А.СОПУЕВ, Н.К.АРКАБАЕВ

Delphi чөйрөсүндө программалоо

- Башкаруучу конструкциялар
- Массивдер
- Функциялар, процедуралар
- Файлдар
- Графика, мультимедия
- Берилгендер базасы

"ПРОГРАММАЛОО" СЕРИЯСЫ

«ПРОГРАММАЛОО»
сериясы

А. Сопуев, Н.К. Аркабаев

DELPHI ЧӨЙРӨСҮНДӨ ПРОГРАММАЛОО

Окуу колдонмо

Ош 2020

УДК 004.43
ББК 32.973-01
С 64

Рецензенттер: физика-мат. илим. доктору, профессор Сатыбаев А.Ж.
физика-мат. илим. кандидаты, доцент Арапбаев Р.Н.

Сопуев А., Аркабаев Н.К.
С 64 Delphi чөйрөсүндө программалоо. Окуу колдонмо.
Ош: ОшМУ, «Билим», 2020. – 150 б.

ISBN 978-9967-18-565-4

Окуу колдонмодо Delphi программалоо тилинде колдонмолорду түзүү технологиясы баяндалып, негизги көңүл объекттерге, объекттердин окуяларына, касиеттерине жана методдоруна, башкаруучу конструкцияларга, массивдерге, функцияларга жана процедураларга, файлдарга, графика жана мультимедиага, берилгендер базасына, объектке-ориентирленген программалоого бөлүнгөн жана мисалдар келтирилген.

Колдонмо студенттер, окутуучулар жана колдонмолорду өз алдынча түзүүнү каалагандар үчүн арналат.

Ош мамлекеттик университетинин Окумуштуулар Кеңешинин чечими менен сунуш кылынды.

С 2404090000-20

УДК 004.43
ББК 32.973-01

ISBN 978-9967-18-565-4

Мазмуну

Кириш сөз 6
1-глава. Delphi программалоо чөйрөсү 7
§ 1.1. Delphi программалоо чөйрөсү 7
1.1.1. Delphi чөйрөсүн орнотуу 7
1.1.2. Проекттердин жумушчу каталогу 7
1.1.3. Delphi нин беш терезеси 8
1.1.4. Delphi деги колдонмонун проекти 10
1.1.5. Проектти сактоо 11
§ 1.2. Объекттердин касиеттери, окуялары жана методдору 12
1.2.1. Объекттердин касиеттери 12
1.2.2. Объекттердин методдору 13
1.2.3. Объекттердин окуялары 13
§ 1.3. Форма жана компоненттер 14
§ 1.4. Тексттик информацияларды кийирүү жана чыгаруу 16
1.4.1. Label компоненти 16
1.4.2. Edit компоненти 17
1.4.3. Memo компоненти 18
1.4.4. Button компоненти 20
1.4.5. BitBtn компоненти 20
1.4.6. RadioButton компоненти 22
2-глава. Delphi программалоо тили 23
§ 2.1. Берилгендердин тиби 23
§ 2.2. Өзгөрмөлөр жана туракттуулар 25
2.2.1. Өзгөрмөлөр 25
2.2.2. Туракттуулар (константалар) 25
2.2.3. Түшүндүрмөлөр (комментарийлер) 26
2.2.4. Типтерди өзгөртүүчү функциялар 26
2.2.5. Берилгендерди калыпка салуу менен чыгаруу 27
2.2.6. Эки сандын суммасын табуу программасы 27
§ 2.3. Тизмелер 30
2.3.1. ListBox компоненти 30
2.3.2. ListBox: жолчолорду кошуу, өчүрүү жана сорттоо 32

2.3.3. ComboBox компоненти ... 33
2.3.4. ComboBox: негизги методдор жана окуялар ... 35
§ 2.4. Диалогдук терезелер ... 36
2.4.1. Кийирүү терезеси ... 36
2.4.2. Билдирүү терезесине чыгаруу ... 38
2.4.3. ShowMessage процедурасы ... 39
2.4.4. MessageDlg функциясы ... 39
3-глава. Delphi тилиндеги башкаруучу конструкциялар ... 43
§ 3.1. Бутактуу алгоритмдерди программалоо ... 43
3.1.1. Шарттуу конструкция ... 43
3.1.2. Тандоо конструкциясы ... 45
§ 3.2. Циклдер, алардын түрлөрү жана колдонулушу ... 47
3.2.1. For конструкциясы ... 48
3.2.2. While конструкциясы ... 50
3.2.3. Repeat конструкциясы ... 51
§ 3.3. Каталыктарды түзөтүү ... 53
4-глава. Массивдер ... 61
§ 4.1. Массивди жарыялоо ... 61
§ 4.2. Массивдердин үстүнөн аткарылган иш-аракеттер ... 62
4.2.1. Массивди чыгаруу ... 62
4.2.2. Массивди киргизүү ... 64
4.2.3. Массивдин минималдык элементин табуу ... 68
4.2.4. Массивден элементти издөө ... 70
4.2.5. Массивди сорттоо ... 72
§ 4.3. Көп ченемдүү массивдер ... 74
5-глава. Символдор жана жолчолор ... 81
§ 5.1. Символдор менен иштөөчү функциялар ... 81
§ 5.2. Жолчолор үчүн стандарттык функциялар ... 83
6-глава. Функциялар жана процедуралар ... 87
§ 6.1. Подпрограмма жана анын түрлөрү ... 87
§ 6.2. Функция жана анын түрлөрү ... 88
§ 6.3. Функция жана анын структурасы ... 88
§ 6.4. Функцияны чакыруу жана колдонуу ... 90
§ 6.5. Функцияны табуляциялоо ... 93

§ 6.6. Процедуралар 96
7-глава. Файлдар 100
§ 7.1. Файлды жарыялоо 100
§ 7.2. Файлдык өзгөрмө 101
§ 7.3. Тексттик файлдар 102
8-глава. Консолдук колдонмолор 106
§ 8.1. Консолдук колдонмону түзүү 106
§ 8.2. Read оператору 107
§ 8.3. Write оператору 107
§ 8.4. Консолдук колдонмо үчүн Windows API функциялары 109
9-глава. Графика жана мультимедиа 111
§ 9.1. Графиктик бет 111
§ 9.2. Сызыктар 112
§ 9.3. Карандаш 113
§ 9.4. Кисть 115
§ 9.5. Тик бурчтук 116
§ 9.6. Бурчтары жумуру тик бурчтук 117
§ 9.7. FillRect жана FrameRect методдору 118
§ 9.8. Текстти чыгаруу 119
§ 9.9. Сынык сызык 122
§ 9.10. Айлана жана эллипс 123
§ 9.11. Жаа 124
§ 9.12. Көп бурчтук 125
§ 9.13. Функциянын графигин тургузуу 126
10-глава. Берилгендер базасы 135
§ 10.1. Берилгендер базасы жана башкаруу системасы 135
§ 10.2. Берилгендер базасынын нормалдык формасы 137
§ 10.3. Телефондордун электрондук справочниги 140
§ 10.4. Microsoft Access менен маалыматтар базасын түзүү 141
§ 10.5. ADO технологиясы 143
Колдонулган адабияттар 150

Кириш сөз

Delphi (Дельфи) — бул объекттердин касиеттерин, окуяларын жана методдорун колдонуу менен визуалдуу түрдө колдонмолорду түзүүгө ылайыкташтырылган чөйрө болуп саналат.

Delphi чөйрөсүндө программалоо тили катары Object Pascal тили алынган жана анын 1-версиясы 1995-жылы 14-февралда Borland фирмасы тарабынан презентация кылынган. Кийинки версияларынын өнүгүү тарыхы төмөнкүдөй:

Delphi 1 (1995), Delphi 2 (1996), Delphi 3 (1997), Delphi 4 (1998), Delphi 5 (1999), Delphi 6 (2001), Delphi 7 (2002).

Delphi 8 (for .NET, 2003), Delphi 2005 (2004), Delphi 2006 (2005), Delphi 2007 (2007), Delphi 2009 (2008), Delphi 2010 (2009).

Delphi XE (2010), Delphi XE2 (2011), Delphi XE3 (2012), Delphi XE4 (2013), Delphi XE5 (2013), Delphi XE6 (2014), Delphi XE7 (2014), Delphi XE8 (2015), Delphi 10 Seattle (2015).

2006-жылы Borland фирмасынан CodeGear (Embarcadero) компаниясы бөлүнүп чыгат жана ал 2007-жылы веб-колдонмолорду PHP программалоо тилинде түзүү үчүн Delphi for PHP пакетин чыгарган. Ошондой эле тармактык технологияларды колдонуу үчүн Delphi for .NET жана Delphi Prism (for .NET, 2008), Delphi Prism 2011 (for .NET) версиялары да чыгарылган.

Окутуу процессинде Delphi 7 версиясы көп колдонулат, анткени Delphi 7 де визуалдык проектирлөө технологиясын жана объекттердин касиеттерин, окуяларын жана методдорун жеңил жана тез өздөштүрүүгө болот, ошондой эле жөнөкөй программалардан баштап берилгендер базасы менен иштей ала турган профессионалдык программаларды түзсө болот.

Delphi 7 де программаларды түзүүнү үйрөнүп алгандан кийин Delphi нин башка версияларында иштеп кетүү анча деле кыйынчылыкты туудурбайт.

Колдонмодогу материалдарды окуп-үйрөнүү менен студент Delphi визуалдык чөйрөдөсүндө маселелерди объекттик-ориентирленген программалоо методдору менен чыгарууга, Windows-колдонмо түзүүгө, анын интерфейсин конструкциялоого, берилгендерди базага сактоо менен маселелерди чечүүнү автоматташтырууга жетишүүсү керек.

1-глава. Delphi программалоо чөйрөсү

§ 1.1. Delphi программалоо чөйрөсү

1.1.1. Delphi чөйрөсүн орнотуу

Delphi пакетинин Borland Delphi 7 версиясында жөнөкөй бир терезелүү колдонмолордон маалыматтар базасын башкарууга чейинки программаларды түзүү өтө ыңгайлуу болгондуктан аны окутуу процессинде кеңири колдонулушууда.

Borland Delphi 7 пакети менен Windows операциялык системасынын чөйрөсүндө иштөөгө болот. Процессорго коюлуучу талаптар: тактык жыштыгы 166 МГц тен кем эмес, оперативдик эси 128 Мбайттан кем эмес болушу керек. Программаны толук орнотуу үчүн 475 Мбайт мейкиндик керек болот.

Borland Delphi 7 пакетинин төрт варианты бар: Personal, Professional, Enterprise жана Architect. Пакеттин деңгээли канчалык жогору болсо, ошончолук мүмкүнчүлүгү кеңири болот. Мисалы, Enterprise пакети обочолонгон базалар менен (мисалы, InterBase) иштей алса, Personal пакетинин андай мүмкүнчүлүгү жок.

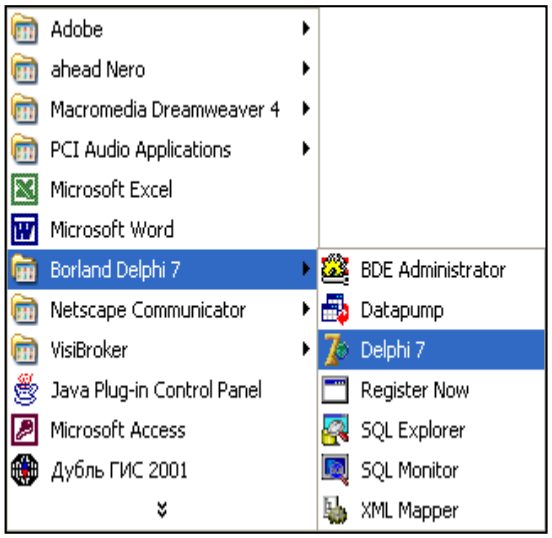
Delphi 7 ни орнотуу үчүн анын диструбитиви жазылган дисктен Delphi Setup Launcher командасын жүктөө керек.

1.1.2. Проекттердин жумушчу каталогу

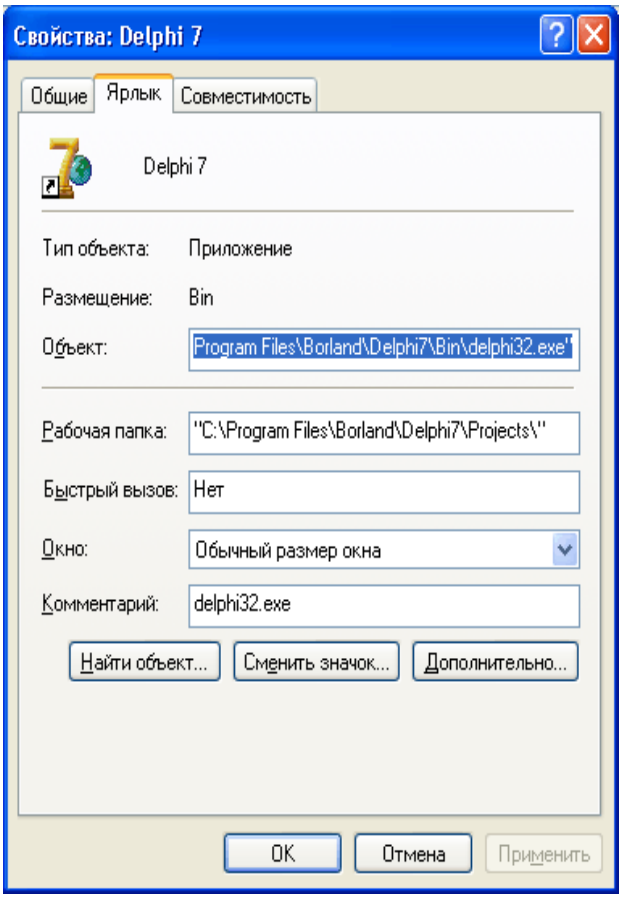
Программаны орноткондон кийин Delphi ни жүктөөгө болот. Бирок эң оболу D:\ дискине жумушчу каталог (проекттердин каталогун) түзүп алуу керек. Мисалы, D:\ дискине **Delphi** деген папка түзүп алабыз: D:\Delphi. Ушул папканы жумушчу каталог кылуу үчүн, б.а. сактала турган проекттер ушул папкага сакталышын камсыз кылуу үчүн төмөнкүдөй киришебиз: Чычкандын көрсөткүчүн

Пуск | Программы | Borland Delphi 7 | Delphi 7

командасына алып келип оң баскычты басып, **Свойства** деген команданы тандаганыбызда (1.1-сүрөт) **Свойства: Delphi 7** терезеси ачылат (1.2-сүрөт). **Рабочая папка** деген талаага проекттерди жазуу үчүн дайындалган папканын дарегин жазабыз, б.а. "D:\Delphi" деп жазабыз. Андан кийин ОК баскычын басабыз. Натыйжада ушул папка проекттерди сактоо учурунда автоматтык түрдө ачылат.



1.1-сүрөт. Delhi ни жүктөө



1.2-сүрөт. Жумушчу папка түзүү

Delphi 7 ни жүктөө

Пуск | Программы | Borland Delphi 7 | Delphi 7 командасы аркылуу ишке ашырылат.

Эгерде жумушчу столдо Delphi 7 нин ярлыгы болсо, аны эки жолу чыкылдатуу менен да жүктөөгө болот.

1.1.3. Delphi нин беш терезеси

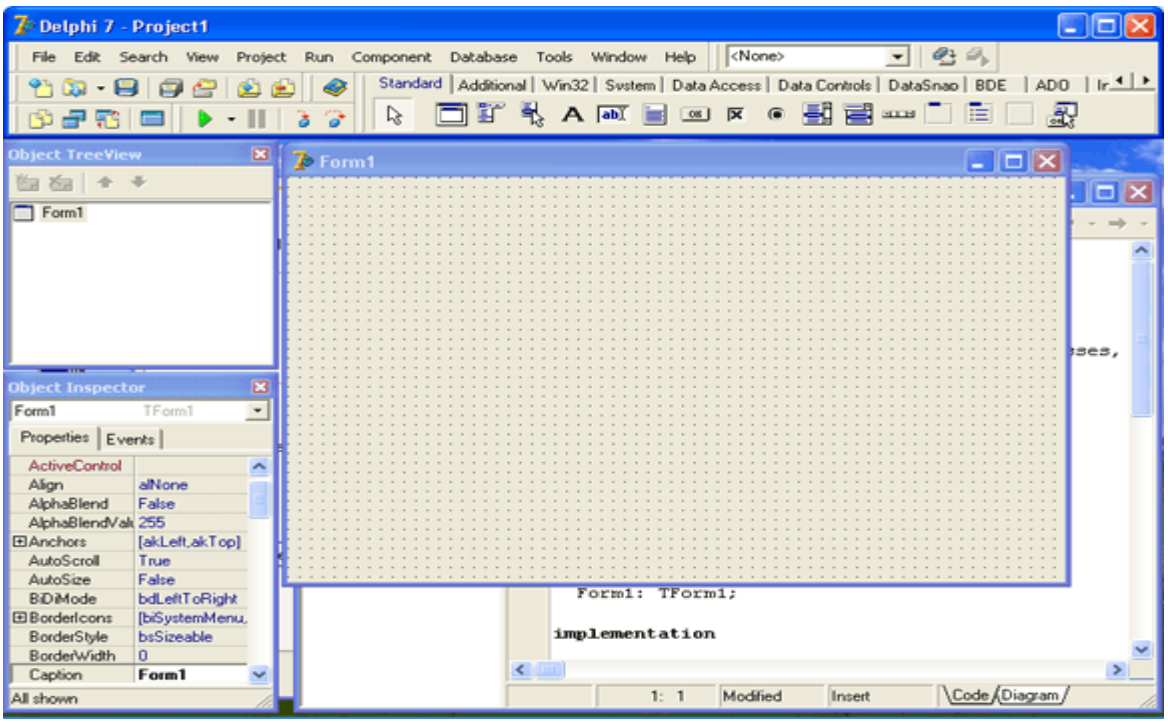
Delphi жүктөлгөндө экранда беш терезе пайда болот (1.3-сүрөт):

1. Башкы терезе – **Delphi 7;**
2. Форманын алгачкы терезеси – **Form1;**
3. Объекттердин касиеттеринин редактору терезеси — **Object Inspector;**
4. Объекттердин тизмесин кароо терезеси – **Object TreeView;**
5. Коддор редакторунун терезеси – **Unitl.pas.**

Ар бир терезени өз алдынча өзгөртүүгө болот. Коддор редакторунун терезеси форманын алгачкы терезеси аркылуу

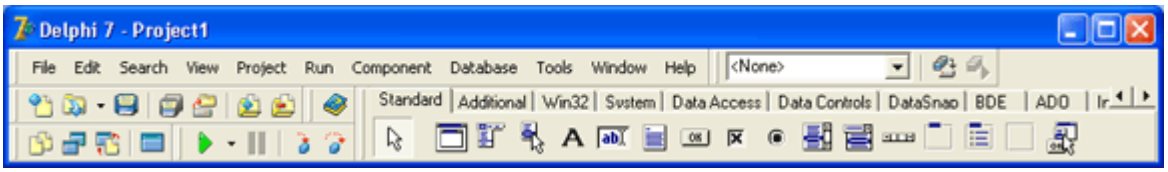
дээрлик жабылып турат, аны көрүү үчүн F12 же **Togle Form / Unit** (F12) баскычын басуу керек.

Коддор терезесинен формалар терезесине өтүү үчүн жогорудагы баскычты кайрадан басуу керек.



1.3-сүрөт. Delphi нин жумушчу экраны

1). Башкы терезеде (Delphi 7) менюлар жолчосу, командалар, инстру-менттер панели жана компоненттер палитрасы жайгашкан (1.4-сүрөт).



1.4-сүрөт. Башкы терезе

2). Форманын алгачкы терезеси (Form1) – колдонмолорду түзүү үчүн ылайыкташтырылган контейнер болуп эсептелет, анткени анын бетине каалагандай компоненттерди жайгаштырууга болот (1.3-сүрөт).

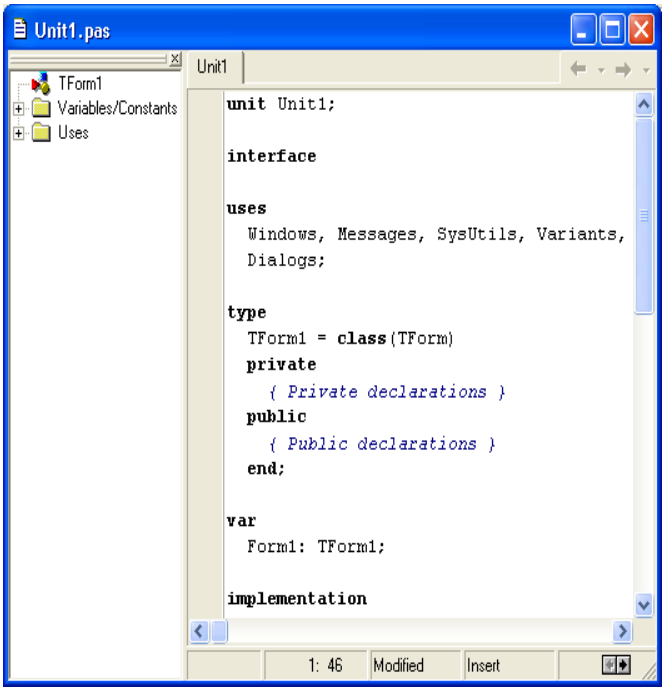
3). Объекттердин инспектору (Object Inspector**)** терезеси – объекттердин касиеттеринин редакторунун терезеси болуп эсептелет жана ал объекттердин касиеттеринин жана окуяларынын маанилерин редактирлөө үчүн колдонулат (1.3-

сүрөт). Объекттин касиеттери **Properties** барагынан тандалат. Объекттин касиеттери объекттин түрүн, абалын жана кыймылын мүнөздөйт. Мисалы, width и height касиеттери форманын өлчөмүн (туурасын жана бийиктигин), top жана left касиеттери – форманын экрандагы абалын, caption касиети – бөрктүн текстин аныктайт.

Объекттин окуялары **Events** барагынан тандалат. Объекттин окуяларынын мисалы катары объектке бир же эки жолу чыкылдатуу, объектти тазалоо же жабуу, клавишаны басуу ж.б.у.с. окуяларды алууга болот.

4). Объекттердин тизмесин кароо (Object TreeView) терезеси – объекттердин формадагы жайгашуу дарагын мүнөздөйт, б.а. объекттерди табуу жана тандоо үчүн колдонулат (1.3-сүрөт).

5). Коддор редактору (Unit1.pas) терезесинде программанын тексти жазылат. Жаңы проект менен ишти баштоонун алдында Delphi тарабынан программанын шаблону сунушталат (1.5-сүрөт).



1.5-сүрөт. Коддор редактору

1.1.4. Delphi деги колдонмонун проекти

Delphi де түзүлгөн ар бир колдонмо (программа) үчүн проект түзүлөт. Проект формалардан, модулдардан жана башка ресурстардан турат. Модул англис тилинде Unit деп аталат. Биз форма менен иштеп жатканда форманын модулу да пайда болот. Проект түзүлүп жатканда төмөнкү файлдар түзүлөт:

1) **Проекттин файлы (.dpr)** – формалар жана модулдар боюнча информацияларды сактоо үчүн колдонулуучу тексттик файл. Мында инициализациялоо жана программаны аткаруу үчүн жүктөө операторлору бар;

2) **Модулдун файлы (.pas) –** программанын коду сакталуучу модул;

3) **Формалардын файлы (.dfm) –** формаларды баяндоо файлы);

4) **Проекттин опцияларынын файлы (.dof) –** проектти баяндоо файлы;

5) **Проекттин конфигурацияларынын файлы (.cfg);**

6) **Ресурстар файлы (.res) –** пиктограммаларды жана файлдардын ресурстарын кармап турат;

7) **Резервдик копиялар (~dp,~df,~pa).**

Компилятор жогорудагы файлдарды пайдалануу менен программаны аткаруучу файлды (EXE-файл) түзөт.

Кеңейтилиши *.dpr болгон файлды башкы модул деп айтабыз. Анын текстин көрүү үчүн Project менюсунан View Source командасын тандоо керек.

1.1.5. Проектти сактоо

Проектти сактоо үчүн **File** менюсунан **Save Project As** командасын тандоо керек. Эгерде проект мурда бир да жолу сакталбаган болсо, анда Delphi эң оболу модулду (коддор редактору терезесиндеги малыматтарды) сактоону сунуштайт. Ошондуктан экранга **Save Unitl As** терезеси чыгарылат. Бул терезеде проект сакталуучу папканы тандап алуу керек жана модулдун аталышын кийирүү керек.

Программа модулдун аталышын Unit1.pas деп атоону сунуштайт. Бирок модулдун аталышын маселени чечүү темасы менен байланышкан ат катары берүү туура болот, мисалы биз сандардын суммасын табуу маселесин карап жатсак, Usumma деп атаганыбыз оң болот. Бул жерде файлдын аталышындагы биринчи тамганын U тамгасы менен байланышы ал модулдун файлы экендигин билдирип турат жана аны проекттен жеңил эле айырмалоого болот.

«Сохранить» баскычын баскандан кийин кийинки терезе, башкача айтканда, проектти сактоо терезеси экранга чыгарылат. Мында проекттин файлынын аталышын тандоо менен проектти сактайбыз. Проекттин аталышын модулдун аталышына жакындаштырып, мисалы, Psumma деп атоого болот. Файлдын аталышындагы биринчи тамганын P тамгасы менен байланышы ал проекттин файлы экендигин билдирип турат.

1.1-эскертүү. *Проект файлдардын жыйындысы*

болгондуктан ар бир проектти өзүнчө папкага сактоо сунуш кылынат.

§ 1.2. Объекттердин касиеттери, окуялары жана методдору

1.2.1. Объекттердин касиеттери

Объектке-ориентирленген программалоо (ООП) тилиндеги негизги бирдик болуп программалык объект эсептелет.

Программалык объект берилгендерди мүнөздөөчү **касиеттерди** кармап турат, берилгендерди кайра иштетүү үчүн **методдорду** колдонуу мүмкүнчүлүгүнө ээ жана **окуяларга** жооп бере алат. Эгерде салыштырып айта турган болсок, объекттер – зат атооч, объекттин касиеттери – сын атооч, объекттин методдору – этиш деп айтсак болот.

Объекттердин **классы** деп объекттерди мүнөздөөчү касиеттердин, методдордун жана окуялардын жыйындысын айтабыз.

Объекттердин классына таандык болгон экземпляр ушул класстын бардык касиеттеринин, методдорунун жана окуяларынын жыйындысына ээ болот, б.а. мураскери болот. Класстын ар түрдүү экземплярлары бирдей окуялардын жыйындысына ээ болот, бирок ал окуялардын маанилери айырмаланышы мүмкүн.

Объекттердин касиеттеринин баштапкы маанилерин программалоо системасынын диалогдук терезесин колдонуу менен аныктап алышыбыз мүмкүн. Ар бир объекттин баштапкы маанилери Properties барагында көрсөтүлгөн.

Объекттердин касиеттерин эки жол менен: колдонмону моделдөө учурунда же программа аткарылып жатканда өзгөртүүгө болот.

1). Объекттердин касиеттерин колдонмону моделдөө учурунда өзгөртүү үчүн Properties барагындагы касиеттердин маанилерин өзгөртүү керек.

2). Объекттин касиеттерин программа аткарылып жатканда өзгөртүү үчүн программалык коддун сол жагында объекттин атын көрсөтүп, андан кийин точка коюуп, объекттин касиетинин атын жазуу керек, ал эми барабардыктын оң жагына касиеттин маанисин жазуу керек:

Объект.Касиет = КасиеттинМааниси;

1.2.2. Объекттердин методдору

Объект кандайдыр бир операцияны же аракетти аткарышы үчүн анын өзүнө таандык болгон **методду** колдонуу керек.

Методдор деп объекттердин үстүнөн аткарылуучу аракеттерди камсыздоо үчүн дайындалган прцедураларды жана функцияларды айтабыз.

Методдор **interface** секциясында жарыяланат, ал эми алардын программалык коддору **implementation** секциясына жазылат.

Көп методдор бир же бир нече аргументтерге ээ. Аргументтер аракетти аткаруунун параметрлерин көрсөтүп берет. Аргументтерге конкреттүү маанилерди ыйгаруу үчүн кош чекит жана барабардык белгиси (**:=**) колдонулат, ал эми аргументтер бири-биринен үтүр (**,**) менен ажыратылат.

Кайсыл объектке кандай метод колдонулуп жаткандыгын белгилөө үчүн чекит (**.**) белгисин колдонобуз, б.а. биринчи объектти жазып, андан кийин чекит коюп, акырында методдун аталышын жазуу керек:

Объект.Метод арг1:=маани, арг2:=маани;

1.2.3. Объекттердин окуялары

Программа иштеп жаткан учурдагы объекттер тарабынан аткарылуучу аракеттер **окуялар** (event) деп аталат.

Delphi де ар бир окуяга ат берилген. Ар бир окуянын аталышы бар. Мисалы, чычкандын баскычын чыкылдатуу - OnClick окуясы, ал эми чычкандын баскычы менен эки жолу (кош) чыкылдатуу - OnDblClick окуясы болуп саналат.

1.1-таблицада Windows дүн кээ бир окуялары келтирилген.

1.1-таблица. Окуялар

Окуялар	Качан аткарылат
OnClick	Чычкандын баскычын чыкылдатканда
OnDblClick	Чычкандын баскычын кош чыкылдатканда
OnMouseDown	Чычкандын баскычын басканда
OnMouseUp	Чычкандын баскычын коюп жибергенде
OnMouseMove	Чычкандын көрсөткүчүн жылдырганда
OnKeyPress	Клавиатуранын баскычын басканда

OnKeyUp	Клавиатуранын баскычын коюп жибергенде
OnCreate	Объекти (форманы, башкаруу элементин) түзгөндө

Окуяга жооп берүү үчүн кандайдыр бир аракет жасалыш керек. Ошондуктан колдонуучунун аракетине программа кандайдыр бир ишти аткаруусу үчүн программист тиешелүү окуяны иштетүүчү процедураны жазышы керек.

Окуяны иштетүү процедурасын түзүү үчүн эң оболу **Object Inspector терезесинен** компонентти тандап алуу керек. Андан кийин ушул эле терезеде **Events** (Окуялар) барагына өтөбүз.

Events (Окуялар) барагынын сол колонкасында тандалган компонент кабыл ала турган окуялардын аталыштары көрсөтүлгөн. Эгерде окуя үчүн иштетүү процедурасы жазылган болсо, анда оң колонкада бул процедуранын аты көрсөтүлгөн.

Окуяны иштетүү функциясын түзүү үчүн процедуранын аты турган талаага чычкандын сол баскычын эки жолу чыкылдатуу керек. Натыйжада коддордун редактору ачылып, окуяларды иштетүү процедурасынын шаблону пайда болот.

§ 1.3. Форма жана компоненттер

Delphi де жаңы проектти даярдоо үчүн эң биринчи жүктөлө турган **баштапкы** форманы түзүү керек. Ал үчүн **Form1** формасынын касиеттеринин маанилерин өзгөртүү жана формага керектүү компонеттерди жайгаштыруу керек.

Форманын касиеттери анын тышкы көрүнүшүн, б.а. өлчөмүн, экрандагы абалын, бөрктөгү текстин, рамкасынын түрүн аныктайт (1.2-таблица).

1.2-таблица. Форманын касиеттери

Касиеттери	**Түшүндүрмөсү**
Name	Форманын аты. Программада форманын аты форманы башкаруу жана анын компонеттерине жетүү үчүн колдонулат
Caption	Бөрктүн (заголовканын) тексти
Width	Форманын туурасы
Height	Форманын бийиктиги
Top	Форманын жогорку чекарасынан экрандын жогорку чекарасынан чейинки аралык

Left	Форманын сол чекарасынан экрандын сол чекарасынан чейинки аралык
BorderStyle	Чек аранын түрү. Чек аранын түрү **кадимки-дей** (bsSizeable), **жөнөкөй** (bsSingle) же **жок** (bsNone) болушу мүмкүн
BorderIcons	Терезени башкаруу кнопкасы
Icon	Диалогдук терезенин бөркүндөгү значогу
Color	Фондун түсү
Font	Шрифт

Форманын касиеттерин өзгөртүү үчүн **Object Inspector** терезеси колдонулат**. Object Inspector** терезесинин жогору жагында объекттин аты жана касиеттеринин маанилери көрсөтүлгөн. **Properties** (Свойства) барагынын сол жагында объекттин касиеттери, ал эми оң жагында алардын маанилери көрсөтүлгөн.

Каалаган компонентти формага жайгаштыруу үчүн чычкандын сол баскычы менен компонентке бир жолу чыкылдатабыз, андан кийин чычкандын сол баскычын формага алып келип, компонент жайгаштырыла турган жерге дагы бир жолу чыкылдатабыз. Натыйжада компонент формага жайгашат. Андан кийин Объекттер инспектору терезесиндеги касиеттерди (Propeties) же окуяларды (Events) тандоо менен компонентти маселелерди чечүүгө колдонобуз.

Формадагы компонетти жок кылуу үчүн эң оболу чычкандын сол баскычы менен аны тандап алуу керек, андан кийин **Del** баскычын басуу керек.

Delphi нин ар бир компонентине компоненттин атынан жана анын катар номеринен турган ат ыйгарылат. Мисалы, формага Edit элементинин эки компонентин жайгаштырсак, анда алардын аттары тиешелеш түрдө Edit1 жана Edit2 болот. Программист Name касиетин өзгөртүү менен компонентке жаңы ат бериши мүмкүн.

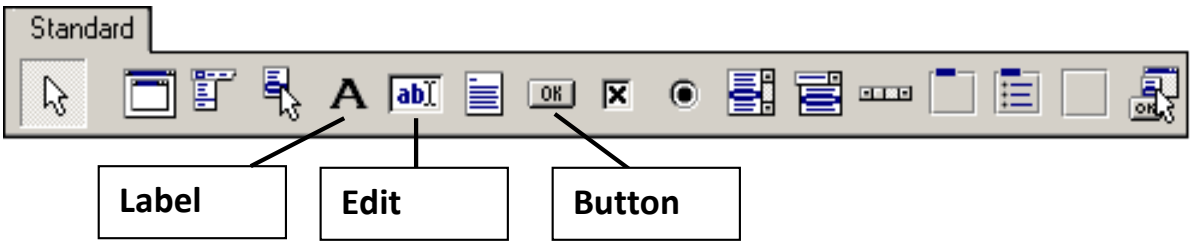
Программанын ишин аяктоо үчүн негизги терезени, б.а. форманы жабуу керек. Ал үчүн формага кнопка жайгаштырып, анын onClick окуясын иштетүүчү процедурага **Close** методун колдонобуз. Программалык код төмөнкүдөй болот:

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

```
    Form1.Close;
end;
```

§ 1.4. Тексттик информацияларды кийирүү жана чыгаруу

Тексттик информацияларды кийирүү жана чыгаруу үчүн **Standard** барагынын **Label, Edit** жана **Memo** компоненттери колдонулат. **Standard** барагында жалпысынан 17 компонент бар



1.6-сүрөт. Standard барагынын компоненттери

(1.6-сүрөт). Алардын ичинен колдонмодо окуяны моделдештирүү үчүн **Button** компоненти колдонулат. Ушул эле максатта Additional барагында жайгашкан **BitBtn** компоненти да колдонулат.

1.4.1. Label компоненти

Label компоненти **A** символу менен белгиленген. Ал текстти форманын бетине чыгаруу үчүн колдонулат. Анын касиеттери тексттин көрүнүшүн жана жайгашышын аныктайт (1.3-таблица).

1.3-таблица. Label компонентинин касиеттери

Касиети	Колдонулушу
Name	Компоненттин аты. Компонентти жана анын касиеттерин тандоо үчүн колдонулат.
Caption	Компоненттин талаасына жазыла турган текст.
AutoSize	Талаанын өлчөмү тексттин узундугуна жараша өзгөрөт.
WordWrap	Учурдагы жолчого батпай калган текст автоматтык түрдө кийинки жолчого өткөрүлөт.
Alignment	Талаанын ичинде тексттин түздөлүшүн аныктайт.
Font	Текстти чагылдыруучу шрифтти аныктайт.
Color	Текст чыгарыла турган областтын фонунун түсүн аныктайт.
Visible	Текстти көрүнө (True) же көрүнбөй (False) турган кылат.

Label компонентине текстти чыгаруу үчүн 2 жолду колдонсо болот:

1). Проектирлөө учурунда Caption касиетинин талаасына текстти кийирсе болот. Кийирилген текст автоматтык түрдө компоненттин бетине чыгат.

2). Программа иштеп жатканда текстти программалык режимде чыгарса да болот. Мисалы,

```
Label1.Caption:='Саламатсынарбы!';
```

Label компонентине чыгарылган текстти программалык режимде өчүрүү үчүн төмөнкү кодду колдонсок болот:

```
Label1.Caption:=' ';
```

1.4.2. Edit компоненти

Edit компоненти бир жолчолуу тексттик талааны билдирет. Edit компоненти колдонуучу тарабынан кийирилген текстти алуу жана аны чагылтуу үчүн колдонулат. Edit компонентинин төмөнкүдөй касиеттери бар (1.4-таблица).

1.4-таблица. Edit компонентинин касиеттери

Касиети	Колдонулушу
AutoSelect	Касиеттин мааниси True болгондо текстти бөлүп алуу аткарылат
BorderStyle	Компоненттин ички бөлүгү менен колдонмонун клиенттик областынын ортосундагы чек араны аныктайт. Эки маанини алат: bsNone – чек ара жок, bsSingle – чек ара бар
CharCase	Компонентте чагылдырылган символдордун регистрин аныктайт, унчукпаганда ecNormal мааниси алынат
Color	Компоненттин түсүнүн фону, унчукпаганда clWindow түсү алынат
MaxLength	Edit талаасына кийирилүүчү символдордун санын чектейт, унчукпаганда 0 мааниси алынат, бул учурда символдордун саны чектелбейт

PasswordChar	Информацияны жашыруу үчүн колдонулат. Эгерде касиеттин мааниси #0 болсо, анда текст нормалдык түрдө чагылтылат. Эгерде касиеттин маанисине каалагандай башка символ жазылса, ошол символ Edit талаасында кайталанып көрсөтүлөт
ReadOnly	Эгерде касиеттин мааниси True болсо, анда текстти редактирлөөгө тыюу салынат
Text	Компоненттин талаасына кийирилген маанилер жолчо тибиндеги (текст түрүндөгү) берилгендерди аныктайт

Edit компонентине текстти кийирүү же чыгаруу үчүн 2 жолду колдонсо болот:

1). Проектирлөө учурунда Text касиетинин талаасына текстти кийирсе болот. Кийирилген текст автоматтык түрдө компоненттин бетине чыгат.

2). Программа иштеп жатканда текстти программалык режимде чыгарса да болот. Мисалы,

```
Edit1.Text:='Delphi!';
```

Edit компонентине чыгарылган текстти программалык режимде өчүрүү үчүн төмөнкү кодду колдонсок болот:

```
Edit1.Text:=' ';
```

1.4.3. Memo компоненти

Memo компоненти көп жолчолуу тексттик талааны аныктайт. Аны жөнөкөй тексттик редактор деп караса да болот. Memo компонентинин кээ бир касиеттери 1.5-таблицада келтирилди.

1.5-таблица. Memo компонентинин кээ бир касиеттери

Касиети	Колдонулушу
Name	Компоненттин аты. Программада компоненттин касиеттерин колдонуу үчүн пайдаланылат
Text	Memo талаасындагы текстти аныктайт

Lines	Memo талаасындагы текстти жолчолордун жыйындысы катары аныктайт. Жолчо өзүнүн номери боюнча аныкталат
Lines.Count	Memo талаасындагы тексттин жолчолорунун саны
Left	Талаанын сол чек арасынан форманын сол чек арасына чейинки аралык
Top	Талаанын жогорку чек арасынан форманын жогорку чек арасына чейинки аралык
Height	Талаанын бийиктиги
Width	Талаанын туурасы
Font	Кийирилүүчү тексттин чагылтуучу шрифт
ParentFont	Ата-энелик форманын шрифтинин касиетин мурастоо белгиси

Memo талаасына кийирилген жолчолор 0 дөн баштап номерленген болот жана алар Lines касиетинде сакталат. Мисалы, i жолчодогу текст Lines[i] түрүндөгү объект болот. Lines[i] объектиси окуу үчүн да жана текст менен иштөөгө байланышкан иштерди аткаруу үчүн да колдонулат. Memo компонентине текстти клавиатурадан же программалык режимде да кийирүүгө болот.

Жаңы жолчолорду жазуу үчүн

- **Add()**
- **Insert()**

методдору колдонулат.

Add() методу жаңы жолчону тексттин эң акырына кошот, ал эми Insert() методу жаңы жолчону номери көрсөтүлгөн жолчонун алдына кошот.

```
Memo1.Lines.Add('Бул жолчо эң акыркы жолчо болот');
Memo1.Lines.Insert(2, 'Бул үчүнчү жолчо болот');
```

- **Delete()** методу номер көрсөтүлгөн жолчону өчүрөт.

```
Memo1.Lines.Delete(i); // i индекстүү жолчо өчүрүлөт.
```

Memo талаасындагы жолчолордун саны төмөнкүдөй аныкталат: n = Memo1.Lines.Count;

1.4.4. Button компоненти

Button компоненти жөнөкөй командалык баскычты аныктайт. Бул компонент баскычтын **OnClick** окуясын ишке ашыруу үчүн колдонулат. (1.6-таблица).

1.6-таблица. Button компонентинин кээ бир касиеттери

Свойство	Колдонулушу
Action	Баскыч менен байланышкан аракетти аныктайт
Cancel	OnClick окуясы Esc баскычы менен ишке ашырылабы дегенди аныктайт
Caption	Баскычтагы жазууну аныктайт
Default	Баскычты басуу Enter баскычын басуу менен тең күчтүү экендигин аныктайт
ModalResult	ModalResult модалдык формасынын касиетинин маанисин аныктайт
TabOrder	Табуляциялоо тизмесиндеги компоненттин позициясын көрсөтөт
TabStop	Баскычка Tab клавишасы менен жетүүгө мүмкүн экендигин аныктайт

1.7-таблица. Button компонентинин негизги методдору

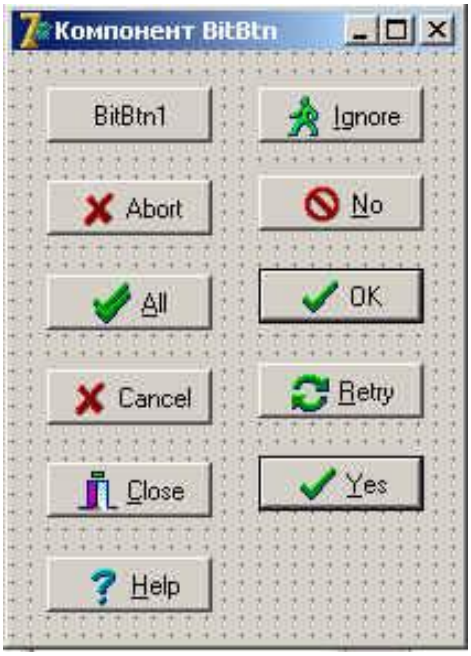
Методдор	Колдонулушу
Click	Click процедурасы. Баскычка чычкан менен чыкылдаганды имитациялайт
SetFocus	SetFocus процедурасы. Элементти активдештирүү менен фокусту берет
OnClick	Баскычка чычкан менен чыкылдатканды билдирет

1.4.5. BitBtn компоненти

BitBtn компоненти Additional барагында жайгашкан. Анын негизги касиеттери Button компонентинин касиеттерине окшош. Ошондуктан ал стандарттык Button классынын мураскери болот.

Бирок, андан айырмаланып, бул баскычтын бетине текст менен кошо сүрөттү да жайгаштырса болот.

Kind касиетин өзгөртүү менен Delphi де алдын ала белгилүү болгон сүрөттөрдү компонентке коюуга болот (1.7-сүрөт).



1.7-сүрөт. BitBtn компоненти. Kind касиетинин маанилери

Мисалы, компонентке bkClose константасын койсок, анда баскыч проекттин формасын жабуу баскычына айланат, бул учурда баскычка эч кандай кодду жазуунун зарылчылыгы жок болуп калат.

BitBtn компонентине колдонуучу өзү каалаган башка сүрөттү койсо да болот. Ал үчүн **Glyph** касиетин пайдалануу керек. Анын баштапкы мааниси None болот. Бул учурда баскычта сүрөт болбойт

BitBtn компонентине растрдык сүрөттү коюу үчүн объекттер инспекторундагы Glyph касиетинин үч чекиттүү баскычын басабыз. Натыйжада экранга Picture Editor диалогдук терезеси чыгарылат. Сүрөттү жүктөө үчүн Load баскычын басабыз жана файлды ачуу диалогдук терезесин пайдаланып керектүү файлды тандоо менен сүрөттү баскычтын бетине чыгарабыз. Баскычка коюлуучу сүрөттүн өлчөмү анча чоң эмес болуш керек. Жүктөлүүчү сүрөттүн файлынын кеңейтилиши **.bmp** түрүндө болгону жакшы болот. Керек болгон учурда сүрөттүн өлчөмүн өзгөртүп койсо болот.

Компонентке коюлуучу сүрөттү Delphi нин курамына кирген Image Editor редактору же Adobe Photoshop программасы менен даярдап алса болот. Кеңейтилиши *.bmp болгон сүрөттү компьютерден издөө системасы аркылуу таап алса да болот. Эгерде сүрөттүн форматы башка болсо, аны bmp форматка, мисалы, Photoshop программасы менен өзгөртүп алса болот.

Текст менен сүрөттүн өз ара жайгашышы Layout касиети менен аныкталат.

- blGlyphRight – сүрөт тексттин оң жагында;
- blGlyphLeft - сүрөт тексттин сол жагында;

- blGlyphBottom - сүрөт тексттин астында;
- blGlyphTop - сүрөт тексттин үстүндө.

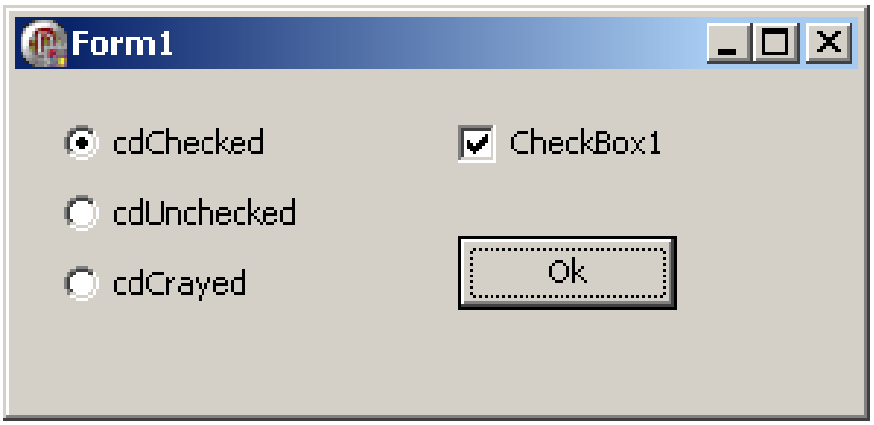
Сүрөт менен тексттин ортосундагы аралык **Spacing** менен берилет. Унчукпаган учурда аралык 4 пикселге барабар болот. Эгерде -1 маанисин алсак, анда текст сүрөт менен баскычтын четинин ортосуна түздөлөт.

1.4.6. RadioButton компоненти

Бул компонент опцияларды тандоонун радио кнопкасы. Радио кнопкалар бир нече опциядан бирөөнү гана тандоого уруксат берет. Радио кнопкалар группа түрүндө иштегендиктен алардын саны бирден көп болуусу зарыл. Радио кнопканын тандалганын билүү үчүн **Checked** касиетин пайдаланабыз. Компонент бизге мурдагы компоненттерден белгилүү болгон касиеттерге жана окуяларга ээ.

Формага үч TRadioButton жана бирден TCheckBox, TButton жайгаштырабыз. Радио кнопкалардын Caption талааларына тиешелеш түрдө cbChecked, cbUnchecked жана cbGrayed маанилерин жазабыз. Экинчи радио кнопканын Checked талаасына true маанисин орнотобуз. Командалык кнопканын OnClick окуясына көрсөтүлгөн программалык кодду жазып, программанын иштөөсүн текшеребиз.

```
if RadioButton1.Checked then CheckBox1.State:=cbChecked;
if RadioButton2.Checked then CheckBox1.State:=cbUnchecked;
if RadioButton3.Checked then CheckBox1.State:=cbGrayed;
```



1.8-сүрөт. RadioButton компоненти менен иштөө

2-глава. Delphi программалоо тили

§ 2.1. Берилгендердин тиби

Delphi программалоо чөйрөсүндө Delphi программалоо тили колдонулат, анын негизин обьектке багытталган Object Pascal тили түзөт.

Программа деп маселени чечүү үчүн түзүлгөн инструкциялардын (көрсөтмөлөрдүн) удаалаштыгын айтабыз.

Инструкциялар бири биринен үтүрлүү чекит менен ажыратылат. Инструкциялар идентификаторлордон турат.

Идентификаторлор төмөнкүлөрдү билдирет:

- *тилдин инструкциясы (if, while, for ж.б.);*
- *өзгөрмөлөр;*
- *турактуулар (бүтүн, чыныгы сан же жолчо);*
- *арифметикалык (+, -, *, /) операцияларды;*
- *логикалык (and, or, not) операцияларды;*
- *подпрограммаларды (процедура же функция);*
- *подпрограммалардын башталышын белгилөө;*
- *блоктун башталышын (begin) жана аягын (end).*

Delphi тилинде берилгендердин 5 түрдүү типтери бар:

- *бүтүн сандар;*
- *чыныгы сандар;*
- *символдор;*
- *жолчолор (символдор жолчосу);*
- *логикалык чоңдуктар.*

Бүтүн сандар тиби. Delphi тилинде бүтүн сандар тиби 7 типке бөлүнөт: ShortInt, SmallInt, LongInt, Int64, Byte, Longword 2.1-таблица).

2.1-таблица. Бүтүн сандар тиби

Тип	**Диапазон**	**Формат**
ShortInt	-128 - 127	8 бит
SmallInt	-32 768 - 32 767	16 бит
LongInt	-2 147 483 648 - 2 147 483 647	32 бит
Int64	-263- 263 - 1	64 бит
Byte	0 - 255	8 бит
Word	0 - 65 535	16 бит

Longword	0 - 4 294 967 295	32 бит

Мындан тышкары универсалдык бүтүн тип болгон Integer тиби да колдонулат жана ал Longint тибине эквиваленттүү.

Чыныгы сандар тиби. Delphi тилинде чыныгы сандар тиби 6 типке бөлүнөт: Real48, Single, Double, Extended, Comp, Currency (2.2-таблица).

2.2-таблица. Чыныгы сандар тиби

Тип	**Диапазон**	**Формат**
Real48	2.9·10-39 - 1.7·1038	6 байт
Single	1.5·10-45 - 3.4·1038	4 байт
Double	5.0·10-324 - 1.7·10308	8 байт
Extended	3.6·10-4951 - 1.1 104932	10 байт
Comp	263+1 - 263-1	8 байт
Currency	-922 337 203 685 477.5808 – 922 337 203 685 477.5807	8 байт

Мындан тышкары универсалдык бүтүн тип болгон Real тиби да колдонулат жана ал Double тибине эквиваленттүү.

Символдук тип. Символдук тип 2ге бөлүнөт: AnsiChar жана WideChar.

- AnsiChar тиби — бул ANSI кодировкасындагы символдор, аларга 0 дөн 255 диапазонундагы сандар туура келет;
- WideChar тиби — бул Unicode кодировкасындагы символдор, аларга 0 дөн 65 535 диапазонундагы сандар туура келет;

Мындан тышкары универсалдык символдук тип болгон Char тиби да колдонулат жана ал AnsiChar тибине эквиваленттүү.

Жолчолук тип. Жолчолук тип 3кө бөлүнөт: ShortString, LongString, WideString.

- ShortString тиби компьютердин эсинде статикалык түрдө жайгаша турган жана узундугу 0 дон 255 символго чейин болгон жолчо болуп саналат;
- LongString тиби компьютердин эсинде динамикалык түрдө жайгаша турган жана узундугу бош эстин көлөмү менен гана чектелген жолчо болуп саналат;

- WideString тиби компьютердин эсинде динамикалык түрдө жайгаша турган жана узундугу бош эстин көлөмү менен гана чектелген жолчо болуп саналат. WideString тибиндеги ар бир символ Unicode-символ болуп саналат.

Мындан тышкары жолчолук тип үчүн String тиби да колдонулат жана ал ShortString тибине эквиваленттүү.

Логикалык тип. Логикалык тип Boolean тибине таандык болот. Ал True (истина, чын Чын) же False (ложь, жалган) болгон эки маанинин бирин гана кабыл ала алат.

§ 2.2. Өзгөрмөлөр жана туракттуулар

2.2.1. Өзгөрмөлөр

Өзгөрмө деп берилгендер сакталган эстин бөлүгүн (областын) айтабыз.

Программа өзгөрмөгө кайрыла алышы үчүн өзгөрмөнүн аты болушу керек. Өзгөрмөнүн аты үчүн латын алфавитинин тамгалары, араб цифралары жана атайын символдор колдонулат. Өзгөрмөнүн атындагы 1-символ сөзсүз тамга болуш керек. Өзгөрмөнүн атында пробелдер (боштуктар) болбош керек. Delphi тилиндеги компилятор чоң же кичине тамганы ажыратпайт. Ошондуктан Summa, summa, SUMMA сөздөрү бир эле өзгөрмөнү аныктайт.

Delphi тилинде колдонулуучу ар бир өзгөрмө алдын ала жарыяланган болушу керек. Өзгөрмөлөрдү жарыялоо төмөнкүдөй форматта болот:

```
var
Аты : тиби;
```

мында ***Аты*** - өзгөрмөнүн аты, ***тиби*** - өзгөрмөнүн тиби.

Мисалы: var: i: Integer; a,b : Real;

2.2.2. Туракттуулар (константалар)

Delphi тилинде туракттуулар эки түрдүү болот: ***кадимки*** жана ***ат коюлган (именованный)***.

1). Кадимки турактуу деп бүтүн же бөлчөк санды, өзүнчө алынган символду же символдор жолчосун, логикалык маанини түшүнөбүз.

2). Логикалык турактуулар. Логикалык айтуулар (туюнтмалар) “Чын” же “Жалган” болушу мүмкүн. “Чын” маанисине True, ал эми “Жалган” маанисине False турактуусу (константасы) туура келет.

2.2.3. Түшүндүрмөлөр (комментарийлер)

Түшүндүрмөлөр (комментарийлер) деп программанын кодун түшүндүрүү (баяндоо) үчүн жазылган, бирок компиляцияланбаган тексттин фрагментин айтабыз.

Түшүндүрмөлөр эки түрдүү болот:

- Бир жолчолуу;
- Көп жолчолуу, б.а. эки же андан көп жолчолуу.

Delphi де түшүндүрмө үчүн төмөнкү конструкциялар колдонулат:

- // – бир жолчодо гана баяндалган түшүндүрмө;
- { } же (* *) – көп жолчолуу түшүндүрмөлөр;

Мисалдар:

i:= i+1; // эсептегичтин мааниси 1 ге чоңойот

s: = IntToStr(i); { IntToStr функциясы i бүтүн санынын жолчо түрүндөгү көрүнүшкө өзгөртүп берет}

2.2.4. Типтерди өзгөртүүчү функциялар

Программаны даярдоодо типтерди өзгөртүү функциялары колдонулушу мүмкүн. Алар төмөнкүлөр: **StrToInt, IntToStr, StrToFloat, FloatToStr.**

- **StrToInt(Edit1.Text)** функциясы **Edit1** талаасындагы символду бүтүн санга айландырып берет;
- **IntToStr(n)** функциясы **n** санын символго айландырып берет;
- **StrToFloat(Edit1.Text)** функциясы **Edit1** талаасындагы символду чыныгы санга айландырып берет;
- **FloatToStr(α)** функциясы α чыныгы санын символго айландырып берет.

2.2.5. Берилгендерди калыпка салуу менен чыгаруу

Берилгендерди калыпка салуу менен чыгаруу үчүн FloatToStrF функциясы колдонулат.

FloatToStrF функциясы жылуучу чекити бар чыныгы санды жолчого калыпка салуу (форматтоо) менен өзгөртөт.

Функцияны колдонуу эрежеси төмөнкүдөй:

function FloatToStrF (r, f, k, m);

мында

r -	чыныгы сан;	
f -	формат:	ffGeneral - универсалдык; ffExponent - илимий; ffFixed – фиксирленген чекити менен; ffNumbe – разряддардын группалары ажыратылат; ffCurency – финансылык.
k	тактыкты билдирет, б.а. жалпы керектүү болгон цифралардын саны;	
m	ондук чекиттен кийинки цифралардын саны	

2.2.6. Эки сандын суммасын табуу программасы

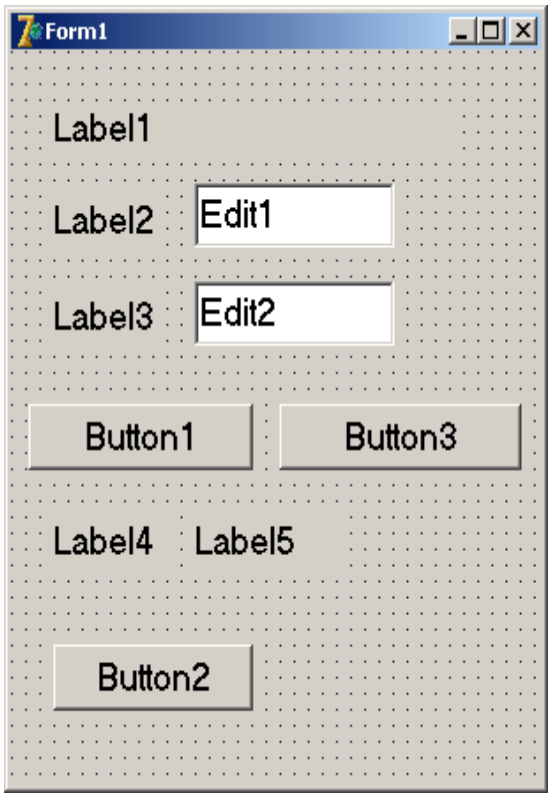
Delphi нин мүмкүнчүлүгүн жана визуалдык проектирлөө технологиясын демонстрациялоо үчүн эки сандын суммасын аныктоочу колдонмону даярдайбыз.

2.2.1-мисал. *Эки санды киргизүүнү жана алардын суммасын табууну талап кылган программа түзгүлө.*

Жаңы программаны түзүү үчүн Delphi ни жүктөйбүз. Эгерде компьютерде башка проект жүктөлгөн болсо **File** (Файл) менюсунан **New** | **Application** (Создать | Приложение) командасын тандайбыз.

Жогорудагыдай эле жаңы проектти даярдоо алгачкы форманы (диалогдук терезени) түзүүдөн башталат. Эң оболу **Form1** формасынын элементтеринин (белгилердин, киргизүү талааларынын, командалык баскычтардын) касиеттеринин маанисин өзгөртүүдөн башталат.

Формага Standard барагынын Label1, Label2, Label3, Label4, Label5, Edit1, Edit2 жана Button1, Button2, Button3 компоненттерин жайгаштырабыз (2.1-сүрөт).



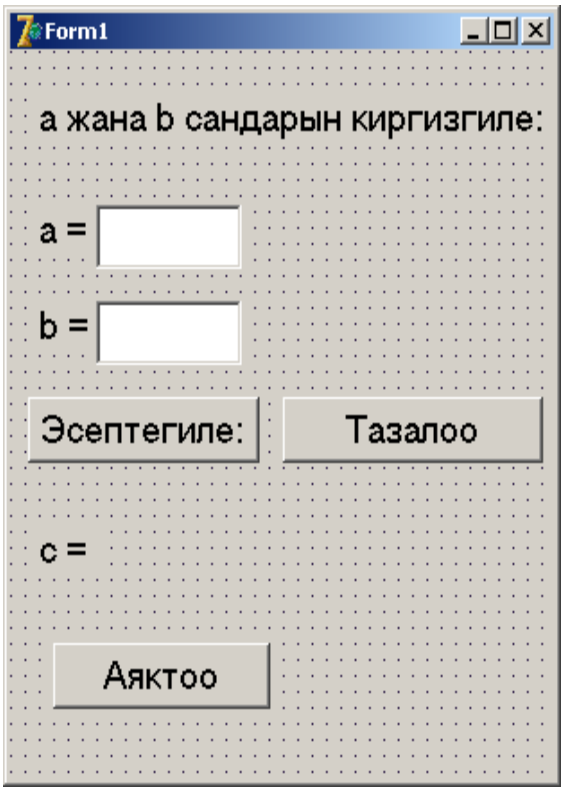
2.1-сүрөт. Эки сандын суммасы

Форманын касиеттерин өзгөртүү үчүн **Object Inspector** терезесин колдонобуз**. Object Inspector** терезесинин жогору жагында объекттин аты жана касиеттеринин маанилери көрсөтүлгөн. Анын эки барагы бар: **Properties** (Свойства) жана **Events** (События). **Properties** барагынын сол жагында объекттин касиеттери, ал эми оң жагында алардын маанилери көрсөтүлгөн. Эгерде **Events** барагына өтсөк, анда деле сол жакта объекттердин касиеттери, ал эми оң жагында алардын маанилери көрсөтүлгөн. Компоненттердин касиеттерин төмөнкү таблицадагыдай өзгөртүп чыгабыз:

2.3-таблица. Формадагы объекттер

Объекттин аталышы	Объекттин касиети	Объекттин касиетинин мааниси
Label1	Caption	a жана b сандарын киргизгиле
Label2	Caption	a=
Label3	Caption	b=
Label4	Caption	c=
Button1	Caption	Эсептегиле
Button2	Caption	Тазалоо
Button3	Caption	Аяктоо
Edit1	Caption	Edit1 жазуусун өчүрөбүз
Edit2	Caption	Edit2 жазуусун өчүрөбүз

Натыйжада биздин форма-быз төмөнкүдөй көрүнүшкө келет (2.2-сүрөт).



2.2-сүрөт. Даяр болгон форма

Edit1 жана Edit2 компоненталары киргизүү-редактирлөө компоненталары болуп эсептелет. Унчукпаганда бул талаалардын терезесинде Edit1 жана Edit2 деген сөздөр жазылган болот. Аларды өчүрүү үчүн **Properties барагынан** Text касиетин тандап, анда жазылган сөздү өчүрөбүз.

“Аяктоо” баскычына эки чыкылдатып коддор терезесине киребиз да анда жазылган программанын даяр шаблонуна төмөнкү сөздү жазабыз:

close;

Андан кийин “Эсептегиле” баскычына эки чыкылдатып коддор терезесине киребиз да төмөнкү программаны жазабыз:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  a:=StrToInt(Edit1.Text);
  b:=StrToInt(Edit2.Text);
  c:=a+b;
  Label5.Caption:=FloatToStr(c);
end;
```

“Тазалоо” баскычына эки жолу чыкылдатып, пайда болгон шаблонго төмөнкү кодду жазабыз:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
Edit1.Text:='';
Edit2.Text:='';
Label5.Caption:='';
end;
```

Программанын иштешин текшерүү үчүн F9 баскычын басабыз жана программанын иштешин текшерип көрөбүз, б.а. “Эсептегиле”, “Тазалоо” жана “Аяктоо” деген баскычтардын туура иштеп жаткандыгын текшеребиз.

Акырында программаны сактайбыз. Ал үчүн

File / Save As …

командасын беребиз. Ачылган Save терезесиндеги Имя файла деген талаага Usumma деген ат берип, “Сохранить” баскычын басабыз. Натыйжада биздин форма сакталат.

Проектти

File / Project Save As …

командасын беребиз жана Psumma аталышы менен сактайбыз. Бул учурда сакталуучу проект компиляцияланат жана exe файл даяр болот.

§ 2.3. Тизмелер

Колдонмолорду түзүүдө **тизмелер** (список) көп колдонулат. Тизмелерди түзүү жана тандоо үчүн Delphi де **ListBox** жана **ComboBox** компоненттери колдонулат. ListBox жана ComboBox компоненттери **Standard** барагында жайгашкан (2.3-сүрөт).

2.3-сүрөт. Standard барагы

ListBox компоненти тизмени чагылдырат жана колдонуучуга керектүү жолчону тандап алууга мүмкүнчүлүк берет. Ал эми **ComboBox** мындан тышкары берилгендерди редакциялоого (оңдоп-түзөөгө) да мүмкүнчүлүк берет.

ListBox компонентинде бир нече тизмени тандап алса болот. Ал эми **ComboBox** компонентинде бир гана жолчону гана тандап алууга мүмкүнчүлүк бар.

2.3.1. ListBox компоненти

ListBox компонентти тексттик информацияны түзүү, сактоо жана кайра иштетүү үчүн колдонулат. **Ал Standard** барагында

жайгашкан.

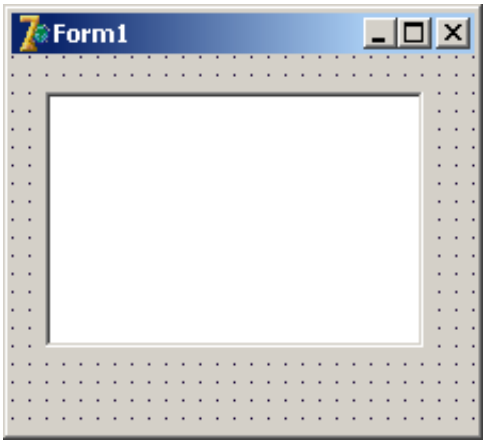
ListBox компоненти тизменин бардык элементтерин чагылдырат. Эгерде бардык тизме компоненттин терезесине батпаса, анда терезеге жылдыруу тилкеси (полоса прокрутки) автоматтык түрдө кошулат.

ListBox компонентиндеги ар бир жолчого **String** тибиндеги тексттик информацияны сактоого болот. ListBox компонентине – жолчолордун массивин жайгаштырууга болот. Бул компоненттин касиеттери 2.3-таблица көрсөтүлгөн.

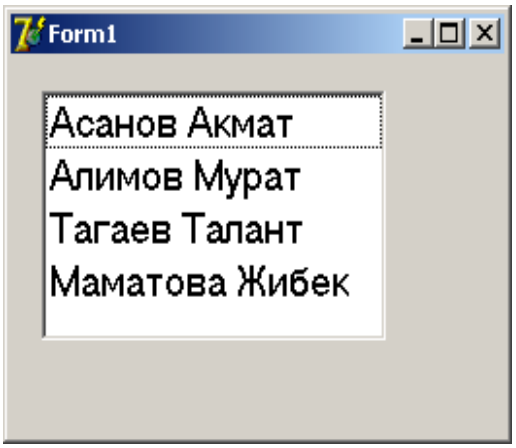
2.3-таблица. ListBox компонентинин касиеттери

Касиети	**Колдонулушу**
Name	Компоненттин аты. Компонентти жана анын касиеттерин тандоо үчүн колдонулат.
Items	Тизменин элементтери.
ItemIndex	Тизменин тандалган элементинин номери. Эгерде эч кандай элемент тандалбаган болсо, анда ItemIndex=-1 болот.
Font	Тизменин элементтерин чагылдыруучу шрифтти аныктайт.

ListBox компонентинин конструкциялоо этабындагы көрүнүшү 2.4-сүрөттө көрсөтүлгөн.



2.4-сүрөт. ListBox компоненти



2.5-сүрөт. ListBox компонентиндеги тизмелер

Жолчолорго 2.3-таблицада көрсөтүлгөндөй компоненттин **Items** касиети аркылуу жетүүгө болот.

Мисалы, ListBox1 компонентасын формага жайгаштырып, анын талаасына тизмени киргизүү үчүн Object Inspector терезесинин **Items** касиетин тандап, андагы үч чекиттүү баскычка чычкандын көрсөткүчүн чыкылдат-канда String List Editor терезеси ачылат. Ага тизменин ар бир элементин өзүнчө жолчого жайгаштырабыз. Ал үчүн ар бир элементти киргизгенден кийин Enter баскычын басабыз. Акыркы элемент киргизилгенден кийин Enter баскычын баспастан туруп ОК баскычын басабыз. Натыйжада тизменин элементтери ListBox1 компонентинин терезесине жазылып калат (2.5-сүрөт).

Компоненттин биринчи жолчосунун номери 0 гө барабар болгондуктан і номерлүү жолчого

ListBox1.Items[i-1];

командасы менен кайрылууга болот.

ItemIndex касиети тизмедеги тандалган жолчонун номерин аныктайт. Мисалы тизменин үчүнчү элементи тандалган болсо, анда ItemIndex=2, ал эми эч кандай элемент тандалбаган болсо, анда ItemIndex=-1 болот.

2.3.2. ListBox: жолчолорду кошуу, өчүрүү жана сорттоо

ListBox компоненти дисктен маалыматтарды жүктөй алат жана информацияларды файлга сактай алат. ListBox компонентине жолчолорду кошуу алгоритми төмөнкү кадамдардан турат:

- Файлдан окуу;
- Тизменин акырына жолчону кошуу;
- Номери і болгон жолчонун алдына жолчо кошуу;
- Түзүү (конструкциялоо) этабында жолчолорду кошуу.

ListBox компонентинде тизмелер менен иштөө жолдору (жолчолорду кошуу, өчүрүү жана сорттоо) 2.4-таблицада келтирилди.

2.4-таблица. Тизмелер менен иштөө

Кадамдар	Аткарылуучу аракеттер

Файлдан окуу жана файлга жазуу	// Файлдан окуу процедурасы
	ListBox1.Items.LoadFromFile('Файлдын аты');
	// Файлга жазуу процедурасы
	ListBox1.Items.SaveToFile('Файлдын аты');

Тизменин акырына жолчолорду кошуу	// Тизменин акырына жолчолорду кошуу.
	ListBox1.Items.Add('Жаңы жолчо');
	{Компоненттеги жолчолор саны **ListBox1.Items.Count** процедурасы менен аныкталгандыктан акыркы жолчонун номери **ListBox1.Items.Count-1** болот, анткени номерлөө 0 дөн башталат}

Номери i болгон жолчонун алдына жолчо кошуу	ListBox1.Items.Insert(i, 'Жаңы жолчо'); // Жаңы жолчонун номери **i** болот.
Түзүү (конструкциялоо) этабында жолчолорду кошуу	Объекттер инспекторуна кирип **Items** касиетин тандап, үч чекиттүү баскычка чыкылдатуу керек. Натыйжада String List Editor редакторунун терезеси ачылат. Керектүү маалыматтарди киргизгенден кийин OK баскычын бассак, формадагы ListBox компонентасынын терезесине тиешелүү маалыматтар жазылып калат.
Бардык тизмени тазалоо	ListBox1.Items.Clear;

Тизмеден элементти өчүрүү	// Тизмедеги экинчи элемент өчүрүү
	ListBox1.Items.Delete(1);

Тизмени сорттоо	ListBox1.Sorted := True;

2.3.3. ComboBox компоненти

ComboBox компонентти тексттик информацияны түзүү, сактоо жана кайра иштетүү үчүн колдонулат. **Ал Standard** барагында жайгашкан.

ComboBox компоненти тизмени чагылдырат, колдонуучуга керектүү жолчону тандап алууга жана берилгендерди редакциялоого (оңдоп-түзөөгө) мүмкүнчүлүк берет.

ComboBox компоненти тизмени түшүүчү (выпадающая) тизме катары чагылтат, ошондой эле бардык тизмени да чагылтууга болот.

ComboBox компоненти бир жолчолуу редактирлөө талаасы болгон Edit жана тизмелерди түзүү үчүн дайындалгани ListBox компоненттеринин комбинациясы болуп эсептелет. Сырттан караганда ComboBox компоненти Edit талаасына окшош болот, бирок талаанын оң жагында жебеси (стрелкасы) бар баскычы бар. Эгерде бул баскычка чычканчанын сол баскычы менен чыкылдатсак, анда ListBox компонентинин тизмелерине окшогон жогортон төмөн түшүүчү тизме пайда болот.

ComboBox компонентинин кийирүү жолчосун пайдаланып тизмеге жаңы элементтерди кошууга, тизмедеги керектүү элементти издөөгө жана тизменин активдүү элементин чагылтууга болот.

ComboBox компонентинин касиеттери Edit жана ListBox компоненттеринин касиеттеринин комбинациясынан турат. Редактирлөө жолчосу менен иштөө үчүн **Text** касиети, ал эми тизмелер менен иштөө үчүн **Items** касиети колдонулат.

Тизмедеги тандалган элементтин номерин **ItemIndex** касиети менен аныктай алабыз. Унчукпаган учурда ItemIndex = -1 деп эсептелет. Номерлөө 0 дөн башталат. Эгерде эч бир жолчо тандалбаган болсо, анда ItemIndex = -1 деп алабыз.

ComboBox компонентинин тизменин элементтерин иштетүү үчүн дайындалган элементтерди кошуу, өчүрүү жана издөө операциялары ListBox компонентинин тизмелерине окшош ишке ашырылат.

ComboBox компонентинин негизги касиеттери 2.5-таблицада берилди.

2.5-таблица. ComboBox компонентинин касиеттери

Касиети	**Колдонулушу**
Action	Берилген компонент менен байланышкан аракетти билдирет
Align	Компонентти контейнерде түздөө режимин аныктайт

DropDownCount	Жылдыруу тилкеси чыкпаган учурдагы жайылган тизменин элементтеринин максималдык санын аныктайт
ItemHeight	Түшүүчү тизмедеги элементтердин бийиктигин пиксел менен аныктайт
ItemIndex	Тизмеде тандалган элементтин номерин аныктайт
Items	Тизмедеги жолчолордун массивин аныктайт
SelLength	Редактирлөө терезесинде тандалган символдордун санын аныктайт
SelText	Редактирлөө терезесинде тандалган текстти аныктайт
Sorted	Тизменин автоматтык тартипте алфавит боюнча сорттоло тургандыгын билдирет

2.3.4. ComboBox: негизги методдор жана окуялар

1). **Add(const s: string):integer** – тизменин акырына s жолчосун кошот жана анын номерин кайтарып берет.

Text касиети менен жолчону кошуу үчүн төмөнкү кодду пайдаланса болот:

```
if ComboBox1.ItemIndex = –1 then
ComboBox1.Items.Add(ComboBox1.Text);
```

2). **Delete(n: integer)** – n номерлүү жолчону өчүрөт.

3). **DeleteSelected** – тандалган жолчону өчүрөт.

Мисалы: ComboBox1.DeleteSelected; тандалган жолчону өчүрөт.

4). **Insert(n: integer; const s: string)** – s жолчосун n чи позицияга койот

5). **Exchange(n1, n2: integer)** – n1 жана n2 номерлүү жолчолорду алмаштырат.

6). **Move(n1, n2: integer)** – n1 номерлүү жолчону n2 позициясына жылдырат.

7). **IndexOf(const s: string):integer** – эгерде тизмесе s жолчосу бар болсо, анада ал жолчонун номерин кайтарат, эгерде мындай жолчо жок болсо, анда -1 маанисин кайтарат.

8). **LoadFromFile(const FileName: string)** – аталышы FileName болгон файлдагы тизмелерди жүктөйт.

9). **SaveToFile(const FileName: string)** – тизмени аталышы FileName болгон файлга сактайт.

10). **Clear** – ComboBox компонентинин ичиндегилерди тазалайт. Мисал: ComboBox1.Clear;

11). **SetFocus** – элементке фокусту берүү жана аны акивация кылуу.

ComboBox компоненти үчүн төмөнкү окуяларды моделдөөгө болот:

OnClick – чычкандын сол баскычын басуу;
onDropDown – тизмени ачуу;
onCloseUp – тизмени жабуу;
OnSelect – элементти тандоо;
OnChange – редактирлөө талаасында текстти өзгөртүү.

§ 2.4. Диалогдук терезелер

Программа иштеп жатканда берилгендерди кийирүүгө жана жыйынтыктарды чыгарууга туура келет. Мындай маселелерди чечүү үчүн Delphi де ***кийирүү*** жана ***билдирүү*** терезелери колдонулат.

2.4.1. Кийирүү терезеси

Берилгендерди кийирүү үчүн стандарттык диалогдук терезе – **кийирүү терезеси** колдонулат. Кийирүү терезеси **InputBox** функциясы менен чакырылат. InputBox функциясынын мааниси колдонуучу тарабынан кийирүү талаасына жазылган жолчого барабар болуп эсептелет.

Жалпы учурда InputBox функциясын чакыруу инструкциясы төмөнкүдөй болот:

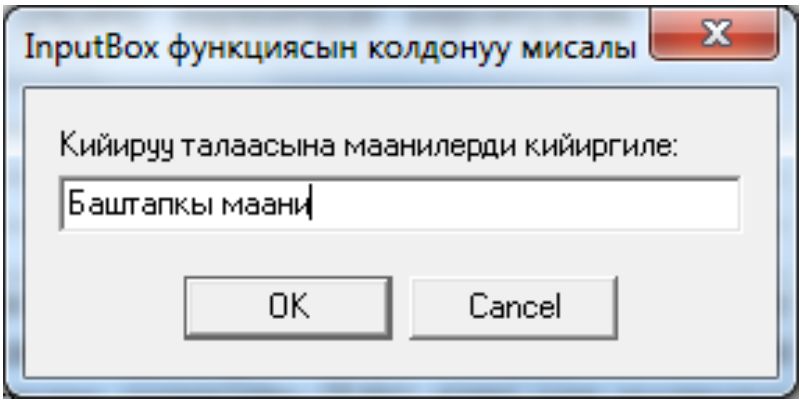
```
var s:string;
begin
    s:=InputBox('Бөрк', 'Түшүндүрмө', 'Маани');
end;
```

мында

- **s** – мааниси кийирүү талаасына жазылган текстке барабар;
- **Бөрк** – кийирүү терезесинин бөркүндөгү текст;
- **Түшүндүрмө** – билдирүүнү түшүндүрүүчү текст;
- **Маани** – экранда пайда болгон терезенин кийирүү талаасындагы текст.

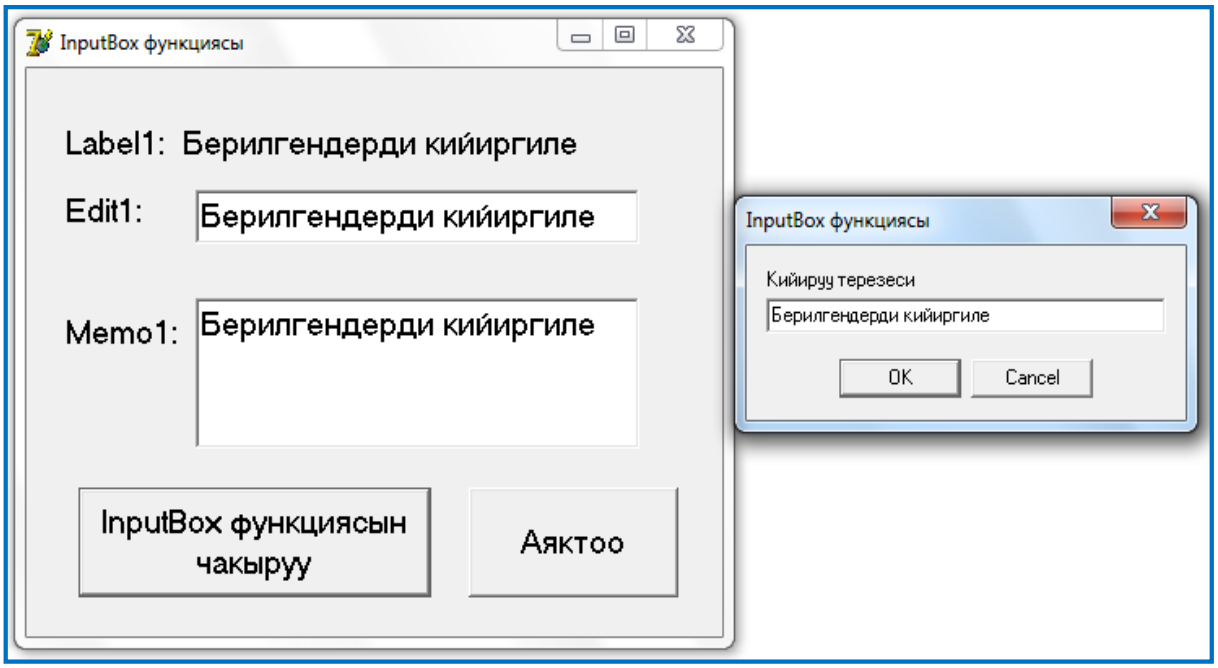
Эгерде колдонуучу программа иштеп жатканда кийирүү талаасына тектти кийирсе жана OK баскычын басса, анда InputBox функциясынын мааниси кийирилген текстке барабар болот.

Эгерде колдонуучу кийирүү талаасына эч кандай маалымат кийирбестен эле **Cancel** баскычын басса, анда InputBox функциясынын мааниси үчүнчү параметрде көрсөтүлгөн текстке барабар болот (2.6-сүрөт).



2.6-сүрөт. Кийирүү терезеси

2.4.1-мисал. InputBox диалогдук терезесинин кийирүү талаасынан белгилер талаасына (Label), редактирлөө талаасына (Edit) жана көп жолчолуу талаага (Memo) маалыматтарды кийиргиле (2.7-сүрөт).



2.7-сүрөт. InputBox функциясы менен кийирүү терезеси

2.1-листинг. InputBox функциясы жана Memo

```
procedure TForm1.Button1Click(Sender: TObject);
var s:string;
begin
        s:=InputBox('InputBox функциясы','Кийирүү терезеси',
                        'Берилгендерди кийиргиле');
        Label2.Caption:=s;
        Edit1.Text:=s;
        Memo1.Clear;
        Memo1.Lines.Add(s);
end;
```

2.4.2. Билдирүү терезесине чыгаруу

Программа аткарылып жатканда жыйынтыктарды билдирүү терезесине чыгарууга болот. Берилгендерди чыгаруу үчүн стандарттык диалогдук терезелерди – билдирүү терезелери колдонулат.

Билдирүү терезелери колдонуучунун көңүлүн буруу зарыл болгондо, мисалы, экранга каталыктарды чыгаруу же кайтарууга болбой турган операцияларды аткаруу сыяктуу билдирүүлөрдү чыгаруу үчүн колдонулат.

Билдирүү терезелери төмөнкү каражаттар менен чыгарылат:

- **ShowMessage процедурасы;**
- **MessageDlg функциясы.**

2.4.3. ShowMessage процедурасы

ShowMessage процедурасы ОК деген баскычы бар жана берилген текст жазылган жөнөкөй диалогдук терезени экранга чыгарат. Процедуранын структурасы төмөнкүдөй:

```
ShowMessage('Билдирүү');
```

мында 'Билдирүү' – ShowMessage терезесине чыгарылуучу текст.

Көп жолчолуу билдирүүнү чыгаруу үчүн chr(13) функциясын колдонууга болот. Бул функция Enter баскычын имитациялайт.

2.4.4. MessageDlg функциясы

MessageDlg функциясы универсалдык билдирүү терезесин аныктайт. **MessageDlg** функциясы экранга билдирүүнү, белгилерди жана тандалган баскычтарды чыгарат.

MessageDlg функциясынын 4 аргументи бар, ал эми анын мааниси **integer** тибинде болот.

Функциянын структурасы төмөнкүдөй:

MessageDlg('Билдирүү',DialogType, Buttons, HelpContext)

мында:

1). 'Билдирүү' - билдирүү тексти, ал жалгыз кавычканын ичине жазылат;

2). DialogType - билдирүүнүн тиби (информациялык, эскертүүчү, критикалык каталык ж.б.). Ар бир типке анык бир значок туура келет. Билдирүүнүн тиби аталышы бар константалар менен берилет (2.6-таблица);

3). Buttons - билдирүү терезесине чыгарылуучу баскычтардын тизмесин аныктайт. Тизме үтүрлөр менен ажыратылган аталышы бар константалардан турушу мүмкүн (2.7-таблица). Жалпы тизме квадраттык кашаанын ичине жазылышы керек.

4). HelpContext - колдонуучу <F1> баскычын басканда экранга справкалык системанын бөлүмүн чыгаруучу параметрди аныктайт. Эгерде мындай справклык система каралбаган болсо, анда бул параметрдин мааниси 0 болушу керек.

MessageDlg функциясын программада колдонуу үчүн төмөнкү форматты пайдаланса болот:

```
var m:integer;
begin
   m:=MessageDlg('Билдирүү',DialogType,
                  Buttons, HelpContext);
end;
```

2.6-таблица. MessageDlg функциясынын константалары

Константа	Билдирүү тиби	Значок
mtWarning	Илеп белгиси символу (Көңүл бургула)	
mtError	Кызыл "x" символу (Каталык бар)	
mtInformation	"i" символу (Информация үчүн)	
mtConfirmation	Суроо белгиси (Тактоо зарыл)	
mtCustom	Билдирүү гана чыгарылат	Значок жок

2.7-таблица. MessageDlg функциясынын баскычтарынын константалары

№	Баскыч	Константа
1	mbYes	"Yes" баскычын чыгарат
2	mbNo	"No" баскычын чыгарат
3	mbOK	"OK" баскычын чыгарат
4	mbCancel	"Cancel" баскычын чыгарат
5	mbHelp	"Help" баскычын чыгарат
6	mbAbort	"Abort" баскычын чыгарат
7	mbRetry	"Retry" баскычын чыгарат
8	mblgnore	"Ignore" баскычын чыгарат
9	mbAll	"All" баскычын чыгарат

Мисалы, билдирүү терезесине **ОК** жана **Cancel** баскычтарын чыгаруу үчүн **Buttons** тизмеси төмөнкүдөй болушу керек: [mbOK, mbCancel].

Жогоруда келтирилген константалардан башка төмөнкү константаларды да колдонууга болот:

mbYesNoCancel = [mbYes, mbNO, mbCancel]

mbYesAllNoAllCancel = [mbYes, mbYesToAll, mbNo, mbNoToAll, mbCancel]

mbOKCancel = [mbOK, mbCancel]

mbAbortRetryCancel = [mbAbort, mbRetry, mbCancel]

mbAbortIgnore = [mbAbort, mbIgnore]

Бул константалар диалогдук терезелерде командалык баскычтардын комбинациясы катарында көп колдонулат.

MessageDig функциясы кайтарып берген маани колдонуучу тарабынан кайсыл константа басылганын аныктап берет (2.8-таблица).

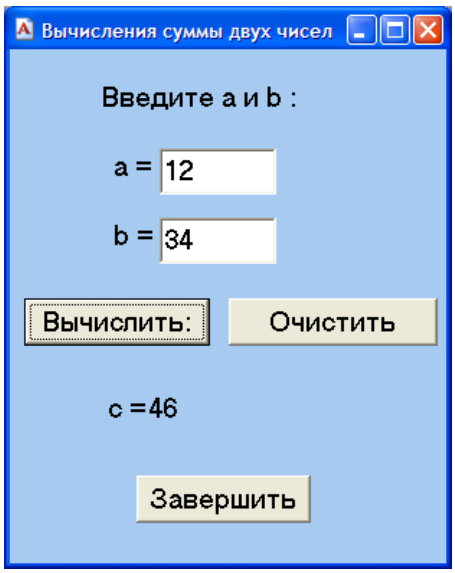
2.8-таблица. MessageDlg функциясынын маанилери

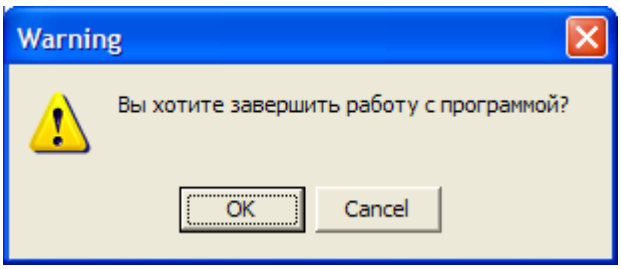
MessageDlg функциясынын маанилери		Басылган баскычтын аталышы
mrOk	1	Ok
mrCancel	2	Cancel
mrAbort	3	Abort
mrRetry	4	Retry
mrIgnore	5	Ignore
mrYes	6	Yes
mrNo	7	No
mrAll	8	All

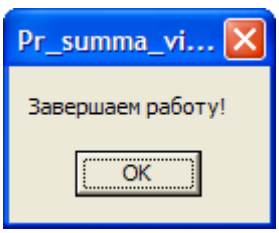
2.1-мисал. Ишти аяктоодо колдонулуучу диалогдордун программасын түзгүлө

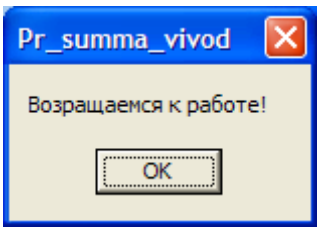
```
unit Un_summa_vivod;
var     Form1: TForm1;
        a,b,c:real;
        ms:integer;
implementation
        {$R *.dfm}
procedure TForm1.Button2Click(Sender: TObject);
```

```
begin
ms:=MessageDlg('Сиз программа менен иштөөнү аяктоону каалайсызбы?', mtWarning, mbOKCancel, 0);
if ms=mrOK then
        begin
           ShowMessage('Ишти аяктап жатабыз!');
        close;
        end;
        if ms=mrCancel then
        begin
           ShowMessage('Иштөөгө кайтып келдик!');
        end;
        end;

procedure TForm1.Button1Click(Sender: TObject);
        begin
          a:=StrToInt(Edit1.Text);
          b:=StrToInt(Edit2.Text);
          c:=a+b;
          Label5.Caption:=FloatToStr(c);
        end;

procedure TForm1.Button3Click(Sender: TObject);
        begin
           Edit1.Text:='';
           Edit2.Text:='';
           Label5.Caption:='';
        end;
```









3-глава. Delphi тилиндеги башкаруучу конструкциялар

§ 3.1. Бутактуу алгоритмдерди программалоо

Бутактануучу процесс деп берилген шарттын аткарылышына карап программанын бул же тигил бутагы аткарыла турган процесстерди айтабыз.

Мындай процесстерди программалоо учун шарттуу «if – then» жана тандоо «case – of» конструкциялары колдонулат.

3.1.1. Шарттуу конструкция

Бутактануучу процесстер белгилүү бир шарттардын аткарылышына, б.а. ал шарттардын “чын” же “жалган” экендигине байланыштуу аткарылат. Бутактануучу процесстерди программалоо үчүн шарттуу конструкциялар колдонулат.

Шарттуу конструкциялардын үч түрү бар:

- *Кыскартылган;*
- *Стандарттык;*
- *Толук.*

Бул конструкциялардын ар бирине токтолуп кетебиз.

1) Кыскартылган шарттуу конструкция

Бул конструкция төмөнкүдөй иштейт: эгерде if кызматчы сөзүнөн кийинки <шарт> «чын» болсо, анда then кызматчы сөзүнөн кийинки «инструкция» аткарылат, ал эми <шарт> «жалган» болгон учурда эч нерсе аткарылбайт.

```
if <шарт> then
begin
   <инструкция>
end;
```

2) Шарттуу конструкциянын стандарттык формасы

Бул учурда конструкция төмөнкүдөй иштейт: эгерде if кызматчы сөзүнөн кийинки <шарт> «чын» болсо, анда then кызматчы сөзүнөн кийинки «1-инструкция» аткарылат, эгерде ал шарт «жалган» болсо, анда else кызматчы сөзүнөн кийинки «2-инструкция» аткарылат.

```
if <шарт> then
begin
    <1-инструкция>
end
else
begin
    <2-инструкция>
end;
```

3) Шарттуу конструкциянын толук формасы

Жалпы учурда конструкция төмөнкүдөй иштейт: эгерде if кызматчы сөзүнөн кийинки <1-шарт> «чын» болсо, анда then кызматчы сөзүнөн кийинки «1-инструкция» аткарылат, эгерде <1-шарт> «жалган» болсо, анда else if кызматчы сөздөрүнөн кийинки <2-шарт> текшерилет. Эгерде <2-шарт> «чын» болсо, анда then кызматчы сөзүнөн кийинки «2-инструкция» аткарылат. Ушул процесс уланып акырында <n-шарт> текшерилет. Эгерде <n-шарт> «чын» болсо, анда then кызматчы сөзүнөн кийинки «n-инструкция» аткарылат, тескери учурда else кызматчы сөзүнөн кийинки <(n+1)-инструкция> аткарылат.

```
if <1-шарт> then
    begin <1-инструкция> end
else if <2-шарт> then
    begin <2-инструкция> end
else if <3-шарт> then
    begin <3-инструкция> end
……….……
else if <n-шарт> then
begin <n-инструкция> end
else
   begin <(n+1)-инструкция>
end;
```

3.1-мисал. *Эки сандын максимумун табуу программасын түзгүлө.*

Тапшырманы аткаруу үчүн формага Label1, Label2, Label3, Edit1, Edit2, Button1, BitBtn1 компоненталарын жайгаштырабыз

3.1.2. Тандоо конструкциясы

Эгерде шарттардын саны эки же андан көп болсо, анда «case–of» тандоо конструкциясы колдонулат. Анын форматы төмөнкүдөй:

```
case «Селектор» of
    К1: begin {1-инструкция} end;
    К2: begin {2-инструкция} end;
    ……
    КN: begin {N-инструкция} end;
    else begin {N+1-инструкция} end;
end;
```

мында «Селектор» - мааниси саналуучу тип боло турган туюнтма, KN - саналуучу типтин константасы. Тандоо конструкциясынын иштөө принциби төмөнкүдөй. Эң оболу Селектордун мааниси аныкталат, андан кийин ал маани саналуучу типтин константалары менен салыштырылат. Эгерде Селектордун мааниси KI мааниси менен дал келсе, анда бул мааниге туура келген I-инструкция аткарылат. Эгерде Селектордун мааниси KI маанилеринин эч бири менен дал келбесе, анда N+1 инструкция аткарылат да, тандоо конструкциясы өз ишин аяктайт.

Ctrl+J командасы менен пайда болгон динамикалык терезеден **case statement** командасын тандасак «case-of» инструкциясынын төмөнкү калыбын алабыз:

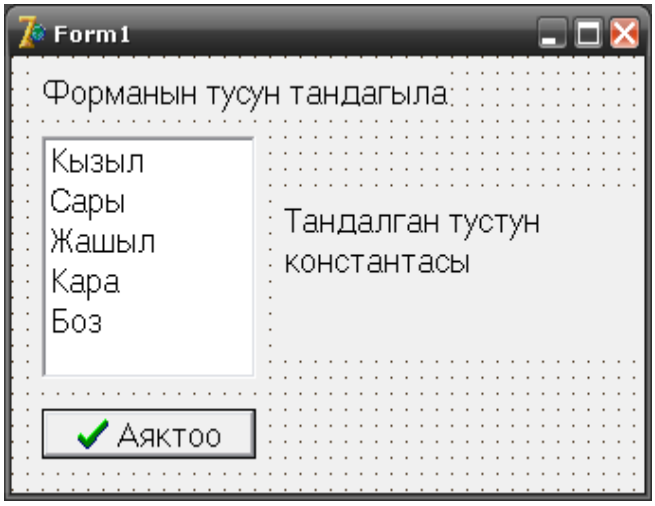
```
case          of
      :       ;
      :       ;
end;
```

“case-of” конструкциясынын негизги параметри катары **ListBox1.ItemIndex** өзгөрмөсүн алабыз жана 3.1-листингде көрсөтүлгөн программалык коддорду жазабыз.

3.2-мисал. *Формадагы терезеде көрсөтүлгөн түстөрдү тандаганда форманын түсү өзгөрө турган программа түзгүлө*.

Тапшырманы аткаруу үчүн форманын моделин 3.1-сүрөттө көрсөтүлгөндөй кылып түзөбүз, б.а. формага Label1, Label2, ListBox1, BitBtn1 компоненталарын жайгаштырабыз.

Label1 компонентинин Caption касиетине “Форманын түсүн тандагыла”, ал эми Label2 компонентасынын Caption касиетине “Тандалган түстүн константасы” деген сөздөрдү жазабыз.



3.1-сүрөт. Түстү тандоо программасы

ListBox1 компонентинин Object Inspector терезесиндеги Items касиетин тандап, андагы үч чекиттүү баскычка чычкандын сол көрсөткүчүн чыкылдатып String List Editor терезесин ачабыз. Пайда болгон терезеге “Кызыл”, “Сары”, “Жашыл”, “Кара”, “Боз” деген сөздөрдү жазып Enter баскычын басабыз. Натыйжада ListBox1 компонентасынын терезесине жогорудагы түстөрдүн тизмеси жазылып калат. BitBtn1 компонентасынын Kind касиетиндеги bkOK баскычын тандайбыз жана Caption касиетине “Аяктоо” деген сөздү жазабыз (3.1-сүрөт).

Андан кийин формадагы ListBox1 компонентинин терезесине эки чыкылдатып коддордун терезесиндеги төмөнкү процедуранын форматын алабыз (3.1-листинг):

3.1-листинг. Форманын түсүн өзгөртүү программасы

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
case ListBox1.ItemIndex of
0  :
  begin
    Form1.Color:=clRed;
    Label1.Caption:='Сиз кызыл түстү тандадыныз';
    Label2.Caption:='Кызыл түстүн константасы '+' clRed';
  end;
1  :
  begin
    Form1.Color:=clYellow;
    Label1.Caption:='Сиз сары түстү тандадыныз';
    Label2.Caption:='Сары түстүн константасы '+' clYellow';
```

```
  end;
2  :
  begin
    Form1.Color:=clAqua;
    Label1.Caption:='Сиз жашыл түстү тандадыныз';
    Label2.Caption:='Жашыл түстүн константасы'+' clAqua';
  end;
3  :
  begin
    Form1.Color:=clBlack;
    Label1.Caption:='Сиз кара түстү тандадыныз';
    Label2.Caption:='Кара түстүн константасы'+' clBlack';
  end;
4  :
  begin
    Form1.Color:=clWhite;
    Label1.Caption:='Сиз ак түстү тандадыныз';
    Label2.Caption:='Ак түстүн константасы'+' clWhite';
  end;
end;
end;
```

§ 3.2. Циклдер, алардын түрлөрү жана колдонулушу

Эгерде маселени чечүү алгоритминде инструкциялардын тобу бир нече жолу кайталанса, анда мындай алгоритмдерди ***циклдик*** деп атайбыз.

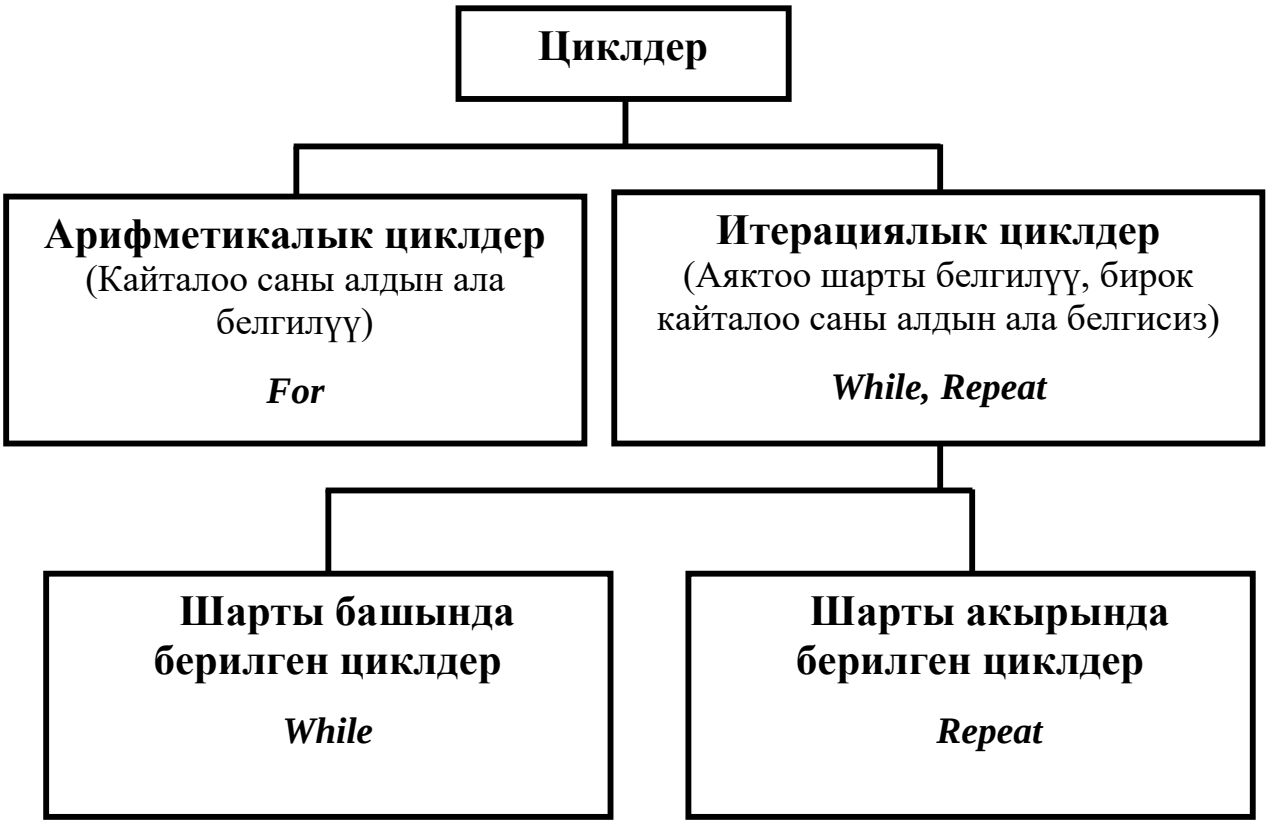
Кайталоо саны алдын ала белгилүү болгон циклдерди ***арифметикалык циклдер*** деп айтабыз.

Эгерде циклди аяктоо шарты гана белгилүү болуп, кайталоо саны алдын ала белгилүү болбосо, анда мындай циклдерди ***итерациялык циклдер*** деп айтабыз.

Delphi тилинде циклдик алгоритмдер боюнча программа түзүүдө төмөнкү инструкциялар колдонулат:

- ***For;***
- ***While;***
- ***Repeat.***

Циклдердин классификациясы



3.1-схема. Циклдердин классификациясы

3.2.1. For конструкциясы

For конструкциясы кайталануучу аракеттердин саны алдын ала белгилүү болгон учурда колдонулат.

Жалпы учурда ***For*** конструкциясынын формасы төмөнкүдөй болот (3.2-схема):

```
for i = n1 to n2 do
begin
      <Кайталануучу инструкциялар>;
end;
```

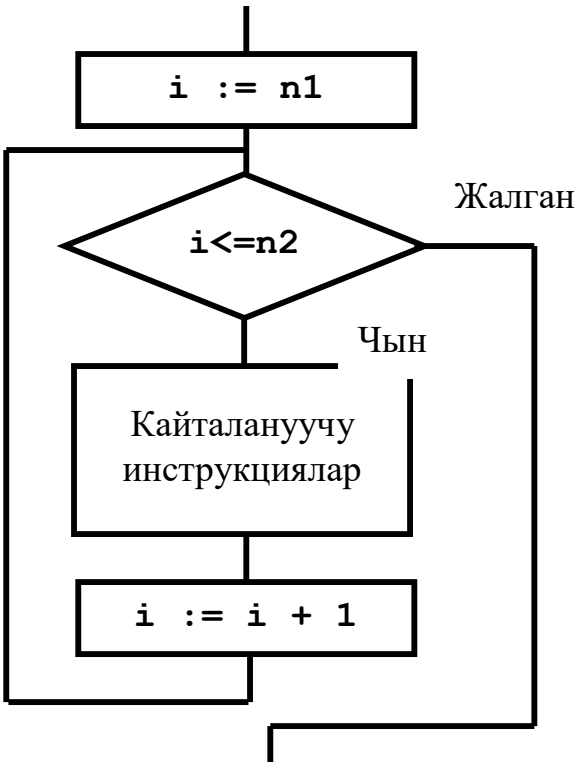
3.2-схема. **For** инструкциясы

мында

- *i* – өзгөрмө-эсептегич, цикл конструкциясынын кайталоо санын аныктайт;
- *n1* – циклдин эсептегичинин баштапкы маанисин аныктайт;
- *n2* – циклдин эсептегичи-нин акыркы маанисин аныктайт;

For инструкциясын колдонгондо өзгөрмө-эсептегич *i* жана *n1*, *n2* маанилери бүтүн типте болушу керек.

Циклдин инструкцияларынын кайталануу санын $n = (n2 - n1) + 1$ формуласы менен аныктоого болот.



3.3-схема. For инструкциясынын блок-схемасы

For инструкциясы төмөн-күдөй аткарылат. Эң оболу *i* өзгөрмө-эсептегичке *n1* маани-си ыйгарылат. Эгерде эсеп-тегичтин мааниси *n2* ден кичине же ага барабар болсо, анда циклдин кайталануучу инструкциялары аткарылат. Андан кийин эсептегичтин маанисине 1 саны кошулат. Эгерде эсептегичтин жаңы мааниси *n2* ден кичине же ага барабар болсо, анда циклдин кайталануучу инструкциялары кайрадан аткарылат. Ошентип, циклдин кайталануучу инструкциялары эсептегичтин мааниси *n2* ден ашып кеткенге чейин аткарылат.

Эгерде $i > n2$ болсо, анда цикл өз ишин аяктайт. Бул учурда башкаруу циклден кийинки турган инструкцияга өткөрүлүп берилет.

For конструкциясынын блок-схемасы 3.3-схемада келтирилген.

3.3-мисал. *1 ден N ге чейинки натуралдык сандардын суммасын тапкыла.*

Чыгаруу. Функциянын маанилерин ***For*** конструкциясын колдонуу менен төмөнкүдөй аныктайбыз. Программанын аткарылышы 3.2-сүрөттө көрсөтүлдү.

Эгерде n1>n2 болсо, анда **to** кызматчы сөзүнүн ордуна **downto** сөзү жазылат да**,** циклдин телосундагы инструкция аткарылгандан кийин эсептегичтин мааниси кичирейет.

```
for i = n1 downto n2 do
begin
      <Кайталануучу инструкциялар>
end;
```

3.4-схема. **For** инструкциясы

3.2.2. While конструкциясы

While конструкциясы циклдеги кайталоо саны алдын ала белгисиз болгон учурда колдонулат. Функциянын же туюнтманын маанисин берилген тактыкта эсептөө, массивдеги же файлдагы берилгендерди издөө учурунда кайталоо саны алдын ала белгисиз болгондуктан while конструкциясын колдонууга болот.

Жалпы учурда while конструкциясы төмөнкүдөй жазылат:

```
while <Шарт> do
begin
      <Инструкциялар>
end;
```

мында <Шарт> - циклдин инструкцияларынын аткарылышын аныктоочу логикалык типтеги туюнтма.

While конструкциясы төмөнкүдөй тартипте аткарылат:

- Берилген шарттагы туюнтманын мааниси аныкталат;
- Эгерде туюнтманын мааниси False (шарт аткарылбаса) болсо, анда while конструкциясы өз ишин аяктайт;
- Эгерде туюнтманын мааниси True (шарт аткарылса) болсо, анда begin жана end кызматчы сөздөрүнүн ортосундагы

инструкциялар (циклдин телосу), б.а. 1-итерация аткарылат. 1-итерация учурунда эсептегичтин маанисин чоңойтүү процедурасы да аткарылышы керек.

- 1-итерациядан кийин кайрадан шарттагы туюнтманын мааниси текшерилип, шарттын чын же жалган экендиги аныкталат. Шарт жалган болсо, циклдин иши аяктайт. Шарт чын болсо, 2-итерация аткарылат.
- Ошентип, бул процесс шарт жалган болгонго чейин кайталана берет.

3.4-мисал. Экранга 1 ден 20 га чейинки сандардын квадратын чыгаргыла.

While конструкциясын колдонобуз.

```
i:= 0;
while i <20 do
        begin
                i:=i+1;
                writeln(i,' ',i*i:3);
        end;
```

3.2.3. Repeat конструкциясы

Repeat конструкциясынын форматы төмөнкүдөй жазылат:

repeat
<Инструкциялар>;
unti1 <Шарт>**;**

мында <Шарт> - циклдин инструкцияларынын аткарылышын аныктоочу логикалык типтеги туюнтма.

repeat конструкциясынын иштөө принциби төмөнкү алгоритм боюнча жүргүзүлөт:

1. Эң оболу repeat жана until кызматчы сөздөрүнүн ортосундагы циклдин телосунун инструкциялары аткарылат;

2. Андан кийин шарттагы туюнтманын маниси эсептелет. Эгерде шарт жалган (False) болсо, анда телонун инструкциялары дагы бир жолу аткарылат;

3. Эгерде шарт чын (True) болсо, анда циклдин иши аяктайт;

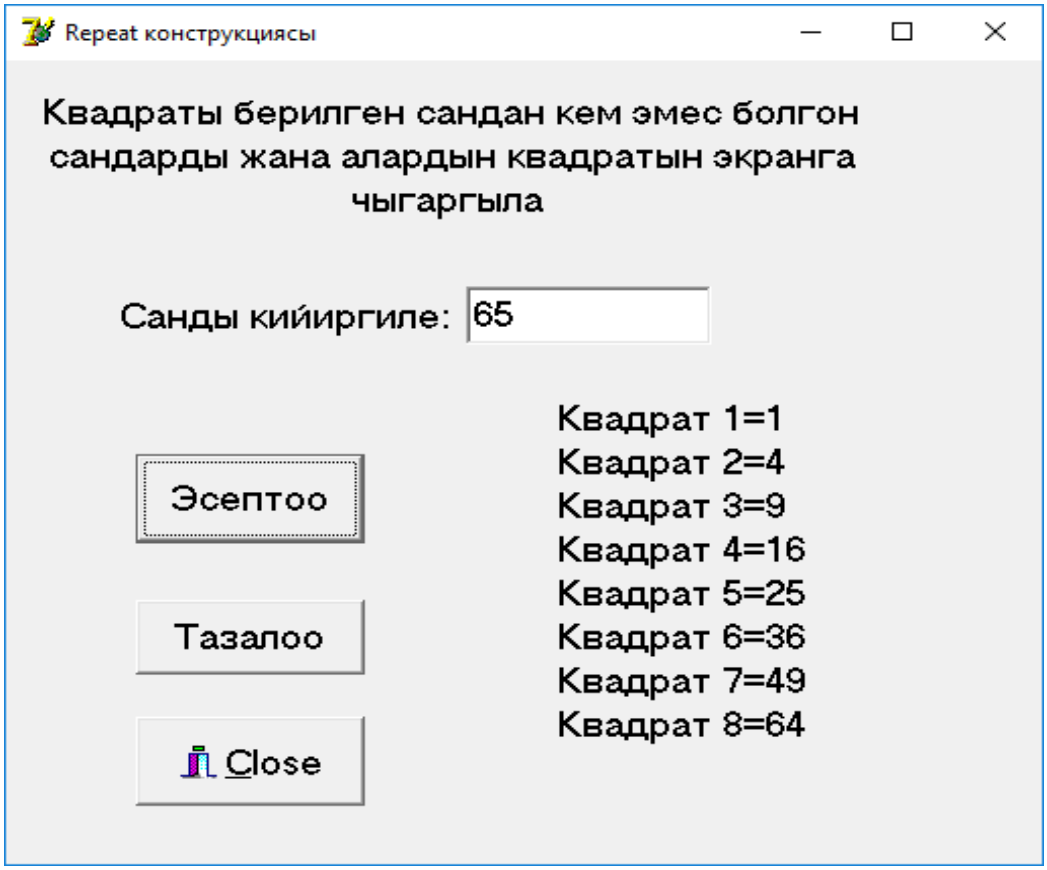
Ошентип, repeat жана unti1 кызматчы сөздөрүнүн ортосундагы телонун инструкциялары шарт аткарылбай калганга чейин улантылат.

3.5-мисал. Квадраты берилген сандан кем эмес болгон сандарды жана алардын квадратын экранга.

Repeat конструкциясын колдонобуз.

3.2-листинг. *Repeat конструкциясын колдонуу листинги.*

```
procedure TForm1.Button1Click(Sender: TObject);
var n,n1,sqrn:integer;
   n2:real;
begin
   n:=1;
   n1:=StrToInt(Edit1.Text);
   n2:=round(sqrt(n1));
   Repeat
     sqrn:=sqr(n);
     Label3.Caption:=Label3.Caption+
     'Квадрат '+IntToStr(n)+'='+IntToStr(sqrn)+#13;
     n:=n+1;
     sqrn:=sqr(n);
   Until (n>=n2) and (sqrn>n1);
end;
```



3.2-сүрөт. Сандардын квадратын чыгаруу

§ 3.3. Каталыктарды түзөтүү

Маселелерди чечүүнүн программасын түзүп жатканда каталыктар пайда болушу мүмкүн. Мындай каталыктарды үч группага бөлүүгө болот:

- Синтаксистик;
- Аткаруу убактысы боюнча каталыктар;
- Алгоритмдик.

3.3.1. Синтаксистик каталыктар. Синтаксистик каталыктарды компиляциялоо убактысы боюнча каталыктар (Compile-time error) деп да коюшат. Мындай каталыктарды жеңил эле оңдоого болот. Аларды компилятор таап берет жана ал каталык жөнүндө маалымат берет. Программист текстке тиешелүү түзөтүүлөрдү кийирип, кайрадан компиляциялоо керек.

Синтаксистик каталыктарга төмөнкү учурларда пайда болот:

- Кызматчы сөздөр туура эмес жазылса;

- Чекит, үтүрлүү чекит жа башка белгилер коюлбай калса;
- Циклде begin end коюлбай калса ж.б.

3.3.2. Аткаруу убактысы боюнча каталыктар. Мындай каталыктарды Delphi де **өзгөчө каталыктар** (exception) деп аташат. Бул каталыктарды деле жеңил эле түзөтүүгө болот. Мындай каталыктар программаны биринчи жолу жүктөгөндө эле же тестирлөө учурунда байкалат.

Өзгөчө каталыктар болгон учурда Debugger Exception Notification терезеси пайда болуп, экранга каталыктын тиби боюнча информация жана каталыктын себебин баяндаган билдирүү чыгарылат. Бул учурда программист эки жолдун бирин тандап алат:

- Программанын аткарылышын токтотот. Ал үчүн Run менюсунан Program Reset (Ctrl+F2) командасын тандоо керек;
- Программанын аткарылышын уланта берет. Ал үчүн Run менюсунан Step командасын тандап, ар бир инструкциянын аткарылышын байкап, каталыкты издейт.

Каталыктарды издөө жана жоюу процесси **түзөтүү** (отладка) деп аталат.

Программанын иштөө жөндөмдүүлүгүн текшерүү процесси – **тестирлөө** деп аталат.

3.3.3. **Өзгөчө каталыктардын алдын алуу.** Delphi де өзгөчө каталар көп учурайт. Өзгөчө каталар пайда болгон учурда каталык жөнүндө маалымат чыгарылып, программанын иштөөсү токтотулат. Ошондуктан бул ситуация өзгөчө учур деп аталат. Мисалы, нөл санына бөлүү – өзгөчө каталыктын классикалык мисалы болуп эсептелет.

Өзгөчө учурду көзөмөлдөө үчүн try/except/end конструкциясы колдонулат:

```
try
    <Каталыктар пайда болуучу инструкциялар>;
except
    <Каталыктан чыгууну камсыздоочу инструкциялар>;
end;
```

Эң оболу программанын негизги варианты болгон try/except секциясындагы инструкциялар аткарылат.

Эгерде бул секциядагы кайсыл бир инструкцияда өзгөчө каталык болсо, анда секциянын калган инструкциялары аткарылбайт да, башкаруу except/end секциясына берилет.

Эгерде бул секциядагы бардык инструкциялар каталары жок аткарылса, анда except/end секциясынын инструкциялары аткарылбайт.

3.5-мисал. Омдун закону боюнча токтун күчүн аныктоо программасын түзгүлө.

Омдун закону боюнча токтун күчүн аныктоо формуласы төмөнкүдөй:

$$I = \frac{U}{R}.$$

Формага маалыматтарга Edit1 Edit1 талаасына цифралардан башка символдорду кийирүүнү тыюу салуу үчүн onKeyPress касиетине төмөнкү программалык кодду жазып алса болот:

```
If not (key in ['0' .. '9', #8, #44, #45]) Then key:=#0;
```

Мында #8 – үтүр баскычынын, , #44 – backspace баскычынын, #45 – минус баскычынын коду, ал эми #0 – эч кандай баскыч басылган жок дегенди билдирет. Жогорудагы программалык код Edit1 талаасына тамгаларды билдирүүчү баскычтарды кийирүүгө тыюу салат.

Негизги программанын текстинде төмөнкүлөрдү эске алабыз:

1-каталык, Edit1 талаасына санды кийирбей аткаруу баскычын басканда пайда болот;

2-каталык, Edit2 талаасына санды кийирбей аткаруу баскычын басканда пайда болот;

3-каталык, Edit2 талаасына нөл санын кийиргенде пайда болот.

Бул каталыктардын алдын алуу үчүн try-except-end конструкциясын колдонобуз. 1- жана 2-каталыктарды болтурбоо үчүн программанын except-end бөлүгүнө 3.3-листингдеги программалык кодду түзүп алабыз.

3.3-листинг. *1- жана 2-каталыктарды болтурбоо коду*

```
on EConvertError do
     begin
       if (Edit1.Text='') then
         begin
           ShowMessage('1-талаага маалымат кийириниз!');
           exit;
         end;
       if (Edit2.Text='') then
         begin
           ShowMessage('2-талаага маалымат кийириниз!');
           exit;
         end;
     end;
```

3-каталыкты болтурбоо үчүн 3.4-листингдеги программалык кодду алабыз:

3.4-листинг. *3-каталыкты болтурбоо коду*

```
except
on EZeroDivide do
    begin
       ShowMessage('Каршылык нөл болушу мумкун эмес!');
    end;
exit;
end;
```

3.3- жана 3.4-листингдеги коддорду бириктирип каталыктарды болтурбоонун жалпы программасы 3.5-листингде келтирилди.

3.5-листинг. *Каталыктарды болтурбоонун толук коду*

```
procedure TForm1.Button1Click(Sender: TObject);
var U,R:real;
    I:real;
begin
       Try
          U:=StrToFloat(Edit1.Text);
          R:=StrToFloat(Edit2.Text);
```

```
        I:=U/R;
      Except
        on EConvertError do
         begin
          if (Edit1.Text='') then
           begin
            ShowMessage('1-талаага маалымат кийириңиз! ');
            Edit1.SetFocus;
            exit;
           end;
          if (Edit2.Text='') then
           begin
            ShowMessage('1-талаага маалымат кийириңиз!');
            Edit2.SetFocus;
            exit;
           end;
         end;
        on EZeroDivide do
         begin
          ShowMessage('Каршылык нөл болушу мүмкүн эмес!');
          exit;
         end;
      end;
      Label4.Caption:='Токтун күчү: I= '+FloatToStrF(I,ffGen-
eral,4,2)+' A';
end;

procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
      if (key in['0'..'9',#8,#44,#45])=False then key:=#0;
end;

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

```
    if (key in['0'..'9', #8,#44,#45])=False then key:=#0;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
     Edit1.Text:='';
     Edit2.Text:='';
     Label4.Caption:='';
     Edit1.SetFocus;
end;
```

Өзгөчө учурдагы каталар менен иштеп жатканда Delphi системасы каталыктардын алдын алуу үчүн аракеттерди жасайт.

Мисалы, нөлгө бөлүү катасы пайда болсо, Delphi системасы 3.4-сүрөттөгү билдирүүнү чыгарат

Бул абалдан чыгып, программага кайтып келүү үчүн ОК баскычын басып, андан кийин баскычтардын Ctrl+F2 (Program Reset) комбинациясын басуу керек.

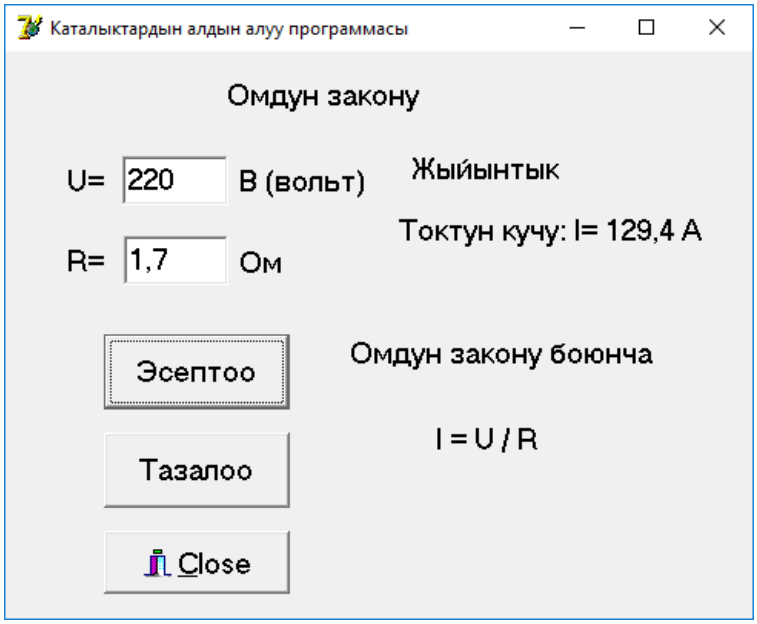
Мындай билдирүүлөрдү чыгарбастан колдонуучунун каталыктар боюнча билдирүүсүнүн чыгышы үчүн, б.а. try/except/end фрагменти менен жазылган коддогу инструкциялардын толугу менен аткарылышы үчүн инструменттер панелиндеги параметрлерге бир нече өзгөртүүлөрдү жасоо керек. Ал үчүн төмөнкү кадамдарды жасайбыз (3.5-сүрөт):

- Delphi Tools менюсун ачабыз;
- Debugger Options... пунктунун
- Language Exceptions барагындагы
- Stop on Delphi Exceptions чекбоксундагы белгини (галочканы) алып

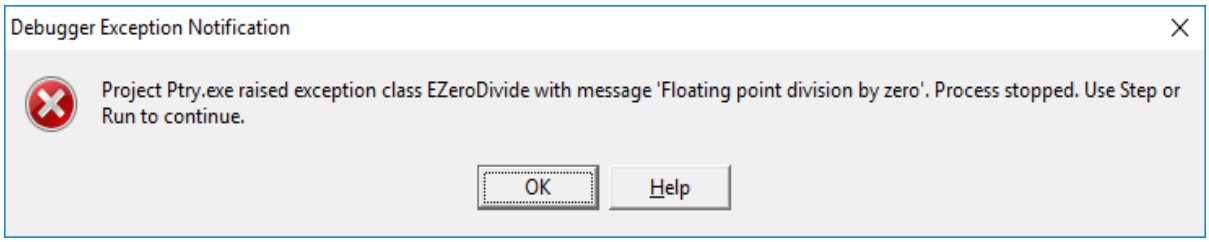
ОК баскычын баскандан кийин Delphi системасынын билдирүүлөрү чыкпайт.

Жогорудагы өзгөртүүлөрдү жасагандан кийин Delphi системасы колдонуучунун билдирүүлөрүн чыгарууга мүмкүнчүлүк берет.

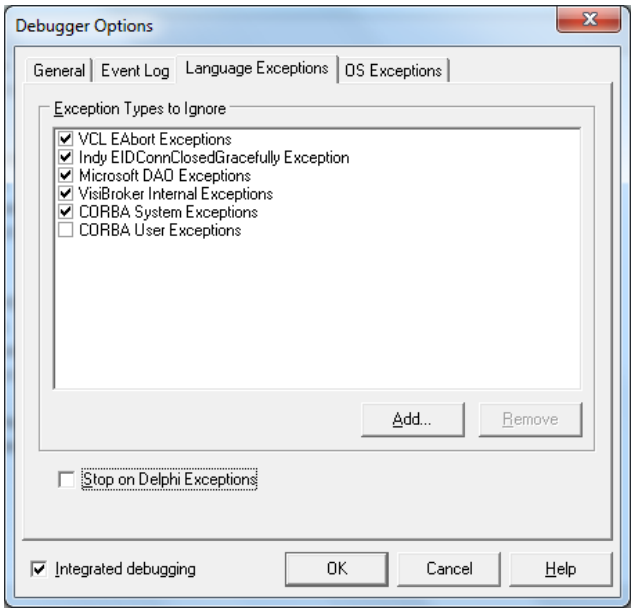
Программа аткарылгандагы жыйынтык 3.3-сүрөттө берилди.



3.3-сүрөт. Токтун күчүн эсептөө

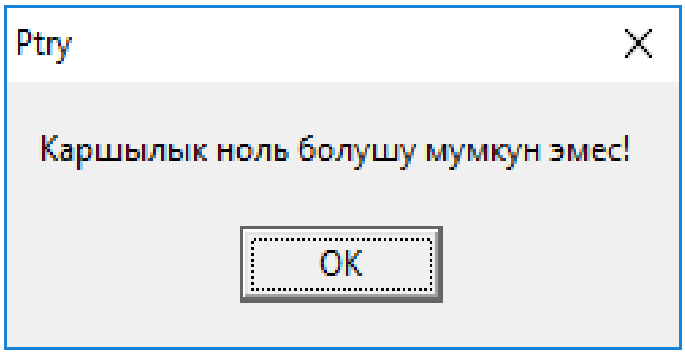


3.4-сүрөт. Delphi Системасынын чыгарган билдирүүсү



3.5-сүрөт. Stop on Delphi Exceptions билдирүүсүн чыгарбоо

Мисалы, нөлгө бөлүү катасы пайда болгон учурда колдонуучунун try/except/end фрагментинде жазган билдирүүсү чыгарылат (3.6-сүрөт).



3.6-сүрөт. Колдонуучунун билдирүүсү

Алгоритмдик каталык учурунда абал башкача болот. Компилятор мындай каталыктарды таба албайт. Каталыкты тестирлөө учурунда гана табуу мүмкүн.

Өзгөчө учурлардын алдын алуу үчүн дагы бир конструкция бар. Бул учур өзгөчө учурдун болушуна карабай коддун белгилүү бир фрагментинин аткарылышы зарыл болгон учурда колдонулат.

Бул конструкция

try/ finally/end

формасында жазылат.

```
try
    <Каталыкты берүүчү инструкциялар>;
finally
    <Каталыктан чыгууну камсыздоочу инструкциялар>;
end;
```

4-глава. Массивдер

§ 4.1. Массивди жарыялоо

Массив деп атайын атка ээ болгон жана ар бир элементи индекстер менен аныкталган бирдей типтеги өзгөрмөлөрдүн жыйындысын айтабыз.

Мисалы, математикада векторлор массивдин мисалы боло алат.

Массивдер типтерди жарыялоо бөлүмүндө төмөнкүдөй жарыяланат:

```
type
name : array[n1..n2] of mun;
```

мында, *name* – массивдин атын, *n1* массивдин элементтеринин диапазонунун төмөнкү, *n2* – жогорку чегин, ал эми *mun* массивдин элементтеринин тибин аныктайт.

Мисалы 100 элементтен турган бүтүн сандардын массиви төмөнкүдөй жарыяланат:

type *bytyn* : **array**[1 .. 100] **of** Integer;

Андан кийин ***bytyn*** типке тиешелүү болгон өзгөрмөлөрдү жарыялоого болот:

var A, B: ***bytyn***;

Массивдерди өзгөрмөлөрдү жарыялоо бөлүмүндө аларды айкын түрдө төмөнкүдөй жарыялоого болот:

```
var
name : array[n1..n2] of mun;
```

var A, B : **array**[1..100] **of** Integer;

Массивдин элементин көрсөтүү үчүн массивдин атын жана квадраттык кашаанын ичине элементтин индексин көрсөтүү керек:

A[5] := 101;
B[30] := 165;

§ 4.2. Массивдердин үстүнөн аткарылган иш-аракеттер

Массивдердин үстүнөн төмөнкүдөй иш-аракеттерди аткарууга болот:

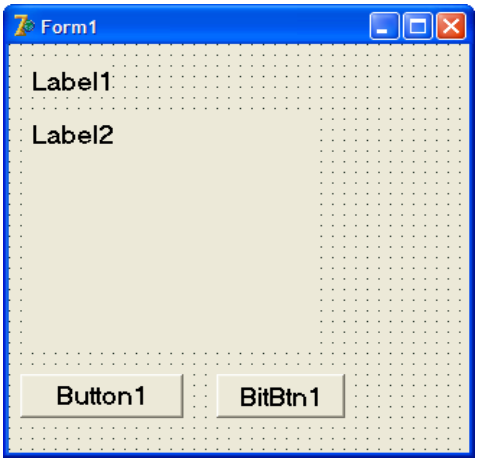
- **Массивди чыгаруу;**
- **Массивди киргизүү;**
- **Массивдин максималдык (минималдык) элементин табуу;**
- **Массивдин берилген элементин издөө;**
- **Массивди сорттоо (иреттөө).**

4.2.1. Массивди чыгаруу

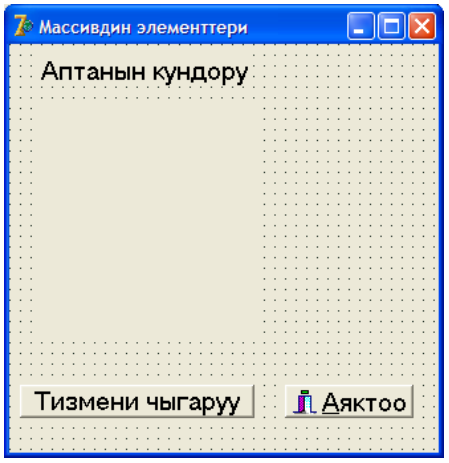
Массивди чыгаруу деп анын элементтерин монитордун экранына чыгарууну түшүнөбүз. Мисалы, массивдин элементтерин чыгаруу үчүн Label компонентин колдонууга болот.

4.1-мисал. *Аптанын күндөрүн чыгаргыла.*

Маселени чыгаруу үчүн формага Label1, Label1, Button1, BitBtn1 компоненталарын жайгаштырабыз (4.1-сүрөт). Компоненталардын касиеттеринен пайдаланып, тиешелүү маанилерди ыйгарабыз (4.2-сүрөт).



4.1-сүрөт. Массивди чыгаруу формасынын макети



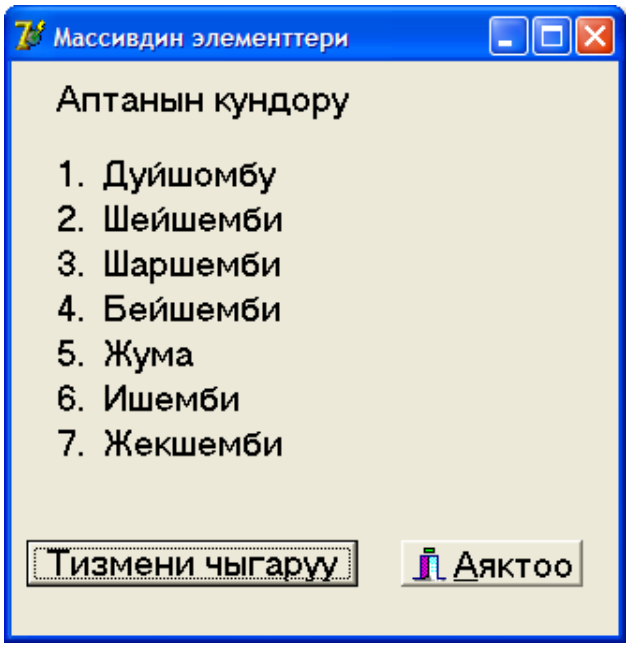
4.2-сүрөт. Компонента-лардын маанилери

Button1 компонентасы үчүн төмөнкү программалык кодду жазабыз (4.1-листинг).

4.1-листинг. Тизмелерди чыгаруу

```
unit Un_day;
```

```
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, Grids, StdCtrls, Buttons;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Button1: TButton;
    BitBtn1: TBitBtn;
Label2: TLabel;
procedure Button1Click(Sender:
TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
  i:integer;
  st:string;
  day : array[1..7] of
string[10]=('Дуйшомбу','Шейшемби','Шаршемби','Бейшемби',
                                    'Жума','Ишемби','Жекшемби');
implementation
        {$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
    for i:=1 to 7 do
    st:=st+IntToStr(i)+'.  '+day[i]+#13;
    Label1.Caption:=st;
end;
end.
```



4.3-сүрөт. Тизмени чыгаруу

BitBtn1 компоненти программанын ишин аяктоо үчүн колдонобуз. Ал үчүн Kind касиетинин Close маанисин «Аяктоо» деген мааниге алмаштырабыз. Программа аткарылганда форма 4.3-сүрөттөгү түргө келет.

4.2.2. Массивди киргизүү

Массивди киргизүү деп программа иштеп жатканда колдонуучудан же файлдан массивдин элементин алууну түшүнөбүз. Бул максатта Additional барагынын StrinGrid компонентин колдонууга болот.

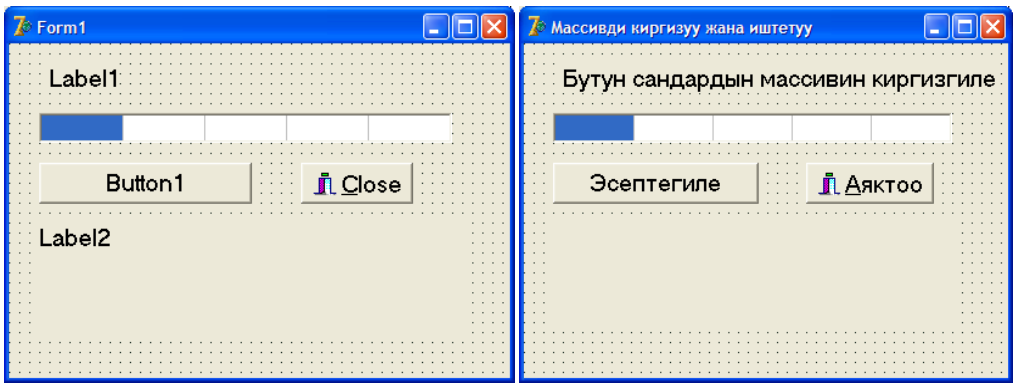
4.2-мисал. *Массивдин элементтеринин арифметикалык орточо маанисин тапкыла.*

Маселени чыгаруу үчүн StrinGrid1 компонентасын формага жайгаштырып, анын касиеттеринин маанисин 4.1-таблицадагыдай тандап алабыз.

4.1-таблица. StrinGrid1 компонентасынын касиеттеринин мааниси

Касиети	**Мааниси**
ColCount	5
FixedCols	0
RowCount	1
DefaultRowHeight	24
Height	24
DefaultColWidth	64
Width	328
Options.goEditing	True
Options.goAlwaysShowEditing	True
Options.goTabs	True

Формага Label1, Label2, Button1 жана BitBtn компоненталарын жайгаштырып, алардын касиеттеринин маанилерин 4.4-сүрөттөгүдөй кылып тандап алабыз.



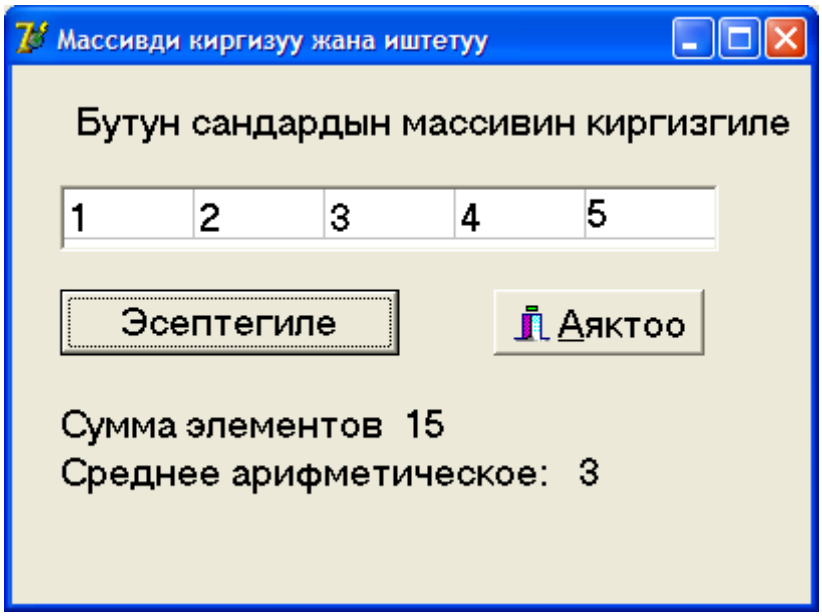
4.4-сүрөт. Колдонмонун башкы формасы

4.5-листинг. Бүтүн сандардын массиви менен иштөө

```
unit Un_massiv;
interface
uses
        Windows, Messages, SysUtils, Variants, Classes,
        Graphics, Controls, Forms,
          Dialogs, StdCtrls, Grids, Buttons;
        type
          TForm1 = class(TForm)
            Label1: TLabel;
            StringGrid1: TStringGrid;
Button1: TButton;
Label2: TLabel;
BitBtn1: TBitBtn;
procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var  Form1: TForm1;
implementation
        {$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var  a:array[1..5] of integer; // массив
  sum_elem: integer;        // сумма элементов
  sr_arif:real;             // среднее арифметическое
  i: integer;               // индекс
begin
          for i:=1 to 5 do
            if Length(StringGrid1.Cells[i-1,0])<>0
              then a[i]:=StrToInt(StringGrid1.Cells[i-1,0])
              else a[i]:=0;
        sum_elem:=0;
        for i:=1 to 5 do
          sum_elem:=sum_elem+a[i];
          sr_arif:=sum_elem/5;
Label2.Caption:='Сумма элементов  '+IntToStr(sum_elem)+#13+
            'Среднее арифметическое:   '+FloatToStr(sr_arif);
```

```
end;
end.
```

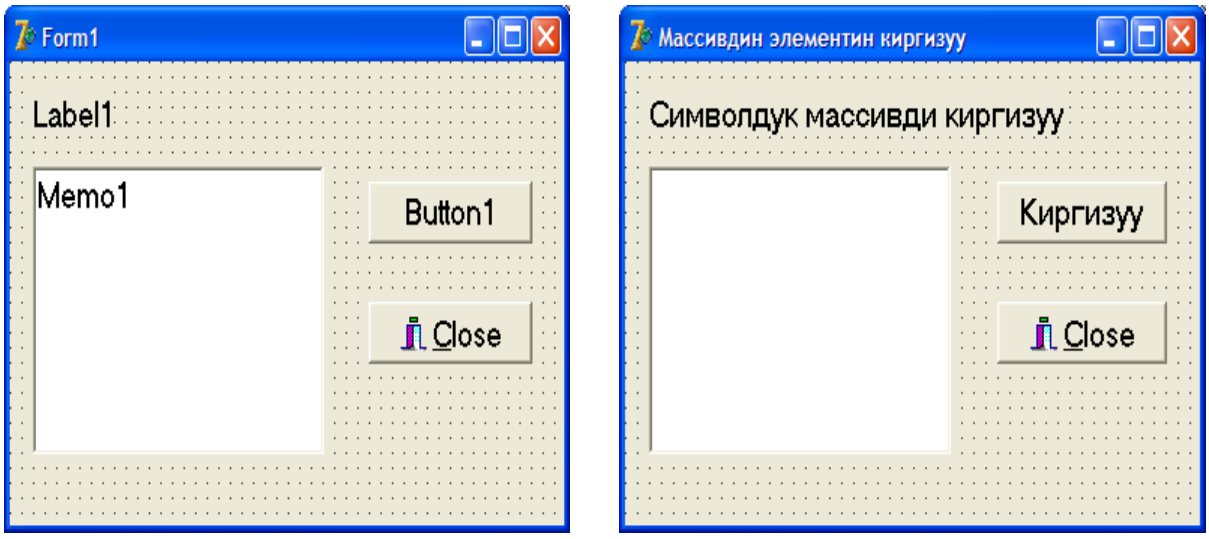
4.5-листинг аткарылганда 4.5-сүрөттөгүдөй жыйынтык алынат.



4.5-сүрөт. Бүтүн сандардын массиви менен иштөө

4.3-мисал. *Символдук массивди киргизүү программасын түзгүлө.*

Элементтери символдор болгон массивди киргизүү үчүн **Memo** компонентин колдонуу ыңгайлуу. Формага Label1, Button1, BitBtn жана Memo1 компоненталарын жайгаштырабыз жана алардын тешелеш касиеттеринин маанилерин 4.6-сүрөттөгүдөй кылып тандап алабыз.



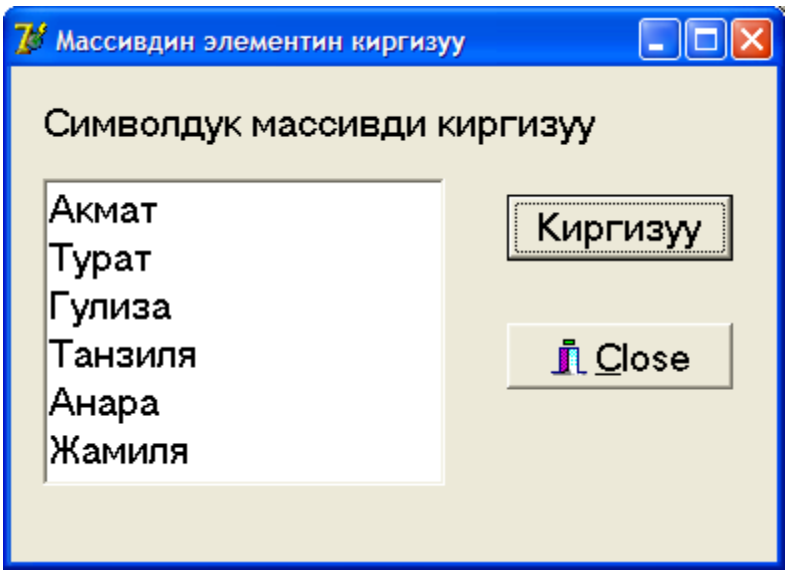
4.6-сүрөт. Колдонмонун башкы бетинин көрүнүшү

Button1 баскычы үчүн программалык кодду төмөнкүдөй жазып алабыз (4.5-листинг).

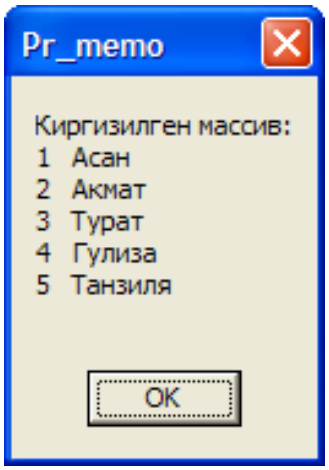
4.6-листинг. Символдук массивдин элементтерин кийирүү

```
unit Un_memo;
interface
uses
        Windows, Messages, SysUtils, Variants, Classes,
        Graphics, Controls, Forms, Dialogs, Buttons, StdCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Memo1: TMemo;
    Button1: TButton;
    BitBtn1: TBitBtn;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
  a: array[1..5] of string[30];//массив
  n:integer; // киргизилген жолчолордун саны
  i:integer; // массивдин элементинин индекси
  st:string;
begin
  n:=Memo1.Lines.Count;
  if n=0 then
    begin
      ShowMessage('Маалыматтар киргизилбеген');
      Exit;
    end;
  if n>5 then
      begin
ShowMessage('Жолчолордун саны массивдин өлчөмүнөн чоң!');
      n:=5;
```

```
    end;
  for i:=1 to n do
    a[i]:=Form1.Memo1.Lines[i-1];
  if n>0 then
    begin
      st:='Киргизилген массив:'+#13;
      for i:=1 to n do
        st:=st+IntToStr(i)+'   '+a[i]+#13;
        ShowMessage(st);
    end;
end;
end.
```



4.7-сүрөт. Массивдин элементтерин Memo талаасына кийирүү



4.8-сүрөт. Кийирилген массивдин элеметтери

4.2.3. Массивдин минималдык элементин табуу

Массивдин минималдык элементин табуу маселесин бүтүн сандардын массивинин мисалында карайлы.

Массивдин минималдык элементин табуу алгоритми төмөнкүдөй:

1). Массивдин 1-элементин минималдык элемент деп алабыз.

2). Тандалган 1-элемент менен калган элементтердин ар бирин салыштырабыз.

3). Эгерде кийинки элемент 1-элементтен кичине болсо, анда ушул элементти эң кичине элемент деп тандап алабыз жана бул процессти акыркы элементке менен салыштырганга чейин улантабыз.

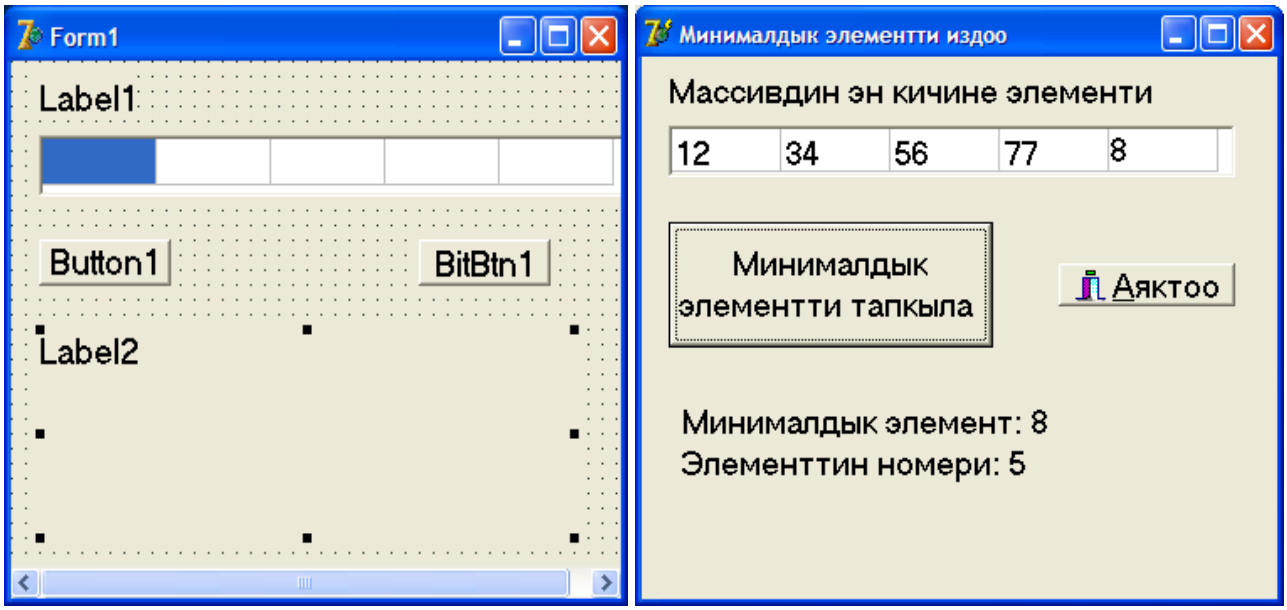
4). Ошентип эң кичине элемент табылат жана аны экранга чыгарабыз.

4.4-мисал. Массивдин эң кичине элементин тапкыла.

Маселени чечүү үчүн StringGrid1, Label1, Label2, Buttonl жана BitBtnl компоненталарын формага жайгаштырабыз.

StringGrid1 компонентасынын касиеттеринин мааниси 4.1-таблицада көрсөтүлгөндөй кылып тандап алабыз.

Buttonl баскычына төмөнкүдөй программалык код жазабыз:



4.9-сүрөт. Массивдин эң кичине элементин табуу

4.7-листинг. *Минималдык элементти табуу.*

```
procedure TForm1.Button1Click(Sender: TObject);
    var
      a: array[1..5] of integer; // массив
      min: integer; // минималдык элементтин номери
      i: integer; // салыштырылуучу элементтин номери
    begin
      for i:=1 to 5 do
         a[i]:=StrToInt(StringGrid1.Cells[i-1,0]);
```

```
        min:=1; // 1-минималдык элемент
      for i:=2 to 5 do
        if a[i]< a[min] then min:=i;
    label2.caption:='Минималдык элемент: '
    +IntToStr(a[min]) +#13+'Элементтин номери: '+
IntToStr(min);
    end;
```

4.2.4. Массивден элементти издөө

Массивдердин үстүнөн төмөнкүдөй операцияларды аткарууга болот: Көп маселелерди чечүүдө массивде бул же тигил маалыматтын, мисалы, массивде “Асанов” деген студенттин фамилиясынын, бар же жок экендигин текшерүүгө туура келет. Мындай маселелерди массивдеги издөөнү уюштуруу маселеси деп атайбыз.

Бул маселени чечүү үчүн ар түрдүү алгоритмдерди колдонууга болот. Алардын ичинен эң жөнөкөйү “жөнөкөй тандоо” алгоритми деп аталат. Алгоритмдин иштөө принциби төмөнкүдөй:

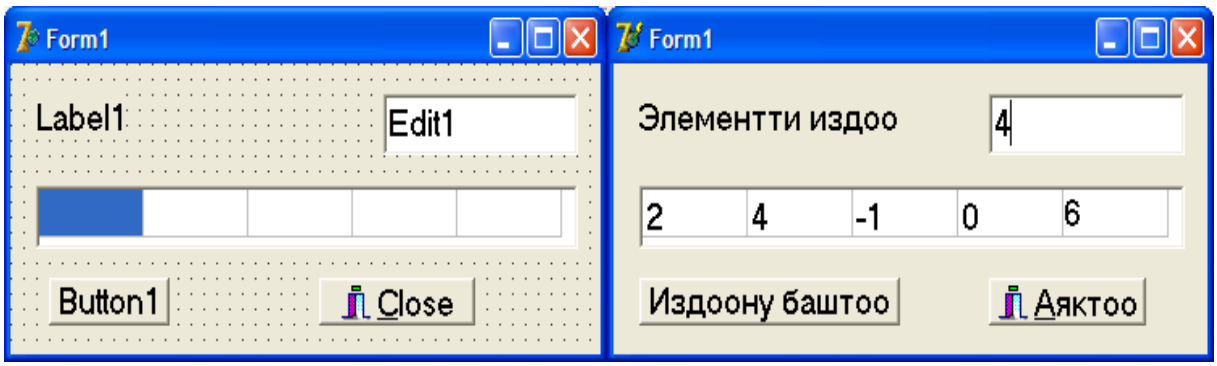
1). Изделүүчү маалыматты үлгү катары кабыл алабыз.

2). “Үлгү” менен массивдин ар бир элементи дал келгенге чейин салыштырылып чыгат.

Бул “жөнөкөй тандоо” алгоритми массив иреттелбеген учурда колдонулат.

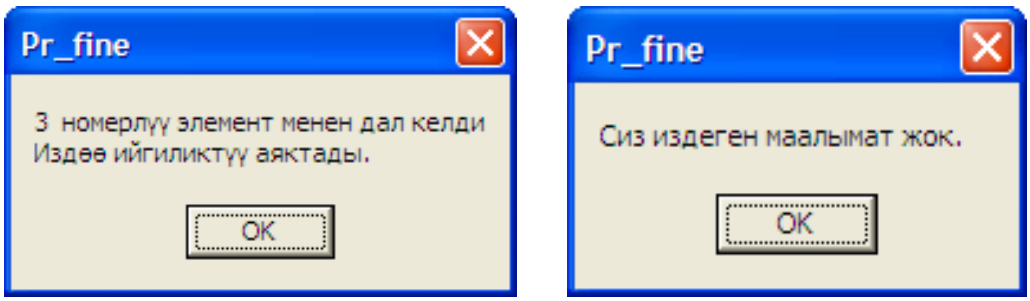
4.5-мисал. Массивдин элементин издөө программасын түзгүлө.

Издөө программасынын макетин 2-сүрөттөгүдөй түзөбүз.



4.10-сүрөт. Массивдин элементин издөө

Программанын жыйынтыгы боюнча маалыматты чыгаруу үчүн ShowMessage функциясын пайдаланабыз.



4.11-сүрөт. Программанын жыйынтыгы боюнча маалымат

Түзүлгөн программанын листинги төмөнкүдөй.

4.8-листинг. *Элементти издөө.*

```
procedure TForm1.Button1Click(Sender: TObject);
var   a: array[1..5] of integer; //массив
      obr: integer; // издөө үчүн үлгү
      found: boolean; // TRUE - үлгүнүн элемент менен дал келиши
      i: integer; // массивдин элементинин  индекси
      begin
      // массивди киргизүү
      for i:=1 to 5 do
      a[i] := StrToInt(StringGrid1.Cells[i-1,0]);
      // үлгүнү киргизүү
      obr := StrToInt(Edit1.text);
      // издөө
      found := FALSE; // керектүү элемент массивде жок болсун
      i:= 1;
      repeat
        if a[i] = obr then found := TRUE
                      else i := i+1;
      until (i > 5) or (found = TRUE);
      if found then
ShowMessage(IntToStr(i)+'  номерлүү элемент менен дал келди'
      +#13+'Издөө ийгиликтүү аяктады.')
      else
      ShowMessage('Сиз издеген маалымат жок.');
      end;
```

4.2.5. Массивди сорттоо

Массивди сорттоо (иреттөө) деп массивдин элементтерин өсүү же кемүү тартибинде жайгаштырууну айтабыз. Бул маселени чечүүнү Additional барагындагы StrinGrid компонентин колдонуу менен көрсөтөбүз.

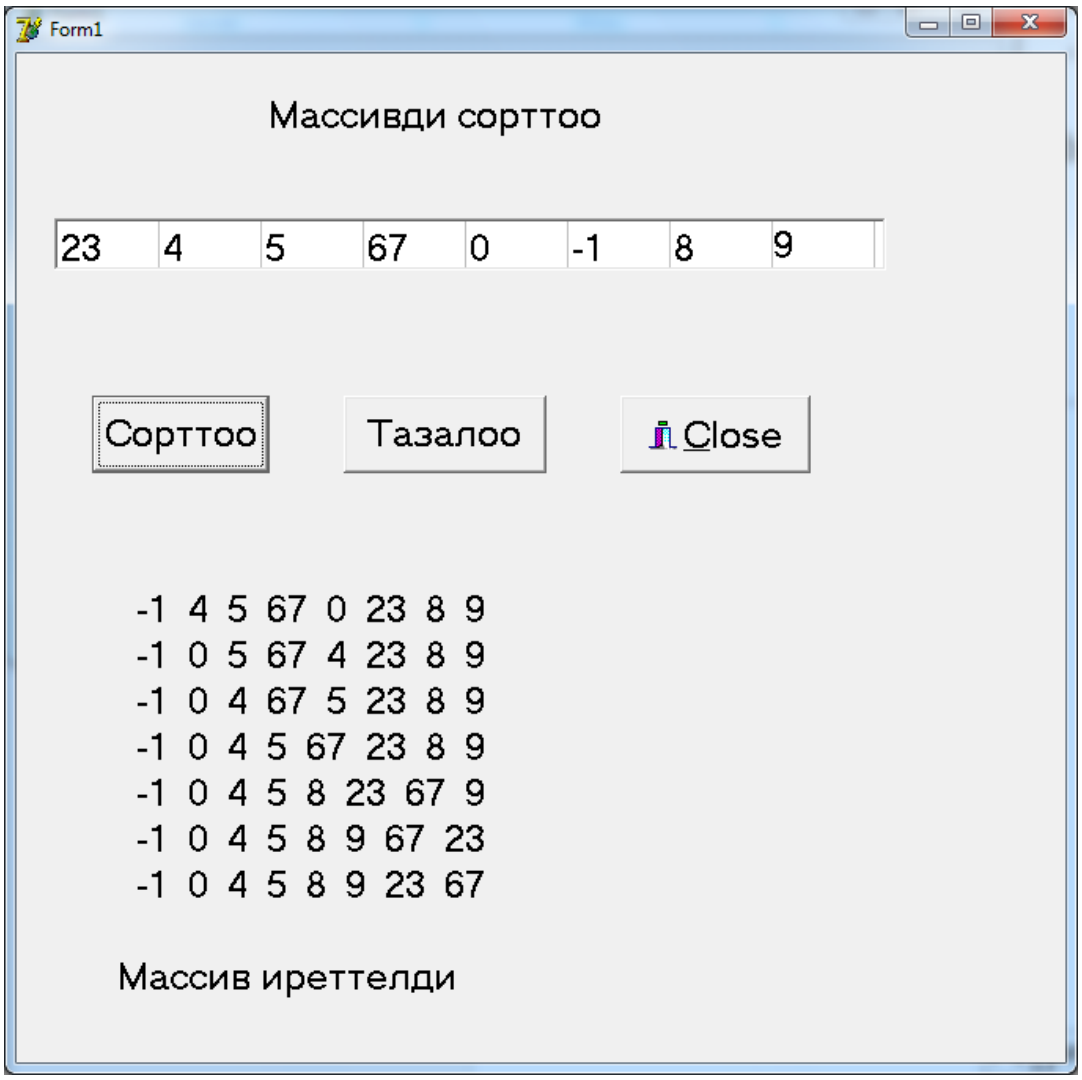
4.6-мисал. 8 элементтүү массивди иреттөө программасын түзгүлө.

Элементтерди иреттөө программасынын макетин 4-сүрөттөгүдөй түзөбүз.

4.9-листинг. *Массивди иреттөө.*

```
procedure TForm1.Button1Click(Sender: TObject);
    var   i:integer; // массивдин индекси
    a:array[1..8] of integer; // массив
    min:integer; //  минималдык элементтин номер
    j:integer; // минималдык элемент менен салыштырылуучу элементтин номери
    buf:integer; // элементерди алмаштыруучу буфер
    k:integer; //  массивди чыгаруунун индекси
    begin
    // массивди кийируу
    Label2.Caption:='';
    For i:=1 to 8 do
    begin
    a[i]:=StrToInt(StringGrid1.Cells[i-1,0]);
    end;
    // иреттоо
    For i:=1 to 7 do
    begin
    min:=i;
    For j:=i+1 to 8 do
    If a[j]<a[min] Then
    min:=j;
    // орундарын алмаштырабыз a[min],  a[i]
    buf:=a[i];
    a[i]:=a[min];
    a[min]:=buf;
    end;
    // массивди чыгаруу
```

```
For k:=1 to 8 do
begin
Label2.Caption:=Label2.Caption+'  '+IntToStr(a[k]);
end;
Label2.Caption:=Label2.Caption+chr(13);
end;
Label2.Caption:=Label2.Caption+chr(13)+'Массив иреттелди';
end;
```



4.12-сүрөт. Программанын жыйынтыгы боюнча маалымат

§ 4.3. Көп ченемдүү массивдер

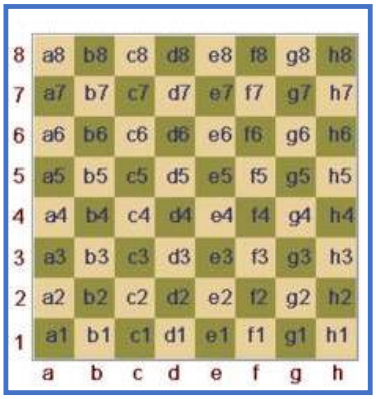
Аныктама. *Эгерде массивдин элементтери эки же андан көп индекстер менен аныкталса, анда мындай массивдерди көп ченемдүү массивдер деп айтабыз.*

Эгерде индекстердин саны экөө болсо, массивди эки ченемдүү массив деп айтабыз.

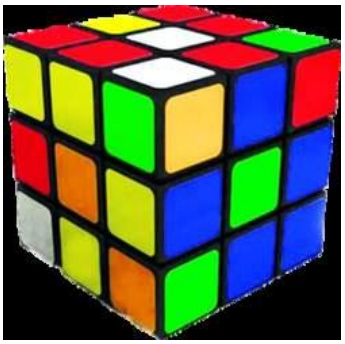
Математикада матрицаны эки ченемдүү массив катары кароого болот. Мисалы, үч жолчодон жана үч мамычадан турган 3x3 өлчөмүндөгү A матрицасын карасак, анын ар бир элементи жолчосунун жана мамычасынын номерлери менен бир маанилүү аныкталат жана ал номерлер индекстер катары каралып, матрицанын элементтери төмөнкүдөй жазылат: $a_{11}, a_{12},...,a_{33}$.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Эки ченемдүү массивдин мисалы катары шахматтык досканы да алууга болот. Шахматтык доскадагы ар бир фигуранын абалы дагы эки индекс менен аныкталат: a1, b1, … , h8.



Үч ченемдүү массивдин мисалы катары Рубиктин кубигин алсак болот, анткени ал 3×3×3 өлчөмүндөгү кичинекей кубиктердин жыйындысынан туруп, анын ар бир кубиги x, y жана z координаталары менен аныкталат.



Эки ченемдүү массивдер типтерди баяндоо бөлүмүндө төмөнкүдөй жарыяланат:

```
type
name : array [n1..n2, m1..m2] of тип;
```

мында, *name* – массивдин атын, *n1* массивдин биринчи индексинин диапазонунун төмөнкү, *n2* – жогорку чегин, *m1* массивдин экинчи индексинин диапазонунун төмөнкү, *m2* – жогорку чегин, ал эми *тип* массивдин элементтеринин тибин аныктайт.

Эки ченемдүү массивдерди өзгөрмөлөрдү жарыялоо бөлүмүндө аларды айкын түрдө төмөнкүдөй жарыялоого болот:

```
var
name : array[n1..n2, m1..m2] of тип;
```

Эки ченемдүү массивдин элементин көрсөтүү үчүн массивдин атын жана квадраттык кашаанын ичине элементтердин индексин көрсөтүү керек:

A[5,6]:= 101;

B[1,30]:= 165;

Эки ченемдүү массивдин элементтерин кийирүү үчүн камтылуучу циклдерди колдонуу ыңгайлуу болот. Мисалы, биринчи цикл жолчолорду кийирсе, ар бир жолчонун элементтерин ички цикл кийире тургандай кылып программа түзүү керек.

4.7-мисал. *Эки ченемдүү массивдин элементтерин кокустук сандар менен толтургула.*

4.10-листинг. *Массивди элементтери менен чыгаруу*

```
procedure TForm1.Button1Click(Sender: TObject);
 var i, j:integer;
             s:string;

  a:array[1..100,1..100] of
  real;
  begin
  // Массивдин кийирүү
        for i:=1 to 5 do
        begin
             for j:=1 to 5 do
               begin
                 a[i,
  j]:=random(9);
               end;
        end;
  // Массивди чыгаруу
  for i:=1 to 5 do
  begin
  for j:=1 to 5 do
  begin
  s:=s+'a[i, j])='+FloatToStr(a[i, j])+' ';
  end;
  s:=s+chr(13);
  end;
  Label2.Caption:=s;
  end;
```

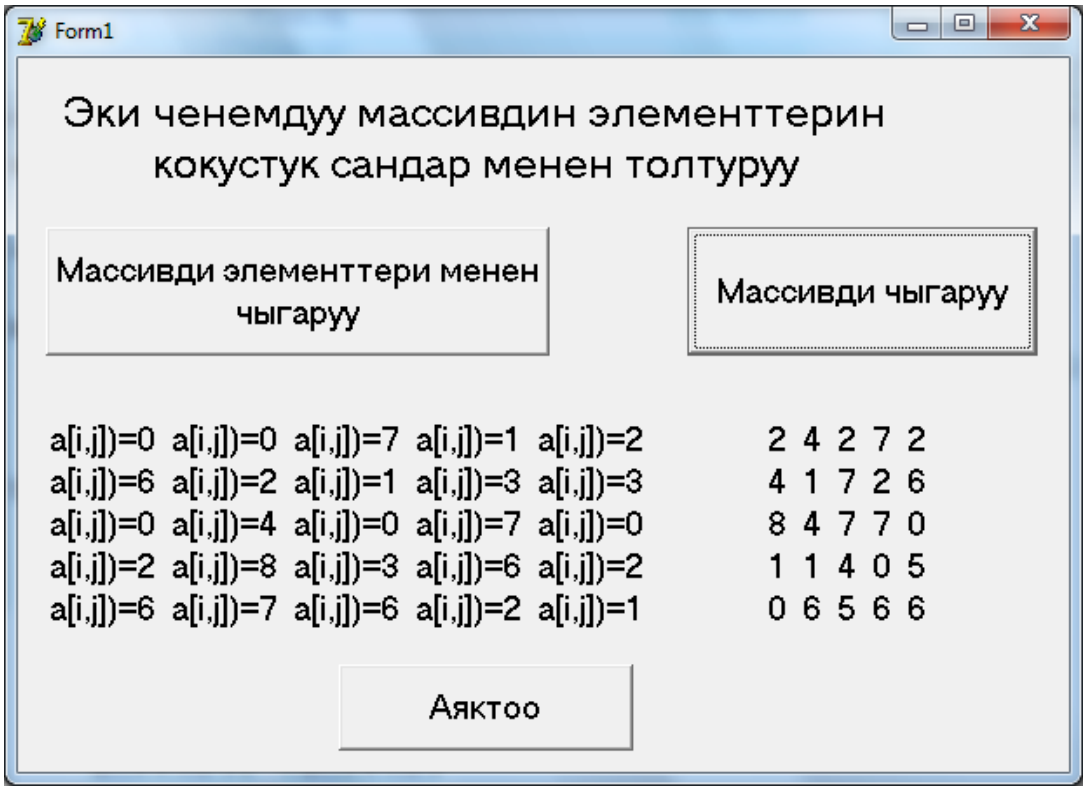
1-листингде камтылуучу циклдерди эң оболу массивдин элементтерин кокустук сандар менен толтуруу, андан кийин экинчи камтылуучу циклдин жардамы менен массивдин элементтерин координаталары боюнча чыгаруу каралган.

4.11-листинг. *Массивди чыгаруу*

```
      procedure TForm1.Button3Click(Sender: TObject);
var i, j:integer;                    // Массивди чыгаруу
        s1:string;                   For i:=1 to 5 do
                                     begin
a:array[1..100,1..100] of              for j:=1 to 5 do
real;begin                           begin
// Массивдин кийирүү                 s1:=s1+FloatToStr(a[i, j])+'  ';
   For i:=1 to 5 do                      end;
   begin                                 s1:=s1+chr(13);
    for j:=1 to 5 do                 end;
     begin                             Label3.Caption:=s1;
       a[i, j]:=random(9);           end;
     end;
   end;
```

4.14-листингде камтылуучу циклдерди массивдин элементтерин кокустук сандар менен толтуруу жана массивди толугу менен чыгаруу каралган.

Программанын жыйынтыгы 4.13-сүрөттө көрсөтүлдү.



4.13-сүрөт. Массивди кокустук сандар менен толтуруу

4.8-мисал. *Эки ченемдүү массивдин элементтерин кокустук сандарды колдонуу менен StringGrid компонентинин ячейкаларына жазгыла.*

Маселени чечүүнү бир нече кадамдарга бөлүп алабыз.

1-кадам. Формага StringGrid компонентин жайгаштырып, андагы фиксирленген жолчону (FixedRows=1) жана мамычаны (FixedCols), берилгендерди кийирүү үчүн керектүү болгон жолчолордун (RowCount=10) жана мамычалардын (ColCount=10) санын, ошондой эле таблицанын ячейкаларын редактирлөөгө мүмкүн боло тургандай кылып таблицанын касиеттерин (Options.goEditing=True, Options.goTab=True, Options.GoAlways-ShowEditor= True) саздап алабыз.

2-кадам. Форма жүктөлүшү менен таблицанын фиксирленген эң жогорку жолчосуна латын алфавитинин тамгалары, ал эми фиксирленген сол мамычасына цифралар чыга тургандай кылып форманын OnCreate касиетине төмөнкүдө программа жазып алабыз:

4.12-листинг. *Мамычалардын аталыштарын кийирүү*

```
procedure TForm1.FormCreate(Sender: TObject);
var i,j:integer;
begin
      // Мамычалардын аталыштарын кийируу
         for i:=1 to 10 do
         StringGrid1.Cells[i,0]:=chr(64+i);
      // Жолчолордун номерлерин кийируу
         for j:=1 to 10 do
         StringGrid1.Cells[0,j]:=IntToStr(j);
end;
```

3-кадам. Кокустук сандарды генерациялоочу random() функциясын колдонуп, камтылуучу циклдерди пайдаланып, формадагы таблицанын ячейкаларын кокустук сандар менен толтурабыз.

4.13-листинг. *Массивди кокустук сандар менен толтуруу*

```
procedure TForm1.Button1Click(Sender: TObject);
var i,j:integer;
begin
// Массивди кокустук сандар менен толтуруу
   for i:=1 to 10 do
     for j:=1 to 10 do
       begin
         StringGrid1.Cells[i,j]:=FloatToStr(random(10));
       end;
end;
```

4-кадам. Таблицанын ячейкаларын кокустук сандардан тазалоочу программаны түзүп алабыз.

4.14-листинг. *Массивди тазалоо*

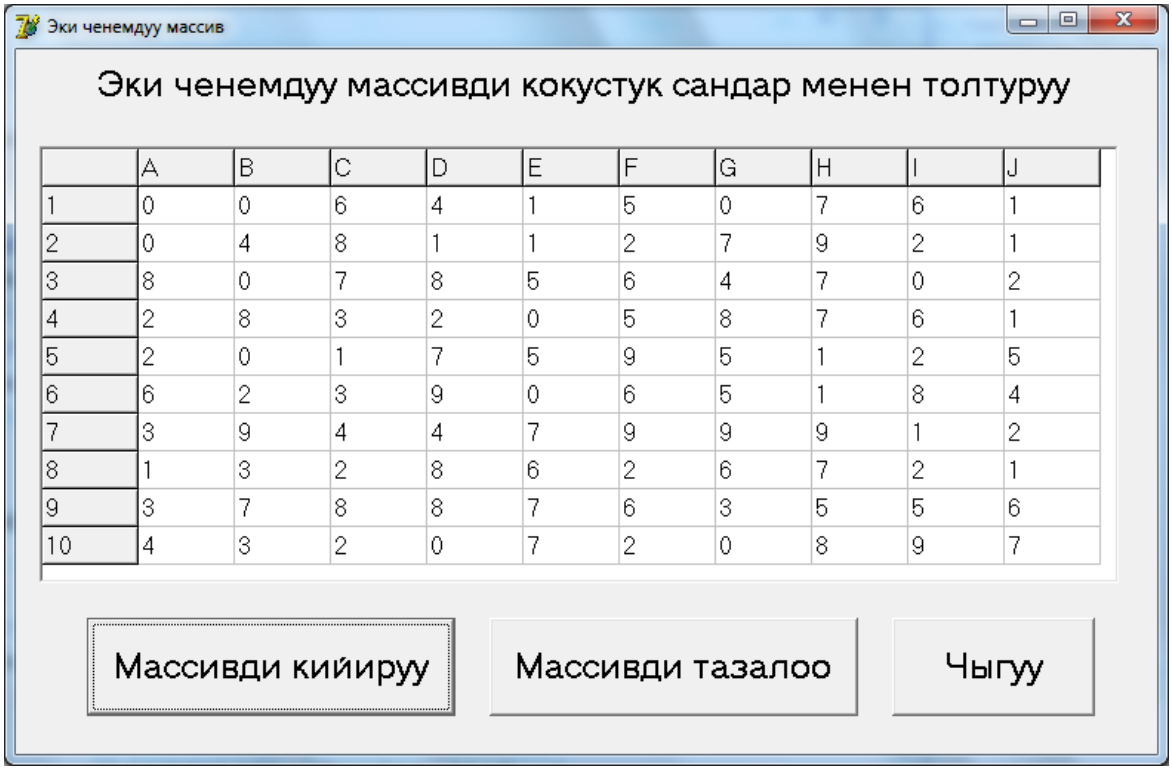
```
procedure TForm1.Button2Click(Sender: TObject);
var i,j:integer;
begin
// Массивди тазалоо
   for i:=1 to 10 do
     for j:=1 to 10 do
       begin
         StringGrid1.Cells[i,j]:='';
       end;
end;
```

5-кадам. Программанын ишин аяктоочу программаны жазып алабыз.

4.15-листинг. *Форманы жабуу.*

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Close;
end;
```

Программанын жыйынтыгы 4.14-сүрөттө чагылдырылды.



	A	B	C	D	E	F	G	H	I	J
1	0	0	6	4	1	5	0	7	6	1
2	0	4	8	1	1	2	7	9	2	1
3	8	0	7	8	5	6	4	7	0	2
4	2	8	3	2	0	5	8	7	6	1
5	2	0	1	7	5	9	5	1	2	5
6	6	2	3	9	0	6	5	1	8	4
7	3	9	4	4	7	9	9	9	1	2
8	1	3	2	8	6	2	6	7	2	1
9	3	7	8	8	7	6	3	5	5	6
10	4	3	2	0	7	2	0	8	9	7

4.14-сүрөт. Эки ченемдүү массивди StringGrid компонентине чыгаруу

StringGrid компоненти

StringGrid компоненти **Additional** барагында жайгашкан. Ал таблицаны элестетет жана анын ячейкаларына символдорду же символдордун жолчосун жазууга болот.

Компонентти колдонууда төмөнкү таблицада келтирилген касиеттерди пайдаланууга болот (4.2-таблица).

4.2-таблица. StringGrid компонентинин касиеттери

Касиети	Мааниси
Name	Компоненттин аты. Программада компонент менен иштөө үчүн колдонулат
ColCount	Таблицанын мамычаларынын саны
RowCount	Таблицанын жолчолорунун саны
Cells	Таблицанын ячейкасы. col номерлүү мамыча менен row номерлүү жолчонун кесилишиндеги элемент cells[col, row] форматында аныкталат

FixedCols	Таблицанын сол жагындагы фиксирленген колонка.
FixedRows	Таблицанын жогору жагындагы фиксирленген колонка
Options.goEditing	Таблицанын ячейкаларында редактирлөө мүмкүнчүлүгүн аныктайт. True — редактирлөөгө уруксат, False — уруксат эмес дегенди билдирет
Options.goTab	Таблицанын кийинки ячейкасына <Tab> баскычы менен өтүүгө уруксат (True) берүүнү же тыюу (False) салууну аныктайт
Options.GoAlways-ShowEditor	Компоненттин редактирлөө режимин аныктайт
Left	Таблицанын сол талаасынын чегинен форманын сол чегине чейинки аралык
Top	Таблицанын жогорку талаасынын чегинен форманын жогорку чегине чейинки аралык
Height	Таблицанын талаасынын бийиктиги
Width	Таблицанын талаасынын кеңдиги
Font	Таблицанын ячейкасында жайгашкан элементтердин шрифти

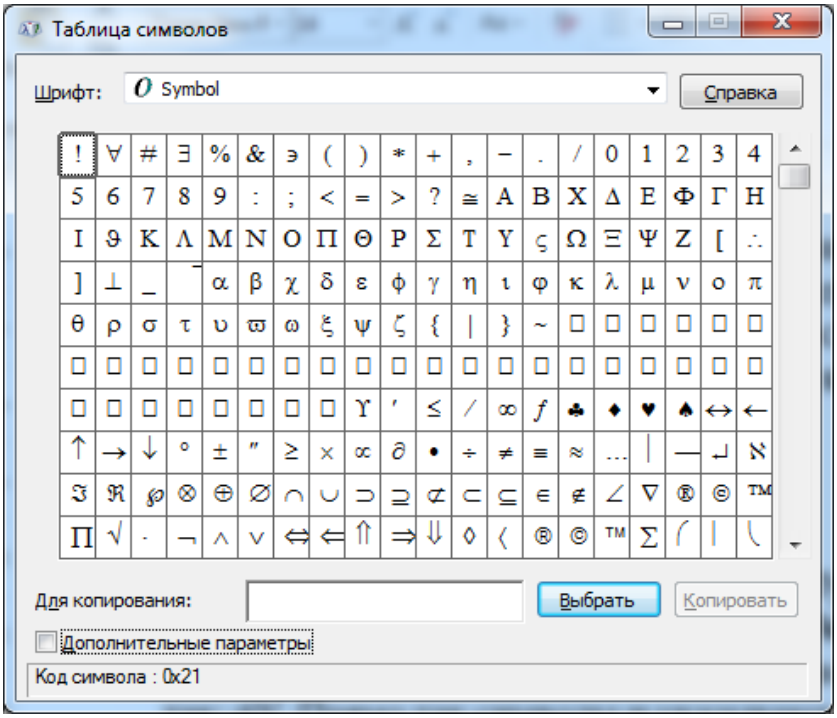
5-глава. Символдор жана жолчолор

§ 5.1. Символдор менен иштөөчү функциялар

Символ деп тамганы, цифраны же кандайдыр бир белгини түшүнөбүз. Тексттер символдордон куралат, ошондуктан символ тексттин бир бирдиги болуп эсептелет.

Символдордун коддук таблицасы 256 позициядан турат, б.а. ар бир символ 0 дөн 256 га чейинки өзүнүн уникалдуу кодуна ээ болот. Эгерде символдун коду N болсо, анда ал символду программада #N деп жазабыз. Эсте 1 символ 1 байт менен аныкталат. Таблицалардын символун төмөнкү маршрут боюнча көрүп алууга болот (1-сүрөт):

- *Пуск*
- *Программы*
- *Стандартные*
- *Служебные*
- *Таблица символов*



5.1-сүрөт. Символдордун таблицасы

Символдор менен иштөөдө негизинен Chr жана Ord функциялары колдонулат. Chr функциясы берилген код боюнча символду кайтарып берсе, Ord функциясы берилген символдун кодун аныктап берет.

5.1.1. Chr функциясы

Chr *функциясы берилген ASCII-коду боюнча символду аныктап берет.*

Функциянын синтаксиси:

Chr (<код>:Byte):Char;

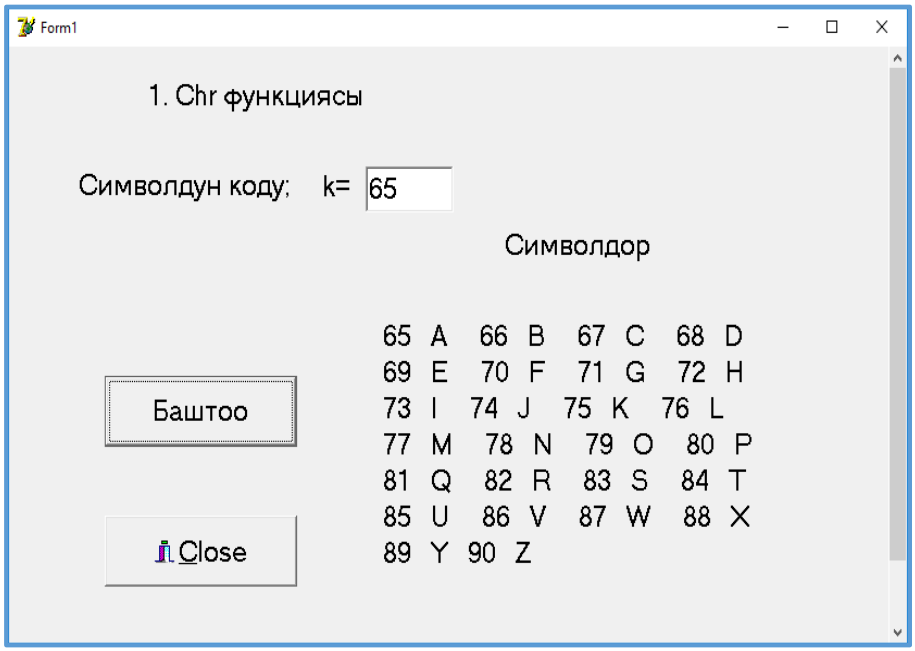
Функцияга кайрылуу форматы төмөнкүдөй:

Chr (<код>);

Функциянын аргументи **byte** тибиндеги сан, ал эми функциянын мааниси - **char** тибиндеги символ болот.

5.1-мисал. Англис тилинин алфавитин экранга чыгаргыла.

Маселени чыгаруу үчүн Chr функциясын пайдаланабыз. 5.1-сүрөттө көрсөтүлгөндөй колдонмонун башкы бетин түзөбүз. Англис тилинин алфавитиндеги А тамгасынын коду 68 ге барабар болгондуктан, формага бул санды кийирип, «Баштоо» баскычын басканыбызда алвафиттин бардык тамгалары экранга чыгарылат (5.1-листинг).



5.1-сүрөт. Коду боюнча символдорду чыгаруу

5.1-листинг. *Коду боюнча символдорду чыгаруу*

```
procedure TForm1.Button1Click(Sender: TObject);
var s:string;
    i,j,k:integer;
begin
  k:=StrToInt(Edit1.Text);
  For i:=0 to 5 do  // Жолчолордун саны
  begin
     For j:=k+i to k+i+3 do  // Мамычалардын саны
     begin
```

```
      s:=s+IntToStr(j)+'  '+chr(j)+'    ';
    end;
    s:=s+chr(13);
    k:=k+3;
  end;
    s:=s+IntToStr(89)+'  '+chr(89)+'  '+IntToStr(90)+'  '+chr(90);
Label4.caption:=s;
end;
```

5.1.2. Ord функциясы

Ord функциясы берилген символдун ASCII-кодун аныктап берет.

Функциянын синаксиси:

Ord (<символ> : char) : integer;

Функцияга кайрылуу форматы төмөнкүдөй:

Ord (<символ>);

Функциянын аргументи **char** тибиндеги символ, ал эми функциянын мааниси - **integer** тибиндеги бүтүн сан болот.

```
ShowMessage('A = '+IntToStr(Ord(A))); // A=65
ShowMessage('C = '+IntToStr(Ord(C))); // C=67
```

§ 5.2. Жолчолор үчүн стандарттык функциялар

Delphi тилинде жолчолор менен иштөөчү стандарттык функциялар бар.

5.2.1. Length функциясы

Length *функциясы жолчонун узундугун, б.а. жолчодогу символдордун санын аныктап берет.*

Функциянын синаксиси:

Length (<жолчо>: string): integer;

Функцияга кайрылуу форматы төмөнкүдөй:

length(<жолчо>);

Функциянын аргументи катары жолчо же жолчо тибиндеги туюнтма боло алат. Функциянын мааниси - жолчодогу символдордун санын аныктап турган сандык маани болот.

Мисалы,

d:=length('Асанов');

болсо, анда d өзгөрмөсүнүн мааниси 6 га барабар болот.

5.2.2. Pos функциясы

Pos *функциясы саптагы жолчонун позициясын аныктайт*. Функциянын синаксиси:

Pos(<сап>: string; <жолчо>: string): integer;

мында <сап> – string тибиндеги берилген жолчолордун жыйындысы, <жолчо> – string тибиндеги изделүүчү жолчолук константа же өзгөрмө.

Pos – функциясы изделүүчү жолчонун саптагы биринчи кездешүүсүнүн позициясын аныктайт жана анын мааниси integer тибиндеги бүтүн сан болот.

Мисалы

p:= pos('ша', 'Бишкек шаары');

инструкциясы аткарылгандан кийин p өзгөрүлмөсүнүн мааниси 8 ге барабар болот. Эгерде сапта мындай жолчо жок болсо, анда pos() функциясынын мааниси нөлгө барабар болот.

5.2.3. Copy функциясы

Copy *функциясы саптагы жолчонун фрагментин бөлүп алат.*

Функциянын синаксиси:

Copy(<жолчо>: string; p, n : integer) : string;

мында <жолчо> – string тибиндеги берилген сап;

- p – бөлүнүп алына турган жолчонун биринчи символунун номери;
- n – бөлүнүп алына турган жолчонун узундугу.

Copy функциясы берилген жолчонун p-позициясынан баштап n символду бөлүп (копиялап) алат жана анын мааниси string тибиндеги жолчо болот.

Мисалы

```
st:= 'Инженер Акматов';
fam:=copy(st, 9, 7);
```

инструкциясы аткарылганда fam өзгөрмөсүнүн мааниси 'Акматов'

болот.

5.2.4. Concat функциясы

Concat *функциясы бир нече жолчолорду бириктирет*.

Функциянын синтаксиси:

Concat (<1-жолчо>, <2-жолчо>,...,<N-жолчо>: string):string;

мында <1-жолчо>,...,<N-жолчо> – string тибиндеги берилген жолчолор. Функциянын мааниси string тибиндеги жолчо болот.

5.2-листингде Concat функциясын колдонуу мисалы келтирилген.

5.2-листинг. *Эки сөзду бириктирүү*

```
var s: string;
begin
    s:= Concat('Асанов', 'Акмат', 'Ошто жашайт');
    // Натыйжада s := 'Асанов Акмат Ошто жашайт' болот.
end;
```

5.2.5. Trim функциясы

Trim функциясы жолчонун башындагы жана акырындагы боштуктарды (пробелдерди) жана башкаруучу символдорду өчүрөт.

Функциянын синтаксиси:

Trim (<жолчо>:string):string;

мында <жолчо> – string тибиндеги берилген жолчо, ал эми функциянын мааниси string тибиндеги жолчо болот.

5.3-листинг. *Trim функциясын колдонуу*

```
var   s1, s2:string;
      l1,l2:integer;
begin
      s1:=#13' Îø ';
      l1:= length(s1);  // l1 := 5
      s2:= Trim(s1);          // s2 := 'Îø'
      l2:= l1-length(s2);     // l2 := 3
      Label1.Caption:=IntToStr(l1)+' '+IntToStr(l2);
      Label2.Caption:=s2;
end;
```

5.2.6. Delete процедурасы

Delete процедурасы жолчонун бир бөлүгүн өчүрөт.

Процедуранын синаксиси:

Delete (<жолчо>: string; n, m: integer);

Процедурага кайрылуу төмөнкүдөй болот:

delete(<жолчо>, n, m);

мында:

- <жолчо> – жолчолук типтеги өзгөрмө же константа;
- n – ушул символдон баштап жолчонун бөлүгү өчүрүлөт;
- m – өчүрүлүүчү жолчонун узундугу.

Мисалы, төмөнкү

```
s:='Бишкек шаары';
delete(s,7,5);
```

инструкциялар аткарылгандын кийин s өзгөрмөсүнүн мааниси 'Бишкек' болуп калат.

5.2.7. Insert процедурасы

Insert *процедурасы бир жолчону экинчи жолчого берилген позициядан баштап койот.*

Процедуранын синтаксиси:

Insert (<жолчо>: string; <сап>: string; n : integer);

мында

- <жолчо> – коюлуучу жолчо;
- <сап> – коюла турган сап;
- n – коюла турган жолчонун позициясынын номери.

5.3-листинг. *Insert процедурасынын жардамында саптын ортосуна сапчаны коюу.*

```
var
  Target : string;
begin
  Target := '12345678';
  Insert('-+-', Target, 3);
  ShowMessage('Target : '+Target);
end;
```

6-глава. Функциялар жана процедуралар

§ 6.1. Подпрограмма жана анын түрлөрү

Программалоо тилдеринде маселелерди чечүү үчүн белгилүү бир инструкциялардын тобун бир нече жолу өзгөртпөстөн кайталоого жана аларды программанын ар кайсы бөлүгүндө аткарууга туура келет. Ошондуктан мындай учурларда түзүлгөн программанын ыкчам иштеши үчүн подпрограмма концепциясы колдонулат.

Delphi тилинде да негизги программалык бирдик болуп подпрограмма эсептелет.

Подпрограмма *деп атайын атка ээ болгон жана белгилүү бир маселени чечүү үчүн дайындалган инструкциялардын жыйындысын айтабыз.*

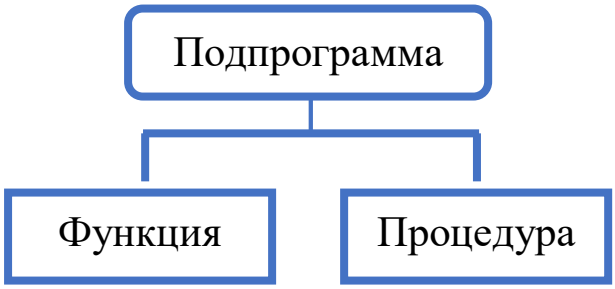
Ар бир подпрограммага атайын ***ат*** берилет жана ал подпрограмманын ***параметрлери*** болот. Подпрограмманын аты аны чакыруу үчүн, ал эми параметрлер маанилерди киргизүү үчүн колдонулат.

Подпрограмманын параметрлери ***формалдуу*** жана ***чыныгы*** деп бөлүнөт. **Формалдуу параметрлер** подпрограмманын жарыялоодо бөлүмүндө жаазылат, ал эми подпрограмманы чакырганда көрсөтүлгөн параметрлер ***учурдагы (чыныгы)*** деп аталат.

Подпрограмма негизги программанын жарыялоо бөлүмүндө жарыяланышы керек. Подпрограмманы негизги программанын каалаган бөлүгүндө чакырып алса болот.

Подпрограмма ***бөрктөн*** жана ***телодон*** турат. Подпрограмманын телосу жарыялардан жана аткарылуучу инструкциялардын удаалаштыгынан турат.

Подпрограмманын эки түрү бар: ***функция жана процедура***.



6.1-схема. Подпрограмманын түрлөрү

6.2-схема. Функциянын түрлөрү

§ 6.2. Функция жана анын түрлөрү

Delphi тилинде функциялар эки түргө бөлүнөт (2-схема).

Delphi системасына коду тиркелген көптөгөн стандарттык функциялар жана процедуралар бар:

- ***Математикалык функциялар:*** abs(x), sin(x), cos(x), tan(x), arctan(x), exp(x), ln(x), sqr(x), sqrt(x), int(x), arcos(x), random(x), ...;
- ***Жолчолорду иштетүү функциялары:*** length(S), concat(S1, S2), copy(S, n, m), delete(S, n, m),...
- ***Типтерди өзгөртүү функциялары*** (IntToStr, StrToint, FloatToStr, StrToFloat,...);

Бирок көп маселелерди чечүүдө стандарттык функциялар жана процедуралар жетишсиз болот, ошондуктан программист тарабынан функцияларды жана процедураларды аныктоого туура келет.

§ 6.3. Функция жана анын структурасы

Функция *деп атайын атка ээ болгон жана маселени чечүүдө бир эле маанини аныктоо үчүн дайындалган инструкциялардын жыйындысын айтабыз.*

Функциянын структурасы төмөнкүдөй: ал бөрктөн жана телодон турат. Функциянын телосу константаларды, типтерди, өзгөрмөлөрдү жарыялоо бөлүмдөрүнөн жана инструкциялар бөлүмүнөн турат.

Жалпы учурда функция төмөнкүдөй жарыяланат:

function ***F*** **(*x1:T1; x2:T2; ...; xN:TN*) : *T*;** // функциянын бөркү

const // константаларды жарыялоо бөлүмү

type // типтерди жарыялоо бөлүмү

var // өзгөрмөлөрдү жарыялоо бөлүмү

// инструкциялар бөлүмү

begin
I1; *//* 1-инструкция
I2; *//* 2-инструкция
...
IN; *//* N-инструкция
F := маани; // маани функциянын аты менен байланыштырылат
end;

мында ***F*** *– функциянын аты,*
x1; x2; ..., xN *– формалдуу параметрлер,*
T1, T2, ..., TN *- формалдуу параметрлердин типтери,*
T *– функциянын тиби,*
I1, I2, ..., IN *– инструкциялар*

6.3-схема. Функцияны жарыялоо структурасы

Функциянын бөркү **function** сөзүнөн башталат, функциянын аты жазылат, тегерек кашаанын ичине параметрлердин тизмеси типтери менен кошо көрсөтүлөт, кош чекит коюлуп, функциянын тиби көрсөтүлөт жана акырында үтүрлүү чекит коюлат.

Бөрктөн кийин константаларды, типтерди, өзгөрмөлөрдү жарыялоо бөлүмдөрү жана инструкциялар бөлүмү жазылат.

Инструкциялар бөлүмүндөгү эң акыркы инструкцияда функциянын атына алынган негизги жыйынтык ыйгарылышы керек.

§ 6.4. Функцияны чакыруу жана колдонуу

Функцияны чакыруу үчүн функциянын атын жазып, андан кийин тегерек кашаалардын ичине чыныгы параметрлерди үтүрлүү чекит менен ажыратып жайгаштырабыз жана эң акырында үтүрлүү чекит коёбуз:

***F* (*a1; a2; ...; aN*);**

мында F – функциянын аты,
a1; a2; ... aN – учурдагы параметрлер, алар
x1; x2; ..., xN - формалдуу параметрлерин алмаштырат.

6.4-схема. Функцияны чакыруу

6.1-мисал. *Программа аткарылып жатканда киргизилген a,b,c сандарынын арифметикалык орточо маанисин функция түшүнүгүн колдонуп аныктагыла.*

Чыгаруу. a, b, c сандарынын арифметикалык орточо маанисин аныктоо үчүн функция түзүп алабыз (4-схема).

```
function f_ort (x1, x2, x3 : real) : real;
    begin
        f_ort := (x1+x2+x3)/3;
    end;
```

6.5-схема. Орточо маанини аныктоочу функция

мында f_ort – функциянын аты, x1, x2, x3 – формалдуу параметрлер, арифметикалык орточо маани функциянын атына ыйгарылат.

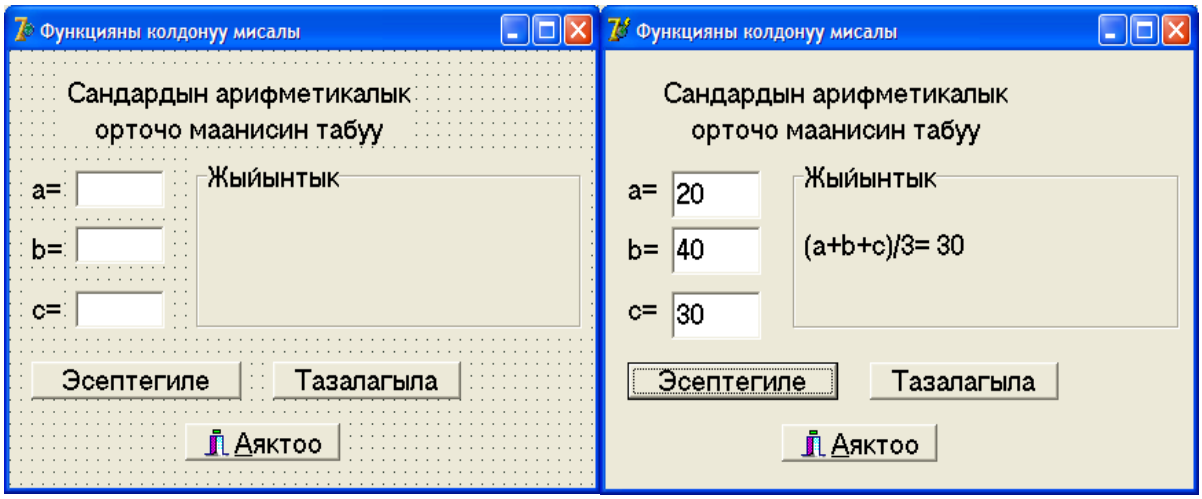
Бул функцияны колдонуу үчүн 9.1-сүрөттө көрсөтүлгөндөй моделди түзүп алабыз.

6.1-листинг. Программанын листинги төмөнкүдөй.

```
unit Un_a_ort;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, Buttons, StdCtrls;
type
        TForm1 = class(TForm)
          Label1: TLabel;
          Label2: TLabel;
          Label3: TLabel;
          Label4: TLabel;
          Edit1: TEdit;
          Edit2: TEdit;
          Edit3: TEdit;
          Button1: TButton;
          Button2: TButton;
          BitBtn1: TBitBtn;
          GroupBox1: TGroupBox;
          Label5: TLabel;
          procedure Button1Click(Sender: TObject);
          procedure Button2Click(Sender: TObject);
  private
          { Private declarations }
  public
          { Public declarations }
  end;
var
        Form1: TForm1;
implementation
       {$R *.dfm}
       function f_ort (x1,x2,x3:real):real;
        begin
           f_ort:=(x1+x2+x3)/3;
        end;
procedure TForm1.Button1Click(Sender: TObject);
       var
        a,b,c,s:real;
       begin
```

```
    a:=StrToFloat(Edit1.Text);
    b:=StrToFloat(Edit2.Text);
    c:=StrToFloat(Edit3.Text);
    s:=f_ort(a,b,c);
    Label5.Caption:='(a+b+c)/3= '+FloatToStr(s);
  end;

procedure TForm1.Button2Click(Sender: TObject);
begin
      Edit1.Text:='';
      Edit2.Text:='';
      Edit3.Text:='';
      Label5.Caption:='';
end;
```

procedure TForm1.Button1Click(Sender: TObject) процедурасындагы a, b, c - өзгөрмөлөрү учурдагы параметрлер болуп, f_ort(x1, x2, x3) функциясы чакырылганда x1, x2, x3 - өзгөрмөлөрү менен алмаштырылат. f_ort(x1, x2, x3) функциясы арифметикалык орточо маанини аныктайт жана ал маани s өзгөрмөсүнө ыйгарылат, ал маани Label5.Caption:='(a+b+c)/3= '+FloatToStr(s1); инструкциясы менен монитордун экранына чыгарылат.



6.1-сүрөт. Функцияны колдонуу мисалы

§ 6.5. Функцияны табуляциялоо

Функцияны табуляциялоо деп белгилүү бир туруктуу кадам менен эсептелген функциянын маанилеринин таблицасын түзүү процессин айтабыз.

Мисал катарында бир өзгөрмөлүү функцияны табуляциялоо маселесин карайбыз, б.а. $y = f(x)$ функциясынын $[a, b]$ сегментиндеги h кадамы менен эсептелген маанилеринин таблицасын түзөбүз. Ал үчүн $[a, b]$ сегментин n бөлүккө бөлүп, h кадамын төмөнкүдөй тандап алабыз: $h = \frac{b-a}{n}$. $[a, b]$ сегментиндеги чекиттерди төмөнкүдөй белгилеп алабыз: $x_0 = a, x_1 = x_0 + h, x_2 = x_0 + 2h, \dots, x_i = x_0 + ih,\ x_n = x_0 + nh = b$.

Бул учурда функциянын маанилери төмөнкүдөй аныкталат: $y_i = f(x_i), i = 0, \dots, n$.

Функциянын маанилеринин таблицасын түзүү үчүн параметрлүү циклди колдонууга болот.

6.2-мисал. $y = \sum_{x=0}^{n} x^2 + x + 8$ *чектүү суммасын эсептөө программасын түзөлү.*

Программаны түзүүдө жогоруда айтылгандай параметрлүү циклди жана фукцияны колдонобуз.

```
function f(x:integer):integer;
    begin
        result:=sqr(x)+x+8;
    end;
```

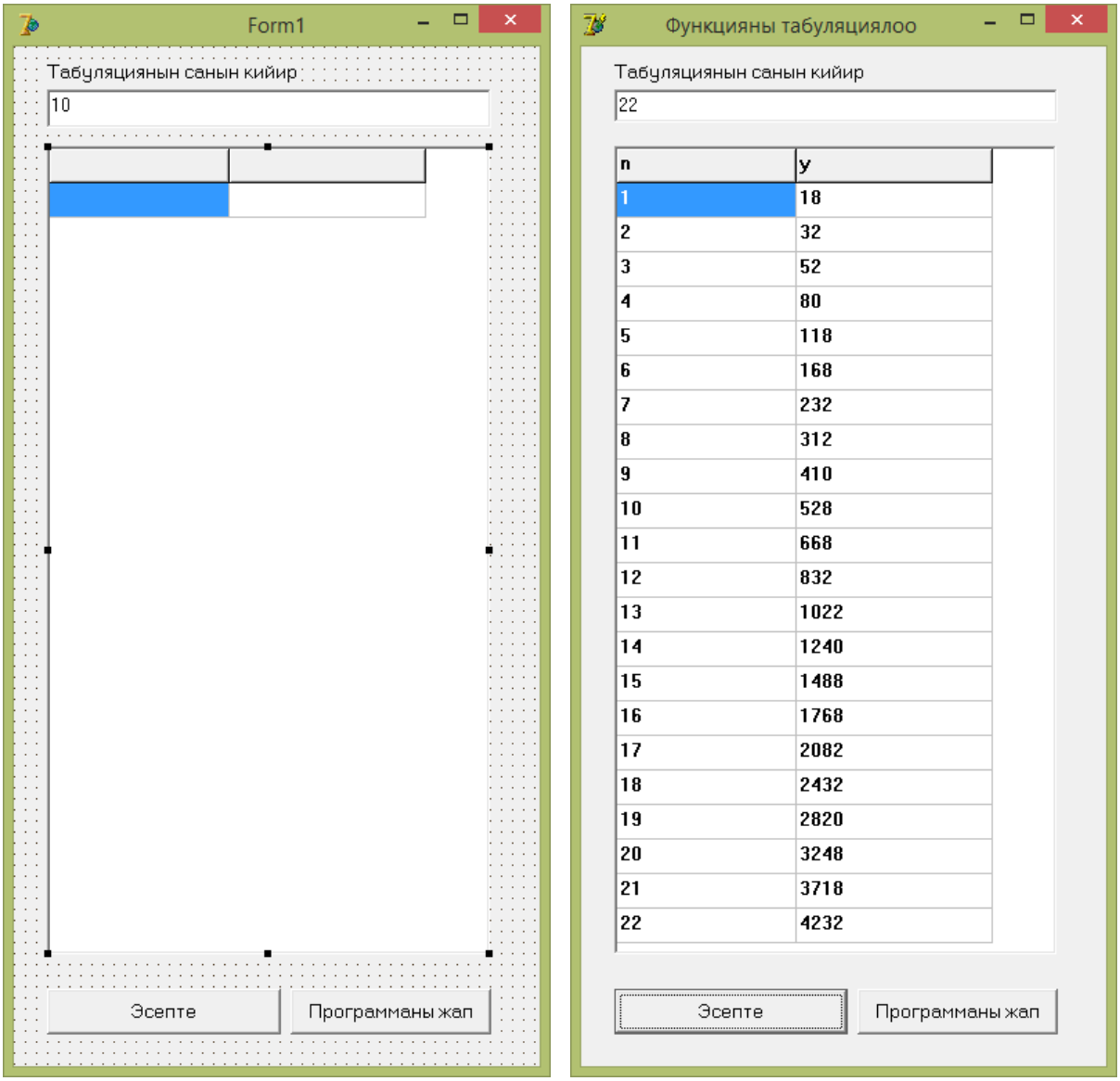
6.5-схема. Орточо маанини аныктоочу функция

6.2-листинг. Программанын листинги төмөнкүдөй.

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, StdCtrls, Grids;

type
  TForm1 = class(TForm)
```

```
    StringGrid1: TStringGrid;
    Button1: TButton;
    Button2: TButton;
    Label1: TLabel;
    Edit1: TEdit;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
function f(x:integer):integer;
 begin
  result:=sqr(x)+x+8;
 end;

procedure TForm1.Button1Click(Sender: TObject);
var n,i,j:integer;
    y:longint;
begin
n:=StrToInt(Edit1.text);
StringGrid1.RowCount:=n;
StringGrid1.Cells[0,0]:='n';
StringGrid1.Cells[1,0]:='y';
 for i:=1 to n do
 begin
  y:=0;
  for j:=0 to i do
   y:=y+f(j);
 StringGrid1.Cells[0,i]:=IntToStr(i);
 StringGrid1.Cells[1,i]:=IntToStr(y);
```

```
 end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
close;
end;

end.
```



6.2-сүрөт. Функцияны табуляциялоо

§ 6.6. Процедуралар

Процедуранын структурасы. Процедура подрограмманын бир түрү болуп эсептелет. Ал эки учурда колдонулат.

1). Подпрограмма эч кандай маанини кайтарып бербеген учурда;

2). Подпрограмма экиден көп маанини кайтарып берген учурда.

Жалпы учурда анын структурасы төмөнкүдөй:

procedure *P* (*var x1:T1; var x2:T2; ...; var xN:TN*); // бөрк

const // Константаларды жарыялоо бөлүмү

type // Типтерди жарыялоо бөлүмү

var // Өзгөрмөлөрдү жарыялоо бөлүмү

// Инструкциялар бөлүмү

begin
I1; // 1-инструкция
I2; // 2-инструкция
...
IN; // N-инструкция
end;

мында ***P*** *– процедуранын аты,*
x1; x2; ..., xN *– формалдуу параметрлер,*
T1, T2, ..., TN *- формалдуу параметрлердин типтери,*
I1, I2, ..., IN – *инструкциялар.*

6.6-схема. Процедураны жарыялоо структурасы

6.2-мисал. *Банкка салынган акчадан түшкөн кирешени эсептегиле.*

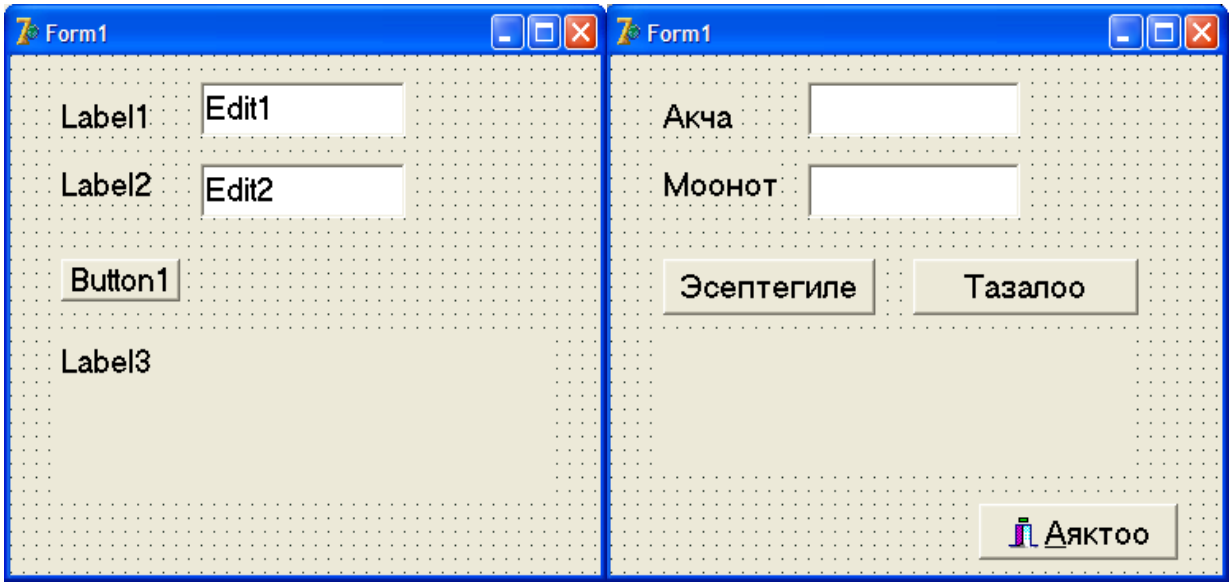
Программаны түзүүдө **FloatToStrF(x, f, k, n)** стандарттык функциясын колдонобуз. Бул функция **x** чыныгы санынын символдук сүрөттөлүшүн берет. Мында **f** – форматты аныктайт,

мисалы **ffCurrency** – финансылык формат, **k –** тактыкты аныктайт, б.а. цифралардын жалпы саны, **n** – ондук чекиттен кийинки цифралардын саны.

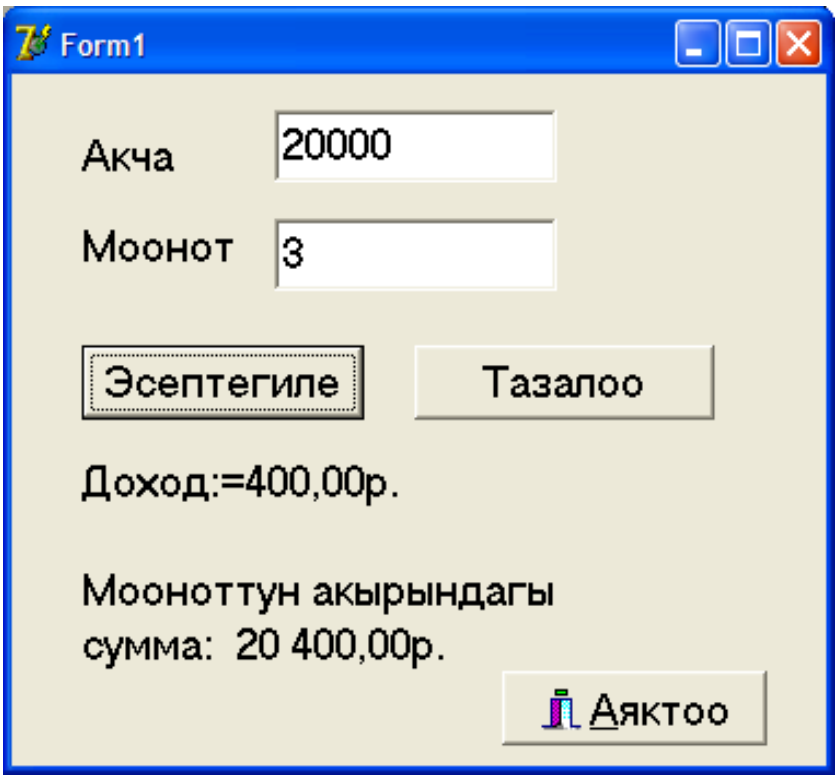
6.2-листинг. *Кирешени эсептөө.*

```
unit Un_dohod;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
Dialogs, Buttons, StdCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Button1: TButton;
          Edit1: TEdit;
          Edit2: TEdit;
          Label3: TLabel;
          BitBtn1: TBitBtn;
          Button2: TButton;
          procedure Button1Click(Sender: TObject);
          procedure Button2Click(Sender: TObject);
        private
          { Private declarations }
        public
          { Public declarations }
        end;
      var
        Form1: TForm1;
      implementation
      {$R *.dfm}

procedure vklad(sum1: real; t:integer;var pr:real;var sum2:real);
var
        p:real;
  begin
          case t of
          1..3 : p:=0.08; // 8 процент
```

```
          4..6 : p:=0.085; // 8.5 процент
          else
              p:=0.1; // 10 процент
  end;
        pr:=sum1*(p/12); // доход
        sum2:=sum1+pr*t;   // мөөнөттүн акырындагы сумма
        end;

procedure TForm1.Button1Click(Sender: TObject);
var
        pr1:real;
        s1,s2:real;
        t1:integer;
begin
        s1:=StrToFloat(Edit1.Text);
        t1:=StrToInt(Edit2.Text);
        vklad(s1,t1,pr1,s2);
        Label3.Caption:='Баштапкы сумма: '+FloatToStr(s1)+#13+
        '1 айдагы киреше:='+FloatToStrF(pr1,ffCurrency,6,2)+#13+
        'Мөөнөттүн акырындагы'+ #13+'сумма:  '+FloatTo-
StrF(s2,ffCurrency,6,2);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
       Edit1.Text:='';
       Edit2.Text:='';
       Label3.Caption:='';
end;
end.
```



6.2-сүрөт. Кирешени эсептөө формасы



6.3-сүрөт. Процедураны колдонуу мисалы

7-глава. Файлдар

§ 7.1. Файлды жарыялоо

Файл деп атайын атка (ысымга) жана кеңейтилишке ээ болгон берилгендердин бирдей типптеги удаалаштыгынан турган физикалык эстин бөлүгүн айтабыз.

Файлдын элементтеринин саны физикалык эстин өлчөмүнө жараша чектелет.

Delphi программалоо тилинде программада алынган жыйынтыктарды клавиатураны колдонбостон туруп эле дискке сакталуучу файлдарга жазууга, файлдагы маалыматтарды окууга жана аларды кайра иштетүүгө болот.

Файлдар негизинен үч типке бөлүнөт:

- *Тексттик файлдар (тексттик файлдагы информациялар ASCII символдорунда сакталат);*
- *Типтештирилген файлдар (файлдагы берилгендерди тиби аныкталган, б.а. алардын тиби бүтүн, чыныгы, символдук, логикалык же жолчолук типте болушу мүмкүн);*
- *Типтештирилбеген (экилик) файлдар (файлдардын универсалдык форматы: графиктик, аудио- жана видеофайлдар, электрондук таблицалар, HTML, ж.б. файлдар).*

Тексттик жана типтештирилген файлдар экилик файлдардын айрым бир учуру болуп эсептелет.

Берилгендердин структурасы сыяктуу эле файл дагы өзгөрмөлөрдү жарыялоо бөлүмүндө жарыяланышы керек.

1). Тексттик файлдарды жарыялоо 7.1-схемада берилди.

f: TextFile;

мында ***f*** *– файлдык өзгөрмөнүн аты,*
TextFile *– тексттик файлдын тиби.*

7.1-схема. Тексттик файлды жарыялоо

2). Типтештирилген файлдарды жарыялоо 7.2-схемада көрсөтүлдү.

f: file of T;

мында ***f*** *– файлдык өзгөрмөнүн аты,*
T *– файлдын тиби.*

7.2-схема. Типтештирилген файлды жарыялоо

Мисалы:

f: ***file of*** *integer; // бүтүн сандардын файлы*
koef: ***file of*** *real; // чыныгы сандардын файлы*
res: ***file of*** *char; // символдор файлы*

3). Типтештирилбеген (экилик) файлдар төмөнкүдөй жарыяланат (7.3-схема):

f: file;

мында ***f*** *– файлдык өзгөрмөнүн аты.*

7.3-схема. Экилик файлды жарыялоо

§ 7.2. Файлдык өзгөрмө

Файлдык өзгөрмөнү файлдын аты менен байланыштыруу. Файлдын ичиндеги маалыматтарды окууга же файлга маалыматтарды жазуу үчүн файлдык өзгөрмөнү конкреттүү файл менен байланыштыруу керек.

Файлдын аты Windows операциялык системасында кабыл алынган эрежелер боюнча берилет. Файлдын аты толук болушу мүмкүн, б.а. файлын аты менен кошо файлга баруучу жол – дисктин аты, каталог жана камтылуучу каталогдор жалгыз апострофтун ичине жазылып көрсөтүлүшү мүмкүн.

AssignFile процедурасы файлдык өзгөрмөнү конкреттүү файл менен байланыштыруу үчүн кызмат кылат жана анын жазылышы 7.4-схемда көрсөтүлдү.

AssignFile(f, ФайлдынАты);

мында f – файлдык өзгөрмөнүн аты.

7.4-схема. Файл менен байланыштыруу

Мисалдар:

```
AssignFile(f, 'c:\result.txt');
AssignFile(f, 'c:\pogoda.txt');
fname:=('usd.txt');
AssignFile(f, fname);
```

Эгерде файлга баруучу жол көрсөтүлбөсө, анда файл аткарылуучу файл сакталган каталогдо жатат деп түшүнүү керек.

§ 7.3. Тексттик файлдар

Тексттик файлдар менен иштөө. Файлдык өзгөрмө менен конкреттүү файлды байланыштыргандан кийин төмөнкү маселелерди чечүүгө болот:

- ***Маалыматтарды файлга жазуу;***
- ***Файлдан маалыматтарды окуу.***

Файлдар менен иштөө үчүн *AssignFile();Rewrite(), Append(), Reset(), Writeln(), Readln(), CloseFile() процедуралары колдонулат.*

1). ***Rewrite(f)*** процедурасы жаңы файлды түзүү менен файлды ачат жана ал маалыматтарды файлга жазуу же кайра жазуу үчүн колдонулат.

Мисалы: *AssignFile(f, 'result.txt'); Rewrite(f);* инструкциялары менен *result.txt* файлы түзүлүү менен ачылат жана ага маалыматтарды жазуу үчүн же мурда бар болгон *result.txt* файлын ачып, ага маалыматтарды кайра жазуу үчүн колдонууга болот.

2). ***Append(f)*** процедурасы менен бар файл ачылат жана ал маалыматтарды файлдын акырына кошуп жазуу үчүн колдонулат.

Мисалы,

AssignFile(f, 'result.txt'); Append(f);

инструкциялары менен *result.txt* файлы ачылат жана файлдын акырына маалыматтарды кошуп жазуу үчүн шарт түзүлөт.

3). ***Reset(f)*** процедурасы менен файл маалыматтарды файлдан окуу үчүн гана ачылат. Мисалы,

AssignFile(f, 'result.txt'); Reset(f);

инструкциялары менен *result.txt* файлындагы маалыматтарды окуу үчүн колдонууга болот.

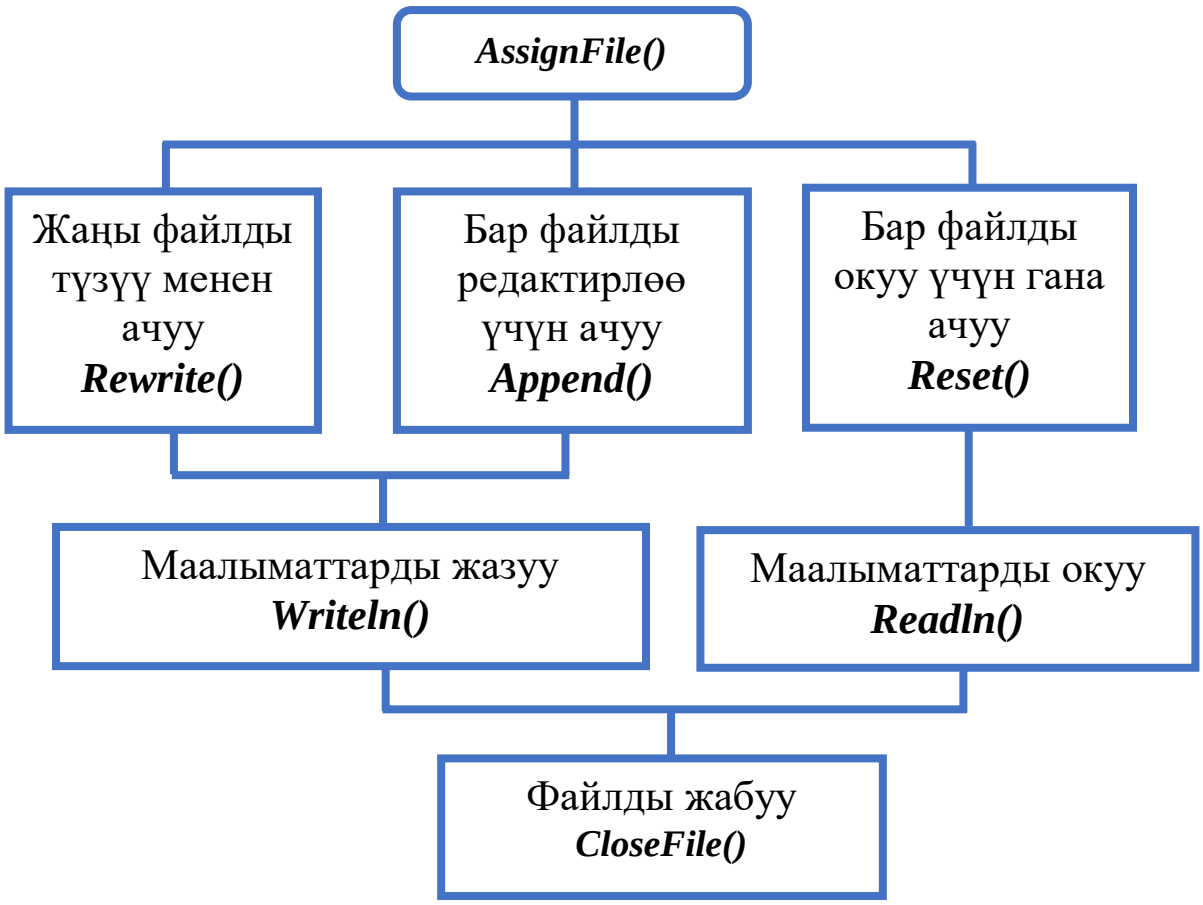
4). *Writeln() жана Readln()* процедуралары тиешелеш түрдө маалыматтарды жазуу жана окуу үчүн колдонулат.

5). *CloseFile(f)* процедурасы ачылган файлды жабуу үчүн колдонулат. Мисалы,

AssignFile(f, 'result.txt'); CloseFile(f);

инструкциялары файлды жабуу үчүн колдонууга болот

Файлдар менен иштөөнү 7.5-схема схема менен көрсөтүүгө болот.

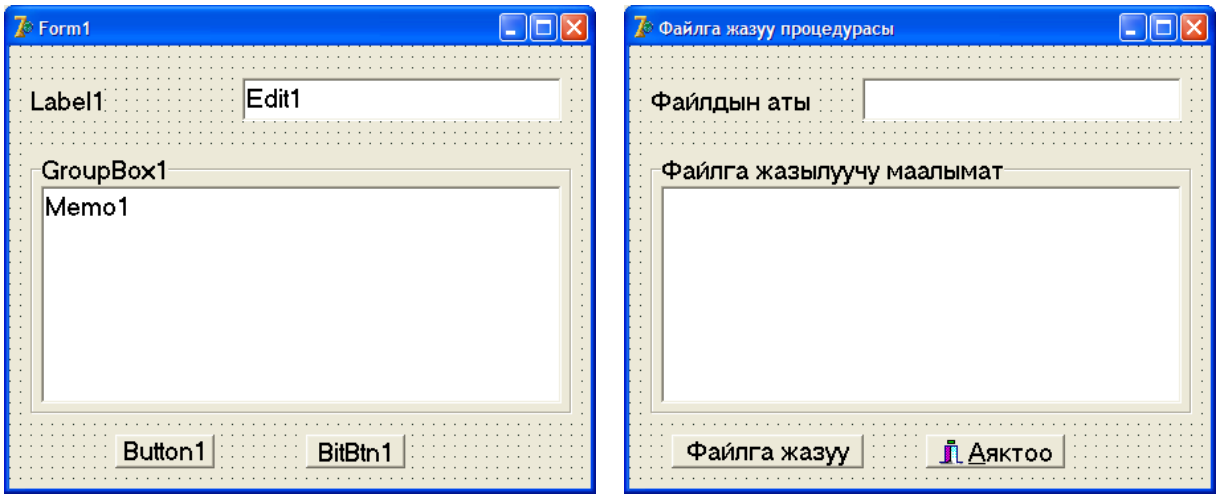


7.5-схема. Файлдар менен иштөө алгоритми

.

7.1-мисал. *Студенттердин тизмесин файлга жазуу программасын түзгүлө.*

Чыгаруу. Маселени чечүү үчүн төмөнкүдөй макет түзүп алабыз (1-сүрөт). Ал үчүн сүрөттө көрсөтүлгөн компоненталарды формага жайгаштырып, касиеттерин саздайбыз (настройкалайбыз).



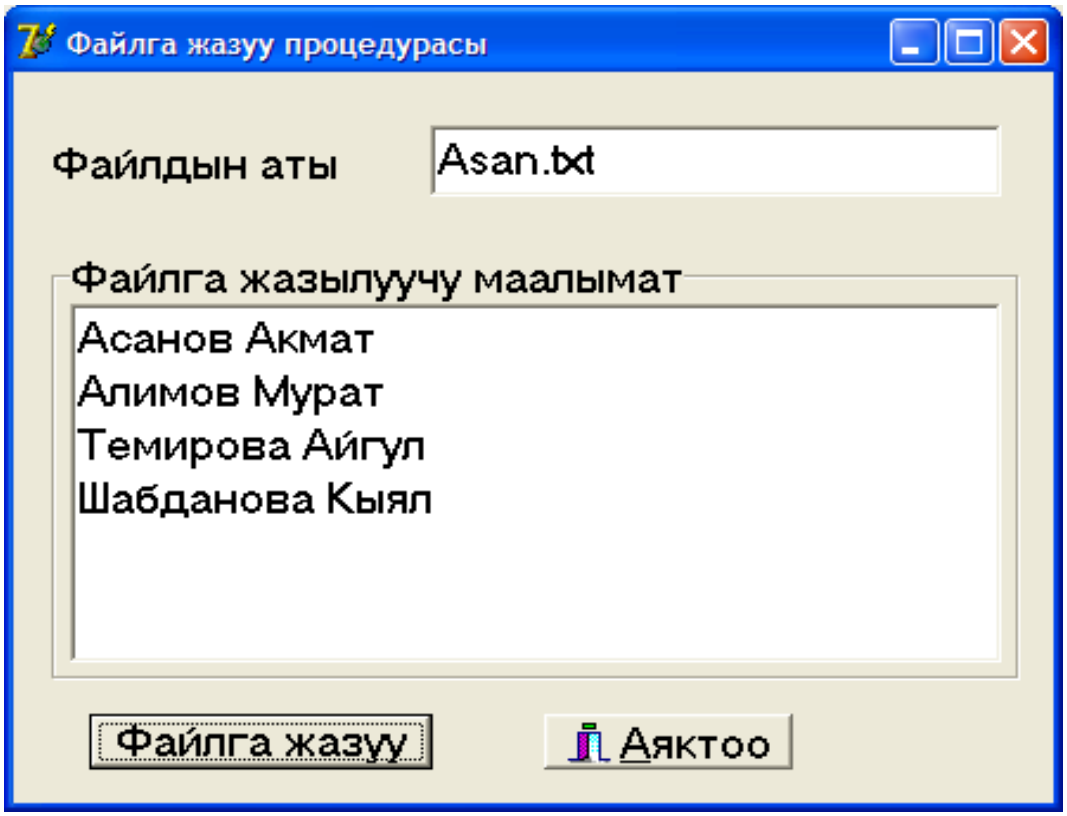
7.1-сүрөт. Файлга жазуу

“Файлга жазуу” баскычына эки чыкылдатып, төмөнкүдөй программа түзөбүз.

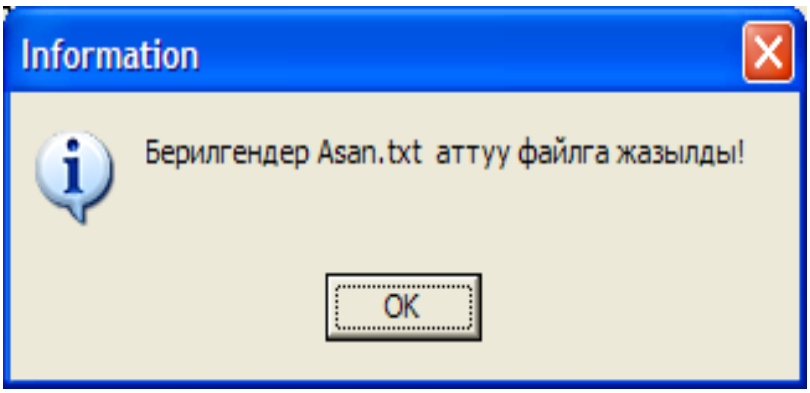
7.1-листинг. *Файлга жазуу процедурасы.*

```
procedure TForm1.Button1Click(Sender: TObject);
var     f : TextFile;
        fname:string;
        i : integer;
begin
        fname:=Edit1.Text;
        AssignFile(f, 'c:\'+ fname);
        Rewrite(f);
        for i:=0 to Memo1.Lines.Count do
          writeln(f, Memo1.Lines[i]);
          CloseFile(f);
          MessageDlg('Берилгендер '+fname+' аттуу файлга жазылды!', mtInformation,[mbOk],0);
end;
end.
```

Программа аткарылган учурда тиешелүү фамилияларды киргизип, “Файлга жазуу” баскычын басканда маалыматтар файлга жазылат (2-сүрөт) жана маалымат монитордун экранына чыгат (3-сүрөт).



7.2-сүрөт. Программа аткарылган учур



7.3-сүрөт. Аткарылган жыйынтык

8-глава. Консолдук колдонмолор

§ 8.1. Консолдук колдонмону түзүү

Эгерде программаны түзүүдө колдонуучунун катышышы зарыл болбосо, анда консолдук колдонмону пайдаланса болот.

Консоль деп – компьютердин эсине информацияларды кийирүүчү-чыгаруучу түзүлүштөр болгон дисплей, клавиатура жана чычканды айтабыз. Консолдук колдонмодо графиктик интерфейс болбойт.

Консолдук колдонмону түзүү үчүн **Console Application** деп аталган программалык коддордун атайын редакторун жүктөп алуу керек. Ал үчүн төмөнкү команданы аткарабыз:

File / New / Other…

Натыйжада **New Items** диалогдук терезеси ачылат. Бул терезеден **Console Application** деген пунктту тандоо менен программалык коддорду редактирлөө терезесин ачабыз. Бул терезеде консолдук колдонмонун проектисинин жалпы структурасын кармап турган даяр шаблон пайда болот (8.1-схема):

```
program Progect1; // Программанын бөркү
{$APPTYPE CONSOLE} // Компилятордун директивасы
uses   // Модулдарды жарыялоо бөлүмү
SysUtils;
      // Типтерди жарыялоо бөлүмү
      // константалар
      // өзгөрмөлөр
      // подпрограммалар
begin
      // Программанын телосу
end.  // Программанын аягы
```

8.1-схема. Консолдук колдонмодогу программанын структурасы

Консолдук колдонмодо программалоонун бардык негизги башкаруучу структуралары иштейт: ыйгаруу инструкциясы, шарттуу жана циклдик инструкциялар.

Консолдук колдонмолорду түзүүдө кийирүү (**read**) жана чыгаруу (**write**) операторлору көп колдонулат.

§ 8.2. Read операторy

Клавиатурадан информацияларды кийирүү үчүн **read** оператору колдонулат жана анын төмөнкүдөй форматтары бар:

read(x1,x2,…,xn);
readln(x1,x2,…,xn);
мында x1,x2,…,xn – кийирилүүчү өзгөрмөлөрдүн тизмеси

8.2-схема. read операторунун форматы

Эгерде программада аткаруу кезеги **read** операторуна келсе, анда аргументте көрсөтүлгөн өзгөрмөнүн мааниси киргизилмейинче аткаруу орундалбайт.

Сандык маанилерди кийирүүдө, мисалы эки санды кийирүүнү аткаруу үчүн алардын арасына жок дегенде бир бошту кту (пробелди), табуляция символун коюу керек же жолчонун акырын билдирүүчү баскычты (**Enter**) басуу керек. Акыркы маанини кийиргенден кийин да **Enter'**ди басуу керек.

readln оператору **read** операторуна окшош, бирок тизмедеги акыркы маани окулгандан кийин, кийинки кезектеги **readln** операторунун өзгөрмөлөрүнө кийирилүүчү маанилер жаңы жолчодон башталат.

§ 8.3. Write операторy

Информацияларды (сандарды, символдорду, жолчолорду, логикалык маанилерди) экранга чыгаруу үчүн **write** оператору колдонулат жана анын төмөнкүдөй форматтары бар:

write(x1,x2,…,xn);
writeln(x1,x2,…,xn);
мында x1, x2, … , xn – чыгарылуучу өзгөрмөлөр, константалар,туюнтмалар

8.3-схема. write операторунун форматы

write жана **writeln** операторлору берилгендердин бардыгын

(өзгөрмөлөрдү, константаларды, туюнтмаларды) экранга удаалаш чыгарып берет.

writeln оператору колдонулган учурда, берилгендер чыгарылып бүткөндөн кийин, курсор жаңы жолчого жылдырылат.

Чыныгы сандар **жылуучу чекит** (плавающая точка) формасында чыгарылат.

Чыныгы сандарды **фиксирленген чекит** формасында чыгаруу үчүн **форматталган чыгаруу** ыкмасын пайдалануу керек:

write(x1:n:m);

мында x1 – чыгарылуучу өзгөрмө, n – чыгаруу үчүн берилген талаанын узундугу (жалпы позициялардын саны), m – жалпы позициялардын ичинен бөлчөк бөлүк үчүн ажыратылган позициянын саны

8.4-схема. write оператору менен форматтуу чыгаруу

8.1-мисал. *Эки сандын суммасын эсептөө программасын консолдук колдонмо түзүү жолу менен түзгүлө.*

Маселени чыгаруу үчүн **File / New / Other…** командасы менен **Console Application** – консолдук колдонмолорду түзүү редакторун жүктөп алабыз жана программаны түзөбүз.

Төмөндө эки сандын суммасын эсептөө программасынын коду келтирилди жана жыйынтыгы көрсөтүлдү (8.1-сүрөт).

8.1-листинг. *Эки сандын суммасын табуу программасы*

```
program P_summa;
      {$APPTYPE CONSOLE}
uses
       SysUtils;
       var a,b,c:real;
begin
      // Эки сандын суммасын табуу программасы
      write('a=');
      read(a);
      write('b=');
      read(b);
```

```
    c:=a+b;
    write('c=',c:6:2);
    readln;
    readln;
end.
```

8.1-листинг аткарылганда эки сандын суммасын эсептөө программасы консолдо 8.1-сүрөттөгүдөй аткарылат.

8.1-сүрөт. Эки сандын суммасын табуу программасы

§ 8.4. Консолдук колдонмо үчүн Windows API функциялары

Консолдук колдонмодо маалыматтарды кийирүү, чыгаруудан башка дагы көптөгөн аракттерди аткарууга болот. Прогаммист үчүн 40ка жакын Windows API функциялары дисплей менен иштөө, чыгуучу символдордун атрибуттарын өзгөртүү, консолдун өлчөмүн өзгөртүү ж.б.у.с аракеттерди аткарууга жадам берет.

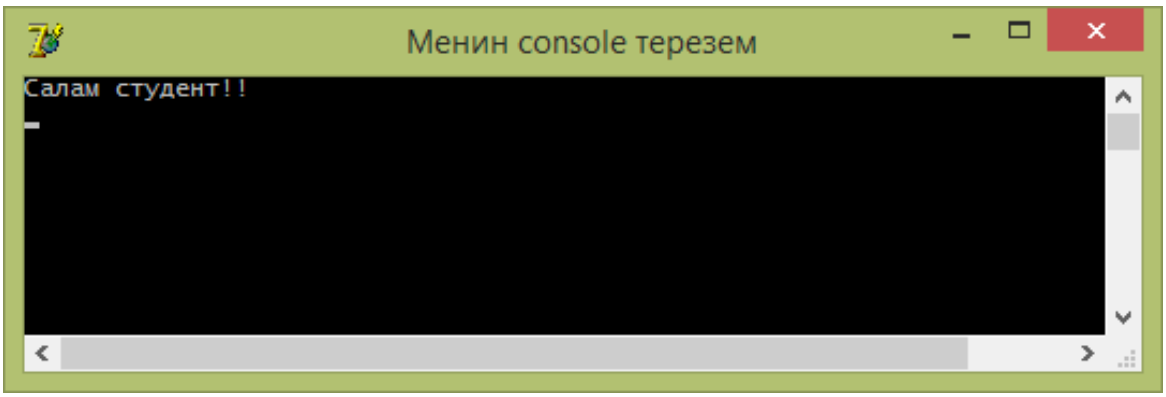
8.2-мисал. Консолдук терезенин бөркүндө чыгуучу текстти өзгөрткүлө.

Маселени чечүүдө API SetConsoleTitle функциясын колдонобуз.

8.2-листинг. *Консолдук терезенин бөркүн өзгөртүү*

```
program ConsoleApplication;
{$APPTYPE CONSOLE}
uses
  Windows,
  SysUtils;
{$R *.RES}
var
  s,sNewTitle, sErrMsg: string;
```

```
begin
  // Кириллица кодировкасын чакыруу
  SetConsoleCP(1251);
  SetConsoleOutputCP(1251);
  s:='Менин console терезем';

  // s жолчосун кириллица кодировкасына өзгөртүү
  CharToOEM(PChar(s),PChar(s));
  sNewTitle := s;

  // бөрктү өзгөртүүгө мүмкүн болбой калган учурда
  if not SetConsoleTitle(PChar(sNewTitle)) then
    begin
      sErrMsg := 'Unable to set caption - ' +
                                          SysErrorMessage(GetLastError);
      MessageBox(0, PChar(sErrMsg), 'Error',
                                    MB_ICONEXCLAMATION + MB_OK);
  end;

  Writeln('Салам студент!!');
  ReadLn;
end.
```



Бул программада эгерде бөрктү өзгөртүүгө мүмкүн болбой калса, анда **SetConsoleTitle** API функциясы **False** маанисин кайтарат жана **GetLastError** функциясы чыккан катанын сандык маанисин берет жана аны экранга чыгаруу үчүн **SysErrorMessage** функциясы тексттик сапка которот.

9-глава. Графика жана мультимедиа

"Графика" деген термин гректин «graphtin» деген сөзүнөн алынып, алгач жазууну, каллиграфияны, сызыктарды чийүүнү, сүрөт тартууну түшүндүргөн. Бирок, өндүрүштүк полиграфиянын өнүгүшү менен графика түшүнүгүнүн мааниси кеңейтилген.

Графика деп сүрөттөрдү, сүрөттөмөлөрдү, чиймелерди, графиктерди, схемаларды, диаграммаларды, түрдүү гравюраларды түзүү үчүн багытталган көркөм өнөрдүн түрүн айтабыз.

Delphi де графиктерди: схемаларды, чиймелерди, сүрөттөмөлөрдү чыгара турган программаны түзүүгө болот. Мындай программалар объекттин (форманын же Image компонентасынын) бетине графиканы чыгарып берет. Объекттин бети **canvas** касиетине туура келет.

Canvas (канвáс) деп растрдык эки ченемдүү сүрөттөмөлөрдү түзүү үчүн ылайыкталган бетти айтабыз.

Ошондуктан объекттин бетине графиктик элементти (түз сызыкты, айлананы, тик бурчтукту ж. б.) чыгаруу үчүн canvas касиетине тиешелүү методду колдонуу керек.

Мисалы, экранга тик бурчтукту чыгаруу үчүн төмөнкү команда берилет:

```
Form1.Canvas.Rectangle (10,10,100,100)
Объект  Касиет        Метод
```

§ 9.1. Графиктик бет

Delphi де графика төмөнкү объекттердин бетине чыгарылат:

- **Форма;**
- **Image компонентасы;**
- **PaintBox компонентасы.**

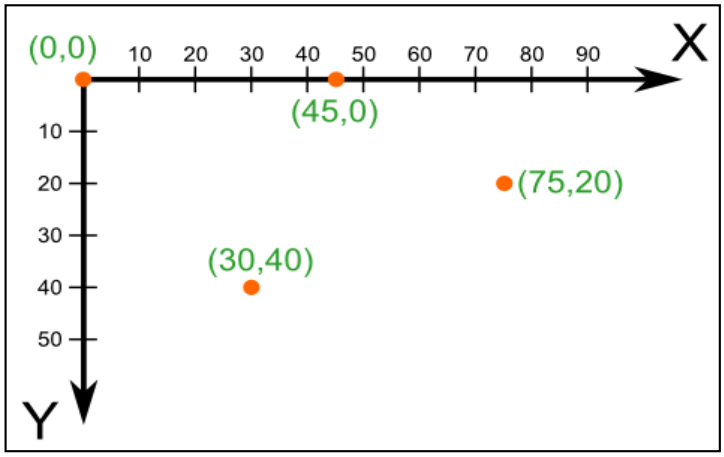
Графиктик элементтерди (түз сызык, айлана, тик бурчтук, ж.б.) объекттин бетине чыгаруу үчүн төмөнкү класстар колдонулат:

- **TCanvas (канвас, холст, чийүү бети);**
- **TPen (карандаш);**
- **TBrush (кисть);**
- **TFont (шрифт).**

Canvas объектинин касиети өзүнө чийүү талаасын,

карандаш, кист жана шрифт объекттерин камтуу менен формага каалагандай графиктик объекттерди – картинкаларды, көп бурчтуктарды, тексттерди ж.б. чыгарууга мүмкүнчүлүк түзүп берет.

Графиктик бет жекече чекиттерден – пикселдерден турат. Пикселдин абалы горизонталдык (X) жана вертикалдык (Y) координаталарынан турат. Сол жогорку пиксел (0,0) координатасына ээ. Координаталар жогортон төмөн жана солдон оңго карай өсөт (9.1-сүрөт).



9.1-сүрөт. Чийүү бетиндеги чекиттердин координаталары

Графиктик беттин өлчөмүн Image компонентасынын Height жана Width касиеттерине же форманын ClientHeight жана ClientWidth касиеттерине кайрылуу менен аныктаса болот.

Жөнөкөй графиктик элементтерди компонентанын (форманын же сүрөттөрдү чыгаруу областына) беттерине чыгаруу үчүн ушул компонентанын Canvas касиетине тиешелүү методдорду колдонуу керек.

§ 9.2. Сызыктар

Түз сызыктын кесиндисин чийүү үчүн LineTo методу колдонулат. Бул методду чакыруу инструкциясы төмөнкүдөй жазылат:

```
Form1.Canvas.LineTo(x,y);
```

LineTo методу чекиттин учурдагы абалынан чакыруу инструкциясындагы көрсөтүлгөн чекиттерге чейин түз сызыктын кесиндисин сызат. Берилген чекитти учурдагы чекит кылуу үчүн MoveTo методун колдонобуз.

```
Form1.Canvas.MoveTo(x,y);
```

Сызыктын көрүнүшү (түсү, жоондугу жана стили) графиктик беттин Pen объектисинин касиетинин маанилер менен аныкталат. 9.1-листингде түз сызыктын кесиндисин чийүү коду келтирилген.

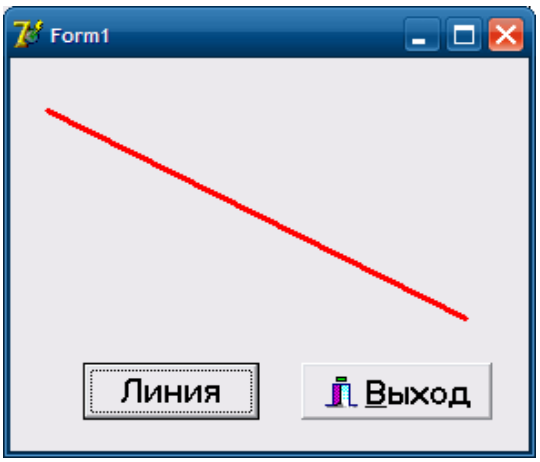


Рис. 9.2. Кесиндини чийүү

9.1-листинг. *Түз сызыктын кесиндисин чийүү коду*

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  With Form1.Canvas do
     begin
        MoveTo(20,30);// Учурдагы чекиттин абалы
        Pen.Width:=3; // Карандаштын жоондугу
        Pen.Color:=clRed; // Карандаштын түсү
        Canvas.LineTo(250,150); // Сызыкты чийүү
     end;
end;
```

§ 9.3. Карандаш

Карандаш классы чекитти, сызыктарды, геометриялык фигуралардын контурларын: тик бурчтуктарды, айланаларды, эллипстерди, жааларды ж.б. чийүү үчүн колдонулат.

Карандашка графиктик беттин **Pen** касиети туура келет. Сызыктын көрүнүшүнүн касиеттерин 9.1-таблицадан көрсө болот.

9.1-таблица. Карандаш объектинин касиеттери

Касиети	Аныктайт
Color	Сызыктын түсүн
Width	Сызыктын жоондугун
Style	Сызыктын түрүн

Color касиети карандаш менен сызылган сызыктын түсүн аныктайт. 9.2-таблицада color касиетинин маанилеринин константалары көрсөтүлгөн.

9.2-таблица. Color касиетинин маанилери

Константа	Цвет	Константа	Цвет
clBlack	Кара	clSilver	Күмүш түстүү
clMaroon	Каштан түстүү	clRed	Кызыл
clGreen	Жашыл	clLime	Ачык жашыл
clOlive	Оливка	clBlue	Көк
clNavy	Кара-көк	clFuchsia	Ачык-кызгылт
clPurple	Кызгылт	clAqua	Бирюза
clTeal	Жашыл-көк	clWhite	Ак

Width касиети сызыктын жоондугун пиксел менен аныктайт. Мисалы, **Canvas.Pen.Width:=2** инструкциясы сызыктын жоондугу 2 пикселге барабар экендигин аныктайт.

Style касиети сызыктын көрүнүшүн (стилин), б.а. үзгүлтүксүз же сынык сызыктардан турган үзүлгөн сызык (пунктирдик сызык) экендигин аныктайт.

Пунктирдик сызыктын жоондугу 1 пикселден чоң болушу мүмкүн эмес. Pen.Width касетинин мааниси 1 ден чоң болсо, анда сызык үзгүлтүксүз сызык катары чагылдырылат.

9.3-таблица. Pen.Style касиети сызыктын көрүнүшүн аныктайт

Константа	Сызыктын көрүнүшү
psSolid	Туташ сызык (үзгүлтүксүз сызык)
psDash	Пунктирдик сызык, узун штрихи бар
psDot	Пунктирдик сызык, кыска штрихи бар
psDashDot	Пунктирдик сызык, кезектешүүчү узун жана кыска штрихи бар
psDashDotDot	Пунктирдик сызык, кезектешүүчү бир узун жана эки кыска штрихи бар
psClear	Сызык чагылдырылбаган учур (эгерде областтын (мисалы, тик бурчтуктун)

	чекарасын чыгарбоо керек болгон учурда колдонулат

§ 9.4. Кисть

Кисть классы контур менен чектелген областтарды боёо үчүн колдонулат жана ага TBrush объектинин **Brush** касиети туура келет. Canvas.Brush касиетинин маанилери 9.4-таблицада келтирилди.

9.4-таблица. TBrush объектинин касиеттери (кисть)

Касиети	Аныктайт
Color	Туюк областты боёо түсү
Style	Областты боёо стили (тиби)

Контурдун ички областын боёого же штрихтөөгө болот. Биринчи учурда областтын фону толук жабылат (көрүнбөйт), ал эми экинчи учурда штрихтелбеген участоктордо областтын фону көрүнүп турат.

9.5-таблица. Brush, Style касиеттеринин маанилери боёо тибин аныктайт

Константа	Областты боёо (толтуруу) тиби
bsSolid	Туташ боёо
bsClear	Область боёолбойт
bsHorizontal	Горизонталдык штрихтөө
bsVertical	Вертикалдык штрихтөө
bsFDiagonal	Жантык сызыгы алдыга багытталган болгон диагоналдык штрихтөө
bsBDiagonal	Жантык сызыгы артка багытталган болгон диагоналдык штрихтөө
bsCross	Клеткага горизонталдык-вертикалдык штрихтөө
bsDiagCross	Клеткага диагоналдык штрихтөө

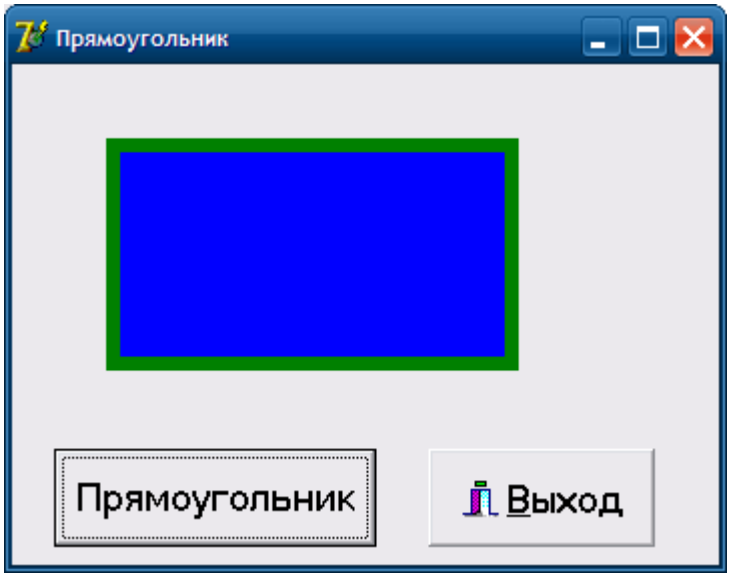
§ 9.5. Тик бурчтук

Тик бурчтук Rectangle методу менен төмөнкү инструкция менен чийилет:

Объект.Canvas.Rectangle(x1, y1, x2, y2);

мында

- объект – тик бурчтук чийиле турган объекттин (компонентанын) аталышы;
- x1, y1 жана x2, y2 – тик бурчтуктун сол жогорку жана оң төмөнкү бурчунун координаталары.



9.3-сүрөт. Тик бурчтук

9.2-листинг. Тик бурчтук

```
procedure TForm1.Button1Click(Sender: TObject);
begin
        With Form1.Canvas do
          begin
            Pen.Width:=7; // Карандаштын жоондугу
            Pen.Color:=clGreen; // Карандаштын түсү
            Brush.Color:=clBlue; // Боёо түсү
            Rectangle(50,40,250,150); // Тик бурчтук
          end;
end;
```

§ 9.6. Бурчтары жумуру тик бурчтук

RoundRec методу бурчтары жумуру тик бурчтукту чиет. RoundRec методун чакыруу инструкциясы төмөнкүдөй:

Объект.Canvas.RoundRec(x1, y1, x2, y2, x3, y3)

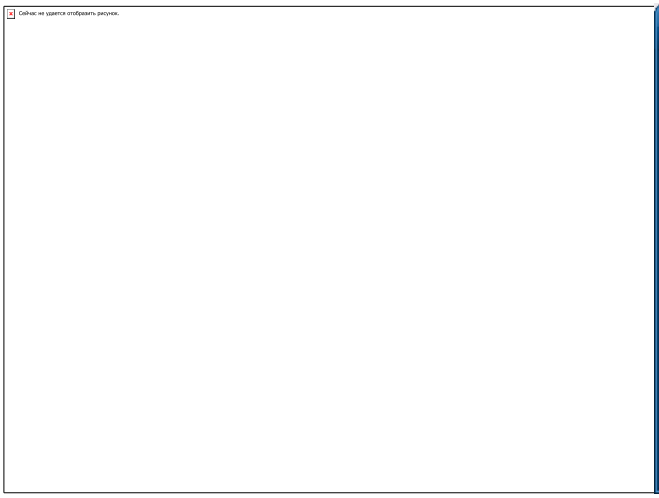
мында

- x1, y1, x2, y2 – бурчтары жумуру болгон тик бурчтукту өз ичине камтып турган тик бурчтуктун бурчтарынын абалын аныктоочу параметрлер;
- x3 жана y3 – жумуру бурчту чийүү үчүн колдонулуучу эллипстин чейрегинин (төрттөн бир бөлүгү) өлчөмү (9.4-сүрөт).

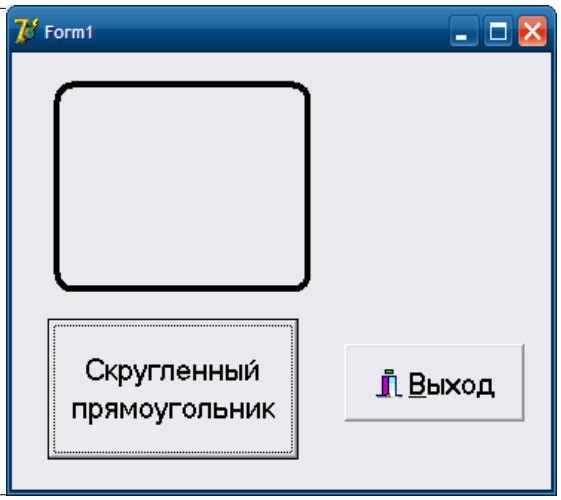
9.3-листингде бурчтары жумуру болгон тик бурчтукту чийүү коду көрсөтүлгөн.

9.3-листинг. Бурчтары жумуру болгон тик бурчтук

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Form1.Canvas.Pen.Width:=4;
    Form1.Canvas.RoundRect(30,20,200,150,20,20);
end;
```



9.4-сүрөт. RoundRec методу



9.5-сүрөт. Бурчтары жумуру тик бурчтук

§ 9.7. FillRect жана FrameRect методдору

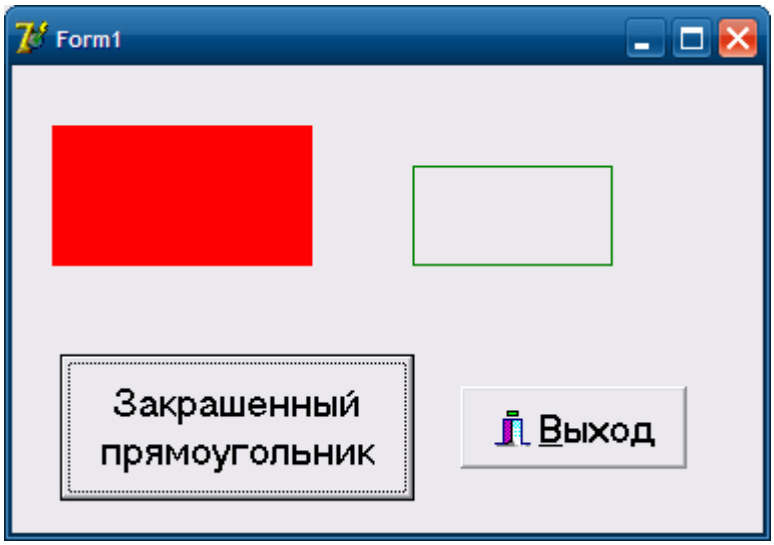
FillRect жана FrameRect методдору менен тик бурчтукту кисть (Brush) инструментин гана колдонуп чийүүгө да болот.

FillRect методу менен боёолгон тик бурчтукту чиебиз, ал эми FrameRect методу менен тик бурчтуктун контурун гана чие алабыз. Бул эки методдун ар биринде структура тибиндеги бир гана TRect параметри колдонулат. TRect структурасынын талаалары тик бурчтуу областтын координаталарын кармап турат, аларды Rect функциясынын жардамы менен аныктоого болот.

9.3-листингде FillRect жана FrameRect методдору менен кызыл түскө боёолгон тик бурчтукту жана чекарасы гана жашыл түстө болгон тик бурчтукту чийүү процедуралары көрсөтүлгөн.

9.3-листинг. FillRect жана FrameRect методдору

```
procedure TForm1.Button1Click(Sender: TObject);
var r1, r2: TRect; // Тик бурчтуктун бурчтарынын координаталары
begin
// Тик бурчтуктун чокуларынын координаталары
    r1:=Rect(20,30,150,100);
    r2:=Rect(200,50,300,100);
    with Form1.Canvas do
    begin
        Brush.Color:=clRed;
        FillRect (r1); // Боёолгон тик бурчтук
        Brush.Color:=clGreen;
        // Тик бурчтуктун чекарасы гана чийилген
        FrameRect(r2);
    end;
end;
```



9.6-сүрөт. FillRect, FrameRect методдору

§ 9.8. Текстти чыгаруу

Графиктик объекттин бетине текстти чыгаруу үчүн TextOut методун колдонобуз. TextOut методун чакыруу инструкциясы төмөнкүдөй көрүнүштө болот:

Объект.Canvas.TextOut(x, y, Текст)

мында

- объект – текст чыгарыла турган объекттин аталышы;
- x, y – текст чыгарыла турган графикалык беттеги чекиттин координаталары;
- Текст – мааниси чыгарыла турган текстти билдирген символдук типтеги өзгөрмө же константа.

Текстти чыгаруу үчүн колдонулуучу шрифт canvas объектинин Font деген касиети менен аныкталат. Font касиети TFont тибиндеги объектти мүнөздөйт.

9.6-таблицада TextOut жана TextRect методдорун колдонуу үчүн керек болгон TFont объектинин касиеттери көрсөтүлгөн.

9.6-таблица. TFont объектинин касиеттери

Касиети	**Аныктайт**
Name	Колдонулуучу шрифттин аталышы. Маани катары шрифттин аталышынкийирүү керек, мисалы Arial

Size	Шрифттин өлчөмүн пункт (points) менен берүү керек. Пункт – полиграфияда колдонулуучу шрифттин өлчөмүн ченөө бирдиги. 1 пункт = 1/72 дюйм
Style	Символдорду жазуу стили: нормалдык, жоон, жантык, асты сызылган, сызылып ташталган. Жазуу стили төмөнкү константалар менен берилет: fsBold (жоон), fsltalic (жантык), fsUnderline (сызылган), fsStrikeOut (сызылып ташталган). style касиети көптүк катары стилдердин комбинациясынан да турушу мүмкүн. Мисалы, "жоон жана жантайган" стилди алуу инструкциясы төмөнкүдөй болот: Объект.Canvas.Font: = [fsBold, fs Italic]
Color	Символдордун түсү. Маани катары Tcolor тибиндеги константаны колдонууга болот

9.1-эскертүү. Текстти чыгаруу областы кисттин (Brush) учурдагы түсү менен боёолгон болот. Ошондуктан текстти чыгаруудан мурда Brush.Color касиетине bsClear маанисин же текст чыгарыла турган беттин түсүнө дал келген түстүн маанисин ыйгарып алуу керек.

9.4-листингде TextOut методун колдонуу менен форманын бетине текстти чыгаруу программасы келтирилген.

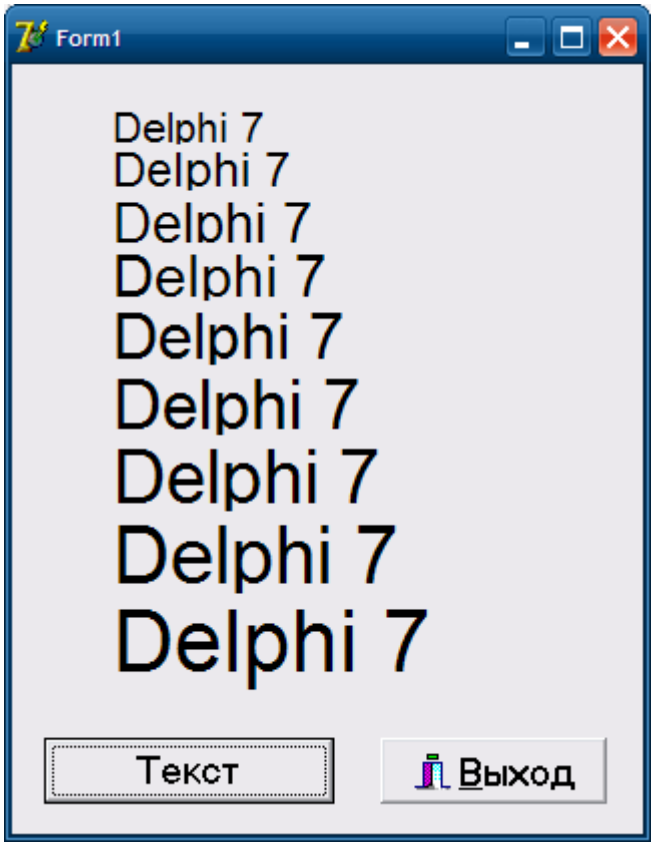
9.4-листинг. Текстти чыгаруу

```
procedure TForm1.Button1Click(Sender: TObject);
var i,y,n:integer;
begin
      y:=0;
      With Form1.Canvas do
       begin
         For i:=0 To 8 do
         begin
            Font.Name := 'Couirer New';
            Font.Size:=16+2*i;
            TextOut(50,20+y,'Delphi 7');
            y:=y+20+3*i;
```

```
        end;
      end;
end;
```

TextOut методу менен текстти чыгаргандан кийин карандаштын көрсөткүчү областтын оң жогорку бурчуна жылдырылат.

Кээде текстти программа иштеп жатканда узундугу белгисиз болгон кандайдыр бир билдирүүдөн кийин чыгаруу керек болуп калат. Мисалы, сандардын сөз менен жазылышынан кийин чыгарылуучу "сом" сөзү болушу мүмкүн. Бул учурда чыгарылган тексттин оң жактагы чекарасын билүү зарыл болуп калат.



9.6-сүрөт. Текстти чыгаруу

TextOut методу менен чыгарылган тексттин оң жактагы чекарасынын координатасын PenPos касиети менен аныктап алса болот.

9.5-листингде TextOut методуннун эки инструкциясын колдонуу менен форманын бетине текстти чыгаруу программасы келтирилген.

9.5-листинг. Текстти чыгаруу

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    with Form1.Canvas do
    begin
        TextOut(50, 30, 'Borland ') ;
        TextOut(PenPos.X, PenPos.Y, 'Delphi 7');
    end;
end;
```

§ 9.9. Сынык сызык

Сынык сызыктарды **Polyline** методу менен сызууга болот. Методдун параметри катары TPoint тибиндеги массивди алууга болот. Массивдин ар бир элементи сынык сызыктын сынуу чекитинин координаталарын кармап турат.

Polyline методу массивдеги биринчи чекит менен экинчисин, экинчи чекит менен үчүнчүсүн, ж.б., акыркы чекиттен мурдагы чекит менен акыркы чекитти удаалаш туташтыруу аркылуу сынык сызыкти чиет.

Практикада эсептөөнүн жыйынтыгын график түрүндө көрсөтүү ыңгайлуу болот. Графикти көрсөтмөлүү жана түшүнүктүү кылуу үчүн координаталык октордон турган координаталык системаларга жайгаштырабыз.

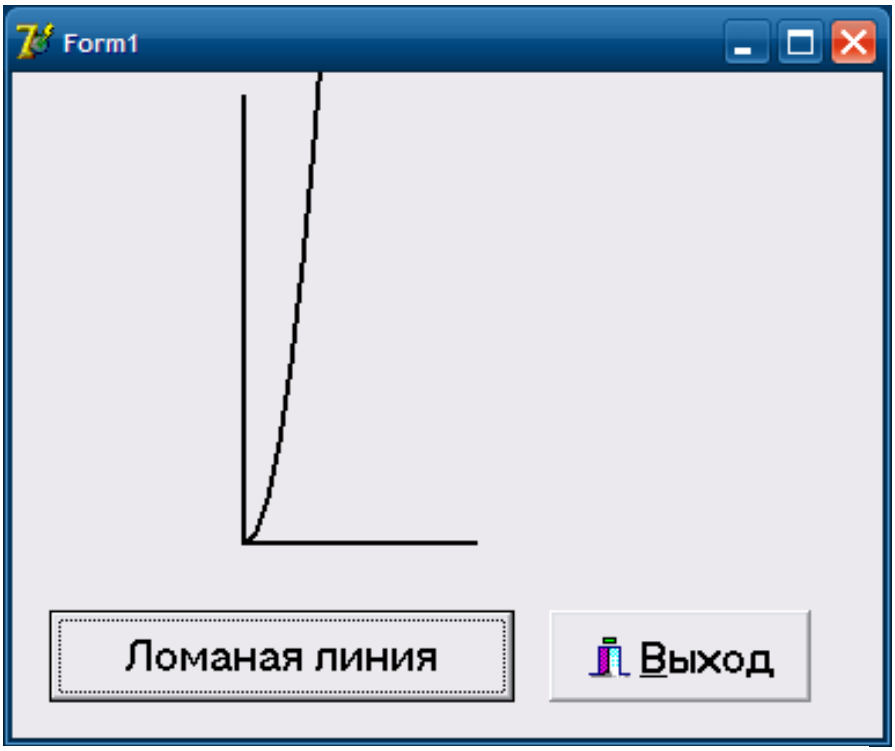
9.6-листингде Polyline методун колдонуу менен функцияны графиги чийилген.

9.6-листинг. Функциянын графиги

```
procedure TForm1.Button1Click(Sender: TObject);
var
    gr:array[0..50] of TPoint; // график – сынык сызык
    x0,y0:integer; // координата башталышынын координаталары
    dx,dy:integer; // шаг координатной сетки по осям X и Y
    i: integer;
begin
       x0:=10; y0:=200; dx:=5; dy:=5;
// gr массивин толтуруу
       for i:=0 to 50 do
         begin
           gr[i].x:=x0+i*dx;
           gr[i].y:=y0-sqr(i)*dy;
         end;
// Графикти тургузуу
       With Form1.Canvas do
begin
       Pen.Width:=2;
       MoveTo(x0,y0); LineTo(x0,10); // Y огу
       MoveTo(x0,y0); LineTo(200,y0); // X огу
       Polyline(gr); // график
end;
```

end;

Polyline методу менен туюк контурларды да чийүүгө болот. Ал үчүн массивдин биринчи жана акыркы чекиттеринин координаталары дал келиш керек, б.а. бир эле чекиттин координаталары болуш керек.



9.7-сүрөт. Сынык сызыктын колдонулушу

§ 9.10. Айлана жана эллипс

Ellipse методу менен параметрлердин маанилерине жараша эллипсти же айлананы чийүүгө болот.

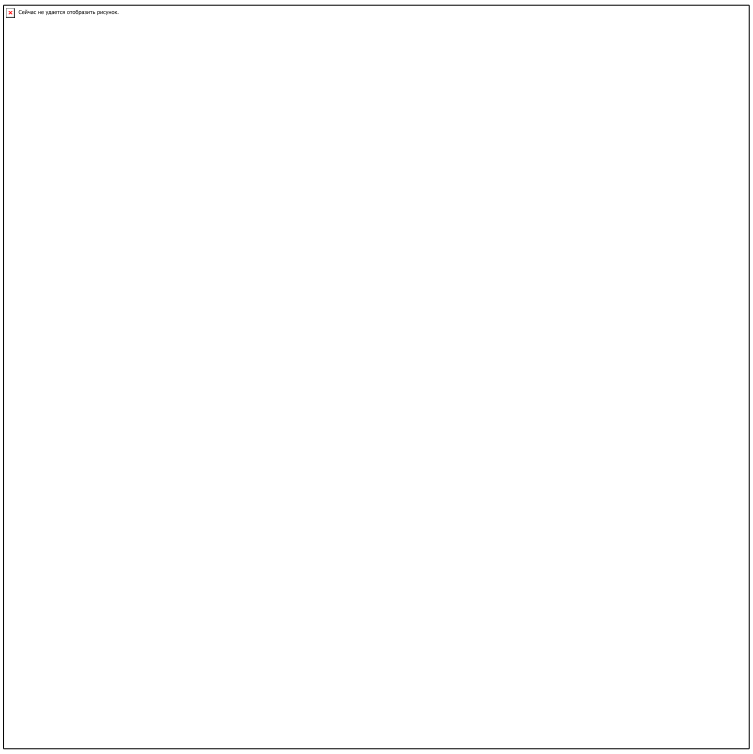
Ellipse методун колдонуу инструкциясы төмөнкүдөй көрүнүштө болот:

Объект.Canvas.Ellipse(x1,y1, x2,y2)

мында

- объект – бетине чийилүүчү объекттин (компонентанын) аталышы;

- x1, y1, x2, y2 – ичине эллипс чийиле турган тик бурчтуктун же ичине айлана чийиле турган квадраттын координаталары (9.8-сүрөт).



9.8-сүрөт. Ellipse методу

Ellipse методунун параметрлеринин маанилери геометриялык фигуранын түрүн аныктайт.

Эллипстин контурунун түсү, жоондугу жана стили Pen касиетинин маанилери менен аныкталат.

Эллипстин ички областын боёо түсү жана стили Canvas бетинин Brush касиетинин маанилери менен аныкталат.

§ 9.11. Жаа

Arc методу менен жааны чийүүгө болот. Arc методун колдонуу инструкциясы төмөнкүдөй көрүнүштө болот:

Объект.Canvas.Arc(x1,y1,x2,y2,x3,y3,x4,y4)

мында:

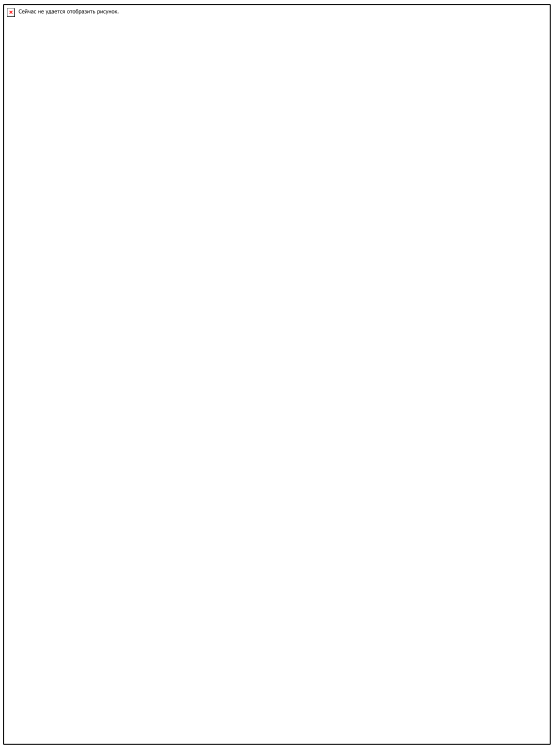
- x1, y1, x2, y2 – бир бөлүгү сызылуучу жааны билдирген эллипсти (айлананы) аныктоочу параметрлер;
- x3, y3 – жаанын баштапкы чекитинин координатасы;
- x4, y4 – жаанын акыркы чекитинин координатасы.

Жаанын баштапкы жана акыркы чекиттери – эллипс менен эллипстин борборунан өткөн жана чокулары (x3, y3), (x4, y4) чекиттери болгон тик бурчтуктун кесилишкен чекиттери катары аныкталат.

Жаа баштапкы чекиттен акыркы чекитке карай саат жебесине карама-каршы багытта чийилет (9.9-сүрөт).

Жаанын сызыгынын түсү, жоондугу жана стили Canvas бетинин Pen касиетинин маанилери менен аныкталат.

Arc методунун параметрлери жааны эллипстин (айлананын) бир бөлүгү катары аныктайт.



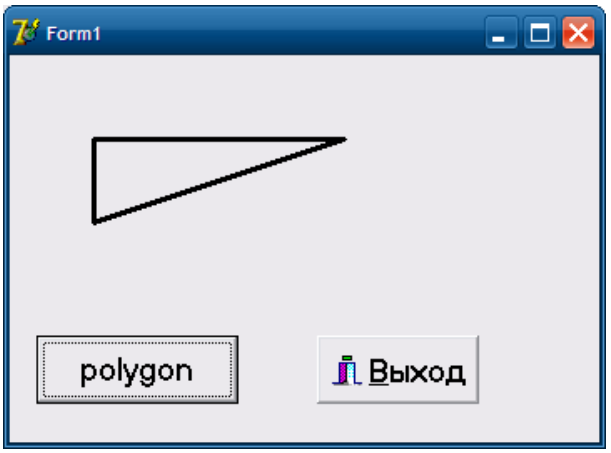
9.9-сүрөт. Arc методу

§ 9.12. Көп бурчтук

Polygon методу көп бурчтукту чиет. Методдун параметри катары TPoint тибиндеги массив алынат.

Массивдин ар бир элементи көп бурчтуктун бир чокусунун координаталарын билдирет.

Polygon методу массивдеги биринчи чекит менен экинчисин, экинчи чекит менен үчүнчүсүн, ж.б., акыркы чекиттен мурдагы чекит менен акыркы чекит менен, акыркы чекитти баптапкы чекит менен удаалаш туташтыруу аркылуу көп бурчтукту чиет.



9.10-сүрөт. Polygon методу

Көп бурчтуктун чек арасынын түсү жана стили Pen касиетинин маанилери, чек ара сызыгы менен чектелген областты боёо түсү жана стили Brush касиетинин маанилери менен аныкталат, ал эми область кисттин учурдагы түсүн жана стилин колдонуу менен боёолот.

9.7-листингде polygon методун колдонуу менен үч бурчтукту чийүү процедурасы келтирилген.

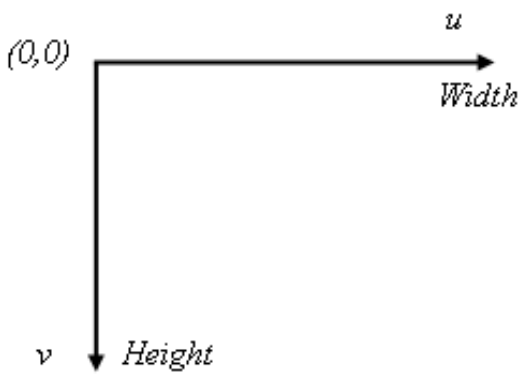
9.7-листинг. Үч бурчтукту чийүү

```
procedure TForm1.Button1Click(Sender: TObject);
var
pol: array[1..3] of TPoint; // Үч бурчтуктун чокулары
begin
```

```
        pol[1].x:=50;
        pol[1].y:=50;
        pol[2].x:=200;
        pol[2].y:=50;
        pol[3].x:=50;
        pol[3].y:=100;
        Form1.Canvas.Pen.Width:=3;
        Form1.Canvas.Polygon(pol);
end;
```

§ 9.13. Функциянын графигин тургузуу

$[a,b]$ сегментинде үзгүлтүксүз болгон $y = f(x)$ функциясынын графигин тургузуу үчүн декарттык координаталар системасында $(x_i, f(x_i))$ чекит-терин тургузуп, аларды түз сызыктын кесиндилери менен туташтыруу керек. Канчалык көп чекит алынса, график ошончолук көрүнүктүү (жылма) болот.

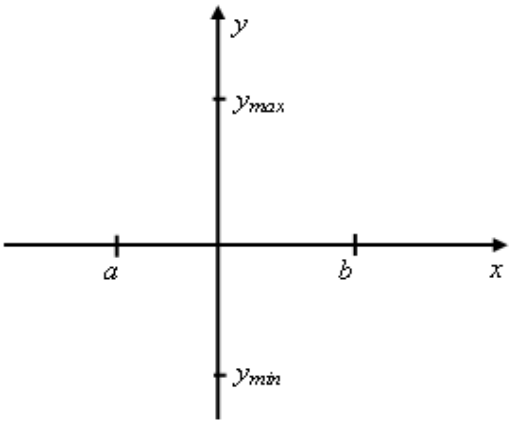


9.11-сүрөт. Ouv координаталар системасы

Функциянын графигин формага же Delphi нин башка компонентасына жайгаштыруу үчүн компонентанын өлчөмдөрүнүн өзгөчөлүгүн эске алуу керек.

Мейли *Ouv* монитордун экранындагы координаталар системасы болсун (9.11-сүрөт). u огу солдон оңго карай багытталган жана анын координаталары *0* дөн *Width* чоңдугуна чейин өзгөрөт ($0 \le u \le Width$), ал эми v огу жогорудан төмөн карай багытталган жана анын координаталары *0* дөн *Height* чоңдугуна чейин өзгөрөт ($0 \le v \le Height$).

Мейли *Oxy* реалдуу (чыныгы) декарттык координаталар системасы болсун (9.12-сүрөт). $y = f(x)$ функциясынын графиги $[a,b, y_{min}, y_{max}]$ тик бурчтугунда жайгашкандыгын байкоо кыйын эмес, мында $a \le x \le b$, $y_{min} \le f(x) \le y_{max}$.



9.12-сүрөт. *Oxy координаталар системасы*

Oxy координаталар системасындагы графикти *Ouv* координаталар системасына көчүрүү үчүн төмөнкү теңдемелер системасын алабыз:

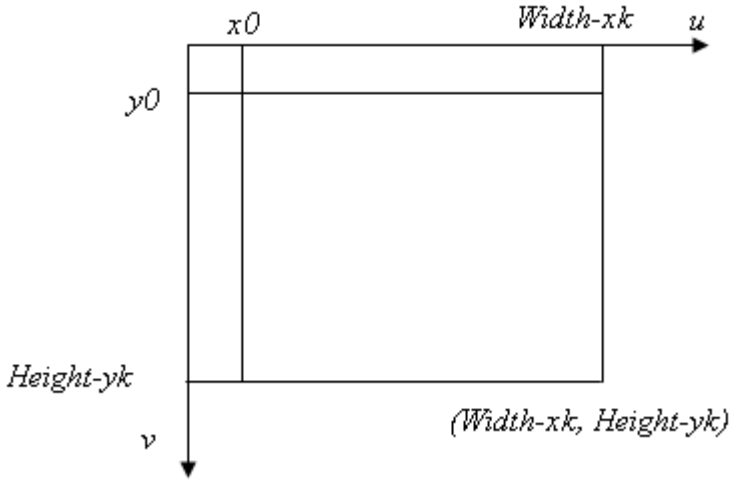
$$\begin{cases} u = cx + d, \\ v = py + q, \end{cases} \qquad (1)$$

мында c,d,p,q - белгисиз константалар.

Айталы *x0,xk,y0,yk* сандары компонентанын сол, оң, төмөн жана жогору чек араларынан тиешелүү түрдө четтөөлөрүн аныктасын (9.13-сүрөт).

$[a,b]$ кесиндисин $[x0, Width - xk]$ кесиндисине өзгөртүп түзүү үчүн төмөнкү системаны алабыз:

$$\begin{cases} c\,a + d = x0, \\ c\,b + d = Width - xk. \end{cases}$$



9.13-сүрөт. Сүрөт тартуу областы

Бул системадан a жана b ны таап алабыз:

$$c = \frac{Width - xk - x0}{b - a},\ d = x0 - c\,a. \qquad (2)$$

Ал эми $[y_{min}, y_{max}]$ кесиндисин $[y0, Height - yk]$ кесиндисине өзгөртүп түзүү үчүн төмөнкү системаны алабыз:

$$\begin{cases} py_{max} + q = y0, \\ p\,y_{min} + q = Height - yk. \end{cases}$$

Мындан p жана q ларды аныктап алууга болот:

$$p = \frac{Height - yk - y0}{y_{min} - y_{max}},\ q = y0 - p\,y_{max} \qquad (3)$$

Жыйынтыгында, (1) сызыктуу өзгөртүп түзүү $[a,b]$ кесиндисин $[x0, Width - xk]$ кесиндисине, ал эми $[y_{min}, y_{max}]$ кесиндисин $[y0, Height - yk]$ кесиндисине (2) жана (3) формулалардын негизинде чагылтат.

Width чоңдугун аныктап жатканда компонентанын сол жана оң жактардагы рамкаларын, ал эми *Height* чоңдугун аныктап жатканда компонентанын бөркүн (заголовкасын) эске алуу керек.

Бирок, графикти чийүү үчүн бизге рамкалардын жана бөрктүн кереги жок. Бул өлчөмдөр *ClientWidt h* (клиенттик областтын туурасы, мында рамкалардын туурасы эске алынбайт) жана *ClientHeig ht* (клиенттик областтын бийиктиги, мында рамкалардын жана бөрктүн өлчөмдөрү эске алынбайт) касиеттеринде берилген. Ошондуктан c, d, p жана q чоңдуктарын аныктоодо төмөнкү формулаларды колдонобуз:

$$\begin{cases} c = \dfrac{ClientWidt\ h - xk - x0}{b - a},\ d = x0 - c\,a, \\ p = \dfrac{ClientHeig\ ht - yk - y0}{y_{min} - y_{max}},\ q = y0 - py_{max}. \end{cases} \qquad (4)$$

Дисплейдин экранына функциянын графигин чыгаруу үчүн төмөнкү алгоритмди пайдаланабыз:

1. $[a,b]$ *сегментин майда бөлүктөргө бөлүп алуу үчүн* N *санын тандап алуу;*

2. x *өзгөрмөсүнүн өзгөрүү кадамын аныктоо:* $h = (b-a)/N$ *;*

3. X, Y массивдерин түзүү: $x_i = a + i\,h,\ y_i = f(x_i), i = \overline{0, N}$ *;*

4. Y тин максималдык (max) жана минималдык (min) маани-лерин эсептөө;

5. (4) формула боюнча c, d, p жана q коэффициенттерин табуу;

6. $u_i = cx_i + d$, $v_i = py_i + q$ массивдерин түзүү;

7. Коңшу чекиттерди LineTo методу менен түз сызыктын кесиндиси аркылуу удаалаш туташтыруу же Pixcels методу менен чекиттерди орнотуу;

8. Координаталар системасын, торчонун сызыктарын жана жазууларды жайгаштыруу.

9.1-мисал. *$[a,b]$ кесиндисинде $y = 2\sin x e^{x/2}$ функциясынын графигин тургузгула.*

1. Жаңы проект түзөбүз (Ugraf, Pgraf).

2. Функциянын графигин чагылтуу үчүн жетишээрдик боло тургандай кылып форманын өлчөмдөрүн өзгөртөбүз (*Width* = 800, *Height* = 700).

3. Формага TImage жана TButton класстарынын бирден компоненталарын жайгаштырабыз. Image1 компонентасы TImage классындагы растрдык картиналарды чыгаруучу объект катары функциянын графигин Button1 баскычын басканда чагылтуу үчүн колдонулат.

9.7-листингде функциянын графигин түзүү программасы келтирилген.

9.7-листинг. Функциянын графигин түзүү

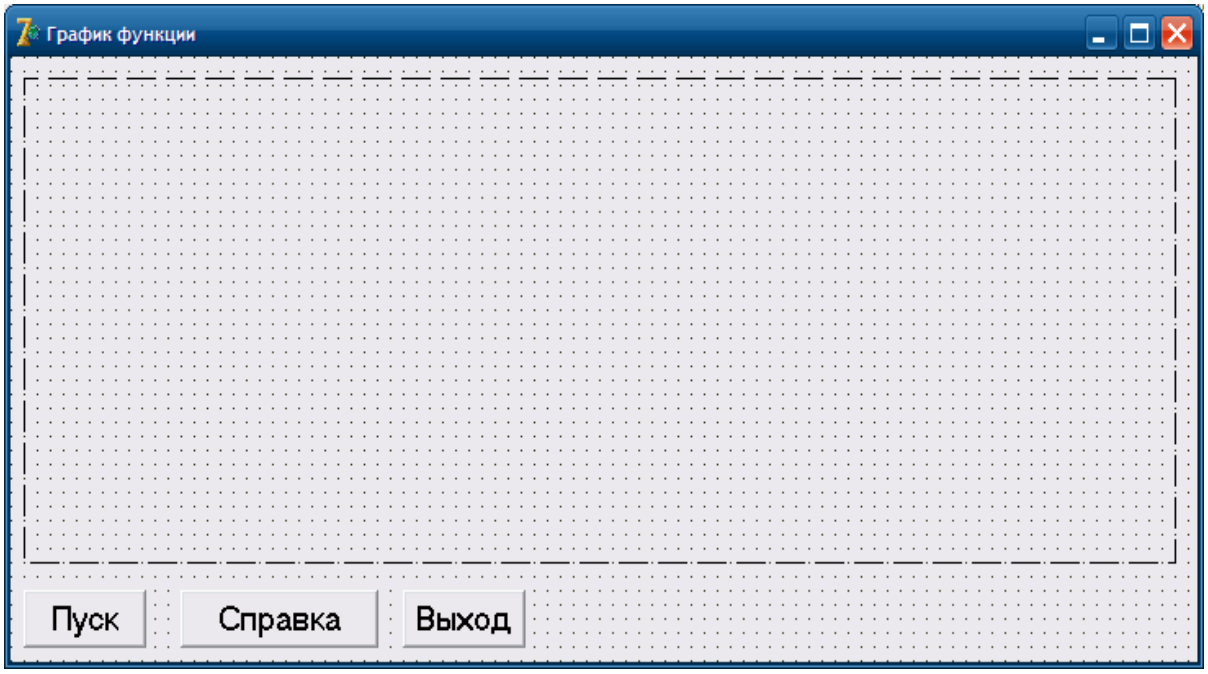
```
unit Ugraf;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, ExtCtrls;
Const
    N=1000; // Кесиндилердин саны
type
  TForm1 = class(TForm)
    Image1: TImage;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
```

```
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
          Form1: TForm1;
implementation
        {$R *.dfm}

// Берилген функция
Function f(x:real):real;
begin
        f:=2*sin(x)*exp(x/5);
end;

// Токтотуп туруу
procedure Delay(s:Integer);
var
        t:Integer;
begin
          t:=GetTickCount+s;
          while GetTickCount<t do
          Application.ProcessMessages;
end;

// Функциянын графигин тургузуу программасы
procedure TForm1.Button1Click(Sender: TObject);
var
     i:integer;
```

```
    a,b:real; // Берилген кесинди
    h:real; // Өсүндү
    x,y:array[0..N] of real; // x,y маанилеринин массиви
    ymin,ymax:real; // Функциянын max жана min маанилери
    x0,xk,y0,yk:integer; // Четтөөлөр
    c,d,p,q:real; // Координаталарды өзгөртүү коэффициенттери
    u,v:array[0..N] of integer; // u,v маанилеринин массиви
begin
      a:=-10;b:=10; // Берилген кесинди
      h:=(b-a)/N; // Өзгөрүү кадамы
      x[0]:=a; y[0]:=f(x[0]); // x, y маанилеринин массиви
      For i:=1 to N do
       begin
         x[i]:=x[0]+i*h; y[i]:=f(x[i]);
       end;

// Максималдык жана минималдык маанилер
      ymin:=y[0];
      ymax:=y[0];
      For i:=1 to N do
       begin
         if y[i]>ymax then ymax:=y[i];
         if y[i]<ymin then ymin:=y[i];
end;

// Координаталар системаларын өзгөртүү: Oxy → Ouv
      x0:=5;xk:=20;
      y0:=5;yk:=20;
      c:=(Form1.Image1.ClientWidth-xk-x0)/(b-a);
      d:=x0-c*x[0];
      p:=(Form1.Image1.ClientHeight-yk-y0)/(ymin-ymax);
      q:=y0-p*ymax;
// u,v маанилеринин массиви
      For i:=0 to N do
```

```
        begin
          u[i]:=trunc(c*x[i]+d); v[i]:=trunc(p*y[i]+q);
        end;

// Координаталар огун чийүү
Form1.Image1.Canvas.MoveTo(x0,y0);
Form1.Image1.Canvas.LineTo(x0,Form1.Image1.ClientHeight-yk);
Form1.Image1.Canvas.MoveTo(u[0],v[0]);
Form1.Image1.Canvas.LineTo(u[N],v[0]);
Form1.Image1.Canvas.TextOut(x0+5,y0+15,FloatTo-
StrF(ymax,ffGeneral,6,3));
Form1.Image1.Canvas.TextOut(x0+5,Form1.Image1.ClientHeight-yk,
FloatToStrF(ymin,ffGeneral,6,3));

// Координата окторунун аталыштарын чыгаруу
Form1.Image1.Canvas.Pen.Color:=clRed;
Form1.Image1.Canvas.Font.Size:=14;
Form1.Image1.Canvas.TextOut(x0+5,y0,'v');
Form1.Image1.Canvas.TextOut(u[N],v[0],'u');
Form1.Image1.Canvas.TextOut(x0+300,Form1.Image1.ClientHeight-
yk,'y=2sin(x)exp(x/5)');

// Функциянын графиги
      Form1.Image1.Canvas.MoveTo(u[0],v[0]);
      Form1.Image1.Canvas.Pen.Width:=2;
      Form1.Image1.Canvas.Pen.Color:=clRed;
      For i:=0 to N do
        Begin
          Form1.Image1.Canvas.LineTo(u[i],v[i]);
          // Form1.Image1.Canvas.Pixels[u[i],v[i]]:=clRed;
          Delay(10); // Токтотуп туруу
        end;
        end;
```

```
// A.chm справкалык файлды чыгаруу
procedure TForm1.Button3Click(Sender: TObject);
var
        h:HWND;
        begin
          h:=FindWindow('HH Parent','index');
          if h=0 then
            WinExec('hh.exe A.chm',SW_RESTORE)
          else
           begin
           ShowWindow(h,SW_RESTORE);
           Windows.SetForegroundWindow(h);
           end;
end;

// Справкалык файлды форма менен кошо жабуу
procedure TForm1.FormClose(Sender: TObject; var Action: TClose-
Action);
var
        h:HWND;
begin
          h:=FindWindow('HH Parent','index');
          if h<>0 then
           SendMessage(h,WM_CLOSE,0,0);
end;

// Форманы жабуу
procedure TForm1.Button2Click(Sender: TObject);
begin
        close;
end;
end.
```



9.14-сүрөт. Графикти чийүү үчүн түзүлгөн форманын модели

Эскертүү. Эгерде Pixels методун колдонсок, анда Form1.Image1.Canvas.LineTo(u[i],v[i]); инструкциясынын ордуна Form1.Image1.Canvas.Pixels[u[i],v[i]]:=clRed; инструкциясын жазабыз.

9.15-сүрөт. Функциянын динамикалык графиги

10-глава. Берилгендер базасы

§ 10.1. Берилгендер базасы жана башкаруу системасы

Берилгендер базасы деп – информациялар жазылган таблицалардын жыйындысын айтабыз.

Берилгендер базасынын (ББ) структурасына жана түзүлүшүнө жараша бир нече моделдерин кароого болот.

- Иерархиялык;
- Реляциялык;
- Тармактык;
- Объектке-ориентирленген;
- Көп звенолуу жана башка.

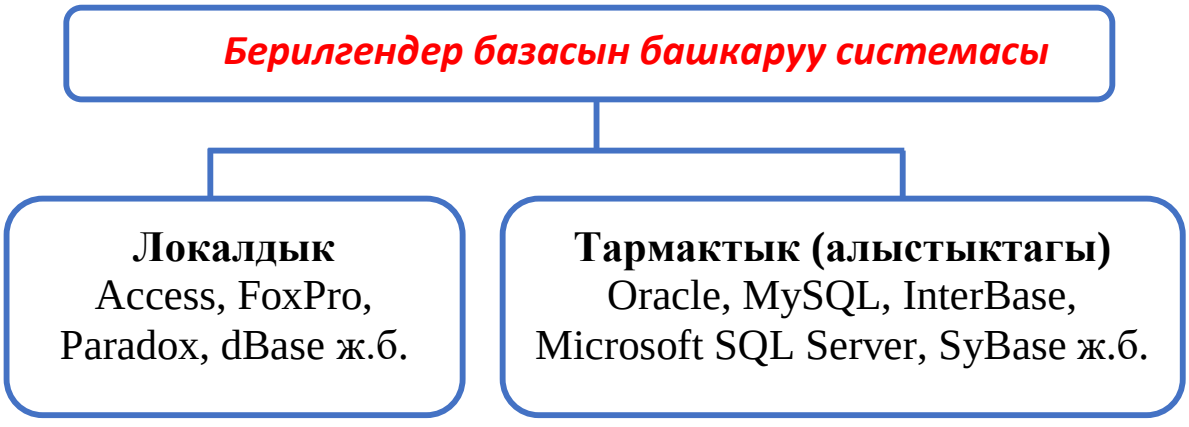
Азыркы учурда реляциялык ББ кеңири колдонулат, анткени алар түзүлүшүнүн жөнөкөйлүгү жана структурасынын ийкемдүүлүгү менен өзгөчөлөнүп турат.

Реляциялык ББ деп логикалык өз ара байланышкан таблицалардын жыйындысын айтабыз.

Берилгендер базасы менен иштөөчү программаны ***Берилгендер базасын башкаруу системасы*** *(ББББС) деп айтабыз.*

Берилгендер базасын түзүү, толтуруу, жаңылоо жана алып салуу ББББС менен аткарылат.

Берилгендер базасы эки типке бөлүнөт: локалдык (персоналдык) жана тармактык (көптөгөн колдонуучулук) (10.1-схема).



10.1-схема. ББББС нын структурасы

Локалдык ББББС бир эле колдонуучу компьютер менен иштеп жаткан учурда колдонулат.

Эгерде колдонуучулардын саны эки же андан көп болсо, анда **тармактык ББББС** колдонулат.

Берилгендер базасынын (ББ) таблицасы жолчолордон жана мамычалардан турат. Жолчолор – **жазуулар** (***записи***), ал эми мамычалар – **талаалар** деп аталат (10.1-таблица). Ар бир талаа өзүнчө атка ээ болушу керек.

10.1-таблица

	1-талаа	2-талаа	3-талаа	4-талаа
1-жазуу				
2-жазуу				
3-жазуу				

Талааларда берилгендердин бир гана тиби сакталышы керек. Бул маалыматтар таблицаны түзгөндө берилиши керек.

Ар бир таблица үчүн **ачкычтар** жана **индекстер** берилиши мүмкүн.

Ачкыч деп таблицадагы ар бир жазууну бир маанилүү аныктаган берилгендердин талааларынын комбинациясын айтабыз.

Ачкычтар **жөнөкөй** жана **татаал (курама)** болушу мүмкүн.

Эгерде ачкыч бир гана талаадан турса, анда аны **жөнөкөй талаа** деп айтабыз. Эгерде ачкыч бир нече талаалардын курамынан турса, анда аны **курама (татаал) талаа** деп айтабыз.

Ачкыч жазуулардын кайталанышына жол бербейт.

Ачкычты түзгөн талааларды **ачкычтык талаалар** деп атайбыз.

Индех деп жазуулар боюнча сорттолуучу таблицанын талааларынын комбинациясын айтабыз.

Индекс түзүлгөн таблицанын талааларынын маанилери ачкычтардан айырмаланып бири-бири менен дал келиши мүмкүн.

Индексти түзүүчү талааларды индекстик талаалар деп айтабыз. Таблицаны сорттоо индекстик талаалардын маанилеринин өсүүсү же кемүүсү боюнча жүргүзүлүшү мүмкүн. Ачкычтык талаалар автоматтык түрдө индекстелет.

Берилгендер базасынын эң жөнөкөй мисалы болуп телефондук маалымдагыч (справочник) болуп эсептелет (10.2-таблица).

10.2-таблица

	Фамилиясы	Аты	Иштеги телефону	Үйдөгү телефону
1	Асанов	Акмат	2-12-37	5-14-67
2	Алиева	Бермет	8-23-56	5-47-92
3	Иванов	Сергей	4-35-46	8-54-63

§ 10.2. Берилгендер базасынын нормалдык формасы

Берилгендер базасын проектирлөө үчүн эң оболу берилгендер базасын нормалдаштырып алуу сунушталат.

Берилгендер базасын нормалдаштыруу процесси деп ашыкча берилген маалыматтарды жоюуну айтабыз. Берилгендер базасын нормалдаштыруу деп таблицаны кайталанбагандай кылып жана карама-каршылык болбогондой кылып эки же андан көп таблицаларга бөлүүнү айтабыз. Нормалдаштыруунун негизги максаты болуп таблицанын ар бир талаасы бир гана жерде кездеше тургандай абалга келтирүү болуп эсептелет.

Таблицаны нормалдаштыруунун негизинен 6 формасы бар, бирок практикада нормалдаштыруунун алгачкы 3 формасы эле колдонулат.

Биринчи нормалдык формада (1НФ) ББнын ар бир талаасы бөлүнбөс *(атомардык)* жана кайталануучу группалар жок болушу керек.

Таблицанын талаасын бөлүнбөс деп айтабыз, эгерде ал талааны майда талааларга бөлүүгө мүмкүн эмес болсо.

Мисалы, студенттин фамилиясын жана анын группасын бир талаага бириктирсек, анда талаанын бөлүнбөстүк талабы аткарылбай калат. Биринчи нормалдык форманын талабы боюнча бул берилгендерди эки талаага бөлүүгө туура келек.

Кайталануучу группалар деп маанилеринин мазмуну бирдей болгон талааларды айтабыз. 10.1-сүрөттөгү таблицада талаанын аталыштары кайталануучу группанын мисалы болот.

Студент 1	Студент 2	Студент 3
Асанов И.А.	Алиева Ж.А.	Сидоров С.С.

10.1-сүрөт. Кайталануучу группалар

Бул таблица биринчи нормалдык форманын эрежесин канааттандырбайт. "Студент 1", "Студент 2" жана "Студент 3" талаалары мааниси боюнча бирдей объекттерди кармап турат, ошондуктан аларды "Студент" деп аталган бир эле талаагы жайгаштыруу керек (10.2-сүрөт).

№	Студент	Группа
1	Асанов И.А.	А1
2	Алиева Ж.А.	А2
3	Сидоров С.С.	А3

10.2-сүрөт. Кайталанбоочу группалар

Чындыгында группада 3 эле студент болбойт да. Эгерде студенттин саны 25 болсо, анда таблица кандай абалга келип калат, таблицада 25 бирдей талаа пайда болот.

10.2-сүрөттө "Студент" талаасы "Фамилия А.А." форматындагы берилгендерди баяндайт. Эгерде оператор берилгендерди "Фамилия Аты Атасы" форматында берсе, анда талаанын бөлүнбөстүк шарты бузулат. Бул учурда талааны үч талаага бөлүүгө туура келет, анткени издөө фамилия боюнча эле эмес, аты же атасы боюнча да изделиши мүмкүн.

Экинчи нормалдык формада (2НФ) таблицада биринчи нормалдык форманын бардык шарттары аткарылышы керек жана ар бир ачкыч болбогон талаа ачкычтык талаалардын толук топтому менен бир маанилүү идентификацияланышы (аныкталышы) керек.

Төмөнкүдөй мисалды карайлы: студенттур акысы ЖОЖ аркылуу каржыланган спорттук секцияларга катышат деп эсептейли. Эгерде таблицаны 10.3-сүрөттөгүдөй кылып түзсөк, анда экинчи нормалдык формаданын шарттары бузулгандыгын байкайбыз.

№	Секция	Төлөм
1	Сүзүү	500
2	Шахмат	400
3	Теннис	450
4	Сүзүү	500
5	Теннис	450

10.3-сүрөт. Экинчи нормалдык форманын бузулушу

Бул таблицанын ачкычтык талаалары болуп "№" - "Секция" талаалары болуп эсептелет. Бирок бул таблицада "Секция" - "Төлөм" байланышы да бар. Эгерде биз №2 жазууну өчүрсөк, анда шахмат секциясы боюнча төлөнүүчү бааны жоготуп койобуз. Натыйжада бул секцияга бир студент жазылганга чейин информацияларды киргизе албайбыз.

Экинчи нормалдык форманын талабы боюнча ар бир ачкыч эмес талаа ачкычтык талаа менен бир маанилүү аныкталышы керек. Секцияга катышуу төлөмү эч качан студенттин номери болгон ачкычтык талаадан көз каранды болбош керек. Ошондуктан, экинчи нормалдык форманын талаптары орундалышы үчүн таблицаны эки таблицага бөлүүгө туура келет: ар бир талаа өзүнүн ачкычтык талаасы менен аныкталышы керек:

№	Секция
1	Сүзүү
2	Шахмат
3	Теннис
4	Сүзүү
5	Теннис

Ключ: №

Секция	Төлөм
Сүзүү	500
Шахмат	400
Теннис	450

Ключ: Секция

10.4-сүрөт. Туура аныкталган экинчи нормалдык форма

Таблицаны ачкычтык эмес талаалар ачкычтык талаалар менен бир маанилүү аныктала тургандай кылып эки таблицага бөлдүк (10.4-сүрөт).

Үчүнчү нормалдык формада (3НФ) таблицанын ачкычтык эмес талааларында транзитивдик көз карандылыктар болбошу керек, б.а. биринчи ачкычка кирбеген каалаган талаанын мааниси

башка биринчи ачкычка кирбеген талаанын маанисине көз каранды эмес.

Студенттердин берилгендер базасында спорттук секцияны уюштуруу үчүн кеткен чыгымды да көрсөтүү керек болсун дейли (10.5-сүрөт).

Секция	Төлөм	Студенттердин саны	Жалпы баа
Сүзүү	500		
Шахмат	400		
Тенис	450		

10.5-сүрөт. Үчүнчү нормалдык форманын бузулушу

Бул мисалда ачкычтык талаа болуп "Секция" талаасы эсептелет. Ал эми "Төлөм" жана "Студенттердин саны" талаалары ачкычтык талаадан көз каранды, бирок алар бири-биринен көз каранды эмес. Бирок "Жалпы баа" талаасы "Төлөм" жана "Студенттердин саны" талааларынан көз каранды, ошондуктан бул жерде үчүнчү нормалдык форманын эрежеси бузулууда.

Бул мисалда "Жалпы баа" талаасын алып салса болот, анткени аны эсептелүүчү талаа катары "Төлөм" жана "Студенттердин саны" талааларынын маанилеринин көбөйтүндүсү катары эсептеп, чыгарып алууга болот.

Ошентип, берилгендерди нормалдаштыруу деп берилгендер базасын проектирлөөнү түшүнөбүз: кандай таблицалар болот, анда кандай талаалар болот, алар кандай типте жана өлчөмдөрү кандай болоору алдын ала аныкталат.

Андан кийин ар бир таблицаны биринчи нормалдык формага келтиребиз. Кийинки кадамда таблицаны экинчи нормалдык формага, андан кийин үчүнчү нормалдык формага келтиребиз, натыйжада берилгендер базасы нормалдык формага келтирилет.

§ 10.3. Телефондордун электрондук справочниги

Бүгүнкү күндө бирге окугандардын, иштегендердин же таанышгардын иштеги, үйдөгү жана мобилдик телефондорунун электрондук справочнигин түзүп алуу зарыл маселелердин бири болуп калды.

10.1-маселе. 10.3-таблицада көрсөтүлгөндөй телефондордун электрондук справочнигин түзүү талап кылынсын.

10.3-таблица. Телефондордун справочниги

Фамилиясы	Аты	Иш телефону	Үй телефону	Мобилдик телефону
Акматов	Мурат	5-23-18	2-12-24	0552-41-35-84
Керимбекова	Айнура	4-36-90	5-56-34	0772-35-23-95

10.3-таблицада, мисалы, Акматов Мурат менен анын иштеги, үйдөгү жана мобилдик телефондору бири бири менен байланышкан маалыматтар болуп эсептелет.

10.1-аныктама. *Бири бири менен байланышкан маалыматтардын жыйындысын* ***маалыматтар базасы*** *деп айтабыз.*

10.1-маселени чечүү үчүн 10.3-таблицага туура келүүчү маалыматтар базасын Microsoft Access программасы менен түзүп жана аны башкарууну Delphi программалоо тилинин каражаты менен жүзөгө ашырабыз.

§ 10.4. Microsoft Access менен маалыматтар базасын түзүү

Проводник менен **C:** каталогунда **BD** аттуу папка түзүп, ага Microsoft Access менен түзүлгөн **Contacts.mdb** аттуу малыматтар базасын сактайбыз. Contacts.mdb базасынын мүнөздүк касиеттери 10.4-таблицадагыдай болсун.

10.4-таблица. Таблицанын мүнөздүк касиеттери

Талаа	**Тип**	**Өлчөмү**	**Түшундурмө**
Fam	Текстовый	50	Фамилиясы
Name	Текстовый	50	Аты
Phone_work	Текстовый	50	Иш телефону
Phone_hous	Текстовый	50	Үй телефону
Phone_mobil	Текстовый	50	Мобилдик телефон

Маалыматтар базасын төмөнкү кадамдар боюнча түзөбүз.

1). Microsoft Access программасын жүктөп, **Файл \ Создать \ Новая база данных...** командасын бергенден кийин имя файла деген талаада базанын атын киргизүү талап кылынат. Базага **Contacts.mdb** деп ат берип, аны **C:\ BD** каталогуна сактайбыз.

2). Маалыматтар базасында 10.4-таблицага туура келүүчү таблицанын макетин **Создание таблицы в режиме конструктора** деген команда менен же **Конструктор** режимин тандап алуу менен түзөбүз (10.6-сүрөт). Таблицаны **Создание таблицы с помощью мастера** же **Создание таблицы путем ввода данных** командалары менен түзсө да болот. Бирок биз үчүн ыңгайлуусу **Конструктор** режими болуп эсептелет.

Имя поля	Тип данных	Описание
Fam	Текстовый	Фамилиясы
Name	Текстовый	Аты
Phone_work	Текстовый	Иш телефону
Phone_hous	Текстовый	Үй телефону
Phone_mobil	Текстовый	Мобилдик телефон

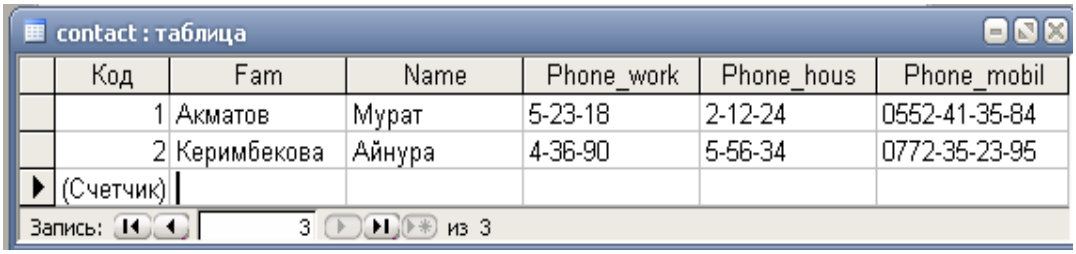
10.6-сүрөт. Конструктор режиминдеги таблица

Макетти сактаганыбызда таблицанын атын сурайт. Анын атын **contact** деп атайбыз. Андан кийин бул таблица менен башка таблицаларды байланыштыруу үчүн **ачкыч** түзүү сунушталат. Ага **Да** деп жооп берсек, анда таблица түзүлөт. Натыйжада макеттин толук формасы төмөнкүдөй болот (10.7-сүрөт).

Имя поля	Тип данных	Описание
Код	Счетчик	
Fam	Текстовый	Фамилиясы
Name	Текстовый	Аты
Phone_work	Текстовый	Иш телефону
Phone_hous	Текстовый	Үй телефону
Phone_mobil	Текстовый	Мобилдик телефон

10.7-сүрөт. Таблицанын талааларынын аталыштары

3). Түзүлгөн таблицаны 10.4-таблицадагы маалыматтар менен толтургандан кийин төмөнкү көрүнүштөгү таблица алынат.



Код	Fam	Name	Phone_work	Phone_hous	Phone_mobil
1	Акматов	Мурат	5-23-18	2-12-24	0552-41-35-84
2	Керимбекова	Айнура	4-36-90	5-56-34	0772-35-23-95
(Счетчик)					

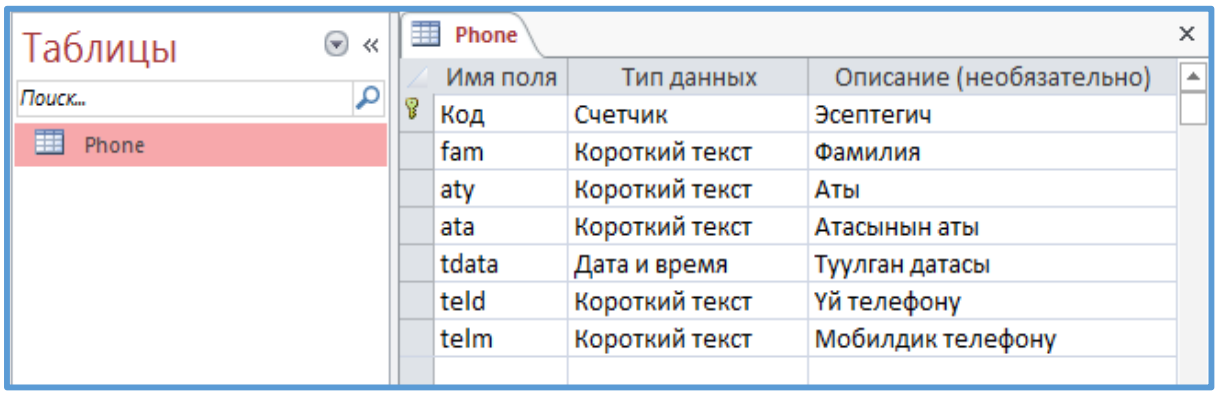
Запись: 3 из 3

10.8-сүрөт. Маалыматтар менен толтурулган таблица

§ 10.5. ADO технологиясы

Delphi’де берилгендер базасы менен иштөө программасын түзүүнү мисалда карайлы. Microsoft Access’те «Contact» деп аталган база түзөбүз жана аны менен иштөө үчүн Microsoft иштеп чыккан ADO (ActiveX Data Object) технологиясын колдонобуз.

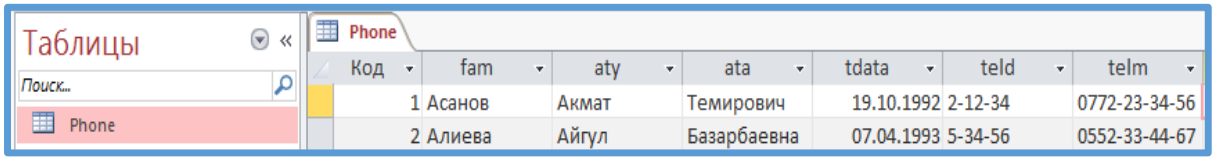
10.5.1. Microsoft Access’те база түзүү. D:\ дискине MIT аттуу каталог ачып, анын ичиндеги Data аттуу папкага Microsoft Access 2013 программасында «Contact.accdb» деп аталган база түзөбүз жана ага «Phone» деп аталган таблицаны (10.9-сүрөт) «Конструктор» режиминде түзүп жайгаштырабыз.



10.9-сүрөт. «Конструктор» режиминдеги«Phone» таблицанын структурасы

Мындан сырткары fam, aty, ata талааларынын «Размер поля» деген касиетине 30 санын, ал эми teld, telm талааларына – тиешелеш түрдө 10, 15 сандарын жазабыз.

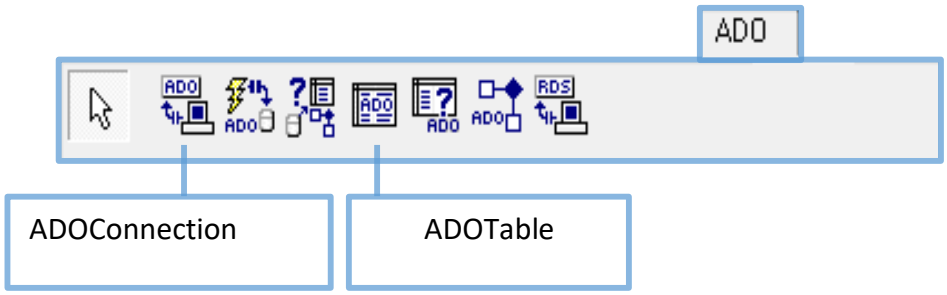
Андан кийин «Режим таблицы» режимине өтүп, таблицага бир нече маалыматтарды кийиребиз (10.10-сүрөт) жана аны сактап алабыз.



10.10-сүрөт. «Phone» таблицасын толтуруу мисалы

10.5.2. ADO технологиясында берилгендер базасы менен иштөө. Delphi 7 программасын жүктөп, жаңы колдонмо түзөбүз.

Башкы формага төмөнкү компоненттерди жайгаштырабыз: ADO барагынан (10.11-сүрөт) **ADOConnection** жана **ADOTable,** Data Access барагынан **DataSource,** ал эми Data Controls барагынан **DBGrid** ком-поненталарын жайгаштырабыз**.**



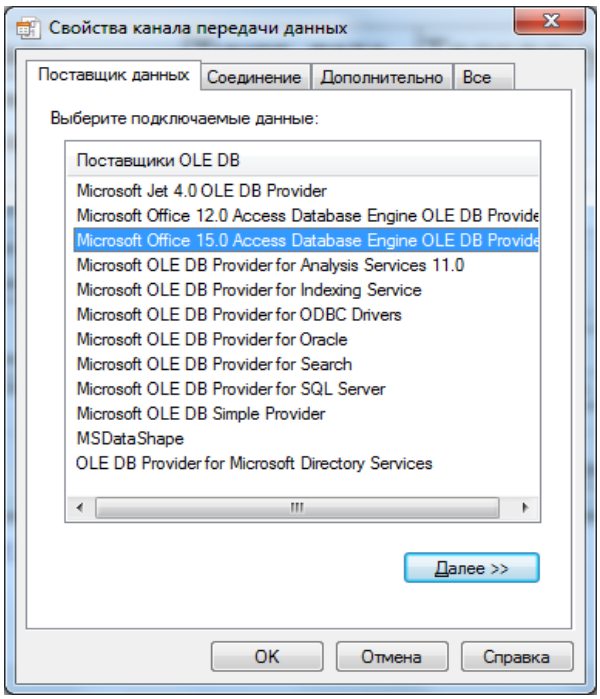
10.11-сүрөт. ADO барагынын компоненттери

1) ADO барагынан **ADOConnection1** компонентасынын формага коюп, анын касиеттерин төмөнкү таблица боюнча саздап алабыз.

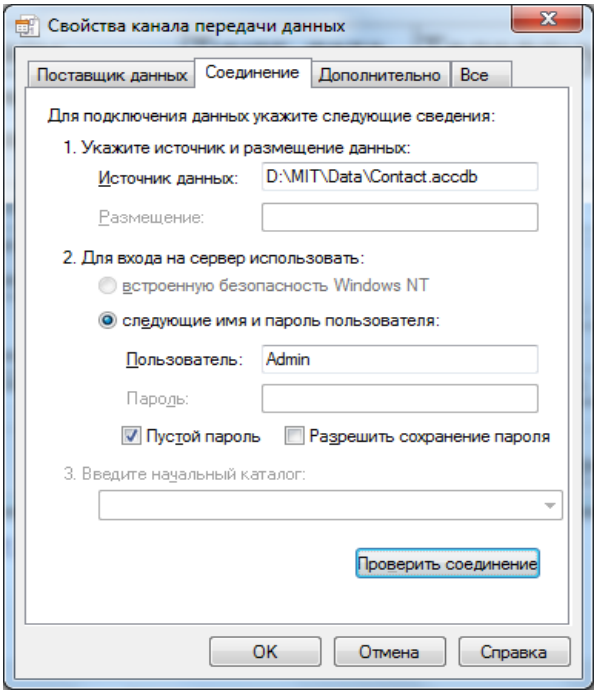
10.5-таблица. ADOConnection1 компонентасынын касиеттери

Касиети	**Мааниси**	**Баяндамасы**
ConnectionString	D:\MIT\Data\Contact.accdb	
LoginPrompt	False	
Mode	cmShareDenyNone	
Connected	True	

ConnectionString касиетинин үч чекиттүү баскычын басканда 10.3-сүрөттөгү «Поставщик данных» деген барактан «Microsoft Office 15.0 Access Database Engine OLE DB Provider» жолчосун тандап, «Далее» баскычын басканда 10.4-сүрөттөгү «Соединение» барагы ачылат. Андан «Источник данных» пунктуна берилгендер базасынын дарегин, б.а. D:\MIT\Data\Contact.accdb жолун көрсөтөбүз, «Пользователь» пунктундагы Admin маанисине макул болуп, «Пустой пароль» деген пунктка «птичка» белгисин коёбуз.

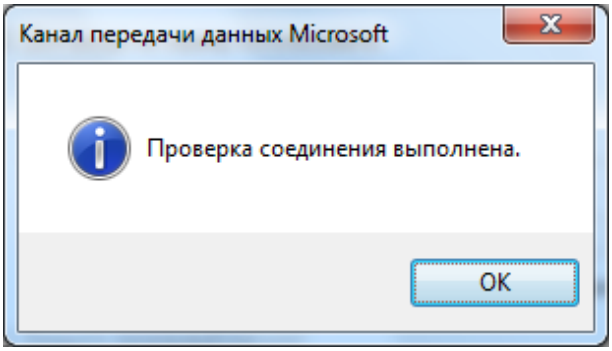


10.12-сүрөт. 1-кадам



10.13-сүрөт. 2-кадам

Берилгендер базасынын программага кошулгандыгын билүү үчүн «Проверить соединение» деген баскычын басабыз. Эгерде байланыш туура болгон болсо, анда монитордун экранына 10.14-сүрөттөгү маалымат чыгарылат. Андан кийин «ОК» баскычтарын басуу менен базаны кошуунун биринчи этабын аяктайбыз.



10.14-сүрөт. Байланышты текшерүү

2) ADO барагынан **ADOTable1** компонентасын формага коюп, анын касиеттерин төмөнкү таблица боюнча саздап алабыз.

10.6-таблица. ADOTable1 компонентасынын касиеттери

Касиети	**Мааниси**	**Баяндамасы**
Connection	ADOConnection1	
TableName	Phone	
Active	True	

3) Data Access барагынан **DataSource1** компонентин формага коюп, анын **DataSet** касиетине ADOTable1 маанисин жазабыз.

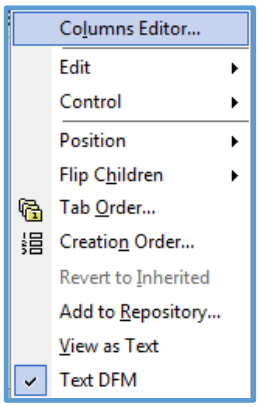
4) Data Controls барагынан **DBGrid1** компонентин формага коюп, анын DataSource касиетине **DataSource1** маанисин ыйгарабыз. Натыйжада формага төмөнкү көрүнүштөгү таблица чыгат.

Код	fam	aty	ata	tdata	teld	telm
1	Асанов	Акмат	Темирович	19.10.1992	2-12-34	0772-23-34-56
2	Алиева	Айгул	Базарбаевна	07.04.1993	5-34-56	0552-33-44-67

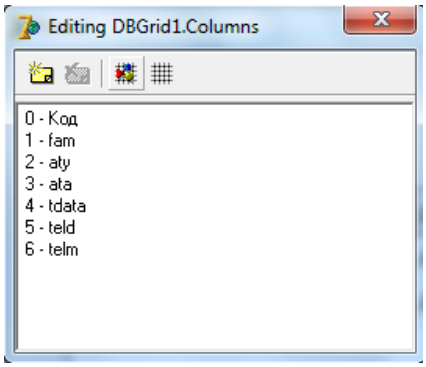
10.15-сүрөт. Маалыматтар менен толтурулган таблица

Бул жерде таблицанын талааларынын аталыштарын жана алардын туурасын саздаш үчүн DBGrid1 компонентасын тандап, андан кийин чычкандын оң баскычын бассак, 10.16-сүрөттөгү динамикалык меню пайда болот. Анын «Columns Editor» деген жолчосун тандасак, анда 10.17-сүрөттөгү «Editing DBGrid1.Columns» терезеси пайда болот.

Андагы ар бир жолчонун касиеттерин 10.6-таблицадагыдай кылып тандап алабыз.



10.16-сүрөт. Мамычалардын редактору



10.17-сүрөт. Мамычалардын аталыштарын өзгөртүү редактору

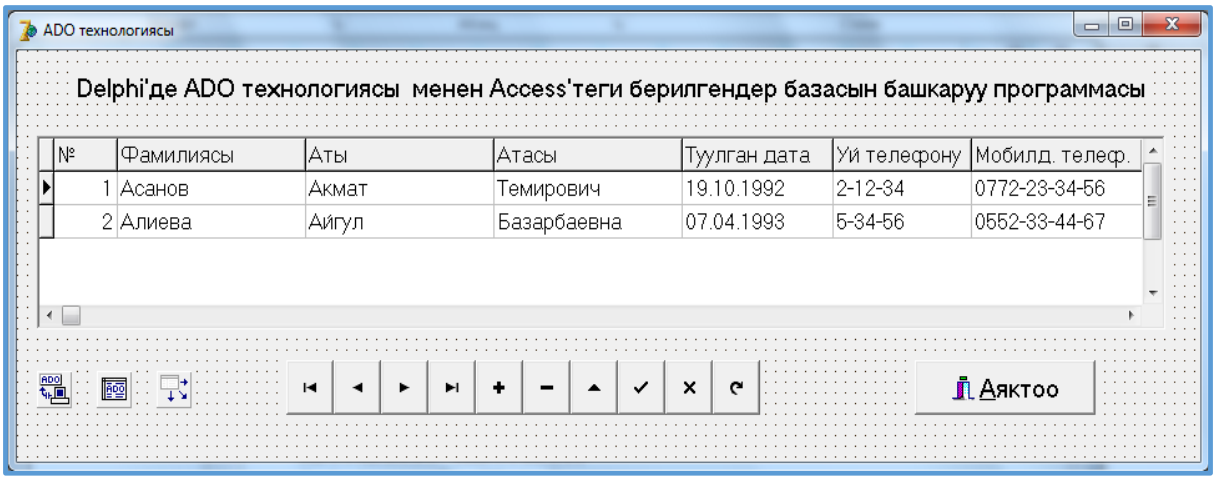
10.7-таблица. Талаалардын мүнөздөмөсү

	Title.Caption	Width
0-Код	№	50
1-fam	Фамилия	150
2-aty	Аты	150
3-ata	Атасы	150
4-tdata	Туулган датасы	120
5-teld	Үй телефону	110
6-telm	Мобилдик телефону	150

5) Data Controls барагынан **DBNavigator1** компонентин формага коюп, анын DataSource касиетине **DataSource1** маанисин жазабыз. Натыйжада DBGrid1 компонентасынын ар бир жолчосу менен иштөөгө мүмкүнчүлүк алабыз.

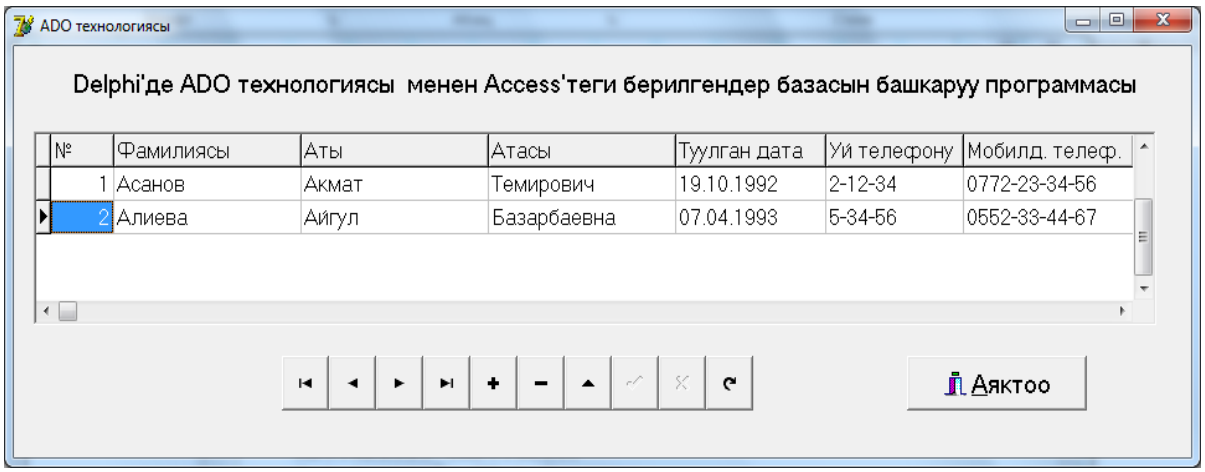
6) Additinol барагынан **BitBtn1** компонентин формага коюп, анын Kind касиетинин **bkClose** маанисин алабыз, ал эми Caption касиетине &Аяктоо деген маанини жазабыз.

Жогорудагы белгиленгендер аткарылгандан кийин биздин формабыз төмөнкүдөй көрүнүшкө келет:



10.18-сүрөт. Форманын көрүнүшү

Эгерде программаны жүктөсөк, анда берилгендер базасы менен иштөө программасы төмөнкүдөй болот:



10.19-сүрөт. Берилгендер базасы

10.1-эскертүү. Эгерде берилгендер базасы ***Microsoft Access 2000*** *же* ***Microsoft Access 2002-2003*** программасында түзүлсө (кеңейтилиши .mbd), анда «Поставщик данных» деген барактан «Microsoft Jet 4.0 OLE DB Provider» жолчосун тандап алуу керек.

10.2-эскертүү. Эгерде берилгендер базасы ***Microsoft Access 2010*** *же* ***Microsoft Access 2013*** программасында түзүлсө (кеңейтилиши .accbd), анда базаны ***База данных Microsoft Access 2000*** *же* ***База данных Microsoft Access 2002-2003*** форматтарында сактап алууга болот.

10.1-листинг. Программанын тизмеги

```
unit Contact;
        interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, Grids, DBGrids, DB, ADODB;
type
          TForm1 = class(TForm)
            ADOConnection1: TADOConnection;
            ADODataSet1: TADODataSet;
            DataSource1: TDataSource;
            DBGrid1: TDBGrid;
            procedure FormActivate(Sender: TObject);
            procedure FormClose(Sender: TObject; var Action:
TCloseAction);
```

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

var
        Form1: TForm1;

implementation

       {$R *.dfm}

procedure TForm1.FormActivate(Sender: TObject);
begin
      try
        ADOConnection1.Open ;
        ADODataSet1.Active :=True;
      except
        on e:Exception do begin
           DBGrid1.Enabled :=False;
           MessageDlg('Ошибка доступа к файлу C:\BD\Contacts.mdb',
                  mtError,[mbOK],0);
        end;
      end;
end;

procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
        if DBGrid1.EditorMode then
          begin
            ADODataSet1.UpdateBatch(arCurrent);
          end;
end;
end.
```

Колдонулган адабияттар

1. Архангельский А.Я. Программирование в Delphi. Учебник по классическим версиям Delphi. – М.: ООО «Бином-Пресс», 2013. – 816 с.

2. Культин Н. Основы программирования в Delphi 7. – 2-е изд., перераб. и доп. - СПб.: БХВ-Петербург, 2009. – 640 с.

3. Культин Н. Delphi 7 в задачах и примерах. - СПб.: БХВ-Петербург, 2006. – 288 с.

4. Осипов Д. Delphi. Профессиональное программирование. – СПб.: Символ-Плюс, 2006. – 1056 с.

5. Осмоналиев А.Б., Аркабаев Н.К. Borland Pascal 7.0. Программалоонун негиздери. 1-бөлүк. – Ош: «Ошоблбасмакана», 2008. – 256 б.

6. Осмоналиев А.Б., Аркабаев Н.К. Borland Pascal 7.0. Программалоонун негиздери. 2-бөлүк. – Ош: «Ошоблбасмакана», 2010. – 304 б.

7. Кайыпбердиев Н.А., Кожобеков К.Г., Мадазимов Р.М. Delphi мисалдар менен. Баштоочулар курсу. – Ош: “ДИП полиграфия”, 2006. – 120 б.