынан кейін көрсетіледі және тақырыпшадан нүктелі үтір «;» арқылы ажыратылады. Кейбір модификаторларды қарастыралық:

– virtual – виртуалды әдіс;
– dynamic – динамикалық әдіс;
– override – қайта анықталатын әдіс;
– message – хабарды өңдеу;
– abstract – абстрактылы әдіс.

Әдепкі түрде класта баяндалатын барлық әдістер *статикалық* болып табылады және қарапайым ішкі бағдарламалар ретінде шақырылады.

Объектілерді құру немесе жою үшін белгіленген әдістер *конструкторлар* және *деструкторлар* деп аталады. Берілген әдістердің баяндалуының қарапайым процедураларды баяндаудан айырмашылығы мынада, олардың тақырыпшаларында **constructor** және **destructor** кілттік сөздері тұрады. TObject базалық класында және көптеген басқа да кластарда конструкторлар мен деструкторлар ретінде **Create** және **Destroy** атаулары қолданылады.

Элементке қатынас құрмастан бұрын оны алдымен конструктордың көмегімен құрып алу қажет. Мысалы:

ObjectA:= TOwnClass.Create;

Конструктор «үймеде» (heap) жаңа объект үшін жады бөледі, реттік өріс үшін нөлдік мән береді, nil мәні – нұсқағыштар мен өріс-кластар үшін, қатарлық өріс бос болып орнатылады, сонымен қатар нұсқағышты құрылған объектіге қайтарады.

Конструктор мен деструкторды қолдануға мысал келтірелік:

```
Type
TShape = class (TGraphicControl)
```

```
private
FPen: TPen;
procedure PenChanged (Sender: TObject);
public
Constructor Create (Owner: TComponent); override;
Destructor Destroy: override;
...
End.

// TShape класының Create конструкторын баяндау
constructor TShape.Create (Owner: TComponent);
begin
inherited Create (Owner); //Иемденген бөліктерін анықтау;
Width:= 65; //Иемденген қасиеттерді өзгерту
Height:= 65;
FPen := Tpen.Create; //Жаңа өрістерді анықтау
FPen.OnChange:= PenChanged;
End;
```

Туынды-класс конструкторында алдымен аналық конструктор шақырылады, сонан соң ғана қалған әрекеттер орындалады. Туынды-класта *override* (қайта анықтау) директивасы аналық класқа жаңа әдісті пайдалануға мүмкіндік береді. *Inherited* кілттік сөзі аналық клас әдістерін шақыру үшін қызмет етеді.

Бақылау сұрақтары:

1. Object Pascal-да қандай типтерді білесіз, әрқайсысына талдау жасаңыз.
2. AnsiChar, WideChar және Char символдық типтерінің бір-бірінен айырмашылығы.
3. Диапазоны үлкен бүтін сандарды баяндау үшін қандай тип қолданылады?
4. Логикалық тип не үшін керек?
5. Тұрақтылар дегеніміз не, қандай түрлері бар?
6. Оператор дегеніміз не, олардың приоритеттері дегенді қалай түсіндіресіз?

7. Өрнек типі мен айнымалы типтерінің сәйкестіктері қалай тексеріледі?

8. Object Pascal-дағы стандартты функцияларды атаңыз.

9. Input Box функциясы не үшін керек?

10. Типтерді түрлендіру функцияларына талдау жасаңыз.

11. MessageDlg функциясына қатынас қалай құрылады?

12. Тілдің операторлары, циклдық операторларға талдау жасаңыз?

13. Құрамдас оператор дегеніміз не?

14. Массивтерді баяндауды қалай көрсетуге болады?

15. Ішкі бағдарлама дегеніміз не, процедурадан функцияның айырмашылығы қандай?

16. Модуль деген не, бағдарламада қалай қолданылады?

17. Кластар мен объектілерге түсінік беріңіз.

18. **FCode**, **FSign** және **FNote** өрістеріне қатынас үшін қандай қасиеттер қолданылады?

Мысалдар

1 мысал. Бүтін сандарды бөлгендегі бүтін бөлікті табу.

1.1) **File/New Application** командасы көмегімен жаңа жобаны құрыңыз. Объектілер инспекторы көмегімен **Caption** (аты) қасиетіне «Бөлуден қалатын бүтін бөлікті табу» мәнін беріңіз. Компоненттер палитрасының **Standart** бетін пайдаланып суретте көрсетілгендей (39-сурет) форманың төменгі жағында **Button** (батырма) екі компонентін орналастырыңыз. Осы компоненттердің (**Object Inspector** объектілер инспекторынан) **Caption** қасиеттеріне **«Операция»** және **«Жабу»** мәндерін беру керек. Келесі тапсырмаларда осы батырмаларды құру операциялары осы жолмен берілетіндіктен, оларды құрудың сипаттамасы келтірілмейді. **«Жабу»** батырмасын тінтуірмен екі рет басып активтендіру қажет және де ашылған оқиғаларды өңдеушіге төмендегі мәтінді жазыңыз:

```
Procedure TForm1.Button2Click(Sender: TObject);
begin
```

Close; {Бағдарлама орындалуының аяқталуы}
End.

1.2) Компоненттер палитрасының **Standart** бетін пайдалана отырып форманың бірінші бөлігінде **Edit** енгізудің үш жолын орналастырыңыз. Мәтіннің модулінде оларға **Edit1, Edit2** және **Edit3** аттары меншіктеледі. Объектілер инспекторы көмегімен барлық енгізу жолдарының **Text** (Мәтін) қасиетінің құрамын жойыңыз.

1.3) Компоненттер палитрасының **Standart** бетін пайдалана отырып, әрбір енгізу жолының сол жағында **Label** белгілерін орналастырыңыз. Модуль мәтінінде оларға **Label1, Label2** және **Label3** аттары меншіктелінеді. Объектілер инспекторы көмегімен бірінші белгінің **Caption** қасиетіне **"бөлінгіш"**, екінші белгіге – **"бөлгіш"**, үшіншіге – **"бөлінді"** мәндерін беріңіз.

1.4) **«Операция»** батырмасын тінтуір арқылы активтендіріңіз (тінтуірдің оң жақ батырмасын қатарынан екі рет басу арқылы). Оқиғаларды өңдеушіге келесі мәтінді жазамыз.

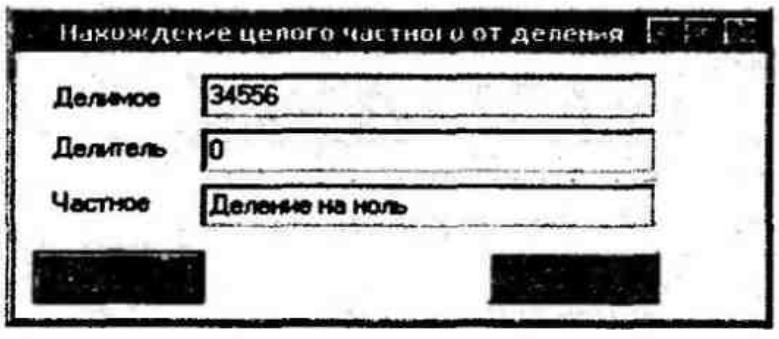
```
Procedure TForm1.Button1Click (Sender: TObject);
LABEL Out; {Таңбаның сипаттамасы}
Var x, y, Res: integer;
Begin
X:=StrToInt (Edit1.Text): {Бөлінгіш}
Y:= StrToInt (Edit2.Text): {Бөлгіш}
IF Y=0 THEN
BEGIN
Edit3.Text:= "Нөлге бөлу";
GOTO out;
End;
Res: =X div Y;
Edit3.Text: = InToStr(Res); {Бөлінді}
Out: {Бос оператордың белгісі}
End.
```

Бұл ішкі бағдарламада соңында бос операторды белгілейтін **out** белгісі енгізілген, бұл операторға өту бөлгіш нөлге тең, бөліндіні есептеуге болмайтын жағдайда орындалады.

Бұл жерде пайдаланатын **StrToInt** функциясы сәйкес енгізу жолында санның мәтіндік көрсетілуін екілік баламаға түрлендіреді.

1.5) Негізгі менюдің **Run/Run** командасы көмегімен бағдарламаны орындауға жіберіңіз.

1.6) Бөлінгіш пен бөлгіштің енгізу жолдарына қандайда бір сандарды енгізіп **«Операция»** батырмасын басамыз. **«Бөлінді»** енгізу жолында бір санды екіншісіне бөлгендегі бүтін сан немесе «Нөлге бөлу» деген фраза шығады (39 сурет).



39-сурет. Бөліндіні табу бағдарламасы

1.7) Сандардың басқа жұбын енгізіңіз де, жаңа бөліндіні алу үшін **«Операция»** батырмасын қайта басыңыз және т.с.с.

1.8) Бағдарлама жұмысын, **Закрыть (Жабу)** батырмасын басу арқылы аяқтаңыз.

2 мысал. Бүтін сандарды енгізіп, ANSI-кодтауда осы санға сәйкес символды, символ жоқ болса, мұндай символ жоқ туралы хабарды шығару.

Тапсырманы келесі түрде шешуге болады:

2.1) Негізгі менюдің **File/New Application** командасы көмегімен жаңа жобаны құрыңыз. Объектілер инспекторы көмегімен форманың **Caption** қасиетіне **«Бүтін саннан символды шығару»** мәнін енгізіңіз.

2.2) Компоненттер палитрасының **Standart** бетін пайдалана отырып, форманың жоғары оң жағында **Edit** енгізудің екі жолын орналастырыңыз. Модуль мәтінінде олар **Edit1, Edit2** аттарына ие болады. Объектілер инспекторы көмегімен барлық енгізу жолдарының **Text** (Мәтін) қасиетінің құрамын жойыңыз.

2.3) Компоненттер палитрасының **Standart** бетін пайдалана отырып, әрбір енгізу жолының сол жағында **Label** белгілерін орналастырыңыз. Модуль мәтінінде олар **Label1, Label2** аттарына ие болады. Объектілер инспекторлары көмегімен бірінші белгінің **Caption** қасиетіне **«Сан»**, екінші белгіге – **«Символ»** мәндерін беріңіз.

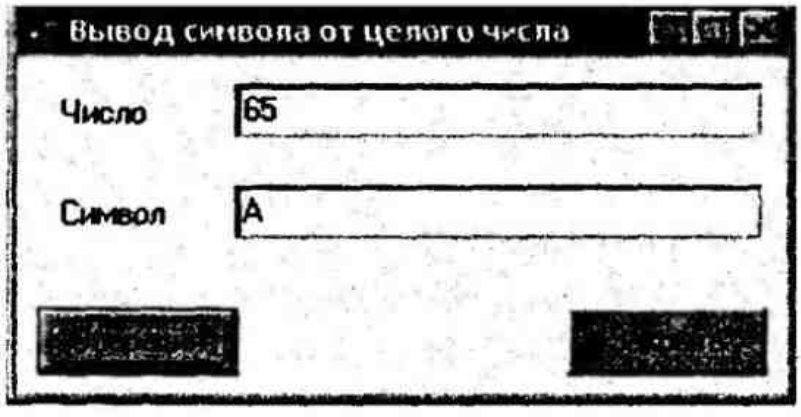
2.4) **«Операция»** батырмасын тінтуірмен активтендіріңіз. Оқиғаларды өңдеушіге келесі мәтінді жазыңыз:

```
Procedure TForm1.Button1Click (Sender:TObject);
Var x: integer;
Begin
X:= StrToInt (Edit1.Text); {Енгізілген сан}
IF (X>31) AND (X<256) THEN
Edit2.Text:=AnsiChar(X); {Символды шығару}
ELSE
Edit2.Text:= 'Символ жоқ';
End.
```

Бұл ішкі бағдарламада біріншіден енгізілген сан екілік формаға түрлендіріледі, содан кейін тексеріледі (осындай кодпен ANSI – кодтауда символ бар ма, жоқ па соны тексереді). Бар болса, ол экранға шығарылады, әйтпесе мұндай символдың жоқтығы туралы хабар шығады.

2.5) Негізгі менюдің **Run/Run** командасы көмегімен бағдарламаны жұмысқа қосыңыз.

2.6) Санды енгізу жолына қандай да бір санды жазыңыз, содан кейін **«Операция»** батырмасын басыңыз. Символды енгізу жолында енгізілген кодқа сәйкес символ шығуы керек (40-суретті қараңыз).



40-сурет. Бөліндіні табу бағдарламасы

2.6) Пункттегі операцияны бірнеше рет орындауға болады.

2.7) Бағдарлама жұмысын **Закрыть (Жабу)** батырмасын басып аяқтаңыз.

3 мысал. Құрамында бүтін сандары бар мәтіндік файлдан бірінші теріс санды оқып, оны экранға шығару. Егер мұндай сан жоқ болса, ол туралы хабарды шығару керек.

3.1) Негізгі менюден **File/New Application** командасының көмегімен жаңа жобаны құрыңыз. Объектілер инспекторы көмегімен форманың **Caption** қасиетіне **«Файлдан теріс санды табу»** мәнін беріңіз.

3.2) Мәтіндік файлды құрып бүтін сандарды жазып қойыңыз. Ол үшін негізгі менюден орындалатын **File/New/New/Text** операциясы көмегімен жаңа мәтіндік файлды ашыңыз. **File1.txt** мәтіндік редактордың жаңа беті ашылады. Бұл бетте кез келген бірнеше бүтін сандарды (бағанға немесе бір жолға бос орынмен бөліп) жазыңыз. Сандарды енгізгеннен кейін файлды **MyFile.txt** деген атпен, негізгі менюдің **File/Save As** командасын пайдаланып сақтаңыз және де **File/Close** командасын пайдаланып, осы файлмен мәтіндік редакторды жабыңыз.

3.3) Интерфейстік бөлімнің айнымалыларды баяндау бөлімінде **Fil** жаңа айнымалыны енгізіңіз (оның типі – **TextFile** мәтіндік файл), бұл бөлім төмендегідей болады:

```
var
Form1: TForm1;
Fi1: TextFile;
```

3.4) Модуль мәтінінде инициализациялау және аяқтау секциясын құрыңыз.

```
INITIALIZATION
AssignFile (Fil, 'MyFile.txt');
Reset (Fil);
FINALIZATION
CloseFile (Fil);
End.
```

3.5) Компоненттер палитрасынан **Standart** бетін пайдалана отырып, форманың жоғарғы жағында **Label** компонентін орналастырыңыз. Модуль мәтінінде ол **Label1** атына ие болады. Объектілер инспекторы көмегімен белгінің **Caption** қасиетіне **«Сан»** мәнін беріңіз.

3.6) **«Операция»** батырмасын тінтуірмен активтендіріңіз. Оқиғаларды өңдеушінің терезесінде келесі мәтінді жазыңыз.

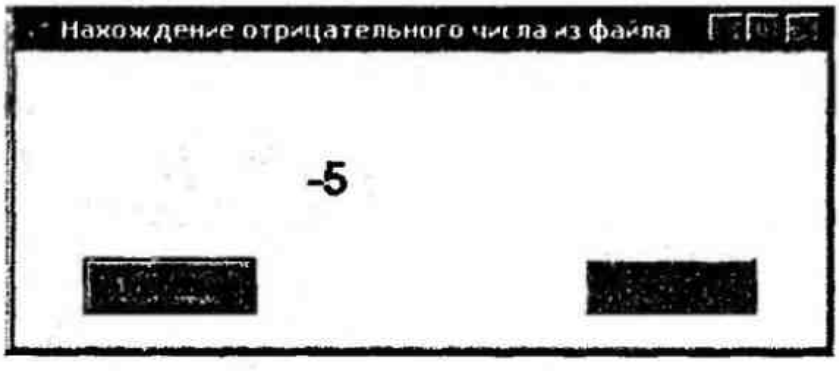
```
Procedure TForm1.Button1Click (Sender: TObject);
Var N: integer;                  {ағымдағы сан}
Begin
N:=0;                            {сан жоқ болуының белгісі}
REPEAT                           {қайталаймыз}
IF not SeekEof (Fil) Then        {егер файлдың соңы
                                 болмаса}
Read (Fil,n);                    {кезекті символды оқимыз}
 Until (n < 0) or SeekEOF (Fil); {сан теріс болғанша}
                                 {немесе файлдың соңы болғанша}
```

```
If N<0 Then
Label1.Caption:=IntToStr (n) {нәтиже шығару}
ELSE
Label1.Caption:= 'Теріс сан жоқ';
End.
```

Бұл ішкі бағдарламада теріс санды оқығанша немесе файлдың соңына жеткенше (Ол **SeekEof** функциясы көмегімен анықталады) мәтіндік файлдан сандарды оқу циклы ұйымдастырылады. Цикл аяқталғаннан кейін нәтиже табылған сан түрінде немесе теріс сан табылмаса, ол туралы хабар түрінде шығарылады. Циклды ұйымдастырған кезде, кезекті санды оқу алдында **Read** процедурасы көмегімен файлда ақпараттың бар-жоқтығы тексеріледі.

3.7) Негізгі менюдің **Run/Run** командасы көмегімен бағдарламаны жұмысқа қосыңыз.

3.7) **«Операция»** батырмасын басқанда экранға табылған теріс сан шығады немесе ол жоқ деген хабар шығады. Егер **«Операция»** батырмасын қайтадан басса, экранға екінші теріс сан шығады және т.с.с. (қараңыз, 41-сурет).



41-сурет. Файлдан теріс санды табу бағдарламасы

3.9) **Закрыть** батырмасын басып бағдарлама жұмысын аяқтаңыз.

4 мысал. – Бүтін санның барлық бөлгіштерін (1 мен сол санның өзінен басқа) табу.

4.1) Негізгі менюдің **File/New Application** командасы көмегімен жаңа жобаны құрыңыз. Объектілер инспекторының көмегімен форманың **Caption** қасиетіне **«Бүтін санның бөлгіштерін табу»** мәнін беріңіз.

4.2) Компоненттер палитрасының **Standart** бетін пайдалана отырып, форманың жоғарғы жағында **Edit** енгізу жолын орналастырыңыз. Модуль мәтінінде ол **Edit1** атына ие болады. Объектілер инспекторы көмегімен енгізу жолының **Text** (мәтін) қасиетінің құрамын жоямыз.

4.3) Компоненттер палитрасының сол бетін пайдаланып, енгізу жолынан солға қарай **Label** белгісін орналастырыңыз. Модульдың мәтінінде ол **Label1** атын иемденеді. Объектілер инспекторының көмегімен белгінің **Caption** қасиетіне «Бүтін сан» мәнін меншіктейміз.

4.4) Компонентер палитрасының **Standart** бетін пайдаланып, форманың **ListBox** (тізім) компонентін орналастырыңыз. Модуль мәтінінде оған **ListBox1** аты меншіктелінеді. Компонентті ені бойынша шамамен форманың үштен бір бөлігіндей созамыз, ал биіктігі бойынша шамамен енгізу жолынан батырмаға дейін созамыз.

4.5) **«Операция»** батырмасын тінтуірмен активтендіріңіз. Оқиғалар өңдеушінің терезесінде келесі операторларды жазыңыз:

```
Procedure TForm1.Button1Click (Sender: TObject);
Var
x, I, half, divider: integer;
begin
ListBox1.Items.Clear;                    {тізімді тазарту}
X:= abs(StrToInt (Edit1.Text));          {зерттелетін сан}
Half:= x Div 2;                          {санның
                                         жартысы}
Divider:=2;                              {ең аз
                                         бөлгіш}
I:=0;                                    {бөлгіштің
                                         реттік нөмірі}
While divider <= half DO
Begin                                    {бөлгіш санның
```

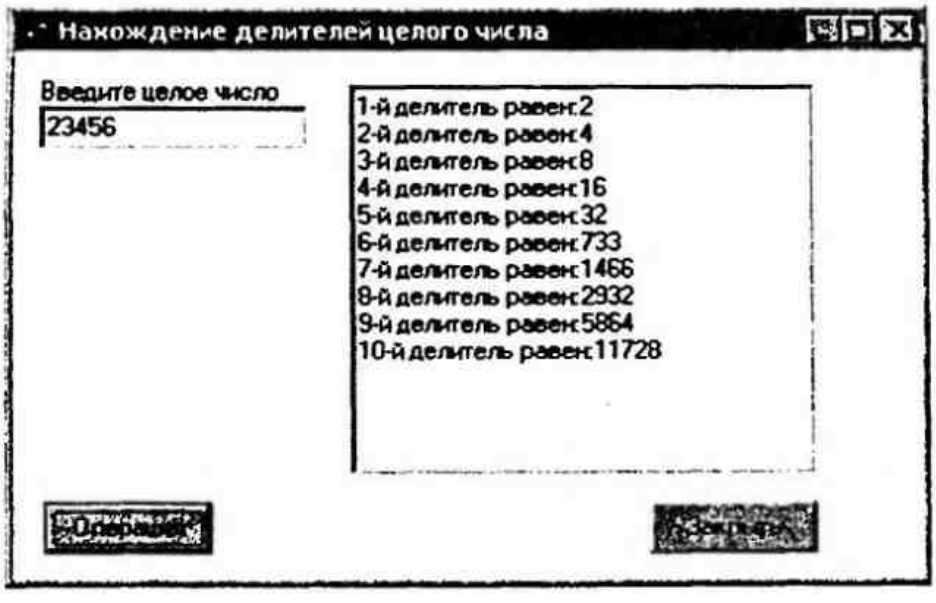
```
                                    жартысынан үлкен
                                     болмайынша...}

  If x mod divider = 0 then         {бөлгішті табамыз}
  Begin
  Inc(i);                           {бөлгіштің келесі
                                    нөмірі}
  ListBox1.Items.Add(IntToStr(i)+
  '-ші бөлгіш тең:' + IntToStr(divider));
  End;
  Inc(divider);
  End;
  If i=0 then
  ListBox1.Items.Add('Жай сан');
  End;
```

Мұндағы, X – енгізілетін сан жазылатын айнымалы. **Half** – мүмкін болатын көбейткіштердің жоғары шекарасын анықтайтын, зерттелетін санның жартысына тең сан жазылатын айнымалы, **Divider** – ағымдық зерттелініп жатқан көбейткіштің айнымалысы, i – табылған көбейткіштің нөмірі.

Бұл ішкі бағдарламада көбейткішті іздер алдында біріншіден, **ListBox1** компонентінің жолдар тізімі тазартылады. Бұл бағдарламаны қайта жүктемей-ақ бірнеше рет пайдалану үшін орындалады. Келесі төрт оператор енгізілген параметрлердің бастапқы мәндерін береді, мұндағы қолданылып отырған *Abs* стандартты функциясы енгізілген санның абсолютті мәнін қалыптастырады. Бұдан кейін бөлгіштерді іздеу үшін **While** циклы ұйымдастырылады. Егер бөлгіш табылса (санды **Divider** кезекті санға бөлгендегі қалдық = 0), **ListBox1** компоненті жолдарының тізіміне табылған бөлгіш туралы ақпараты бар жаңа жол қосылады. Әрбір итерация соңында **Divider** параметрінің мәні бірге арттырылады. Цикл зерттелініп жатқан санның жартысынан асатын мән болғанша орындалады. Ішкі бағдарлама соңында бірде-бір бөлгіш табылмаған жағдайда, санның *жай* (бөлгіштері жоқ) екендігі туралы хабар шығады.

4.6) Негізгі менюдің **Run/Run** командасы көмегімен бағдарламаны іске қосыңыз.

4.7) Енгізу жолына қандай да бір бүтін санды енгізіңіз, содан кейін «**Операция**» батырмасын басыңыз. Экранда санның барлық бөлгіштерінің тізімі шығуы керек. 4.7 пункттің операцияларын бірнеше рет орындауға болады (42-суретті қараңыз).



42-сурет. Бүтін санның бөлгіштерін табу бағдарламасы

4.8) **Закрыть** батырмасын басу арқылы бағдарлама жұмысын аяқтаңыз.

5 мысал. Экранға Z-тен А-ға дейінгі әріптерді шығару.

5.1) Негізгі менюдің **File/New Application** командасы көмегімен жаңа жобаны құрыңыз. Объектілер инспекторы көмегімен форманың **Caption** қасиетіне «Z-тен А-ға дейінгі әріптерді экранға шығару» мәнін беріңіз.

5.2) Компоненттер палитрасының **Standart** бетін пайдаланып, форманың жоғарғы жағында **Label** компонентін орналастырып, оны форманың сол жағына жылжытамыз. Модуль мәтінінде ол **Label1** атын алады.

Объектілер инспекторы көмегімен белгінің **Caption** қасиетінің құрамын өшіреміз.

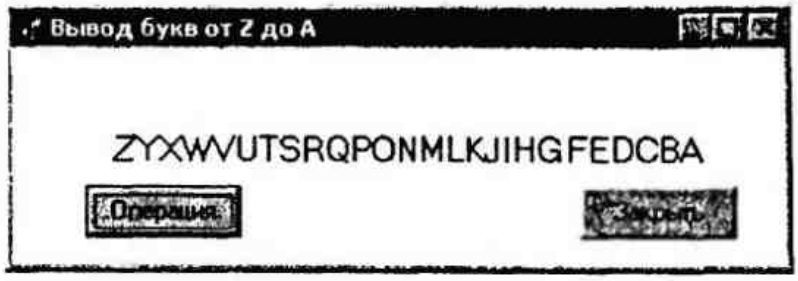
5.3) **«Операция»** батырмасын тінтуірмен активтендіріңіз. Оқиғаларды өңдеушінің терезесінде келесі операторларды жазыңыз.

```
Procedure TForm1.Button1Click (Sender: TObject);
Var
I: char;
Begin
For i:= 'Z' DOWNTO 'A' DO
Label1.Caption:= Label1.Caption+i+ '';
End;
```

Бұл ішкі бағдарламада цикл параметрі ретінде тек бүтін сандар ғана емес, сондай-ақ кез.келген реттік типті (біздің жағдайымызда – символдар) басқа параметрлерді де пайдалануға болатыны көрсетілген. Әрбір итерацияда белгінің мәтініне кезекті символ мен бос орын қосылады.

5.4) Негізгі менюдің **Run/Run** командасы көмегімен бағдарламаны іске қосыңыз.

5.5) **«Операция»** батырмасын басқаннан кейін, экранда соңынан басталып үлкен латын әріптерінің тізімі шығады (43-суретті қараңыз).



43-сурет. Z-тен A-ға дейінгі әріптерді шығаратын бағдарлама

5.6) **«Жабу»** батырмасын басу арқылы бағдарлама жұмысын аяқтаңыз.

6 мысал. Бүтін сандардың мәтіндік файлында бірінші оң санды тауып, оны экранға шығару.

6.1) Негізгі менюдің **File/New Application** командасы көмегімен жаңа жобаны құрыңыз. Объектілер инспекторы көмегімен форманың **Caption** қасиетіне **«Файлдан бірінші оң санды табу»** мәнін енгізіңіз.

6.2) 4.3 мысалдың 4.3.2 пунктін пайдаланып, мәтіндік файлды құрыңыз.

6.3) Интерфейстік бөліктегі айнымалыларды баяндау бөлімінде **Fil** жаңа айнымалыны енгізіңіз (оның типі – **TextFile** мәтіндік файл), бұл бөлім төмендегідей болады:

```
var
Form1:TForm1;
Fil: TextFile;
```

6.4) Модуль мәтінінде инициализациялау және аяқтау секциясын құрыңыз.

```
INITIALIZATION
AssignFile (Fil, 'MyFile.txt');
Reset (Fil);
FINALIZATION
CloseFile (Fil);
End.
```

6.5) Компоненттер палитрасының **Standart** бетін пайдаланып, форманың жоғарғы жағында **Label** компонентін орналастырыңыз. Модуль мәтінінде ол **Label1** атын алады. Объектілер инспекторы көмегімен белгінің **Caption** қасиетінің құрамын өшіріңіз.

6.6) **«Операция»** батырмасын тінтуірмен активтендіріңіз. Оқиғаларды өңдеушінің терезесінде келесі мәтінді жазыңыз.

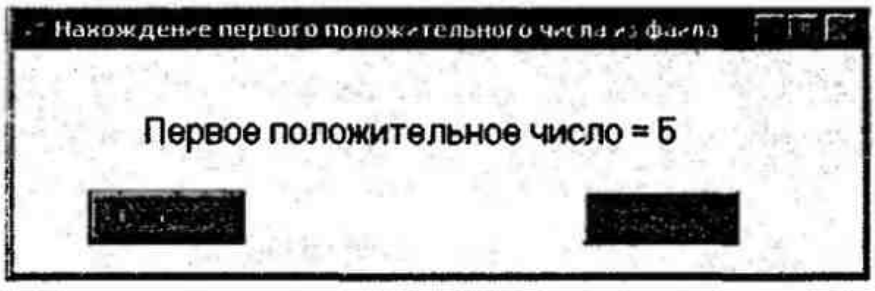
```
Procedure TForm1.Button1Click (Sender: TObject);
Var
```

```
N: integer;                          {Ағымдағы сан}
Yes: Boolean;                        {Сан табылмағанының
                                      белгісі}
Begin
Yes:=False;                          { Сан табылмады }
WHILE NOT SeekEof (Fil) DO           {Файлдың соңы
                                      болмаса}
Begin
 Read(Fil, n);                       {Кезекті санды оқу}
 If N<0 then
 Continue;                           {Теріс сан – келесі
                                      санға көшу}
 Label1.Caption:= 'Бірінші оң сан=' +IntToStr(N);
 Yes:=True;                          {Сан табылды}
 Break;
 End;
 If NOT Yes THEN
 Label1.Caption:= 'Оң сан жоқ'
End.
```

Бұл ішкі бағдарламада *N* айнымалысы файлдан оқылатын санды жазу үшін пайдаланылады, *Yes* айнымалысында оң санның табылғаны немесе табылмағанының белгісі шығады. *Yes* айнымалысына алдын ала **False** мәні беріледі, ол оң санның табылмағанын білдіреді. Содан кейін файлдық айнымалы *Fil* мен айнымалы n арқылы **Read** стандартты процедурасының көмегімен файлдан бүтін сандарды кезекпен оқу үшін **WHILE** циклы ұйымдастырылады. Егер енгізілген сан теріс болса, кезекті итерация **Continue** процедурасы көмегімен тоқтатылады. Егер де сан оң болса, оның мәні **Label1** белгісінің мәтіні түрінде экранға шығарылады, санның табылғаны туралы белгі орнатылады да, **Break** процедурасы көмегімен циклдан шығу жүзеге асырылады. Цикл аяқталғаннан кейін оң сан табылмаса, соған сәйкес хабар шығатын болады.

6.7) Негізгі менюден **Run/Run** командасы көмегімен бағдарламаны жұмысқа қосыңыз.

6.8) **«Операция»** батырмасын басқанда экранға табылған оң сан шығады немесе ол жоқ деген хабар шығады. Егер **«Операция»** батырмасына қайтадан басса, экранға екінші оң сан шығады және т. с. с. (қараңыз 44-сурет).



44-сурет. Файлдан бірінші оң санды табу

6.9) **Закрыть** батырмасына басып бағдарлама жұмысын аяқтаңыз.

7 мысал. Экранға ANSI-кодтаудың символдарын шығару.

AnsiChar типті мәндердің жиынын қарап шығу үшін, экранға (0-ден 31-ге дейінгі кодты қызметтік символдарынан басқа) ANSI-кодтаудың барлық символдарын шығару. Ол үшін келесі іс-әрекеттерді орындау керек:

7.1) Негізгі менюдің **File/New Application** командасы көмегімен жаңа жобаны құрыңыз. Объектілер инспекторы көмегімен форманың **Caption** қасиетіне **«ANSI – кодтаудың символдарын шығару»** мәнін беріңіз.

7.2) Компоненттер палитрасының Standart бетін пайдаланып, форманың жоғарғы жағына **Memo** компонентін (Мәтіндік редактор) орналастырып, оны форманың жоғарғы жартысына созу керек. Бұл компонент формада **Memo1** деп аталатын болады. Объектілер инспекторында **Memo1** компонентінің **Lines** (жолдар) қасиетін ашып, мәтінді **(Memo1)** өшіреміз.

7.3) **«Операция»** батырмасын тінтуірмен активтендіріңіз. Оқиғаларды өңдеушіге келесі мәтінді жазыңыз:

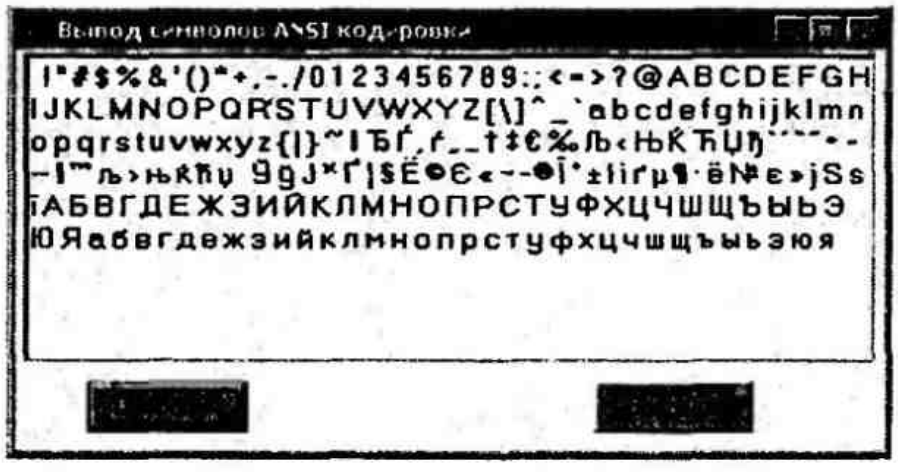
```
Procedure TForm1.Button1Click (Sender: TObject);
Var
```

```
i: Byte;
                Begin
For i:= 32 TO 255 DO
Memo1.Text:= Memo1. Text + AnsiChar(i) + '';
End;
```

Бұл ішкі бағдарламада **Memo1** компонентінде орналасқан, **Text** (Мәтін) қасиетімен анықталатын мәтінге 32-ден 255-ке дейін өзгеретін, i-кодты символдарды (AnsiChar(i)) қосу үшін **(For)** циклы ұйымдастырылған. Қосылатын символдар бір-бірінен бос орынмен (" ") бөлектенеді.

7.4) Негізгі менюдің **Run/Run** командасы көмегімен бағдарламаны жұмысқа қосыңыз.

«Операция» батырмасын басыңыз, нәтижесінде **Memo** элементі бос орынмен бөлектенген, ANSI-кодтаудың барлық символдары бар мәтінді алады (45-суретті қараңыз).



45-сурет. ANSI-кодтаудың символдары бар форманы шығару

7.5) **Закрыть** батырмасын басып бағдарламалардың жұмысын аяқтаңыз.

8 мысал.* String *типті жолдың ағымдық сипаттамаларын шығару.

8.1) Негізгі менюдің **File/New Application** командасы көмегімен жаңа жобаны құрыңыз. Объектілер инспекторы көмегімен форманың **Caption** қасиетіне «String

типті жолдың сипаттамаларын экранға шығару» мәнін беріңіз.

8.2) Модуль мәтініне келесі өзгертулерді енгізіңіз:

– Айнымалыларды баяндау бөлімінің алдына {$H+} компиляторының директивасын орналастырыңыз, ол *String* типті жолдармен **«ұзын»** жолдар секілді жұмыс орындалатынын білдіреді. Айнымалыларды баяндау бөлімінде *String* типті жаңа инициализацияланған айнымалыны енгіземіз, бұл бөлімнің түрі төмендегідей болады.

```
{$H+}
Var
Form1: TForm1;
St: string= "Жолдың сипаттамаларын шығару";
```

Бұл айнымалы **Edit** енгізу жолында терілген мәтінді орналастыру үшін қажет. Бұл енгізу жолындағы мәтіннің типі *String* болғанымен, келешекте өр түрлі максималды ұзындықты *String* типті жолдардың түрлері пайдаланылады.

– Оқиғалар өңдеушісінде **«Операция»** батырмасын басып, келесі мәтінді жазыңыз:

```
Procedure TForm1.Button1Click(Sender: TObject);
Begin
Str:= Edit1.text; {Енгізілген жол}
ListBox1.Items.Clear; {парақ-жәшік құрамы тазартады}
{әрбір жаңа жолдан ақпаратты парақ-жәшік құрамына қосады}
ListBox1.Items.Add ('Жолдың жалпы өлшемі:'
                    +InttoStr(SizeOf(Str)));
ListBox1.Items.Add ('Жолдың ағымды өлшемі:'
                    + InttoStr(Length(Str)));
ListBox1.Items.Add ('Жолдың мәтіні:'+Str);
ListBox1.Items.Add ('Str[1]'+Str[1]); {Бірінші символ}
End.
```

8.3) Компоненттер палитрасының **Standart** бетін пайдаланып, формаға **ListBox** (тізім) компонентін орналас-

тырыңыз. Модуль мәтінінде ол **ListBox1** атқа ие болады. Компонентті ұзындығы бойынша форманың үштен бір бөлігіне, ал ені бойынша шамамен енгізу жолынан батырмаға дейінгі арақашықтыққа созу керек.

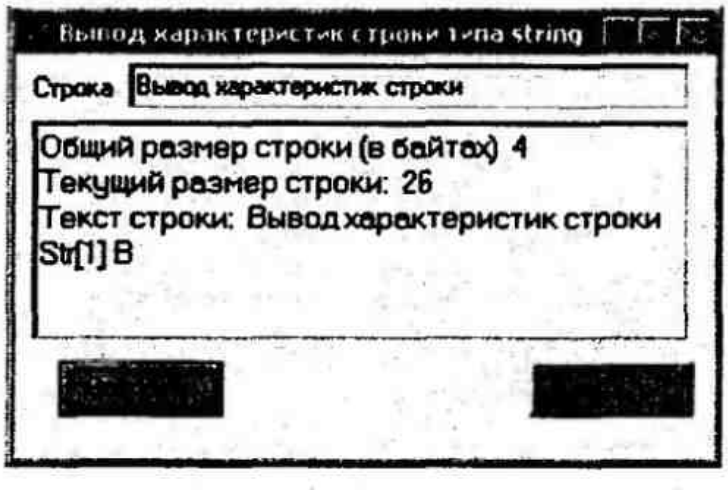
8.4) Форманы құрған кезде пайда болатын **OnCreat** (құру) оқиғасын өңдейтін **FormCreat** оқиғаларды өңдеушіде келесі мәтінді жазыңыз:

```
Procedure TForm1. FormCreat (Sender: TObject);
begin
Edit1.text:= Str;
end;
```

Бұл өңдеуші форманы құрған кезде енгізу жолының **Text** қасиетіне **Str** инициализацияланған жолдың мәнін орналастырады.

8.5) Негізгі менюдің **Run/Run** командасы көмегімен бағдарламаны жұмысқа қосыңыз. Бұл кезде енгізу жолында **Str** жолының алғашқы ақпараты (**«Жолдың сипаттамасын шығару»**) шығады.

8.6) **«Операция»** батырмасын басқанда **ListBox** элементінде **Str** жолында бастапқы орналастырылған ақпарат туралы хабар шығады (46-суретті қараңыз).



46-сурет. **String** типті жолдарды параметрлерімен формаға шығару

8.7) Әрі қарай **Edit1** енгізу жолындағы ақпаратты өзгертуге болады және жолдың параметрлері қалай өзгеретінін **«Операция»** батырмасын басып көре беруге болады.

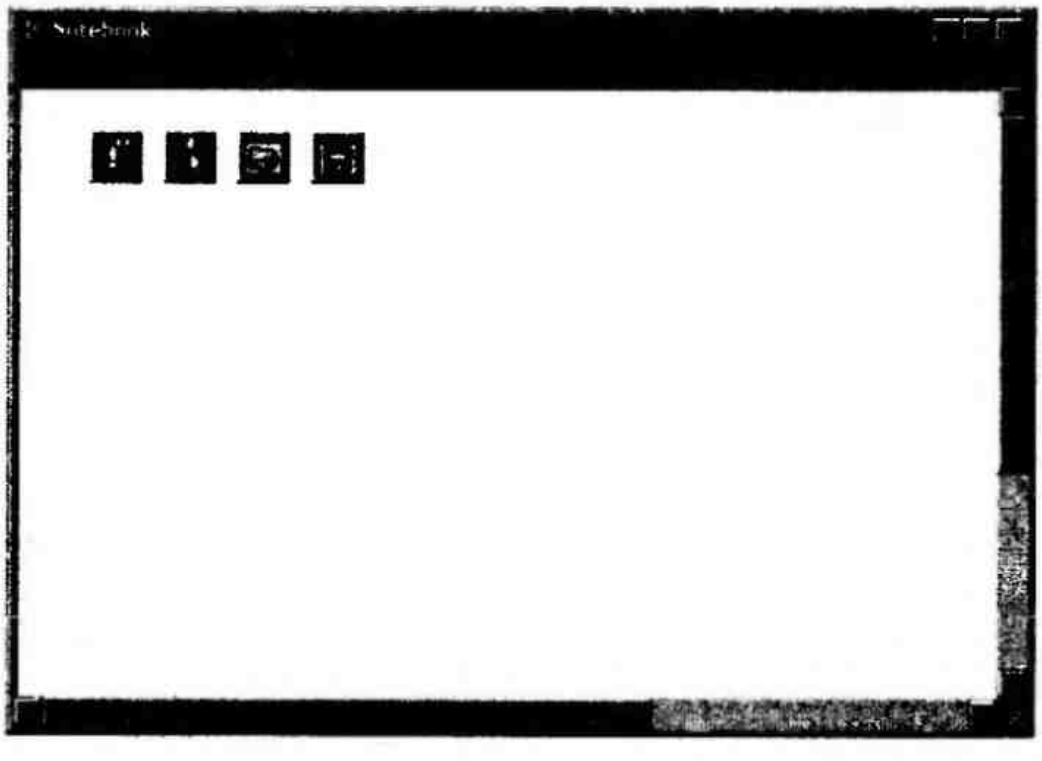
8.8) Айнымалыларды баяндау бөлімінің алдында орналасқан {$H+} компилятордың директивасын жолдың типін **«қысқа»** деп баяндайтын {$H-} директивасына өзгертіңіз. Барлық эксперименттерді қайталаңыз.

8.9) Екі нұсқадағы эксперименттерді {$H} компилятордың директивасымен қайталаңыз, бірақ **Str** жолдың типіне максималды өлшем беріңіз. Мысалы:

```
Str: STRING[10];
```

8.10) **Закрыть** батырмасын басып бағдарлама жұмысын аяқтаңыз.

9 мысал. Мәтіндік файлдары құру, өңдеу, сонымен қатар *.Btp графикалық файлдарды қарап шығу мүмкіндіктері бар мәтіндік редактор түріндегі қосымшаларды құру (47- суретті қараңыз).



47-сурет. Мәтіндік редактор

Осы қосымшаны құру үшін келесі қадамдарды орындау керек :

9.1) Жүйелік менюде **File/New Application** командасын таңдап, жаңа жобаны құрыңыз.

9.2) **Note.pas** және **Notebook.dpr** (немесе файлдардың аттарын өздеріңіз тағайындауыңызға болады) файлдардың берілген аттарымен жобаны сақтаңыз **(File/Save Project As...)**.

9.3) **Object Inspector** терезесінде **Caption** қасиетіне **Notebook** сөзін жазып, негізгі форманың атын өзгертіңіз.

9.4) **Standart** салымында орналасқан **MainMenu** объектісін формаға орналастырыңыз. Оның үстінен тінтуірдің оң жақ бастырмасын басып, **MenuDesigner**-ді жүктеңіз немесе **MainMenu** объектісінің **Items** қасиетінде **(Object Inspector** терезесі) 3 нүктесі бар батырманы басыңыз. Меню пункттерін құру мен өңдеудің терезесі ашылады. Көк түспен менюдің ағымды пункті белгіленеді. **Caption (Object Inspector** терезесі) қасиетінде меню пунктінің атын енгізіңіз (мысалы, **Файл**). Содан кейін оның үстінен тінтуір батырмасын басыңыз, оның астында менюдің жаңа пункті пайда болғанын көресіз. Оның үстінен тінтуір батырмасын басып, **Caption** қасиетінде ішкі пункттің атын енгізіңіз (мысалы, **Жаңа**). Енгізгеннен кейін ішкі пункт пайда болады, оған ат беріңіз (**Ашу**) және т.с.с. Негізгі менюдің келесі пункттері мен ішкі пункттерін құрыңыз: **Файл – «Жаңа, Ашу, Сақтау, Қалай сақтау керек... Шығу және Көмек» Автор туралы**. Менюдің қандай да бір пункті немесе ішкі пункті үшін **«Жылдам»** батырмаларды құру үшін келесі іс-әрекеттерді орындау керек: таңдалған пункт немесе ішкі пункттің Caption қасиетінде атауының алдына & белгісін қоясыз (мысалы, &Жаңа), **Short Cut** қасиетінде құлап ашылатын тізімнен батырмалар комбинациясын таңдағаннан кейін (мысалы, CTRL+N), бұл комбинацияны менюдің пунктінің жанында көресіз. Басқаларға да осы секілді. Бәрін орындағаннан кейін **MenuDesigner** меню құру мен өңдеудің терезесін жабыңыз.

9.5) Қалқып шығатын меню (яғни тінтуірдің оң жақ батырмасын басу арқылы шақырылатын меню) құру үшін формада **Popup Menu** объектісін орнатыңыз, ол **Standart** салымындағы **MainMenu** объектісінің жанында орналасқан. Өңдеу терезесін шақыру мен меню пункттерін құру 9.4 қадамына толығымен сәйкес келеді. Менюдің келесі пункттерін құрыңыз: **болдырмау CTRL+N; Қиып алу CTRL+X; Көшіру CTRL+C; Орналастыру CTRL+Y.**

9.6) Формада **Memo** объектісін орналастырыңыз (Standart салымы). **Align** қасиетінде (Object Inspector терезесі) **alClient** мәнін орнатыңыз, ал **Font** қасиетінде мәтін әріптерінің қарпі мен өлшемін көрсетіңіз. **Memo** объектінің терезесінде **Memo1** сөзін алып тастау үшін **Lines** қасиетінде 3 нүктесі бар батырманы басып, экранға объект құрамын өңдейтін терезені шығарасыз, сол жерде **Memo1** сөзін өшіріп, **OK**-ді басыңыз. **PopupMenu** қасиетінде **PopupMenu1** мәнін орнатыңыз (жолдармен жұмыс істейтін барлық визуалды объектілердің құрамында менюдің қалқып шығатын стандартты өңдеушісі болады, бірақ біздің мақсатымыз өзіміздің қалқып шығатын менюімізді құруды үйрену). **Scroll Bars** қасиетінде **ssBoth** мәнін орнатыңыз (үлкен мәтіндерді қарап шығудың ыңғайлылығы үшін айналдыру сызықтары шығады).

9.7) Формада **Image** объектісін (Additional салымы) орналастырыңыз. **Align** қасиетінде (Object Inspector терезесі) **alClient** мәнін орнатыңыз.

9.8) Формада **OpenDialog** және **SaveDialog** объектілерін (**Dialogs** салымы) орналастырыңыз. Бұл компонентер ыңғайлы интерфейсте файлдарды ашу мен сақтауға мүмкіндік береді. **OpenDialog** пен **SaveDialog** объектілерінің **Filter** қасиетінде файлдардың қандай типтерін ашуға және сақтауға рұқсат екенін көрсетуге болады. Бұл қасиетте 3 нүктесі бар батырманы басқанда, **Filter Editor** сұхбаттық терезесі ашылады. Мұнда **Filter Name** графасында файлдардың аттары, ал **Filter** бағанында сәйкес файлдардың кеңейтілулері көрсетіледі (4.1 кестені қараңыз).

24-кесте

Сәйкес файлдардың кеңейтілуі

Filter Name	Filter
Text files	*.txt
Graphic files	*.bmp
All files	*.*

9.9) Келесі қадамда негізгі және қалқып шығатын менюдің пункттерін активтендірген кезде орындалатын (қандай да бір оқиғаларды сипаттайтын) процедураларды құру мен анықтауыңыз қажет. **Файл** меню пунктін және **Жаңа** ішкі пунктін басыңыз. Форма терезесі қолданылатын барлық кластар, объектілер мен процедуралар сипатталған тереземен алмастырылады (терезелерді ауыстырып қосу үшін **F12** батырмасын қолданыңыз). Бұл терезеде **Procedure TForm1.N1Click(Sender: TObject);** жаңа процедура пайда болады, оған келесі операторларды енгізіңіз:

```
begin
if memo1.CanUndo <> false then
if messagedlg ('Save change?' ,mtconfirmation, [mbYes,
mbNo], 0) = mrYes
then memo1.Lines.SaveToFile (s);
memo1.Clear;
save1.Enabled:=false;
end;
```

Осы кезде Object Inspector терезесінде менюдің осы ішкі пунктінің барлық қасиеттері мен оқиғалары көрсетіледі. Ол **N1** атты объект болып табылады (Name қасиетін қараңыз). Егер сіз қасиеттерден (Properties) оқиғаларға (Events) ауыссаңыз, онда **onClick** оқиғасы орындалатын OnClick процедурасының атын көрсетіп тұрғанын байқайсыз.

9.10) Менюдің басқа пункттері үшін осы әрекеттерді орындаңыз.

Файл->>Ашу:

```
Procedure TForm1.N2Click(Sender: TObject);
Begin
If opendialog1.Execute then
If opendialog1.Filterindex=2 then

Begin
Memo1.Visible:= false;
Image1.Picture.LoadFromFile(opendialog1.FileName);
Form1.HorzScrollBar.Range:=1000;
Form1.VertScrollBar.Range:=1000;
End

Else
Begin
Memo1.Visible:= true;
Form1.HorzScrollBar.Range:=0;
Form1.VertScrollBar.Range:=0;
Memo1.Lines.LoadFromFile(opendialog1.FileName);
S:= opendialog1.FileName;
Save1.Enabled:=true;
End;
Form1.caption:= 'Notebook-'+s;
End;
```

Файл->>Сақтау:

```
Procedure TForm1.N3Click(Sender: TObject);
Begin
Memo1.Lines.SaveToFile(s);
Memo1.ClearUndo;
End;
```

Файл->>Қалай сақтау керек... :

```
Procedure TForm1.N4Click(Sender: TObject);
Var
BackupName: string;
```

```
FileHandle: integer;
begin
if SaveDialog1.Execute then
begin
if FileExists(SaveDialog1.FileName) then
begin
backupName:= ExtractFileName(SaveDialog1.FileName);
backupName:= ChangeFileExt(BackupName,'.BAK');
if not RenameFile(SaveDialog1.FileName, BackupName)
then
raise Exception.Create ('Unable to create backup file.');
end;
FileHandle:= FileCreate (SaveDialog1.FileName);
S:=SaveDialog1.FileName;
end;
FileClose(FileHandle);
S:= ChangeFileExt(s,'.txt');
Deletefile(SaveDialog1.FileName);
Save1.Enabled:=true;
memo1.Lines.SaveToFile(s);
memo1.ClearUndo;
end;
```

Файл->>Шығу:

```
Procedure TForm1.N5Click(Sender: TObject);
Begin
if Memo1.CanUndo <> false then
if messagedlg ('Save change?', mtconfirmation,
[mbYes,mbNo], 0) = mrYes
then memo1.Lines.SaveToFile(s);
close;
end;
```

Көмек->>Автор туралы:

```
procedure TForm1.N6Click(Sender: TObject);
begin
messagedlg ('Блокнот'#13#10'© 2000 өз аты-жө-
ніңізді енгізіңіз', mtInformation, [mbOk], 0);
end;
```

9.11) Қалқып шығатын меню пункттерінің процедуралары келесі ретпен орындалады: **PopupMenu** объектісінің **Items** қасиетіне батырманы басыңыз. **MenuDesigner** ашылған терезесінде менюдің **Болдырмау** пунктін басыңыз, **Object Inspector** терезесінің **Events** оқиғаларында **OnClick** оқиғасының аты қатарында тінтуірдің батырмасын 2 рет басыңыз. Ашылған циклға төмендегіні жазыңыз:

```
procedure TForm1.N7Click(Sender: TObject);
begin
Memo1.Undo;
end;
```

Осы секілді қалған пункттерге:

Қиып алу:

```
procedure TForm1.N8Click(Sender: TObject);
begin
Memo1.CutToClipboard;
end;
```

Көшіру:

```
procedure TForm1.N9Click(Sender: TObject);
begin
Memo1.CopyToClipboard;
end;
```

Қою:

```
procedure TForm1.N10Click(Sender: TObject);
begin
Memo1.PasteFromClipboard;
end;
```

Object Inspector терезесіндегі жоғарғы құлап ашылатын тізімнен **Form1** объектісін таңдап, **Events** салымында **OnActivate** және **OnClose** оқиғаларына келесі проце-

дураларды жазыңыз (оқиғаға процедураны құру үшін оқиға атының жанынан, бос орын үстінен қатарынан 2 рет басыңыз):

```
procedure TForm1. FormActivate(Sender: TObject);
begin
Save1.Enabled:= false;
end;

procedure TForm1.FormClose(Sender: Tobject; var
Action: TcloseAction);
begin
If memo1.CanUndo <> false then
If messagedlg ('Save change?',mtconfirmation, [mbYes,
mbNo], 0) = mrYes
then memo1.Lines.SaveToFile(s);
end;
```

Var айнымалыларды сипаттау терезесінде **Form1: TForm1;** жазбаларынан кейін **S:string;** жазып қойыңыз.

9.12) Қосымшаны жұмысқа қосу үшін **F9** батырмасын басыңыз. Қателер туралы хабар шықса, синтаксисті тексеріңіз, сіз дұрыс жаздыңыз ба соны тексеріңіз немесе оқытушыдан көмек сұраңыз. Процедурада қолданылатын объектінің қандай да бір қасиеті не үшін қажет екенін білу үшін объект атына меңзерді қойып **F1**-ді басыңыз.

10 мысал. Қатынас операцияларын қолдану.

Y=F(X) функциясының графигін құру керек болсын, ол мынадай түрге ие:

0<=X<50	Y=0,
50<=X<100	Y=X-50,
100<=X<150	Y=50,
150<=X<200	Y=-(X-200),
200<=X<250	Y=0,

Барлық өлшемдері пиксельдермен берілген.

Бұл графиктің формасы трапеция түрінде болады (48-сурет).

10.1) Негізгі менюдің **File/New Application** командасы көмегімен жаңа жобаны құрыңыз. Объектілер инспекторы көмегімен форманың **Caption** (аты) қасиетіне *«Y=F(X) қатынастар операциясы»* мәнін беріңіз.

10.2) Компоненттер палитрасының **Additional** бетін пайдаланып, графика орналастыру үшін **Image** (бейне) компонентін формаға орналастырыңыз. Компонент форма модулінде Image1 атқа ие болады. Width (ені) және Heigth (биіктік) қасиеттері көмегімен компоненттің енін 300 пиксель, ал биіктігін – 150 пиксель деп белгілеңіз.

10.3) Компоненттер палитрасының **Standart** бетін пайдаланып Image компонентінің астына **Button** компонентін орналастырып, оның **Caption** (аты) қасиетіне **«Операция»** мәнін жазыңыз.

10.4) Операция батырмасын тінтуірмен активтендіріңіз. Оқиғаларды өңдеушінің терезесінде келесі мәтінді енгізіңіз:

```
procedure TForm1.Button1Click (Sender: TObject);
VAR i: word;

begin
{горизонталь осьті саламыз}

image1.Canvas.MoveTo(25, 100);
image1.Canvas.LineTo(25+250,100);

{вертикаль осьті саламыз }

image1.Canvas.MoveTo(25,125);
image1.Canvas. LineTo(25,125-100);

{қарындаш сипаттамаларын береміз}

image1.Canvas.pen.Color:=clRed; {қызыл сызық}
```

```
image1.Canvas.pen. Width:=2; {сызық қалыңдығы 2
пиксель}
{график саламыз}
image1.Canvas.MoveTo(25,100); {график басы}

For i:=1 to 250 do
if i < 50 then                                    {1-аймақ}
image1.Canvas.LineTo(i+25,image1.Canvas.PenPos.y)
else if i < 100 then                              {2-аймақ}
image1.Canvas.LineTo(i+25,image1.Canvas.PenPos.y-1)
else if i<150 then                                {3-аймақ}
image1.Canvas.LineTo(i+25,image1.Canvas.PenPos.y)
else if i<200 then                                {4-аймақ}
image1.Canvas.LineTo(i+25,image1.Canvas.PenPos.y+1)
else                                              {5-аймақ}
image1.Canvas.LineTo(i+25,image1.Canvas.PenPos.y);
end;
```

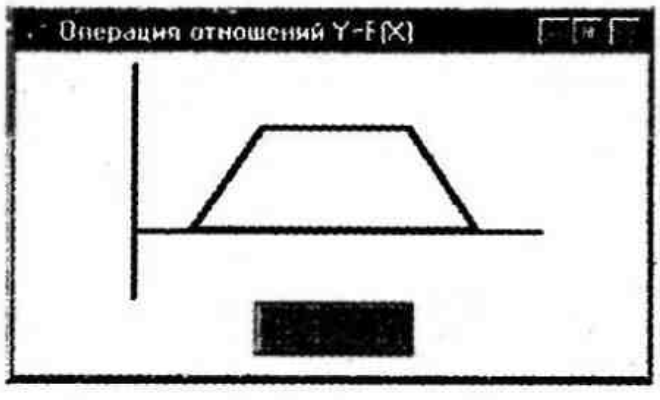
Оқиғаларды өңдеушіде алдымен координаттардың горизонталь осі бейнеленеді. Ол үшін алдымен графикалық бейненің көрінбейтін меңзері **MoveTo** әдісінің көмегімен бейнеленетін сызықтың басына орнатылады. Координаттар пиксельдермен берілетінін есте сақтау керек және де кез келген терезе үшін координаттар басы жоғары сол жақ бұрыштан басталады. Әрі қарай **LineTo** әдісі көмегімен берілген жерден бастап координаттары әдісте көрсетілген нүктеге дейінгі түзу сызық салынады. Егер қарындаш сипаттамаларын өзгертпесе, онда сызықтың ені 1 пиксель, түсі қара болады.

Графиктің өзін бейнелеген кезде қарындаштың қызыл түсті (clRed) және қалыңдығы 2 пиксель болатын сипаттамасы қойылады. Сонымен қатар, бейнелегенде қарындаш орнын анықтайтын, **Canvas-PenPos** қасиеті (бұл қасиет екі координатаны қарындаштың X және Y орналасуын береді) қолданылады. Бұл жағдайда қарындаштың Y координатасы ескеріледі, өйткені оның X координатасы **for** циклда, әрбір итерациядан кейін 1-ге арттырылып, өзгереді. Y координатасы кезекті итера-

цияда горизонтальды бөлікті бейнелеген кезде өзгермей тұрады немесе графиктің осы бағыттағы көлбеулік бұрышы 45° болғанда 1-ге азаяды, яғни теріс бағыттағы көлбеу кезінде 1-ге артады. Әрбір итерацияда X координатасының ағымды мәні қандай ағымды диапазонға түсіп тұрғаны анықталады, ол үшін қатынас операциясы қолданылады, содан кейін нақты диапазонға байланысты сәйкес сызық таңдалады. Барлық диапазонды талдап шығу үшін **if** конструкциясы пайдаланылады.

10.5) Негізгі менюдің **Run/Run** командасы көмегімен бағдарламаны жұмысқа қосыңыз.

10.6) **«Операция»** батырмасын басыңыз. Формада берілген график шығу керек (48-суретті қараңыз).



48- сурет. Графиктік бейнелеу

11 мысал. Жұмыс кезінде басқару элементтері құрылатын бағдарлама жасау.

Жалғыз ғана формадан тұратын, орындалуы кезінде басқару элементтері құрылатын бағдарлама жазу. Компоненттердің әр түрін құруды қарастырайық. Бағдарламаға аздаған өзгерістер енгізіп, бұл тізімді өзгертуге болады. Бұл жағдайда барлық компоненттер бір орында ғана бейнеленетін болғандықтан, олардың формада орын ауыстыру мүмкіншілігін ескерейік. Автоматтық режимде орындалатын компоненттің **DragMode** қасиетіне **dmAutomatic** мәні беріледі, осы мүмкіншілікке байланысты, тінтуір арқылы активтендірілмейді, бірақ табуляция пернелері көмегімен активтендіруді орындауға

болады. Мұнымен бір мезгілде компоненттің бастапқы сыртқы түрін көруге болады. Құрылып жатқан компоненттерді реттеу үшін **Caption** қасиеті көмегімен компоненттерге кезекті реттік нөмірден тұратын аттарды қою керек. Осы сәтте **Text** (мәтіндік редактор) қасиеті көрсетілгенін байқауға болады. Құрылып жатқан компоненттер алғашында берілген (компонент активтендірілуі мүмкін, батырма басылуы мүмкін, бірақ ешқандай іс-әрекет орындалмайды, енгізу жолына мәтінді енгізуге болады, бірақ оны қолдануға болмайды, т.с.с.) операциялардан басқа іс-жүзінде ешқандай әрекет орындай алмайды. Компонент пайдалы әрекеттерді орындау үшін осы іс-әрекеттерді жасаушы нақты операторларды жазу керек. Бұл факт компоненттерді жобалау кезінде компоненттер палитрасы көмегімен құру артығырақ, дұрысырақ болатынын көрсетеді.

Тапсырманы шешу үшін келесі операцияларды орындаңыз:

11.1) Негізгі менюдің **File/New Application** командасы көмегімен жаңа жобаны құрыңыз. Объектілер инспекторы көмегімен форманың **Caption** (аты) қасиетіне **«Басқару элементтерін құру»** мәнін беріңіз.

11.2) Форманың жоғарғы оң жақ бұрышында құрылып жатқан компоненттің типін беру үшін **ComboBox** енгізудің аралас жолын орналастырамыз. Бағдарламада ол **ComboBox1** деген атқа ие болады.

Осы компоненттің **Items** қасиетін пайдаланып, бағдарламада құруға болатын компоненттердің типтерін ашылатын тізімге орналастырыңыз. Ол үшін объектілер инспекторындағы енгізудің аралас жолының **Items** қасиетін тінтуірмен активтендіріп, осы қасиеттің редакторында келесі мәтінді жазыңыз:

```
Tlabel
Tedit
Tmemo
```

Tbutton
TcheckBox
TlistBox
TspeedButton
TstringGrid

11.3) Форма модулінің интерфейстік бөлімінің **uses** секциясында көрсетілмеген, модульдерде орналасқан бірнеше компоненттер құрылатын болғандықтан, модульдерді қосу керек. Олар секцияның соңғы жолында келтірілген:

Uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, ExtCtrls, Buttons, Grids, ComCtrls;

11.4) Құрылатын барлық компоненттер **Tcontrol** класынан туындайтын болғандықтан, осындай кез келген компонентті құрған кезде **Tcontrol** класының айнымалыларын қолдануға болатын еді. Бірақ компонентті құру процесінде қажет болып қалатын бірқатар сипаттамалар нақты компоненттерде жалпыға ашық болады. Оларға еркін қатынас құруға болатын болса, **Tcontrol** класында олар қорғалған және де оларды кластан тыс қолдануға болмайды. Осыған байланысты **Tcontrol** класынан туындаған жаңа **TmyControl** класын енгізіңіз. Бұл жаңа класта қосымша әдістер көмегімен қорғалған сипаттамаларға қажетті әрекеттерді іске асыруға болады. Бұл әдістерде қажетті сипаттамаларға қатынас құруға мүмкіндік бар. Ол үшін интерфейстік бөлімнің типтерді баяндау бөлімінде **TForm1** -ді баяндағаннан кейін сол типті қосыңыз:

```
type
TForm1=class (TForm)
....
End;
```

```
TmyControl=CLASS(Tcontrol)
Procedure InsertCaption(Acaption: STRING);
End;
```

Бұл класта InsertCaption әдісі құрылып жатқан компоненттің тақырыпшасын (Caption қасиеті) құрады. Модульдің орындалатын бөлімінде (implementation) әдістің мәтінін орналастырыңыз:

```
Procedure TmyControl1.InsertCaption(Acaption:
STRING);
Begin

Caption:= Acaption;
End;
```

Компонент атына оның реттік нөмірін қосу керек болғандықтан, модульдің интерфейстік бөлімінің типтерді баяндау бөлімінде **Num** инициализацияланған айнымалысын енгізіңіз:

```
type
Form1: Tform1;
Num: integer=1;
```

11.5) ComboBox енгізудің аралас жолы көмегімен құрылатын компоненттің нақты типін тандап, **«Операция»** батырмасын басып, осы компонентті құрыңыз. Ол үшін **«Операция»** батырмасын тінтуірмен активтендіріңіз, оқиғаларды өңдеушінің ашылған терезесінде келесі мәтінді жазыңыз:

```
Procedure TForm1.Button1Click(Sender: TObject);

Var s: string;
 Control: TmyControl;              {басқару элементі}
Begin
s:=UpperCase(ComboBox1.Text); {бас әріпті мәтін}
if s= 'TLABEL' then
```

```
Tcontrol(Control):= Tlabel.Create(Form1)  {белгі}
else if s= 'TEDIT' then
Tcontrol(Control):= Tedit.Create(Form1) {енгізу жолы}
else if s= 'TMEMO' then
Tcontrol(Control):= Tmemo.Create(Form1) {мәтін
                                          редакторы}
else if s= 'TBUTTON' then
Tcontrol(Control):=Tbutton.Create(Form1) {батырма}
else if s= 'TCHECKBOX' then
Tcontrol(Control):=TcheckBox.Create(Form1)
                                          {өшіргіш}
else if s= 'TLISTBOX' then
Tcontrol(Control):=Tlistbox.Create(Form1) {сызғышы
                                          бар тізім}
else if s= 'TSPEEDBUTTON' then
Tcontrol(Control):=TspeedButton.Create(Form1)
                                  {бейнесі бар батырма}
else if s= 'TSTRINGGRID' then
Tcontrol(Control):=TstringGrid.Create(Form1) {жолдар
                                          кестесі}
else
begin
Messagedlg ('Рұқсат етілмеген элемент',mtWarning,
[mbOk], 0);
exit;
end;

{компонент параметрлерін беру}

Control.InsertCaption('Комп.N'+IntToStr(Num)); {аты}

Inc(Num);

Control.DragMode:=dmAutomatic; {Автомат.
                                Орын ауыстыру}
Form1.InsertControl (Control);  {Control тізімінде
                                орналастыру}
end;
```

Әдіс төмендегідей жұмыс істейді. Алдымен **ComboBox1** енгізудің аралас жолының мәтіні **UpperCase** әдісі көмегімен үлкен әріптерге түрлендіріледі, содан кейін қандай компонентті құру керек екені талданады да, компонент, **Create** конструкторы көмегімен құрылады.

Егер символдар тізімі дұрыс болмаса, онда «рұқсат етілмеген элемент» деген хабар шығып, әдіс өзінің жұмысын аяқтайды. Егер де символдар тізімі дұрыс болса және сәйкес компонент құрылса, оған **InsertCaption** әдісі көмегімен сәйкес ат беріледі, құрылған компоненттер санының санағышы арттырылады, компоненттер орын ауыстыруының автоматты режимі орнатылып, компонент **InsertControl** әдісі көмегімен форманың **Control** тізіміне орналастырылады.

11.6) Форма бетінде компоненттерді орналастыру режимін ұйымдастырыңыз. Компоненттерді орын ауыстырудың автоматты режимі алдыңғы әдіспен орнатылып қойылған. Енді компонентті формада орын ауыстыруға рұқсат ету және оны сол жерде қалдырып қоюды қамтамасыз ету керек. Ол үшін, алдымен форманы белгілеңіз, объектілер инспекторының **Events** бетінде форманың **OnDragOver** оқиғасын тінтуірмен активтендіріңіз. Оқиғаларды өңдеушінің ашылған терезесінде келесі мәтінді белгілеңіз:

```
Procedure TForm1.FormDragOver (Sender, Source:
Tobject; X,Y: integer; State: TdragState; var Accept:
Boolean);

begin
Accept:=True;
end;
```

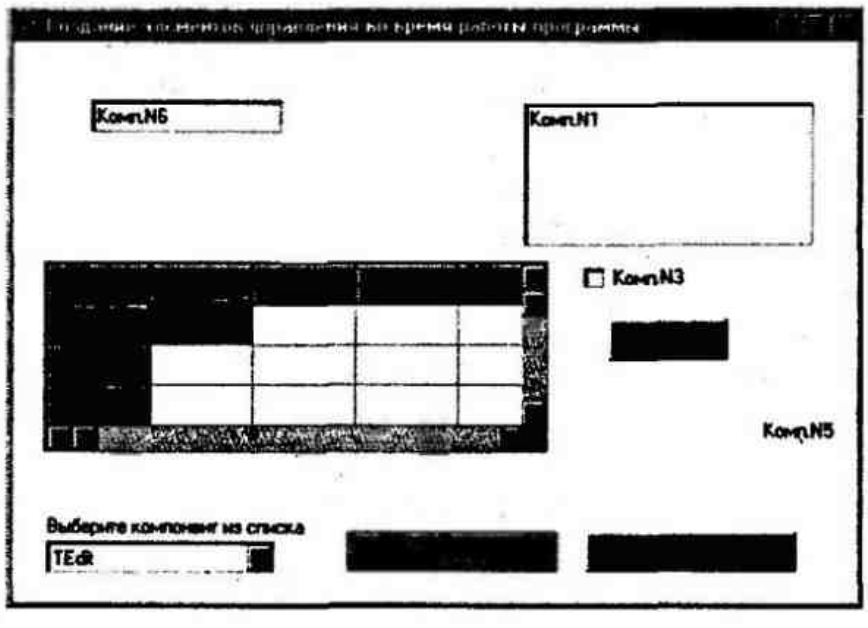
Бұл өңдеуші, орыны ауыстырылып жатқан компоненттің орын ауыстыру және (біздің жағдайда формаға) орналастыру мүмкіншілігін анықтайтын **Accept** параметріне **True** мәнін меншіктейді.

Осы төрізді форманың **OnDragDrop** оқиғасын тінтуірмен активтендіріп, оқиғаларды өңдеушіге төмендегі мәтінді жазыңыз:

```
Procedure TForm1.FormDragDrop(Sender, Source:
Tobject; X,Y: integer);
begin
With Source as Tcontrol do
begin
Left:=X;
Top:=Y;
end;
end;
```

Бұл әдіс компонентті тасымалдаған кезде, тінтуірдің координаталарымен анықталатын, оның жоғарғы сол жақ бұрышының жаңа координаттарын береді.

11.7) Негізгі менюдің **Run/Run** командасы көмегімен бағдарламаны жұмысқа қосыңыз. Экранда құрылған форма пайда болады. Енгізудің аралас жолынан компоненттің типін таңдап, **«Операция»** батырмасын басыңыз. Форманың жоғарғы сол жақ бұрышында сәйкес компонент шығуы керек (49-суретті қараңыз).



49-сурет. Құрылған компоненттері бар форманы шығару

11.8) Компонентті форманың кез келген басқа жеріне ауыстырып, жаңа компонентті құрыңыз. Осындай компоненттердің шексіз көп санын құруға болады

11.9) **Закрыть** батырмасын басып, бағдарлама жұмысын аяқтаңыз.

12 мысал. Күнтізбе бағдарламасын құру.

Бағдарламада бетбелгі көмегімен кез келген айды таңдап, ағымды айдың күнтізбесі шығарылуы қажет, сонымен қатар жылды беру мен апта күндерін шығару мүмкіншілігін қарастыру керек.

Тапсырманы шешу үшін келесі операцияларды орындаңыз:

12.1) Негізгі менюдің **File/New Application** командасы көмегімен жаңа жобаны құрыңыз. Объектілер инспекторы көмегімен форманың **Caption** (аты) қасиетіне **«Күнтізбе»** мәнін беріңіз.

12.2) Компоненттер палитрасының Win95 бетін пайдаланып, формаға **TabControl** (бет белгілер) компонентін орналастырыңыз. Модуль мәтінінде ол **TabControl1** деген атқа ие болады. Бұл компонентті форманың барлық ені бойынша созу керек. Осы компоненттің Tabs қасиетін активтендіріңіз. Ашылған терезеде әрқайсысы әр айдың алғашқы үш әрпінен тұратын 12 жолды жазыңыз.

12.3) **TabControl1** бет белгілер өрісінде компоненттер палитрасының **Samples** бетінен **Calendar** күнтізбе компонентін орналастырыңыз. Бағдарламада ол **Calendar1** атын алады. Компонентті ені бойынша барлық форманы алатындай етіп созыңыз. Күнтізбе компонентінің барлық жұмыс аймағын алатындай етіп, **TabControl1** компонентінің биіктігін кішірейтіңіз. Күнтізбе әдеттегідей күнтізбе секілді көріну үшін **StartOfWeek** қасиетіне 1 мәнін меншіктеңіз. Бұл жағдайда апта дүйсенбіден басталады.

12.4) Жоғарғы жағында, форманың ортасында орналасатындай етіп бағдарламаның аты үшін **Label** белгісін орналастырыңыз. Белгінің **Font** қасиетін активтендіріп,

ашылған терезеде **Size** қаріп өлшемін 14 деп орнатып, белгінің **Caption** қасиетінде **«Күнтізбе»** атын беріңіз.

12.5) Осы белгіден төмен солға қарай компоненттер палитрасының **Samples** бетінен **SpinEdit** енгізу жолын орналастырыңыз. Енгізу жолы бағдарламада **SpinEdit1** атқа ие болады. Оны жолды көрсету үшін пайдаланыңыз. Бұл компоненттің үстіне **Label** белгісін орналастырыңыз, оның **Caption** қасиетіне **«Жыл»** мәнін беріңіз.

12.6) Енгізу жолының оң жағына компоненттер палитрасының стандарт бетіне **Edit** енгізу жолын орналастырыңыз. Бұл компонентке апта күні шығарылады. Бұл жерде қандай да бір ақпаратты жазуға тыйым салу үшін **ReadOnly** қасиетіне **true** мәнін меншіктеңіз. Бұл енгізу жолының жоғарғы жағында **Label** белгісін қосып, оның **Caption** қасиетін **«Апта күні»** мәнін енгізіңіз.

12.7) Бет белгілерді айлармен байланыстырыңыз, сонда күнтізбе осы ай үшін шығады. Ол үшін **TabControl1** компонентін белгілеңіз (оның кез келген бір бет белгісіне тышқанның курсорын орналастырып), объектілер инспекторының **Events** бетінде **OnChange** оқиғасын тышқанмен активтендіріңіз. Дайын болған оқиғалар өңдеушісіне келесі мәтінді жазыңыз:

```
procedure TForm1. TabControl1Change (Sender:
Tobject);
begin
Calendar1.Month:= TabControl1.Tabindex+1;

end;
```

Мұнда **Calendar1** компонентінің **month** (ай) қасиетіне активті бет белгінің 1-ге арттырылатын реттік нөмірінің мәні меншіктеледі.

12.8) Жыл енгізілетін жолды күнтізбемен байланыстырыңыз. Ол үшін:

SpinEdit1 енгізу жолын тінтуірмен белгілеңіз, **Events** бетінде **OnChange** оқиғасын активтендіріңіз. Оқиғалар өңдеушінің терезесінде келесі мәтінді енгізіңіз:

```
procedure TForm1. SpinEdit1Change (Sender: Tobject);
begin

Calendar1.Year:=SpinEdit1.Value;
end;
```

Бұл өңдеушіде **Calendar1** компонентінің **Year** қасиетіне **SpinEdit1** енгізу жолының **Value** мәні меншіктеледі. Бұл жаңа мән, тінтуір арқылы басқа бір компонентті белгілеп немесе табуляция пернесін қолданып, **SpinEdit1** енгізу жолынан шығып кеткен жағдайда ғана меншіктеледі.

12.9) Күнтізбені апта күні көрсетілетін жолмен байланыстырыңыз. Ол үшін **Calendar1** компонентін тінтуірмен белгілеп **Events** бетінде **OnChange** оқиғасын активтендіріңіз. Оқиғаларды өңдеушінің терезесінде келесі мәтінді орналастырыңыз:

```
Procedure TForm1. Calendar1Change (Sender: Tobject);
begin
CASE dayofweek(calendar1. CalendarDate) OF

1:edit1.Text:= 'Жексенбі';
2:edit1.Text:= 'Дүйсенбі';
3:edit1.Text:= 'Сейсенбі';
4:edit1.Text:= 'Сәрсенбі';
5:edit1.Text:= 'Бейсенбі';
6:edit1.Text:= 'Жұма';
7:edit1.Text:= 'Сенбі';
end;

end;
```

12.10) Бағдарламаны жұмысқа қосқан мерзіммен анықталатын күнтізбенің бастапқы күйін беріңіз. Ол үшін форманың өзін тінтуірмен белгілеп, **Events** бетінде **OnCreat** оқиғасын активтендіріңіз. Оқиғаларды өңдеушінің терезесінде келесі мәтінді орналастырыңыз:

```
Procedure TForm1.FormCreate (Sender: Tobject);
Begin
Calendar1.CalendarDate:=Date;
SpinEdit1.Value:=calendar1.Year;
TabControl1.TabIndex:=calendar1.Month-1;
end;
```

Мұнда Date стандартты функция қолданылады, оған компьютерде орнатылған ағымды мерзім беріледі. Алынған мән **calendar1** компоненттің **CalendarDate** қасиетіне меншіктеледі. **SpinEdit1** енгізу жолының **Value** қасиетіне ағымды жылды анықтайтын, **calendar1** компонентін **Year** қасиетінің мәні меншіктелінеді. Осымен қоса, ағымды айды ескере отырып, **TabControl1** компонентінің сәйкес бет белгісі активтелінеді.

12.11) Негізгі менюдің **Run/Run** командасы көмегімен бағдарламаны жұмысқа қосыңыз. Экранда күнтізбесі бар құрылған форма шығуы керек (50-суретті қараңыз).

50-сурет. Күнтізбе бағдарламасы

13 мысал. Сұхбаттық терезе көмегімен форма сипаттамаларын беруге болатын жобаны құру.

Сұхбаттық терезенің көмегімен форма сипаттамаларын беретін жоба құрыңыз. Сұхбаттық терезеде форма өңін салатын қылқалам стильдері және форма жақтауының көптеген батырмалары берілетін болады.

Ол үшін келесі операцияларды орындаңыз:

13.1) Негізгі менюдің **File/New Application** командасы көмегімен жаңа жобаны құрыңыз. Объектілер инспекторы көмегімен форманың **Caption** (аты) қасиетіне **«Форма сипаттамаларын беру»** мәнін беріңіз.

13.2) Жобаға оң жағында орналасқан екі батырмасы бар сұхбаттық терезенің дайындығын қосыңыз. Ол үшін Delphi-дің негізгі меню көмегімен орындалатын **File/New/Dialog/StantartDialog** операциясын пайдаланыңыз. Мұндай операция екеу, батырмалары оң жағында орналасқан сұхбаттық терезеге сәйкес келетінін таңдау керек (сәйкес суретте көрсетіліп тұрады). Осы сұхбаттық терезеге сәйкес келетін айнымалының аты – **OkRightDlg**. Ішкі жақтау шекарасының ішінде бірінің астында бірі орналасатындай етіп 8 ауыстырып-қосқышты (компоненттер палитрасының **RadioButton** компоненттері) және бір белгіні (Label компонент) қоя алатындай етіп, терезе өлшемін биіктігі бойынша үлкейтіңіз. Бағдарлама мәтінінде **Bevel1** атын иемденетін ішкі жақтауды да осыған сәйкес үлкейтіңіз.

Ішкі жақтауға қатарынан екі **Label** белгісін орналастырыңыз. Объектілері инспекторы көмегімен бұл белгілерге **«қыл қалам стилі»** мен **«жақтау батырмалары»** аттары беріледі. **«Қыл қалам стилі»** белгісінің астына бір-бірінің астында болатындай сегіз **RadioButton** ауыстырып-қосқыштарды орналастырыңыз. Бұлардың бәрі бағдарламада **RadioButtonN** аттарына ие болады, мұндағы N – ауыстырып-қосқыштың реттік нөмірі (1-ден 8-ге дейін). Осы ауыстырып-қосқыштардың **Caption** қасиетін пайдаланып жоғарыдан төмен қарай, келесі атауларды беріңіз.

Тұтас
Бос
Оң қиғаш
Сол қиғаш
Түзу тор
Қиғаш тор
Көлденең тор
Тік тор

Жақтау батырмаларының белгісінің астына бірінің астына бірі орналасатындай етіп **Standart** бетінен 4 **CheckBox** ажыратқышын орналастырыңыз. Олар бағдарламада **CheckBoxN** аттарына ие болады, мұндағы N – ажыратқыштардың реттік нөмірі (1-ден 4-ке дейін). Осы ажыратқыштардың **Caption** қасиетін пайдаланып, оларға жоғарыдан төменгі ретпен келесі атауларды беріңіз:

Жүйелік меню
Минимизация
Максимизация
Анықтама

13.3) Сұхбаттық терезені ашқанда ауыстырып-қосқыштар мен ажыратқыштар негізгі терезе күйіне сәйкес келетін күйде тұруы керек болғандықтан, сұхбаттық терезенің экрандағы бейнесінің алдында шығатын, осы терезенің **OnShow** оқиғаларды өңдеушісін жазыңыз. Ол үшін сұхбаттық терезені белгілеп, объектілер инспекторының **Events** бетінде **OnShow** оқиғасын активтендіріңіз. Оқиғаларды өңдеушінің терезесінде келесі мәтінді орналастырыңыз:

```
Procedure TOKRightDlg.FormShow(Sender: Tobject);
Begin
CASE form1.Brush.style OF                {толтыру стилі}
bsSolid: RadioButton1.Checked:=True;     {тұтас}
bsClear: RadioButton1.Checked:=True;     {бос}
bsBDiagonal:RadioButton1.Checked:=True;  {сол
                                          қиғаш}
```

```
bsFDiagonal:RadioButton1.Checked:=True; {оң қиғаш}
bsCross:RadioButton1.Checked:=True; {тор}
bsDiagCross:RadioButton1.Checked:=True; {қиғаш
                                          тор}
bsHorizontal:RadioButton1.Checked:=True;
                                          {көлденең}
bsVertical:RadioButton1.Checked:=True;  {тік}
END;

{жақтаудағы батырма}

If blSystemMenu IN form1.BorderIcons THEN {жүйелік
                                          меню}
CheckBox1.Checked:=True;      {ауыстырып-қосқыш
                               қосылған}

ELSE
CheckBox1.Checked:=False;     {ауыстырып-қосқыш
                               өшірілген}
If biMinimize IN form1.BorderIcons THEN {терезе
                  минимизацияланған} (кішірейтілген)
CheckBox2.Checked:=True;
ELSE
CheckBox2.Checked:=False;
If biMaximize IN form1.BorderIcons THEN {терезе
                  максимизацияланған} (үлкейтілген)
CheckBox3.Checked:=True;
ELSE
CheckBox3.Checked:= False;
If biHelp IN form1.BorderIcons THEN     {анықтама}
CheckBox4.Checked:=True;
ELSE
CheckBox4.Checked:= False;

End;
```

Бұл өңдеушіде алдымен форма қыл қаламының қазіргі стиліне байланысты белгіленген ауыстырып-қосқыш

орнатылады. Содан кейін форма жақтаушыларына орналасқан батырмаларға сәйкес ажыратқыш орнатылады.

13.4) Сұхбаттық тереземен барлық операциялар орындалғаннан кейін, құрылып жатқан сұхбаттық терезенің типі мен осы терезені анықтайтын айнымалы төмендегідей болады:

```
Type
TOKRightDlg=class(TForm)
OkBtn: Tbutton;
CancelBtn: Tbutton;
Bevel1: Tbevel;
Label2: Tlabel;
RadioButton1: TradioButton;
RadioButton2: TradioButton;
RadioButton3: TradioButton;
RadioButton4: TradioButton;
RadioButton5: TradioButton;
RadioButton6: TradioButton;
RadioButton7: TradioButton;
RadioButton8: TradioButton;
CheckBox1:TcheckBox;
CheckBox2:TcheckBox;
CheckBox3:TcheckBox;
CheckBox4:TcheckBox;

Procedure FormShow(Sender: Tobject);

private

{private declarations}

Public

{public declarations}

end;

var

 OKRightDlg: TOKRightDlg;
```

13.5) Негізгі формаға өтіп, **"Операция"** батырмасын активтендіріңіз. **OnClick** оқиғаны өңдеушінің терезесінде келесі мәтінді енгізіңіз:

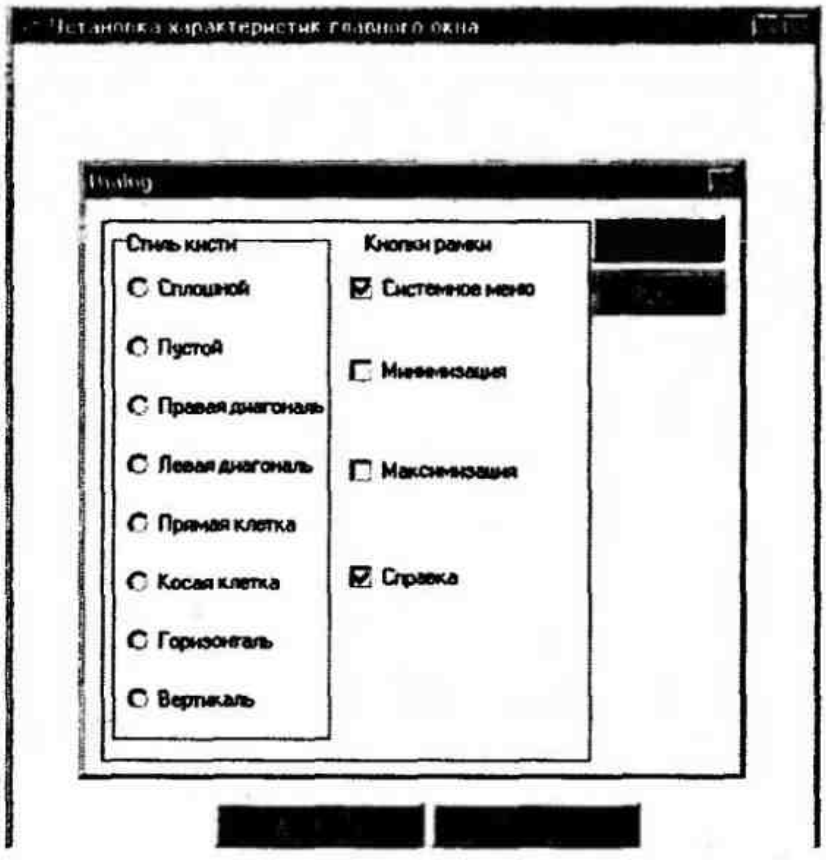
```
Procedure TForm1.Button1Click(Sender:TObject);
Var Icons: TborderIcons;
Begin

If OKRightDlg.ShowModal=mrOK THEN
BEGIN {кисть стилін орнату}

If OKRightDlg. RadioButton1.Checked THEN {тұтас}
Form1.Brush.Style:=bsSolid
ELSE
IF OKRightDlg. RadioButton2.Checked THEN {бос}
Form1.BrushStyle:=bsClear
ELSE
IF OKRightDlg. RadioButton3.Checked THEN {сол
                                          қиғаш}
Form1.Brush.Style:=bsBDiagonal
ELSE
IF OKRightDlg. RadioButton4.Checked THEN {оң
                                          қиғаш}
Form1.Brush.Style:=bsFDiagonal
ELSE
IF OKRightDlg. RadioButton5.Checked THEN {тор}
Form1.Brush.Style:=bsCross
ELSE
IF OKRightDlg. RadioButton6.Checked THEN {қиғаш
                                          тор}
Form1.Brush.Style:=bsDiagCross
ELSE
IF OKRightDlg. RadioButton7.Checked THEN {көлденең}
Form1.Brush.Style:=bsHorizontal
ELSE
IF OKRightDlg. RadioButton8.Checked THEN {тік}
Form1.BrushStyle:=bsVertical;
```

```
Icons:=[]; {жақтау батырмаларын орнату}

If OKRightDlg.CheckBox1.Checked THEN
Icons:=Icons+[biSystemMenu]; {жүйелік меню}
If OKRightDlg.CheckBox2.Checked THEN
Icons:=Icons+[biMinimize]; {терезе кішірейтілген}
If OKRightDlg.CheckBox3.Checked THEN
Icons:=Icons+[biMaximize]; {терезе үлкейтілген}
If OKRightDlg.CheckBox4.Checked THEN
Icons:=Icons+[biHelp]; {анықтама}
BorderIcons:=Icons;
Refresh; {енгізілген өзгерістерді белгілейді}
End;
End;
```

Бұл өңдеушіде сұхбаттық терезе модальды режимде жұмысқа қосылады және ол қалай (қандай батырма көмегімен) жабылғаны талданады. Егер терезе **ОК** бастырмасы көмегімен жабылса, енгізілген ақпаратты талдау орындалады, қарсы жағдайда енгізілген ақпарат ескерілмейді. Талдау процесі кезінде алдымен қылқаламның әр түрлі стильдеріне сәйкес келетін ауыстырып-қосқыштардың қайсысы таңдалғаны анықталады, содан кейін қылқалам үшін осы стиль тағайындалады. Содан кейін терезе жақтауының әр түрлі батырмаларына сәйкес келетін ажыратқыштың қайсысы қосылғаны талданып, форма терезесі үшін сәйкес батырмалар орнатылады. Онан соң енгізілген өзгерістерді көрсету үшін, **Refresh** әдісі көмегімен терезе бейнесі жаңартылады.

13.6) Негізгі менюдің **Run/Run** командасы көмегімен бағдарламаны жұмысқа қосыңыз. **«Операция»** батырмасын басыңыз. Экранда сұхбаттық терезе ашылуы қажет (51-суретті қараңыз). Негізгі формадағы қылқаламының әр түрлі стильдері мен осы форманың жақтауындағы батырмалардың жиындарын беруге болады. Сұхбаттық терезені **ОК** немесе **Cancel** пернелері көмегімен жаба отырып, формадағы өзгерістерге көңіл бөліңіз. Қылқаламның **bsClear** стилін берген кезде және жақтаудың

Help батырмасының орындалуында ерекшелік бар (ол әрқашанда шыға бермейді).



51-сурет. Негізгі терезенің сипаттамаларын орнату

13.7) **Закрыть** батырмасын басып, бағдарлама жұмысын аяқтаңыз.

IV ТАРАУ

WINDOWS 9X, WINDOWS NT ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕРІ ҮШІН БАҒДАРЛАМАЛЫҚ ЖАБДЫҚТАРДЫ ЖОБАЛАУ. ДИНАМИКАЛЫҚ МАССИВТЕР

4.1. Динамикалық массивтер құру

Динамикалық массивтердің нақты ұзындығы мен нақты өлшемі болмайды. Осындай массивтердің элементтерін сақтауға қажетті жады массив элементтеріне мәндерді меншіктеген кезде немесе SetLength процедурасын шақырған кезде динамикалық түрде үлестіріледі.

A: array [1..100] of string; секілді жай статикалық массивтерді баяндаумен қатар, бағдарламаларда динамикалық массивтерді де баяндау мүмкіндігі туындады.

Динамикалық массивтер Object Pascal-ға Delphi 4-тен бастап енгізілген. Динамикалық массивті баяндағанда оның негізгі сипаттамалары (элементтердің өлшемі - dimensions – және типі секілді) анықталады, бірақ элементтер саны көрсетілмейді.

Баяндау синтаксисі:

<Аты> **array of** <базалық тип>

Мысалы, келесі бағдарламалық код

A: **array of** integer;

B: **array of array of string**; екі динамикалық массивтің баяндалуын көрсетеді. Мұндағы А бүтін сандардың бір өлшемді массиві, ал В – жолдың екі өлшемді массиві.

Динамикалық массивті баяндағанда оған жады бөлінбейтінін ескеру керек, компиляторға неше элемент сақтау керегін көрсету үшін SetLength процедурасы қолданылады. Мысалы, SetLength (A, 20) операторы жадыда Integer типті жиырма мәнді сақтауға қажет аймақты бөліп береді. Динамикалық массивтердің индекстері әрқашанда 0-ден басталады, сондықтан да А массивінің элементтеріне қатынас құру мысалында 0-ден 19-ға дейінгі индекс мәндерін пайдалану қажет.

Динамикалық массивтер түрінде баяндалған айнымалылар шын мәнінде жай сілтемелер болып табылады; олар Delphi-дің алдыңғы версияларындағы ұзын жолдарға ұқсас қолданылады (яғни, компилятор жадының белгіленген аймағына бағыттайтын сілтемелерді есептейді; қашан олардың саны нөлге тең болғанда, жады босатылады). Сонымен, динамикалық массивтің элементтері алып жатқан жадыны босату үшін осы айнымалыға nil мәнін меншіктесе жеткілікті (әрине Finalize процедурасын да шақыруға болады, бірақ бұл әдіс, егер басқа ешқандай айнымалы берілген жадыға сілтеме жасамаса ғана жадыны босатады).

Осылайша, келесі операторлар бір-біріне бара-бар:

```
A:= nil;
Finalize (A);
SetLength (A, 0);
```

Ескерту

Динамикалық массивтерге ешқашан New және Dispose процедураларын қолданбаңыз.

Динамикалық массивтермен жұмыс істегенде келесі мысалда келтірілген ерекшелікті есте сақтау керек. Бағдарламада екі динамикалық массив – X және У берілген делік. X:=У операторын орындағанда, X массивінің ұзындығы У массивінің ұзындығына тең болады және де X айнымалысында У айнымалысы нұсқап тұрған де-

ректерді көрсететін нұсқағыш сақталады. Басқа сөзбен айтқанда динамикалық массивтердің ұзын жолдары мен статикалық массивтерінен айырмашылығы динамикалық массивтерді меншіктеу операциясы кезінде өз элементтерінің мәндерін көшірмейді. Мысалы, келесі бағдарламалық код орындалғаннан кейін

```
Var
A,B: array of integer;
begin
SetLength (A, 1);
A [0] := 1;
B := A;
B[0] := 2;
end;
```

A[0] мәні 1-ге емес, 2-ге тең болмақ. Бұдан басқа, «динамикалық массив» типті айнымалыларды салыстырғанда олардың элементтерінің мәндері емес, оларда сақталатын нұсқағыштар салыстырылады. Мысалы:

```
var
A, B: array of integer;
begin
SetLength (A, 1);
SetLength (B, 1);
A[0] := 2;
B[0] := 2;
end;
```

бағдарламалық коды орындалғанда (A := B) өрнегі False мәнін, ал (A[0]= B[0]) – өрнегі True мәнін қайтарады (возврат).

Динамикалық массивтің ұзындығын кішірейту үшін *Copy* функциясын қолданамыз және шыққан нәтижені сол «динамикалық массив» типті айнымалының өзіне меншіктейміз. Мысалы, A := Copy(A, 0, 20) операторы индексі 19-дан үлкен A массивінің барлық элементін кесіп тастайды.

Динамикалық массив инициализацияланғаннан кейін оған: Length – ұзындығы, High – индекстің ең үлкен мәні, Low – индекстің ең кіші мәні стандартты функцияларын қолдануға болады. Егер массив ұзындығы нөлдік болса, онда High функциясы 1 мәнін қайтарады. Динамикалық массив айнымалысының өзі массив басына нұсқауыш болып табылады. Егер массивке жадыда орын бөлінбеген болса, онда айнымалының мәні nil-ге тең. Бұл жай нұсқауыш емес. Оны « ^ » операциясымен белгілеуге болмайды.

Көп өлшемді массивтерді баяндау үшін **array of** – құрылымының керекті санын қолдану керек. Мысалы, келесі бағдарламалық код

```
type
TMessageGrid = array of array of string;
var
Msgs: TMessageGrid;
```

қатарлардың екі өлшемді массивін баяндайды. Бұл массивті инициализациялау үшін SetLength процедурасында екі бүтін санды параметрлерді көрсету керек

```
SetLength (Msgs, 4, 5);
```

Осылайша, Msgs айнымалысы 4*5 жолдан тұратын массивке сілтеме жасайды. Тікбұрышты массивті құруға да болады, мысалы, екінші параметр тұрақты емес, жол нөміріне қатысты өзгеріп отырады. Бұл жағдайда SetLength процедурасы арқылы алдымен бірінші параметр беріледі. Мысалы:

```
SetLength(A2, 3);
```

операторы бірінші өлшемді 3-ке тең деп алады. Жадыда 3 жолды динамикалық массивке A2[0], A2[1], A2[2] орын бөлінеді. Бұл массивтердің өлшемі анықталмаған. Енді, мысал үшін олардың біріншісінің өлшемін 4 деп берейік

```
SetLength (A2[0], 4);
```

ал келесінің өлшемін 5 деп алайық:

```
SetLength (A2[1], 5);
...
```

Төменде кез келген N өлшемді астыңғы үшбұрышты матрицаны құру және толтыру бағдарламасы берілген:

```
var A2: array of array of integer;
N, i1, i2, m: integer;
begin
N := 3;
M := 1;
SetLength(A2, N); {Жол санын беру – N}
For i1 := 0 to N do {Жол бойынша цикл}
begin
SetLength(A2[i1], i1+1); {баған саны жол нөміріне тең}
For i2 := 0 to i1 do
begin
A2 [i1, i2] := m; {жолды толтыру}
Inc(m); {m-ді 1-ге өсіру}
end;
end;
end;
```

Бағдарлама екі өлшемді массив құрады. Мұндағы жол саны – N , ал әр жолдағы бағана саны жол нөміріне тең. Бұл мысалдан көрініп тұрғандай динамикалық массивтер статикалық массивтермен салыстырғанда қолданатын жады көлемін кішірейтуге мүмкіндік береді.

Қорыта айтсақ, динамикалық массивтерді процедуралар мен функцияларға нақты параметр ретінде беруге болады.

```
function CalcSum (A: array of integer): integer;
var
```

```
Index: integer;
begin
Result := 0;
For Index := Low(A) to High (A) do
Result := Result + A[index];
end;

var
AAA: array of integer;
Sum: integer;
 begin
SetLength (AAA, 4);
AAA[0] := 1; AAA [1] := 2;
AAA[2] := 4; AAA [3] := 4;
Sum := CalcSum(AAA);
end;
```

4.2. Сандық массивтерді өңдеу функциялары

25-кестеде келтірілген функциялар **math** модулінде анықталған. Бұл функцияларды компилятор деп түсіну үшін, **uses** операторы арқылы қосамыз.

25-кесте

Сандық массивтерді өңдеу функциялары

MaxIntValue (const Data: array of Integer): Integer	Бүтін санды массив элементтерін таңбасымен қамтып максимал мән қайтарады.
MinIntValue (const Data: array of Integer): Integer	Бүтін санды массив элементтерін таңбасымен қамтып минимал мән қайтарады.
MinValue (const Data: array of Double): Double;	Нақты санды массив элементтерін таңбасымен қамтып минимал мән қайтарады.
Sum(const Data: array of Double): Extended	Data массиві элементтерінің қосындысын қайтарады.
MaxValue (const Data: array of Double): Double	Нақты санды массив элементтерін таңбасымен қамтып максимал мән қайтарады.

4.3 Жадыда бейнеленген файлдармен жұмыс

Win32 жүйесінің қолайлы тәсілдерінің бірі – дискідегі файлдарға, олардың жадыдағы сақталатын мәліметтеріне қатынауға мүмкіндік береді. Бұл тәсіл жадыда бейнеленген файлдар арқылы жүзеге асады. Жадыға бейнеленген файлдар арқылы барлық файлдық енгізу-шығару операцияларын орындау қажеттілігінен құтылуға болады. Оның орнына виртуалды мекен кеңістігінің кейбір аумағы резервтеліп, дискідегі файлдың физикалық жадысы осы резервтелген жады кеңістігінің мекенімен байланысады. Бұдан кейін файлда сақталған мәліметке нұсқауыш арқылы сілтеме жасауға болады. Ол нұсқауыш резервтелген аумақта әрекет етеді.

Жадыға бейнеленген файлдарды қолдану

Мұндай файлдарды қолданатын 3 аумақ бар:

- Win 32 жүйесі, жадыға бейнеленген файлдар көмегімен Exe және Dll файлдарын жүктеп, орындайды. Мұндай файлдардың жүктелу уақыты азаяды.
- Жадыға бейнеленген файл сақтайтын мәліметтерге қатынас құру бейнеленген жады аумағына бағыттайтын нұсқауыш арқылы жүзеге асырылады. Мұндай жағдайларда файлдарға қатынау процесі жеңілдетіліп қана қоймай, файлдарды әр түрлі жолмен буферлеу сұлбасын пайдалану қажеттігі жойылады.
- Жадыға бейнеленген файлдар бір компьютерде атқарылатын әр түрлі процестер арасында деректерді бөлу мүмкіндігін береді.

Жадыға бейнеленген файлдардың тек екінші қолдануын қарастырайық.

Жадыда бейнеленген файлдар құрамына қатынас құру

Мұндай файлды құрғанда, ол процесс виртуалды жадының мекендік кеңістігінің кейбір аумағымен байла-

нысады. Бұл байланысты орнату үшін файл бейнеленетін объект құру қажет (*file – mapping object*). Файл мазмұнын қарау және түзету үшін (*file view*) қарау терезесі болу керек; бұл арқылы нұсқаушының көмегімен файл мазмұнына, сонымен қатар жадының кейбір аумақтарына қатынас құруға мүмкіндік алуға болады.

Бұл көру терезесіне жазу кезінде жүйе файл мәліметтерін КЭШ-теу, буферлеу және жүктеу әрекеттерін орындайды, сонымен қатар жадыны бөлу мен босатуды жүзеге асырады. Ал сізге тек жадының кейбір аумағында орналасқан мәліметтерді өзгерту ғана қалады (мұндағы файлдық енгізу-шығару операциялары толығымен жүйеге жүктелген). Жадыға бейнеленген файлдармен жұмыс жасаудың қолайлылығы осында жатыр, себебі бұрын қарастырған тәсілдермен салыстырғанда файлдарды басқару міндеті елеулі жеңілдетіледі. Сонымен қатар бұл операциялардың орындалу жылдамдығы жоғары болмақ.

Келесі бөлімдерде жадыға бейнеленген файлдарды құру және ашуға қажетті әрекеттер тізбегі келтірілген.

Файлды құру немесе ашу

Жадыға бейнеленген файлды құру немесе ашу үшін алдымен бейнелеуге жататын файл дескрипторын алу керек. Бұл үшін келесі функциялардың біреуін қолданамыз: *FileCreate( )* немесе *FileOpen( ). FileCreate( )* функциясы **SysUtils.pas** модулінде анықталған, ол былай беріледі:

```
function FileCreate (const FileName: string): integer;
```

Бұл функция *FileName* атты файл құрады. *FileOpen( )* функциясы берілген режимді пайдаланып, бұрыннан бар файлды ашады. Функцияның дұрыс орындалуының нәтижесінде файл дескрипторының нақты мәні қайтарылады. Қарсы жағдайда Invalid_Handle_Value тұрақтысымен анықталған мән қайтарылады.

FileOpen( ) функциясы **SysUtils.pas** модулінде анықталған, оның түрі мынадай:

Function FileOpen (const FileName: **string**; Mode: Word): integer;

Бірінші параметр жадыға бейнелейтін файлдың толық жолы. Ал екінші параметр ретінде келесі кестеде келтірілген қатынау режимдерінің біреуін қолдануға болады:

26-кесте

Файлға қатынау режимдері (fmOpenXXXX)

Қатынау режимі	Мәні
fmOpenRead	Тек файлдан оқуды орындауға мүмкіндік береді
fmOpenWrite	Тек файлға жазуды орындауды мүмкіндік береді
fmOpenReadWrite	Оқу, жазуды орындауға мүмкіндік береді.

Erep Mode параметрі ретінде 0 алсақ, онда сіз мәліметтерді оқи да алмайсыз, оларды файлға жаза да алмайсыз. Нөлдік мән файлды әр түрлі атрибуттарын (төлсипат) алу үшін қолданылады. Берілген файлдың әр түрлі қосымшаларымен бірге пайдалануды OR (немесе) операциясымен анықтауға болады. Ол үшін 26-кестедегі режимдерді 27-кестедегі режимдердің біреуімен қатар қолдану қажет.

27-кесте

fmShareXXXX қатынау режимдері

Бөлу режимі	Мәні
1	2
fmShareCompat	Файлды бөлу механизмі Dos 1.x және 2.x басқару блоктарымен үйлесімді. Бұл мән fmShareXXXX басқа режимдерімен бірге қолданылады.
fmShareExciusive	Бөлу рұқсат етілмейді

1	2
fmShareDenyWrite	FmOpenWrite режимінен басқа бағдарламалаудың файлды ашу әрекеті сәтсіз болмақ.
fmShareDenyRead	FmOpenRead режимімен басқа бағдарламалардың файлды ашу әрекеті сәтсіз болмақ.
fmShareDenyNone	Басқа бағдарламалардың кез келген қатынаулы файлды ашу әрекеті сәтті болмақ.

Файлдың нақты дескрипторын қайтарып, жадыға бейнеленетін объектіні алуға болады.

Жадыға бейнелеу объектісін құру

Файлды бейнелеудің аталған немесе аталмаған объектісін құру үшін *CreateFileMapping( )* функциясын қолданыңыз, ол функцияның анықталуы:

function *CreateFileMapping (hFile: THandle;*
IpFileMappingAttributes: PSecurityAttributes;
flProtect, dwMaximumSizeHigh, dwMaximumSizeLow: DWord;
IpName: PChar): THandle;

CreateFileMapping функциясы жүйеге файлдың бейнеленетін объектісін құру керектігі жөнінде ақпарат береді. *hFile* бірінші параметр *FileOpen( )* және *FileCreate( )* функцияларына алдыңғы қатынас құрудан алынған файл дескрипторы болып табылады. Бұл файл *flprotect* параметрімен үйлесімді қорғаныш белгілерімен ашылады. *IpFileMappingAttributes* параметрі файлды бейнелеу объектісі үшін қорғау атрибутына сілтеме жасайтын *PSecurityAttributes* типті нұсқауыш болып табылады. Бұл параметр әдетте әрқашан нөлге тең.

flProtect параметрі файлды қарау терезесіне қолданылатын қорғаныш типін анықтайды. Оның мәні дескрипторын алу үшін ашқан файл атрибуттарымен үйлесімді болуы керек. 28 кестеде *flProtect* параметріне меншіктелетін түрлі атрибуттар келтірілген.

28-кесте

Қорғаныс атрибуттары

Қорғаныш атрибуттары	Мәні
PAGE_READONLY	Бұл файлдың мәліметтерін оқуға болады. *FileCreate( )* функциясы көмегімен құрылу керек немесе *FileOpen( )* функциясын шақыру арқылы ашылуы керек, қатынас құру режимі *fmOpenRead* болуы қажет.
PAGE_READWRITE	Бұл файл үшін оқу да, жазу да рұқсат етілген. Файл *fmOpenReadWrite* қатынау режимінде ашылуы керек.
PAGE_WRITECOPY	Бұл файл үшін оқуға да, жазуға да рұқсат етілген, бірақ жазу кезінде файлда жаңа беттің көшірмесі құрылады. Файл *fmOpenWrite* және *fmOpenReadWrite* қатынау режимдерінде ашылу керек.

flProtect параметрінде разряд бойынша OR логикалық операторын қамтитын бөлім атрибуттарын қолдануға болады. Атрибуттардың мәндері 29-кестеде келтірілген.

29-кесте

Бөлім атрибуттары

Бөлім атрибуттары	Мәні
SEC_COMMIT	Бөлімнің барлық беті үшін жүйелік жады аумағында физикалық орын бөліп береді. Бұл мән бастапқы түрде (по умолчанию) белгіленеді.
SEC_IMAGE	Файлды бейнелеу мен атрибуттары туралы файл бейнесінен алынады. Бұл атрибут тек суреттердің орындамалық файлдарына қолданылады. (Ескерту: бұл атрибут Windows 95 жүйесінде есепке алынбайды).
SEC_NOCACHE	Жадыға бейнеленген беттер бүркемеленбейді, сондықтан файлға жазылатын мәліметтердің барлығы файлдың дискідегі мәліметтеріне қолданылады (Ескерту: бұл атрибут Windows 95 жүйесінде есепке алынбайды).
SEC_RESERVE	Бөлім беттерін физикалық орын бөлусіз резервтейді.

dwMaximumSizeHigh параметрі файлды бейнелеу объектісінің максимал өлшемінің үлкен 32 разрядын береді. Егер сізге өлшемі 4Гбайттан жоғары файлдарға қатынас құру керек болмаса, бұл мән әр уақытта нөлге тең болады.

dwMaximumSizeLow параметрі файлды көрсету объектісінің максимал өлшемінің кіші 32 разрядын береді. Бұл параметрдің нөлдік мәні – файлды көрсету объектісінің максимал өлшемі бейнеленетін файлдың өлшеміне тең екендігін көрсетеді.

IpName параметрі файлды бейнелеу объектісінің атын анықтайды. Бұл мән (\) символынан басқа кез келген символға ие бола алады. Егер бұл параметрдің мәні бұрыннан бар файлды бейнелеу объектісіне сәйкес болса, онда бұл функция *flProtect* параметрімен берілген атрибуттар арқылы сол бейнелеу объектісіне қатынас құруды қажет етеді. *IpName* параметрінің мәні ретінде *nil* мәнін беруге болады, бұл аталмаған файлды бейнелеу объектісін құруға әкеледі.

CreateFileMapping( ) функциясы сәтті орындалған жағдайда файлды бейнелеу объектісімен байланысты дескриптор қайтарылады. Егер *CreateFileMapping( )* функциясы керек объектіні құра алмаса, ол *nil* мәнін қайтарады. Сәтсіздікті табу үшін *GetLastError( )* функциясын шақыру керек. Егер құрылатын бейнелеу объектісі бұрыннан бар ұқсас типті объектіге нұсқаса, онда *GetLastError( )* функциясы ERROR_ALREADY_EXISTS мәнін қайтарады.

Бейнелеулерді құру үшін қолданылатын файлдар дескрипторлары негізінде енгізу-шығару операцияларын орындайтын функциялар Windows 95 жүйесінде қолданылмайды. Файлды бейнелеу объектісін алып, файл мәліметтерін процестің мекендік кеңістігінде бейнелеуге болады.

Файл деректерін процестің мекендік кеңістігіне бейнелеу

Файл мәліметтерін процестің мекендік кеңістігіне бейнелеу үшін *MapViewOfFile( )* функциясы қолданылады, ол былайша анықталады:

function MapViewOfFile(
hFileMappingObject: THandie;
dwDesiredAccess: DWord;
dwFileOffsetHigh,
dwFileOffsetLow,
dwNumberOfBytesToMap: DWord): Pointer;

hFileMappingObject параметрі *CreateFileMapping( )* немесе *OpenFileMapping( )* функцияларының көмегімен құрылған, бейнелеудің ашық объектісімен байланысты дескриптор.

dwDesiredAccess параметрі файл мәліметтеріне қатынас құру тәсілдерін көрсетеді де, 30-кестеде көрсетілген мәндердің біреуін қабылдайды.

30-кесте

Файл мәліметтеріне қатынас құру

Параметр мәні	DwDesiredAccess сипаттамасы
FILE_MAP_WRITE	Файл мәліметтерін оқып, жазуға мүмкіндік береді. Ол үшін CreateFileMapping() функциясында Page_Read_Write атрибуты қолданылуы керек.
FILE_MAP_READ	Файл мәліметтерін оқып, жазуға мүмкіндік береді. Ол үшін CreateFileMapping() функциясында Page_Read_Write немесе Page_Read атрибуттары қолданылуы керек.
FILE_MAP_ALL_ACCESS	FileMapWrite процедурасын қолданғанда берілетін қатынас құруды қамтамасыз етеді.
FILE_MAP_COPY	Көшіруді қамти отырып жазуға қол жеткізу мүмкіндігін береді. Файлға жазу кезінде жазылатын беттің жабық (private) көшірмесі құрылады. Бұл жағдайда CreateFileMapping() функциясы Page_Read_Only, Page_Read_Write және Page_Read_ Copy атрибуттарымен қолданылуы керек.

dwFileOffsetHigh параметрі файлдың ығысуының үлкен 32 разрядын береді, сол жерден оның бейнеленуі басталады.

dwFileOffsetLow параметрі файлды ығыстырудың кіші 32 разрядын береді, сол жерден оның бейнеленуі басталады.

dwNumberOfBytesToMap параметрі бейнелеуге арналған файлдың байт санын анықтайды. Нөлдік мән бүтін файлдың бейнеленуін көрсетеді.

MapViewOfFile( ) функциясы сәтті орындалғанда бейнелеудің бастапқы орнын қайтарады, ал қарсы жағдайда *nil* мәнін қайтарады, сонда қате себептерін анықтау үшін *GetLastError( )* функциясын шақыру қажет.

Файлды қарау терезесін босату

UnmapViewOfFile( ) функциясының көмегімен шақырылатын процесс аймағында файлдың бейнеленуін жоюға болады. Бұл функция былайша анықталады.

```
function UnmapViewOfFile(lpBaseAddress: Pointer): BOOL;
```

Бұл функцияның lpBaseAddress жалғыз параметрі бейнеленетін аймақтың базалық мекеніне сілтейді, ол *MapViewOfFile( )* функциясымен қайтарылатын мәнге сәйкес келеді.

Файлмен жұмысты аяқтағаннан кейін *UnmapViewOfFile( )* функциясын міндетті түрде шақыру қажет, әйтпесе жадының бейнеленетін аймағын сіздің процесіңіз аяқталмайынша жүйе босатпайды.

Бейнелеу объектісін жабу

FileOpen( ), CreateFileMapping( ) функцияларына қатынас құру операциялық жүйе ядросының объектілерінің ашылуымен байланысты, сондықтан да олардың жабылуына сіз жауаптысыз. Мұны *CloseHandle( )* функциясының көмегімен орындауға болады. Ол төмендегі түрде анықталады:

```
function CloseHandle(hObject: THandle): bool;
```

CloseHandle( ) функциясы сәтті аяқталғанда функция **True** мәнін, ал қарсы жағдайда **False** мәнін қайтарады, онда қатенің себептерін анықтау үшін *GetLastError( )* функциясының нәтижесін тексеру қажет.

4.4. Көпағынды қосымшалар құру

Win32 операциялық жүйесінде қосымшаларды бірнеше ағынды орындау мүмкіндігі қарастырылған. Win32 жүйесінің 16 разрядты Windows және Dos жүйелерінен артықшылығы осында, бүгінде бір қосымшада түрлі өңдеу типтерін бір мезгілде қатар жүргізуге болатын жабдық пайда болды.

Ағын туралы түсінік

Ішкі процесс, **ағын (поток – thread)** – жеке процестің ішінде бағдарламаның орындалу жолын анықтайтын операциялық жүйенің объектісі. Win32 жүйесінің әрбір қосымшасы кемінде бір стандартты немесе негізгі деп аталатын ағынды қамтиды, бірақ қосымшалар басқа мақсаттардағы қосымшалар ағынын құра алады. Ағын көмегімен жекелеген ішкі бағдарламалардың бір уақытта орындалуын қамтамасыз ететін жабдықтар іске қосылады. Әрине, компьютер бір ғана процессормен жабдықталған болса, онда мұндай әрекет орындалмайды, бірақ әрбір ағынды өңдеу үшін жүйе кезек-кезек белгілі бір уақыт бөліп отырса (секундтың ұсақ бөліктерінде), онда бірнеше қосымша бір уақытта жұмысқа қосылған тәріздес әсер тууы мүмкін.

Ағындардың практикалық қолданылуы

Ағындар – Windows бағдарламашылары үшін өте қуатты жабдық болып табылады. Әдетте ағындар электронды кестелерде сақталатын мәліметтер бойынша есептеулер жүргізуге немесе мәтіндік процессор құжат-

тарын баспаға шығарғанда қолданылады. Сондағы бағдарлама құрушының мақсаты – процестердің керекті өңделуін және қолданушы интерфейстің жұмысы үшін тиімді уақытты қамтамасыз ету. Қолданушы интерфейспен байланысқан визуалды компоненттер кітапханасының (VCL) бөлігі мынаған негізделген, кез келген уақытта оған тек бір ағын ғана қатынас құра алады, сондықтан қолданушы интерфейсін бірнеше ағынмен байланыстырып басқару мүмкін емес. VCL мынаны талап етеді, қолданушы интерфейспен басқарудың барлық түрі қосымшаның негізгі ағынының контекстінде өтуі керек.

TThread объектісі

Delphi ортасы Object Pascal объектісіндегі *TThread* деп аталатын API ағындық объектісін инкапсуляциялайды. Бұл объект TObject объектісінен тікелей туындайды, сондықтан да ол компонент болып табылмайды. *TThreadExecute( )* тәсілі абстрактылы. Ол дегеніміз – *TThread* класының өзі абстрактылы дегенді білдіреді. Және сіз *TThread* класының объектісін құра алмайсыз. Сізге тек *TThread* класынан туындайтын экземплярын құруға болады. *TThread* класынан туындайтын экземплярды құрудың ең оңай жолы **New Items** сұхбаттық терезесінде *TThread Object* элементін таңдау. **New Items** терезесі **Filea New** командасымен ашылады. *TThread Object* элементін таңдағаннан кейін *New TThread Object* сұхбаттық терезесі ашылады. Бұл терезеде жаңа объект атын енгізу керек. Содан кейін Delphi сіздің объектіні қамтитын модуль құрады. *TThread* класының функционалдық туындысын құру үшін *Execute( )* жалғыз тәсілін қайта анықтау керек.

```
procedure TMyTThread. Execute;

var
i: dWord;
s: string;
```

```
begin
 for i:=1 to 1000 000 do
 begin
 s:= IntToStr (Round (Random(1000) * exp (Sqrt (i) ) ) );
 Form1. Label1.Caption:= s;
end;
end;
```

Ағын орындалуға *Create( )* конструкторын шақыру арқылы жіберіледі. **TThread** класының *Create( )* конструкторына берілетін бір ғана логикалық параметр ***CreateSuspended*** деп аталады. Ол ағынды тоқтатылған жағдайда басталу керек пе, жоқ па, соны көрсетеді. Егер оның мәні **False** болса, бұл объектінің *Execute( )* тәсілі автоматты түрде шақырылады. Егер оның мәні **True** болса, онда ағынның нақты жұмысының қосымшаның белгілі нүктесінде басталуына *Resume( )* тәсілін қолдану керек, ол өз кезегінде *Execute( )* тәсілін шақыратын болады.

```
procedure TForm1.Button1Click(Sender: TObject);
var
NewTThread: TMy Thread;
begin
NewTThread:= TmyTThread.Create(False);
end;
```

Ағындық объектінің қосымша қасиеттерін орнату үшін *CreateSuspended* мәні **True** етіп алынады.

Ағын жұмысын аяқтау

Егер *Execute( )* тәсілін орындау аяқталса, TThread ағынының объектісі аяқталған болып саналады. Бұл жағдайда Delphi-дің *EndTThread ( )* стандартты процедурасы шақырылады, ол өз кезегінде Win32-нің *API ExitTThread* функциясын шақырады. Бұл функция міндетті түрде ағын стегін және API ағындық объектісін

босатады. TThread объектісін қолданып болғаннан кейін сәйкес Object Pascal объектісін міндетті түрде жою керек. TThread объектісін босатудың ең қарапайым жолы, оның *FreeOnTerminate* қасиетінің мәнін **True** деп белгілеу. Мұны *Execute( )* тәсілін орындау аяқталғанға дейін жасауға болады. TThread объектісі ағын жұмысының аяқталуында жүзеге асырылатын *OnTerminate* оқиғасын қамтиды. Ағынның орындалуын мәжбүрлі түрде аяқтау үшін *Terminate* тәсілі қолданылады. Ол өз мәнін **True** деп алғанда ағын жұмысының тоқтатылуы керектігін білдіреді. *Execute( )* алдын ала шығу қажеттігін анықтау үшін *Terminated* қасиетінің жағдайын тексеруді өз мойнына алатынын есте сақтау керек.

Synchronize( ) тәсілі

TThread класында *Synchronize( )* тәсілі анықталған. Ол кейбір кластарға қосымшаның негізгі ағындарынан орындалуға мүмкіндік береді. *Synchronize( )* тәсілін анықтау:

```
procedure Synchronize (Method: TThreadMethod);
```

Method параметрінің типі *TThreadMethod* (ешқандай параметрлерге ие болмайтын процедуралық тәсілді белгілейді), ол былайша анықталады:

```
Type TThread Method = procedure of Object;
```

Method параметрі ретінде берілетін тәсіл қосымшаның негізгі ағынынан орындалатын тәсіл болып табылады.

Приоритеттер және кесте құру

Операциялық жүйе арқылы жеке ағынға берілген уақыт шамасы, оған белгіленген приоритетке байланысты. Бөлек ағынның жалпы приоритеті процесс приоритеті мен ағын приоритетінің бірігуімен анықталады.

Win32 процесінің приоритет класы 4 класты қолдайды: *Idle, Normal, High* және *Realtime*. Олар жүйеде жүктелген бөлек процестің приоритетін суреттейді. Кез келген приоритетке бастапқы түрде *Normal* класы меншіктеледі. Белгілі бір приоритетті процесті тудыру үшін бұл белгілердің кез келгенін *CreateProcess( )* функциясының *dwCreationFlag* параметрімен логикалық түрде біріктіру (OR операциясы көмегімен) керек. Бұл белгілерді берілген процестің приоритет класын динамикалық түрде жұмысқа икемдеу үшін қолдануға болады. Барлық кластарды 4-тен 24-ке дейінгі (24 саны қамтылады) сандар мәндерімен өрнектелетін приоритет деңгейімен көрсетуге болады. Ағынның жалпы приоритетін анықтаудағы екінші құраушы бөлік – жеке ағынның салыстырмалы приоритеті. Приоритетті класс процеспен, ал салыстырмалы приоритет процесс ішіндегі бөлек ағындармен байланысты екенін айта кету керек. Ағынға 7 мүмкін болатын приоритеттердің біреуін меншіктеуге болады: *Idle* (күту), *Lowest* (ең төмен), *BelowNormal* (қалыптыдан төмен), *Normal* (қалыпты), *Above Normal* (қалыптыдан жоғары), *Highest* (ең жоғарғы) немесе *TimeCritical* (уақыт бойынша критикалық).

Ағындарды синхрондау

Бірнеше ағынмен жұмыс істеген кезде ағындардың белгілі мәліметтер мен ресурстарға қатынауын синхрондау керек. Мысалы, бір қосымшада бір ағын файлды жадыға оқу үшін белгіленсе, ал екіншісі – файлдағы символ санын санау үшін арналған, әрине файл толығымен жадыға жүктелмейінше сіз оның символ санын анықтай алмайсыз. Бірақ әрбір операция өз ағынында орындалатындықтан, операциялық жүйе оларды мүлдем өзара байланыспаған деп қарастыра алады. Бұл қиындықты шешу үшін, файлды жадыға жүктейтін ағын аяқталмай, символ санау ағыны өз жұмысын бастамайтындай етіп бұл екі ағынды синхрондау керек. Win32-де синхрондаудың түрлі тәсілдері қарастырылған, олар:

критикалық бөлімдер (critical section), мьютекстер (mutexes), семафорлар (semaphores) және оқиғалар (events).

Критикалық бөлімдер

Бұл ағындарды синхрондаудың ең оңай тәсілі. Критикалық бөлім – бір ағынмен бір мезгілде орындала алатын код бөлімі. Егер қандай да бір болмасын бір деректерді инициализациялайтын кодты критикалық бөлімге орналастырсақ, онда басқа ағындар бірінші ағын орындалуын аяқтамайынша және мәліметтер дұрыс инициализацияланбайынша бұл код бөліміне кіре алмайды. Критикалық бөлімді қолдануға дейін оны Win32 жүйесінің *InitializeCriticalSection( )* процедурасымен инициализациялау керек, ол процедура былай анықталады:

```
procedure InitializeCriticalSection (var
LpCriticalSection: TRTLCriticalSection);
```

LpCriticalSection параметрі сілтеме бойынша берілетін TRTLCriticalSection типті жазба. Инициализацияланбаған жазбаны LpCriticalSection параметріне беру ғана керек, бұл жазба сол сәтте процедурамен толтырылады.

Жазбаны толтырғаннан кейін кейбір код блоктарын *EnterCriticalSection( )* және *LeaveCriticalSection( )* функциялары шақырылуының арасына қою арқылы қосымшада критикалық бөлім құруға болады. Бұл процедураларға InitializeCriticalSection() процедурасымен бұрынырақ толтырылған TRTLCriticalSection жазбасы параметр ретінде беріледі. TRTLCriticalSection жазбасымен жұмыс аяқталғаннан кейін DeleteCriticalSection() процедурасын шақыру арқылы жадыны босату қажет.

Мьютекстер

Өз әрекеті бойынша мьютекстер (MUTual Exclusions) критикалық бөлімдерге ұқсас, бірақ екі айырмашылығы

бар. Біріншіден, мьютекстерді процесс шекараларынан өтіп ағындарды синхрондауға қолдануға болады. Екіншіден: мьютекстерге ат беріп, осы ат бойынша қосымша дескрипторлар құруға болады. Мьютекстер құруға арналған функция CreateMutex() деп аталады, ол былайша баяндалады:

function Create Mutex (lpMutexAttributes: PsecurityAttributes; bInitialOwner: BOOL; lpName: pChar): Thandle;

lpMutexAttributes параметрі **PsecurityAttributes** жазбасына сілтеме. Әдетте бұл параметрдің мәні **nil** деп алынады, ал бұл жағдайда жасырын әрекет ететін қорғаныс атрибуттары қолданылады. bInitialOwner параметрі мьютексті құратын ағынды сол мьютекстің иемденушісі ретінде санауға болады ма, жоқ болмайды ма, соны анықтайды, егер осы параметр мәні False болса, мьютекс иесі болмайды. lpName параметрі мьютекс атын білдіреді. Мьютекспен жұмыс істеп болған соң оны *API CloseHandle( )* функциясы көмегімен жабу керек.

WaitForSingleObject функциясы

Hhandle функциясының параметрімен берілген *API* объектісіне қатынас құру мүмкіндігі берілгенге дейін, ағымдағы ағынды күту күйіне ауыстыруға арналған. Бұл кезде күту күйі *dwMilliseconds* параметрімен берілген миллисекундтағы уақыт интервалы біткенше созыла алады. Мьютекс ешқандай ағынға бағынбаса да қатынас құруға боларлық деп саналады. Уақыттың нақты жиілігімен қатар, *DwMilliseconds* параметрі нөлдік мәнге де ие бола алады, бұл мән объект жағдайын тексеруді және дереу қайтаруды білдіреді. Бұл функция қайтара алатын мәндер: **WAIT_ABANDONED** – берілген объект мьютекс объектісі болып табылады, бірақ бұл мьютекстің иемденуші-ағыны ол босатылғанға дейін аяқталады. Мұндай мьютекс беті қайтарылған деп саналады

(abandoned mutex), мьютекске ие болу құқығы шақырушы ағынға беріледі, ал мьютекс қатынас құру мүмкіндігінен айырылады.

WAIT_OBJECT_0 – берілген объектінің жағдайы қатынас құруға болады деп қарастырылады.

WAIT_TIMEOUT – берілген уақыт интервалы өтті, бірақ объект жағдайы қатынас құруға болмайды деп саналады.

Егер мьютекс ағынға жатпаса ол қатынас құруға мүмкіндік береді.

WaitForSingleObject функциясын шақыратын бірінші ағынға иемденуші болу құқығы беріледі, ал мьютекс объектісінің жағдайы қол жетерлік емес болып табылады. Ағын параметр ретінде мьютекс дескрипторын бере отырып *ReleaseMutex( )* функциясын шақырғанда иелік құқығы алынып тасталынады, ал мьютекс қайтадан қол жетерлік болып табылады.

Семафорлар

Ағындарды синхрондаудың тағы бір жолы API семафорлық объектілерін қолдану. Семафорларда мьютекстер жұмысының принципі қолданылған, бірақ оған маңызды бір бөлім қосылды. Оларда синхрондау кодының бөлімдеріне алдын-ала анықталған ағындар саны кіру үшін ресурстарды есептеу мүмкіндігі қарастырылған. Семафорды құру үшін *CreateSemaphore( )* функциясы қолданылады, ол төмендегіше баяндалады:

```
function CreateSemaphore (lpSemaphoreAttributes:
PSecurity Attributes;
lInitialCount, lMaximumCount: Longint; lpName:
PChar): THandle;
```

CreateSemaphore( ) функциясымен берілетін бірінші параметр lpSemaphoreAttributes жазбасына сілтеме болып табылады, мұндағы nil мәні стандартты қорғаныш атрибуттарын қолдану келісіміне сәйкес келеді. lInitial-

Count параметрі семафорлық объект санағышының бастапқы мәнін білдіреді. Бұл мән 0-ден 1MaximumCount-қа дейінгі диапазонда бола алады. Семафор параметрдің мәні 0-ден үлкен болғанша қатынас құру мүмкіндігіне ие болады.

WaitForSingleObject( ) немесе басқа күту функциялары ағынды босатқанда, семафор санағышының мәні азайтылады. Керісінше ReleaseSemaphore() функциясын шақырғанда санағыш мәні артады.

LMaximumCount параметрі арқылы семафорлық объекті санағышының максимал мәні беріледі. Егер семафор кейбір ресурстарды санау үшін қолданылса, бұл мән рұқсат етілген ресурстардың жалпы санына тең болу керек. LpName параметрі семафор атын сақтайды. *ReleaseSemaphore( )* функциясы семафор санағышының мәнін өсіру үшін қолданылады. Бұл функция ReleaseMutex()-не қарағанда көбірек параметрлерді қамтиды. ReleaseSemaphore() функциясының баяндалуы:

```
function ReleaseSemaphore (hSemaphore: THandle;
lReleaseCount: Longint; lpPreviousCount: Pointer) : BOOL;
```

LReleaseCount параметрі көмегімен семафор санағышының мәні қандай санға кішірейтілетіні көрсетіледі. Семафор дескрипторын босату үшін *CloseHandle( )* функциясын қолданады.

Оқиғалар

Ағындарды синхрондаудың тағы бір жолы оқиғалардың (event) API объектісін қолдану. Оқиға екі жағдайда – рұқсат етілген және рұқсат етілмеген бола алатын объект. Бір ағын оқиғаны қатынас құруға мүмкіндік жоқ жағдайға қойып, кез келген есептеулерді біткен соң оқиғаға қатынас құру жағдайын жасайды. Кез келген басқа ағын бірінші ағын есептеулерінің нәтижесінде өз әрекеттерін *WaitForSingleObject( )* функциясы арқылы синхрондай алады, осылайша оған параметр ретінде

бірінші ағынмен анықталған объект-оқиғаны береді. Сондықтан бірінші ағын өз есептеулерін бітіріп объект-оқиғаны қатынаулы жағдайға келтіргенде, екінші ағын ол туралы хабарландырылып, өз жұмысын жалғастырады.

Оқиға ***API CreateEvent()*** функциясы арқылы құрылады:

Function *CreateEvent (lpSemaphoreAttributes: PSecurityAttributes; bManualReset, bInitialState: BOOL; lpName: PChar) : THandle;*

LpSemaphoreAttributes және *lpName* параметрлері *CreateSemaphore( )* және *CreateMutex( )* функцияларындағы параметрлерге өте ұқсас. Егер *bInitialState* параметрі **True** болса, онда құрылғаннан кейін оқиғаға қол жеткізу мүмкіндігі бар, керісінше жағдайда оқиғаға қол жеткізу мүмкіндігі жоқ. *BManual Reset* параметрі оқиға *WaitForSingleObject( )* функциясының сәтті аяқталуынан кейін оқиғаның автоматты түрде қол жеткізуге мүмкін емес жағдайға келтірілетінін анықтайды. Егер ол True болса, онда оқиға рұқсат етілмеген жағдайға қолмен көшірілуі керек.

SetEvent( ) және *ResetEvent( )* функциялары оқиғаларға қол жеткізу немесе қол жеткізу мүмкін емес жағдайға сәйкесінше келтіреді. Объект-оқиғаны жою үшін *API CloseHandle( )* функциясын қолданады.

Бақылау сұрақтары:

1. Динамикалық массив дегенді қалай түсінесіз?
2. **math** модулінде анықталған функцияларға түсінік беріңіз.
3. Жадыда бейнеленген файлдар дегенді қалай түсінесіз?
4. Виртуалды жады деген не? Оның аумағына қатынас құру қалай жүзеге асырылады?
5. Қорғаныс атрибуттарын атаңыз.
6. Файлмен жұмысты бастау және аяқтау функцияларының форматын көрсетіңіз.

7. Көпағынды қосымша деген не?

8. Ағынды орындауға жіберу конструкторының форматын көрсетіңіз.

9. Ағындарды синхрондау дегенді қалай түсіндіруге болады?

10. Мьютекстер мен Семафорларға, Оқиғаларға түсінік беріңіз.

ӨЗІНДІК ЖҰМЫСҚА АРНАЛҒАН ТАПСЫРМАЛАР

Тақырып: «Динамикалық массивтер»

1) Төмендегі суретте көрсетілгендей, екі өлшемді массив құру. Параметрлер ретінде үшбұрыштар саны және үшбұрыштағы қатарлар саны беріледі.

2) Төмендегі суреттегідей екі өлшемді массив қалыптастыру керек. Параметрлер ретінде үшбұрыштар саны және үшбұрыштағы бағандар саны алынады.

3) Үш өлшемді массив құру. Параметрлер ретіне «сатылар» саны және «бір сатының» ені беріледі. Деректерді суретте көрсетілген түрде шығаруға болады.

4) 2 нұсқадағы массивтен үш өлшемді матрица құру. Параметр ретінде үшбұрыштар саны, үшбұрыштардағы бағана саны және үшбұрыштардағы жол ені берілген.

5) n-ші жолдағы бағана саны мен жол ені жол нөміріне тең 3 өлшемді массив құру, басқа сөзбен айтқанда пирамиданың бір ширегіне ұқсас.

6) Пирамида тәріздес үш өлшемді массив құрыңыз.

Тақырып «Жадыға бейнеленетін файлдармен жұмыс»

1) Файлдың барлық бас әріптерін кіші әріптерге ауыстыру.

2) Файлдағы белгілі бір символдың қайталануын санау.

3) Файлдағы белгілі бір символды басқа символмен ауыстыру.

4) Файлдағы белгілі бір символдар тізбегін іздеу.

5) Файлдағы белгілі бір символдар тізбегін басқа тізбекке ауыстыру.

6) Жадыда бейнеленген файлдың құрамын басқа файлға көшіру.

Тақырып «Көпағынды қосымшалар құру»

1) Қосымша ағындарды қолданып, фондық режимде берілген диапазондағы жай сандарды есептейтін бағдарламаны жаз. Табылған мәндерді шығаруды есептеулермен бірге бағдарламаның негізгі формасында жүргіз.

2) Екі қосымша ағынды қолданып есептеулер жүргізетін бағдарламаны жаз. Екі ағынды синхрондауды критикалық бөлімдер механизмі арқылы жаса.

3) Екі қосымша ағынды қолданып есептеулер жүргізетін бағдарлама жаз. Екі ағынды синхрондауды Мьютекс механизмі арқылы жаса.

4) Екі қосымша ағынды қолданып есептеулер жүргізетін бағдарлама жаз. Екі ағынды синхрондауды семафор механизмі арқылы жаса.

5) Екі қосымша ағынды қолданып есептеулер жүргізетін бағдарлама жаз. Екі ағынды синхрондауды объект-оқиғалар механизмі арқылы жаса.

ӨТІЛГЕН МАТЕРИАЛДАР БОЙЫНША БІЛІМ ТИЯНАҚТАУҒА АРНАЛҒАН ТЕСТ СҰРАҚТАРЫ

1. Delphi 6 интегралданған өзірлеу ортасы көптерезелі жүйе және бастапқыда келесі терезелерді қамтиды:

A) Бас терезе, Объектілер бұтағы шолушысының терезесі, Объектілер инспекторы терезесі;
B) Бас терезе, Объектілер инспекторы терезесі, Форма терезесі, Код редакторының терезесі, Код жетекшісінің терезесі;
C) Бас терезе, Объектілер бұтағы шолушысының терезесі, Объектілер инспекторы терезесі, Форма терезесі, Код редакторының терезесі;

D) Бас терезе, Объектілер инспекторы терезесі, Форма терезесі, Код редакторының терезесі;

E) Бас терезе, Объектілер бұтағы шолушысының терезесі, Объектілер инспекторы терезесі, Форма терезесі, Код редакторының терезесі, Код жетекшісінің терезесі.

2. Қосымшаның қолданушы интерфейсі құрастырады:

A) Әзірлеушінің Компоненттер палитрасынан таңдап, формаға орналастыратын компоненттері;

B) Әзірлеушінің Редактор терезесінде жазатын бағдарламалық кодтарын;

C) Объектілердің барлық қасиеттері мен оқиғаларын бейнелеуді;

D) Форма модульдерінің барлық объектілерін бейнелеуді;

E) Әзірлеушінің тек Компоненттер палитрасынан таңдап, формаға орналастыратын визуалды компоненттерін.

3. Қатынас құрудың қандай модификатормен көрсетілген айнымалылары мен әдістері жалпы қатынас құруға ыңғайлы?

A) Protected
B) Published
C) Public
D) Private
E) Class.

4. Компоненттер оқиғаларының қасиеттері мен өңдеушілерін анықтайтын терезені таңдаңыз:

A) Object Inspector
B) Form1
C) Unit1.pas
D) Delphi7 – Project1
E) Exploring Unit1.pas

5. Delphi жобаның құрамына келесі элементтер кіреді:

A) Жоба коды (dpr), модульдер (pas), файлдардың резервті көшірмелері, жоба параметрлері (dof), орта параметрлері (cfg);

B) Формаға сипаттама (dfm), модульдер (pas), орта параметрлері (cfg);

C) Жоба коды (dpr), форма сипаттамалары (dfm), модульдер (pas);

D) Жоба коды (dpr), форма сипаттамалары (dfm), модульдер (pas); жоба параметрлері (dof), орта параметрлері (cfg), ресурстарға сипаттама (res);

E) Жоба коды (dpr), форма сипаттамалары (dfm), модульдер (pas), файлдардың резервті көшірмелері.

6. Бағдарлама модульдерін құрастыруға арналған терезені таңдаңыз:

A) Object Inspector
B) Form1
C) Unit1.pas
D) Delphi7 – Project1
E) Exploring Unit1.pas

7. Форманы жобалау кезіндегі компоненттер қасиеттері қайсысының көмегімен жұмысқа баптала ды:

A) Компоненттер палитрасы;
B) Объектілер инспекторы;
C) Код инспекторы;
D) Жобалар менеджері;
E) Формалар жобалағышы.

8. Батырма компоненті жататын класс типін таңдаңыз:

A) TEdit
B) TButton

C) TPanel
D) TLabel
E) TRadioButton

9. Компонент биіктігін анықтайтын қасиет атын таңдаңыз:

A) Caption
B) Width
C) Height
D) Top
E) Font

10. «Тінтуірді басу» оқиғасының атын таңдаңыз:

A) OnClick
B) OnMouseMove
C) OnMouseUp
D) OnMouseDown
E) OnKeyPress

11. Логикалық операцияларға қайсысы жататынын көрсетіңіз:
A) div
B) mod
C) not
D) =
E) >=

12. Бүтін типті мәнді қатарға түрлендіру үшін қандай функцияның қолданылатынын көрсетіңіз:

A) IntToStr
B) StrToFloat
C) FloatToStr
D) StrToReal
E) RealToStr

13. Жоба параметрлерін орнату үшін қолданылатын терезені көрсетіңіз:

A) Project Options
B) Object TreeView
C) Object Inspector
D) Exploring Unit1.pas
E) Form1

14. Басқарушы батырмаларға қай компоненттер жатады:

A) TBitBtn, TSpeedButton
B) TButton, TSpeedButton
C) TSpeedButton, TEdit
D) TButton, TBitBtn
E) TButton, Tedit

15. Класс ішінде қатынас құрудың модификаторымен байланысты айнымалыларды көрсетіңіз?

A) Protected
B) Published
C) Public
D) Private
E) class

16. Берілген кодтағы қателерді түзетіңіз: i1:=125; Edit1.Text:= FloatToStr (i1);

A) i1:=125; Edit1.Text:= IntToStr (i1);
B) i1:=125; Edit1.Text:= IntToStr (125);
C) i1:=125; Edit1.Text:=FloatToInt(125);
D) i1:=125; Edit1.Text:=StrToFloat(i1);
E) i1:=125; Edit1.Text:=StrToInt(i1);

17. Берілген кодтағы қателерді түзетіңіз:
r1:= -3.1E-4; Edit2.Text:= IntToStr(r1);

A) r1:= -3.1E-4; Edit2.Text:= StrToInt (r1);
B) r1:= -3.1E-4; Edit2.Text:= FloatToStr(r1);
C) r1:= -3.1E-4; Edit2.Text:= IntToStr(-3.1E-4);
D) r1:= -3.1E-4; Edit2.Text:= IntToStr(r1);
E) r1:= -3.1E-4; Edit2.Text:= StrToFloat (r1);

18. Ұсынылған нұсқалардан деректердің нақты типі үшін қолданылатын мәнді таңдаңыз:

A) 100
B) -3.1E-4
C) '11,05'
D) -375
E) '105'

19. Нақты типті мәнді қатарға түрлендіру үшін қандай функция қолданылады?

A) IntToStr
B) StrToFloat
C) FloatToStr
D) StrToReal
E) RealToStr

20. Edit1 компонентінің мәнін Button1 компонент қасиетінің мәніне меншіктеу процедурасының дұрыс нұсқасын таңдаңыз:

A) procedure TForm1Button1Click;
Begin
Button1.Caption:=Edit1.Text; end;
B) procedure TForm1Button1Click(Sender: TObject)
Begin
Button1.Text:=Edit1.Text; end;
C) procedure TForm1Button1Click(Sender: TObject)
Begin
Button1.Name:=Edit1.Text; end;
D) procedure TForm1Button1Click(Sender: TObject)

Begin
Button1.Caption:=Edit1.Text; end;
E) procedure TForm1Button1Click(Sender: TObject)
Begin
Button1.Caption=Edit1.Text;

21. Бір объектіде деректерді және оларды өңдеу тәсілдерін біріктіруді қалай атауға болады?

A) иелену
B) полиморфизм
C) инкапсуляция
D) виртуализация
E) әдіс

22. Бұрынырақ анықталғандардың негізінде жаңа кластар құруды және объект-туындылар өзінің түпкілерінің қасиеттерін иемденетінін қалай атауға болады?

A) иелену
B) полиморфизм
C) инкапсуляция
D) виртуализация
E) әдіс

23. Полиморфизм дегеніміз – бұл:

A) Объект-түпкіні объект-туындының аттас әдістерімен алмастыру мүмкіндігі;
B) Бұрынырақ анықталғандардың негізінде жаңа объектілер құру;
C) Объектілерді кеңейту мүмкіндігі;
D) Бір объектіде деректерді және оларды өңдеу әдістерін біріктіру;
E) Жаңа объектілер құру және бір объектіде деректер мен оларды өңдеу әдістерін біріктіру мүмкіндігі.

24. Жасырын өрістерді сипаттау секциясын және директива объектісі сипаттамасындағы әдістерді ашады:

A) Public
B) Private
C) Var
D) Protected
E) Class

25. Класта инкапсуляцияланған деректер мынадай атауға ие:

A) қасиет
B) әдіс
C) класс
D) өріс
E) объект

26. Компоненттер тізімінен визуалды еместерін көрсетіңіз:

1) TLabel
2) TPanel
3) TMainMenu
4) TEdit
5) TButton
6) TTimer
7) TPopupMenu
8) TMemo

A) 1, 2, 3, 4
B) 3, 7, 6
C) 3, 4, 6, 2
D) 3. 7, 6, 2
E) 1, 2, 3, 7

27. Меншіктеу операторының қайсысы дұрыс жазылған:

A) StrToFloat(Edit1.Text):=x;
B) X:= StrToFloat(Edit1.Text);

C) Edit1.Text:=StrToFloat(x);
D) StrToFloat(x):=Edit1.Text;
E) Label1.Caption=StrToInt(x);

28. Таңдау тізімі қай компонент болып табылады:

A) TPanel
B) TListBox
C) TMemo
D) TLabel
E) TComboBox

29. Жалпыға арналған контейнер дегеніміз – . . . компоненті.

A) TEdit
B) TLabel
C) TPanel
D) TMemo
E) TComboBox

30. Берілген кодтан дұрыс жазылғанын таңдаңыз:

A) if Edit1.Text=Edit2.Text
then Edit3.Text:='Иә'; else Edit3.Text:='Жоқ';
B) if Edit1.Text:=Edit2.Text
then begin Edit3.Text:='Иә' else Edit3.Text:='Жоқ';
C) if Edit1.Text=Edit2.Text
then Edit3.Text:='Иә' else Edit3.Text:='Жоқ';
D) if Edit1.Text:=Edit2.Text
then Edit3.Text:='Иә' else Edit3.Text='Жоқ';
E) if Edit1.Text:=Edit2.Text
then Edit3.Text='Иә' else Edit3.Text='Жоқ';

31. Таңдау операторының дұрыс жазылған нұсқасын таңдаңыз:

A) case Edit1.Text of
1 : Edit3.Text:='1';

2 : Edit3.Text:='10';
else Edit3.Text:='000000'; end;
B) case StrToInt(Edit1.Text) of
1 : Edit3.Text:='1';
2 : Edit3.Text:='10';
else Edit3.Text:='000000'; end;
C) case Edit1.Text of
1 : Edit3.Text:='1';
10 : Edit3.Text:='10';
else Edit3.Text:='00000'; end;
D) case FloatToInt(Edit1.Text) of
1 : Edit3.Text:='1';
10 : Edit3.Text:='10';
else Edit3.Text:='000000'; end;
E) case IntToStr (Edit1.Text) of
1 : Edit3.Text:='1';
2 : Edit3.Text:='10';
else Edit3.Text:='000000'; end;

32. StringGrid компоненті қай панельде орналасқан:

A) Additional
B) Dialogs
C) Standart
D) Samples
E) System

33. Таңдалған қатарға байланысты таңдауды жүзеге асыратын компонент қасиетін көрсетіңіз:
CaseListBox1.XXXXXXXXX of
0: y:=x/400;
1: y:=x/16380;
2: y:=x/28.35;
3: y:=x/28.35*16;
4: y:=437.5*x/28.35;
End;

A) Add
B) ItemIndex

C) Items
D) Lines
E) Text

34. Берілген тізімнен әдістер атауларын көрсетіңіз:
1) Add 2) Items 3) Lines 4) Clear 5) SetFocus 6) Text

A) 1, 4, 5
B) 1, 2, 5
C) 2, 3, 4
D) 3, 4, 5
E) 1, 2, 4, 5

35. Тәуелсіз ауыстырып-қосқыш дегеніміз қай компонент?

A) TGroupBox
B) TCheckBox
C) TMemo
D) TScrollBar
E) TRadioButton

36. TCheckBox компонентінің Иә/Жоқ шешімі қай қасиетте көрсетіледі?

A) Lines
B) State
C) Text
D) Caption
E) Name

37. Форманың басты менюі қай компоненттің көмегімен қалыптасады?

A) TSaveDialog
B) TPopupMenu
C) TMenu

D) TMainMenu
E) TOpenMenu

38. Файлды таңдаудың сұхбаттық терезесі қай компоненттің көмегімен шақырылады?

A) TSaveDialog
B) TMainMenu
C) TOpenDialog
D) TPictureDialog
E) TFontDialog

39. Басты меню пунктін құру үшін MenuDesigner шақырылады:

A) File/Open меню пунктін таңдау арқылы;
B) Объектілер инспекторында Caption қасиетін таңдау арқылы;
C) Компонентті тінтуірмен екі рет басу арқылы;
D) Объектілер инспекторында Items қасиетін таңдау арқылы;
E) Объектілер инспекторында Name қасиетін таңдау арқылы.

40. Мәтіндік файлдар қандай қызметтік сөздің көмегімен баяндалады?

A) File
B) TextFile
C) Text
D) Text80
E) File of Char

41. StringGrid кесте ұяшығының құрамындағыларды қандай қасиет көрсетеді?

A) Cols
B) ColCount

C) RowCount
D) Cells
E) Name

42. TStringGrid компоненті мыналардың қайсысын сақтауға арналған?

A) Кез келген типті объектілерді;
B) Мәтіндік қатарды;
C) Сандарды;
D) Сандар мен қатарларды;
E) Көріністерді.

43. StringGrid1 кестесінің нөлдік жол және екінші баған қиылысында орналасқан қатарға қатынас құру үшін қай нұсқаны таңдаған жөн?

A) StringGrid1. Cells[2,0];
B) StringGrid1. Cells[0,2];
C) StringGrid1. Cells(0,2);
D) Cells[0,2];
E) StringGridCells[2,0];

44. Форма құруға арналған әдіс:

A) Create
B) New
C) Show
D) Release
E) Hide

45. Форманы қосымшаға қосу үшін қай әрекет орындалу қажет?

A) Контекстік меню көмегімен формалар редакторын шақыру;
B) Project – Options меню пунктін таңдау;
C) File – NewForm меню пунктін таңдау;

D) Объектілер инспекторында жаңа форма таңдау;
E) Project – Add to Project меню пунктін таңдау.

46. Барлық файлдарды байланыстыратын (қосымша осылардан тұрады) файлды қалай атаймыз?

A) Бағдарламалық код
B) Формалар
C) Ресурстар
D) Жоба
E) Кітапхана

47. Инструменттер панелін құруға арналған арнайы компонент:

A) TPanel
B) TImageList
C) TToolBar
D) TStatusBar
E) TTImage

48. Жоба файлының кеңейтілуі:

A) Dof
B) Dfm
C) Pas
D) Res
E) dpr

49. Форма файлының кеңейтілуі:

A) Dof
B) Dfm
C) Pas
D) Res
E) Dpr

50. Бағдарламаның нәтижесі қандай?
procedure TForm1.Button1Click(Sender: TObject);
var a, b,c : integer;

```
begin
a:=StrToInt(Edit1.Text);
b:=StrToInt(Edit2.Text);
c:=StrToInt(Edit3.Text);
a:=a+b;
c:= a-b;
Label1.Caption:='a= "+IntToStr(a)+
"b= "+IntToStr(b)+ "c= "+IntToStr(c);
```

Егер енгізу үшін мына сандар берілсе: 12; 13; 14

A) a=13 b=14 c=12
B) a=25 b=13 c=14
C) a=25 b=13 c=12
D) a=12 b=13 c=14
E) a=25 b=14 c=12

51. Бағдарламаның нәтижесі қандай?

```
procedure TForm1.Button1Click(Sender: TObject);
var a, b,c : integer;
begin
a:=StrToInt(Edit1.Text);
b:=StrToInt(Edit2.Text);
c:=StrToInt(Edit3.Text);
Label1.Caption:='a= "+IntToStr(a+b)+
"b= "+IntToStr(b div 2)+ "c= "+IntToStr(c-10);
```

Егер енгізу үшін мына сандар берілсе: 5; 15; -2

A) a=5 b=15 c=-2
B) a=20 b=-7 c=-12
C) a=20 b=1 c=-12
D) a=20 b=7 c=-12
E) a=20 b=1 c=12

52. Бағдарламаның нәтижесі қандай?

```
procedure TForm1.Button1Click(Sender: TObject);
var a, b,c : integer;
begin
```

```
b:=a-5;
c:=b-a;
Label1.Caption:='a= "+IntToStr((a+b) div 4)+
"b= "+IntToStr(b)+ "c= "+IntToStr(c);
```

Егер енгізу үшін мына сандар берілсе: -10; 4; 6

A) a=-6 b=-15 c=-5
B) a=-10 b=4 c=6
C) a=1 b=-15 c=-5
D) a=6 b=4 c=-5
E) a=-6 b=-15 c=6

53. Бағдарламаның нәтижесі қандай?

```
procedure TForm1.Button1Click(Sender: TObject);
var a, b,c : integer;
begin
b:=a-5;
c:=b-a;
Label1.Caption:='a= "+IntToStr(a+b*2)+
"b= "+IntToStr(b)+ "c= "+IntToStr(c);
```

Егер енгізу үшін мына сандар берілсе: -10; 4; 6

A) a=-6 b=-15 c=-5
B) a=-12 b=-15 c=-5
C) a=12 b=-15 c=-5
D) a=-6 b=15 c=-5
E) a=-40 b=-15 c=-5

54. Бағдарламаның нәтижесі қандай?

```
procedure TForm1.Button1Click(Sender: TObject);
var a, b,c : real;
begin
a:=StrToFloat(Edit1.Text);
b:=StrToFloat(Edit2.Text);
c:=StrToFloat(Edit3.Text);
b:=a / 3;
c:=C-a;
Label1.Caption:='a= "+FloatToStr(a*2)+
```

"b= "+FloatToStr(b)+ "c= "+FloatToStr(c);
Егер енгізу үшін мына сандар берілсе: 0.5; 15.5; 12.5

A) a=0.5 b=15.5 c=12.5
B) a=1 b=1.17 c=12
C) a=0.5 b=0.6 c=12.5
D) a=1 b=1.17 c=12.5
E) a=1 b=0.6 c=12

55. Компонентке қатынас құруға рұқсат беру немесе тыйым салу үшін қандай қасиет қызмет етеді?

A) Enabled
B) Visible
C) Caption
D) Text
E) Name

56. Компоненттің экранда көріну мүмкіндігін басқару үшін қызмет ететін қасиетті көрсетіңіз:

A) Enabled
B) Visible
C) Caption
D) Text
E) Font

57. Бағдарламаның дұрыс нәтижесін таңдаңыз:

```
procedure TForm1.Button1Click(Sender: TObject);
var i,j,k,x:integer;
begin
 i:=7;j:=19;k:=0;
 if j>k then x:=20;
 if i>k then x:=10;
 Edit1.text:=IntToStr(x);
```

A) 20
B) 19

C) 10
D) 100
E) 7

58. Келесі бағдарламаның орындалу нәтижесі қандай?

```
procedure TForm1.Button1Click(Sender: TObject);
var a, s1,s2 : real; i:integer;
begin
 A:=1;
 S1:=0;
 S2:=0;
 FOR I:=1 TO 3 DO
 BEGIN
 S1:=S1-A*I;
 S2:=S2+A*I*I;
 A:=-A;
 Label1.Caption:='a= "+FloatToStr(a);
 Label2.Caption:='s1= "+FloatToStr(s1);
 Label3.Caption:='s2= "+FloatToStr(s2);
end; end;
```

A) a=-1 s1=-2 s2=6
B) a=-1 s1=0 s2=0
C) a=1 s1=-2 s2=6
D) a=1 s1=0 s2=6
E) a=-1 s1=-2 s2=0

59. Келесі бағдарламаның дұрыс нәтижесін таңдаңыз

```
procedure TForm1.Button1Click(Sender: TObject);
var i,j,k:integer;
begin
i := 4; j := 9;
repeat
i := i + j;
j := j -1;
until i>= j;
Label1.Caption:='i='+IntToStr(i);
```

```
Label2.Caption:='j='+IntToStr(j);
```

A) i=13 j=8
B) i=13 j=9
C) i=4 j=8
D) i=4 j=8
E) i=3 j=8

60. Келесі бағдарламаның дұрыс нәтижесін таңдаңыз:

```
procedure TForm1.Button1Click(Sender: TObject);
var a, b : integer;
begin
a := 19; b := 7;
while b <= a do begin
a := a - 1;
b := b+3;
Label1.Caption:='a= "+IntToStr(a);
Label2.Caption:='b= "+IntToStr(b);
```

A) a=19 b=7
B) a=16 b=19
C) a=16 b=7
D) a=18 b=10
E) a=16 b=10

61. A[7,7] массивінің тақ жолдарында орналасқан элементтердің ішінен минималды элементті табу үшін бағдарламаның дұрыс үзіндісін таңдаңыз:

A)
```
begin
min:=strToInt(StridGrid1.Cells[0,0]);
for i:=1 to 7 do begin
if (I mod 2)<>0 then continue
else begin
for j:=1 to 7 do
   if StrToInt(StridGrid1.Cells[j,i])<min then
min:=j+1) then begin
StrToInt(StridGrid1.Cells[j,i]); end; end; end;
```
B) begin

```
min:=strToInt(StridGrid1.Cells[0,0]);
for i:=0 to 6 do begin
if (I mod 2)<>0 then continue
else begin
for j:=0 to 6 do
        if StrToInt(StridGrid1.Cells[j,i])>min then
min:=StrToInt(StridGrid1.Cells[j,i]); end; end; end;
```

C)
```
begin
min:=strToInt(StridGrid1.Cells[0,0]);
for i:=0 to 6 do begin
if (I mod 2)<>0 then continue
else begin
for j:=0 to 6 do
   if StrToInt(StridGrid1.Cells[j,i])<min then
min:=StrToInt(StridGrid1.Cells[j,i]); end; end; end;
```

D)
```
begin
min:=strToInt(StridGrid1.Cells[0,0]);
for i:=0 to j+1 do begin
if (I mod 2)<>0 then continue
else begin
for j:=0 to 5 do
   if StrToInt(StridGrid1.Cells[j,i])<min then
min=StrToInt(StridGrid1.Cells[j,i]); end; end; end;
```

E)
```
begin
min:=strToInt(StridGrid1.Cells[0,0]);
for i=0 to 6 do begin
if (I mod 2)<>0 then continue
else begin
for j=0 to 6 do
   if StrToInt(StridGrid1.Cells[j,i])<min then
min=StrToInt(StridGrid1.Cells[j,i]); end; end; end;
```

62. ShowMessage процедурасы қандай батырмалы хабарлау терезесін көрсетеді?

A) YES
B) OK
C) CANCEL

D) NO
E) OK, CANCEL

63. MessageDlg функциясының қызметі:

A) Экранның ортасында хабарлау терезесін көрсетеді және қолданушы жауабын алуға мүмкіндік береді;
B) Хабарлау терезесін көрсетеді;
C) Қолданушы жауабын алуға мүмкіндік беретін терезені көрсетеді;
D) Әзірлеуші көрсеткен экрандағы орында хабарлау терезесін көрсетеді және қолданушы жауабын алуға мүмкіндік береді;
E) Экранның жоғарғы сол жақ бұрышында хабарлау терезесін көрсетеді.

64. OpenDialog сұхбатын қолданудың дұрыс нұсқасын көрсетіңіз:

A) Procedure TForm1.Button1Click(Sender: TObject);
Begin
If OpenDialog1.Execute then

Memo1.Text.LoadFromFile(OpenDialog1.FileName);
End;
B) Procedure TForm1.Button1Click(Sender: TObject);
Begin
If OpenDialog1.Execute then

Memo1.Lines.SaveToFile(OpenDialog1.FileName);
End;
C) Procedure TForm1.Button1Click(Sender: TObject);
Begin
If OpenDialog1.Execute then

Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
End;
D) Procedure TForm1.Button1Click(Sender: TObject);

Begin
If OpenDialog1.Execute then
Memo1LinesLoadFromFile(OpenDialog1FileName),
End;
E) Procedure TForm1.Button1Click(Sender: TObject);
Begin
If OpenDialog1.Execute then
Memo1.Caption.LoadFromFile(OpenDialog1.File-Name),
End;

65. Келесі жазба нені білдіреді:
OpenDialog1.Filter:='Мәтіндік файлдар|*.txt;*.doc|Барлық файлдар|*.*;

A) Маскалы сүзгі барлық файлдар үшін қалыптасатын болады;
B) Маскалы сүзгі тек мәтіндік файлдар үшін қалыптасатын болады;
C) Екі маскалы сүзгі қалыптасады – графикалық файлдар үшін маскалы және барлық файлдар үшін маскалы;
D) Мәтіндік файлдардан басқа барлық файлдар үшін маскалы сүзгі қалыптасатын болады;
E) Екі маскалы сүзгі қалыптасады – тек мәтіндік файлдар үшін және барлық файлдар үшін маскалы.

66. Алдағы уақытта ашу мақсатында файлды таңдауға арналған компонент қайсысы?

A) TSaveDialog
B) TOpenPictureDialog
C) TFindDialog
D) TOpenDialog
E) TFontDialog

ТЕСТ СҰРАҚТАРЫНЫҢ ДҰРЫС ЖАУАПТАРЫ

№	Дұрыс жауабы	№	Дұрыс жауабы	№	Дұрыс жауабы
1	E	27	B	53	E
2	A	28	B	54	B
3	C	29	C	55	A
4	A	30	C	56	B
5	D	31	B	57	C
6	C	32	A	58	A
7	B	33	B	59	A
8	B	34	A	60	B
9	C	35	B	61	C
10	A	36	B	62	B
11	C	37	D	63	A
12	A	38	C	64	C
13	A	39	C	65	E
14	D	40	B	66	D
15	D	41	D		
16	A	42	B		
17	B	43	B		
18	B	44	A		
19	C	45	C		
20	D	46	D		
21	C	47	C		
22	A	48	E		
23	A	49	B		
24	B	50	C		
25	A	51	D		
26	B	52	A		

ҚОРЫТЫНДЫ

Оқу құралында бағдарламалауды енді бастап жүрген бағдарламашыларға арналған көптеген материалдар қамтылған. Кітапта жалпы бағдарламалау жайлы түсінік, бағдарламалаудың негізгі ұғымдары, деректер құрылымы және олармен жұмыс жасау туралы нақты түсініктер берілген.

Кітапта тақырыптарға теориялық мәлімет беріліп, мысалмен тұжырымдалып отырады. Бұл бағдарламашыға танысқан материалды тәжірибе жүзінде пайдаланып көруге мүмкіндік береді. Әр тарау соңындағы бақылау сұрақтары оқушылардың өткен материалдарды қайталап, өз білімдерін тиянақтауына жағдай жасайды.

Мектеп қабырғасында «Ақпараттану курсын» оқып келген бірінші курс білімгерлері Паскаль тілін толық біледі деп айту қиын. Кейбір мектептерде бағдарламалау тілдері оқытылмайды. Осыған орай жоғары және арнаулы орта кәсіптік мамандар дайындайтын оқу орындары студенттеріне бұл оқу құралы бағдарламалау тілін үйренуде жақсы көмекші болады деп сенеміз.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

1. Культин Н. Delphi 6. Программирование на Object Pascal. «БХВ-Петербург», – Санкт-Петербург, 2001.

2. Гофман В., Хомоненко А. Delphi 6 в подлиннике. «БХВ-Петербург», – Санкт-Петербург, 2001.

3. Аманжолова С.Т. Работа с визуальными объектами. М/у к лабораторным работам по дисциплине «Проектирование программных средств ЭВМ, систем и сетей» для специальности 3701. – Алматы: КазНТУ, 2001.

4. Аманжолова С.Т. Проектирование программных средств для Windows 9x, Windows NT. М/у к лабораторным работам по дисциплине «Проектирование программных средств ЭВМ, систем и сетей» для специальности 3701. – Алматы: КазНТУ, 2001.

МАЗМҰНЫ

«Кәсіптік білім» сериясы

А.Ә. ШАЙҚҰЛОВА
С.Т.АМАНЖОЛОВА
Н.Т.АСҚАРОВА

БАҒДАРЛАМАЛЫҚ ЖАСАҚТАМАНЫҢ ҚАЗІРГІ ЗАМАНҒЫ ЖАБДЫҚТАРЫ

Оқулық

Редакторы **Н. Рамазанова**
Техникалық редакторы **Р. Тұрлынова**
Корректоры **Б. Мұқышева**
Көркемдеуші редакторы **Ж. Қазанқапов**
Компьютерде беттеген **Е. Ашилов**

Басуға 17.07.10 қол қойылды.
Пішімі 84x108 $^{1}/_{32}$. Қағазы офсеттік.
Офсеттік басылыс. Шартты б. т. 11,34.
Тапсырыс №208*. Таралымы 1000 дана.

«Фолиант» баспасы
010000, Астана қаласы, Ш. Айманов көшесі, 13

«Фолиант» баспасының баспаханасында басылды
010000, Астана қаласы, Ш. Айманов көшесі, 13