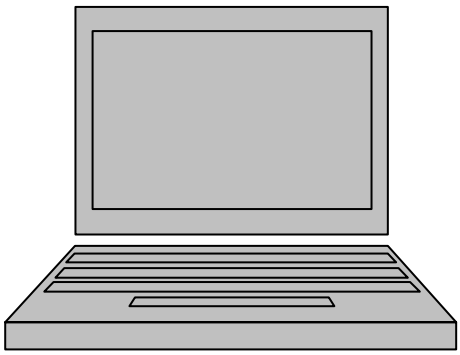
СРЕДСТВА РАЗРАБОТКИ ПРИЛОЖЕНИЙ ДЛЯ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ



♦ **ИЗДАТЕЛЬСТВО ТГТУ** ♦

УДК 004.382.7
ББК ←973я73-5
Е924

Утверждено Редакционно-издательским советом университета

Р е ц е н з е н т
Кандидат технических наук, доцент
М.Ю. Серегин

Е924 Средства разработки приложений для персональных компьютеров : метод. указания / Сост. О.В. Ефремов. – Тамбов : Изд-во Тамб. гос. техн. ун-та, 2006. – 40 с. – 100 экз.

Рассмотрены вопросы разработки приложений для операционных систем семейства Windows.

Предназначены студентам, обучающимся по специальности 261201 "Технология и дизайн упаковочного производства" и магистрантам, обучающимся по направлению 150400.26 "Технологические процессы, машины и оборудование комплексной химической переработки растительных полимеров".

УДК 004.382.7
ББК ←973я73-5

Министерство образования и науки Российской Федерации

ГОУ ВПО "Тамбовский государственный технический университет"

СРЕДСТВА РАЗРАБОТКИ ПРИЛОЖЕНИЙ ДЛЯ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

Методические указания

Тамбов
Издательство ТГТУ
2006

УЧЕБНОЕ ИЗДАНИЕ

СРЕДСТВА РАЗРАБОТКИ ПРИЛОЖЕНИЙ ДЛЯ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

Методические указания

С о с т а в и т е л ь
ЕФРЕМОВ Олег Владимирович

Редактор Е.С. М о р д а с о в а
Инженер по компьютерному макетированию М.Н. Р ы ж к о в а

Подписано в печать 26.12.2006.
Формат 60 × 84/16. Бумага офсетная. Гарнитура Times New Roman.
2,3 уч.-изд. л. Тираж 100 экз. Заказ № 865

Издательско-полиграфический центр
Тамбовского государственного технического университета
392000, Тамбов, Советская, 106, к. 14

ВВЕДЕНИЕ

Современное общество характеризуется повсеместным распространением информационных и компьютерных технологий. Умение работать на компьютере де-факто стало одним из требований к специалисту, причем в совершенно различных областях деятельности – от бизнеса до фундаментальной науки.

Конечно, быть высококвалифицированным программистом дано не каждому, однако знание принципов алгоритмизации, умение разбить задачу на составляющие, установить последовательность и взаимосвязь этапов ее решения могут помочь в любом деле.

В этой связи уместно будет кратко вспомнить основные этапы развития техники программирования.

Принято считать, что первым программируемым компьютером был *Colossus II*, созданный в 1944 г. в Великобритании. «Программировался» он достаточно своеобразно (с нынешней точки зрения): посредством изменения положения множества переключателей. Однако при этом он уже мог настраиваться на решение различных вычислительных задач.

Затем развитие компьютеров привело к появлению алгоритмических языков и систем программирования, которые занимают важное место в программном обеспечении современных ЭВМ. Основное их назначение – освободить программиста от необходимости работать на языке машинных команд. Язык программирования, с которым работает система программирования, называется ее входным языком. Системы программирования именуются по названию своего входного языка. Например: Бейсик-система, Паскаль-система. Иногда в название систем включаются префиксы, обозначающие, например, фирменное происхождение системы. Очень популярны системы с приставкой "Турбо": Турбо-Паскаль, Турбо-Си и др. Это системы программирования, разработанные фирмой *Borland*.

Системы программирования – это универсальные средства работы с информацией. С их помощью можно решать вычислительные задачи, обрабатывать тексты, получать графические изображения, осуществлять хранение и поиск данных и т.д., в общем, делать все, что делают средства прикладного программного обеспечения. Кроме того, сами эти средства (графические и текстовые редакторы, СУБД и др.) – это программы, написанные на языках программирования, созданные с помощью систем программирования.

Языки программирования претерпели большие изменения с тех пор, как в сороковых годах началось их использование.

Первые языки программирования были очень примитивными и мало чем отличались от формализованных упорядочений двоичных чисел (единиц и нулей), понятных компьютеру. Использование таких языков было крайне неудобно с точки зрения программиста, так как он должен был знать числовые коды всех машинных команд, должен был сам распределять память под команды программы и данные. На языках машинных команд трудно поддерживать структурную методику программирования.

Для того, чтобы облегчить общение человека с ЭВМ, были созданы языки программирования типа Ассемблер. Переменные величины стали изображаться символическими именами. Числовые коды операций заменились на мнемонические (словесные) обозначения, которые легче запомнить. Язык программирования приблизился к человеческому языку, но удалился от языка машинных команд. Чтобы ЭВМ могла работать на языке Ассемблера, потребовался транслятор – системная программа, переводящая текст программы на Ассемблере в эквивалентные ей машинные команды. Языки типа Ассемблер – машиноориентированные, потому что они настроены на структуру машинных команд конкретного компьютера. Разные компьютеры с разными типами процессоров имеют разный Ассемблер.

Вообще, первая программа была написана для аналитической машины Чарльза Бебиджа Адой Лавлейс. Она теоретически разработала некоторые приемы управления последовательностью вычислений, которые используются и по сей день, описала одну из важнейших конструкций практически любого языка программирования – цикл.

Революционным моментом в истории языков программирования стало появление системы кодирования машинных команд с помощью специальных символов, предложенной Джоном Моучли (Пенсильванский университет). Система команд увлекла одну из сотрудниц его компании – Грейс Мюррей Хоппер, которая посвятила свою жизнь компьютерам и программированию. Она вспоминает, что стала «третьим в мире программистом первого в мире большого цифрового компьютера» («Марк-1»).

В 1951 г. Хоппер создала первый в мире компилятор и ею же был введен сам этот термин. Компилятор осуществлял функцию объединения команд и в ходе трансляции производил организацию подпрограмм, выделение памяти компьютера, преобразование команд высокого уровня (в то время псевдокодов) в машинные команды.

Середина 1950-х гг. характеризуется стремительным прогрессом в области программирования. Роль программирования в машинных кодах стала уменьшаться. Начали появляться языки программирования нового типа, выступающие в роли посредника между машинами и программистами. Первым и наиболее распространенным был Фортран (1954 г. – фирма *IBM*).

В середине 1960-х гг. Томасом Курцем и Джоном Кемени был создан специализированный язык программирования, который состоял из простых английских слов – *BASIC* (1964 г.). В начале этого десятилетия все существующие языки программирования высокого уровня можно было пересчитать по пальцам, однако впоследствии их число достигло трех тысяч. В практической деятельности используется не более двух десятков. В 1960-е гг. были предприняты попытки преодолеть эту «разноголосицу» языков путем создания универсального языка программирования. Первым детищем был *PL*/1, 1967. Затем на эту роль претендовал АЛГОЛ-68 (1968 г.). Предполагалось, что эти языки вытеснят все остальные, но этого не случилось.

Языки программирования служат разным целям и их выбор определяется предпочтениями пользователя, пригодностью для данного компьютера и данной задачи. А задачи для компьютера бывают самые разнообразные: вычислительные, экономические, графические, экспертные и т.д. Такая разнотипность решаемых компьютером задач и определяет многообразие языков программирования. По всей видимости, в программировании наилучший результат достигается при индивидуальном подходе, исходящем из класса задачи, уровня и интересов программиста. Например, Бейсик широко употребляется при написании простых программ; Фортран являлся классическим языком программирования при решении на ЭВМ математических и инженерных задач; язык Кобол (1960 г.) был задуман как основной язык, ориентированный на деловые задачи, для массовой обработки данных в сфере управления и бизнеса; Пролог был разработан как язык программирования для создания систем искусственного интеллекта.

В конце 1950-х гг. плодом международного сотрудничества явился язык Алгол (алгоритмический язык). Он предназначался для записи алгоритмов, которые стоятся в виде последовательности процедур, применяемых для решения поставленных задач. Он значительно повлиял на развитие других языков.

Развитие идеи Алгола о структуризации разработки алгоритмов нашло отражение при создании в начале 1970-х гг. языка Паскаль швейцарским ученым Никлаусом Виртом. Хотя Паскаль первоначально создавался как учебный язык, его качества оказались настолько высоки, что им охотно пользуются и профессиональные программисты.

Не менее впечатляющей, в том числе и финансовой, удачи добился француз Филип Кан, разработавший систему Турбо-Паскаль. Суть его идеи состояла в объединении последовательных этапов обработки программы – компиляции, редактирования связей, отладки и диагностики ошибок – в едином интерфейсе.

Период с конца 1960-х до начала 1980-х гг. характеризуется бурным ростом числа различных языков, но, как это ни парадоксально, это был и период кризиса программного обеспечения. В январе 1975 г. руководство Пентагона распорядилось навести порядок в хаосе трансляторов и учредило даже специальный комитет для разработки универсального языка. Из сотни представленных проектов было выбрано два, победитель был объявлен в мае 1979 г. – язык АДА. Язык АДА – прямой наследник языка Паскаль. Он предназначен для создания и длительного (многолетнего) сопровождения больших программных систем, допускает возможность параллельной обработки, управления процессами в реальном времени и многое другое.

Большой отпечаток на современное программное обеспечение наложил язык Си (первая версия – 1972 г.), являющийся очень популярным в среде разработчиков систем программного обеспечения (включая и ОС). Си сочетает в себе черты как языка высокого уровня, так и машинно-ориентированного языка, допуская программиста ко всем машинным ресурсам, что не обеспечивают такие языки, как Бейсик и Паскаль.

Многие языки, первоначально разработанные для больших и малых ЭВМ, в дальнейшем были хорошо приспособлены к ПК. Хорошо вписались в «персоналки» не только Паскаль, Бейсик, Си, Лого, но и Лисп, Пролог – языки искусственного интеллекта.

В течение многих лет программное обеспечение строилось на основе операциональных и процедурных языков, таких как Фортран, Бейсик, Паскаль, Ада, Си. И сегодня они играют значительную роль при создании прикладных программных средств. Однако по мере эволюции языков программирования получили широкое распространение и другие, принципиально иные, подходы к созданию программ.

Классическое операциональное и/или процедурное программирование требует от программиста детального описания того, как решать задачу, т.е. формулировки алгоритма и его

специальной записи. При этом ожидаемые свойства результата обычно не указываются. Основные понятия языков этих групп – оператор и данные. При процедурном подходе операторы объединяются в группы – процедуры. Структурное программирование в целом не выходит за рамки этого направления, оно лишь фиксирует некоторые полезные приемы технологии программирования.

Принципиально иное направление в программировании связано с методологиями (иногда говорят – «парадигмами») непроцедурного программирования. К ним можно отнести объектно-ориентированное и декларативное программирование. Объектно-ориентированный язык создает окружение в виде множества независимых объектов. Каждый объект ведет себя подобно отдельному компьютеру, их можно использовать для решения задач как «черные ящики», не вникая во внутренние механизмы их функционирования. Из языков объектного программирования, популярных среди профессионалов, следует назвать прежде всего *C++*, для более широкого круга программистов предпочтительны среды типа *Delphi* и *Visual Basic*.

Выше приведена укрупненная классификация языков программирования (критерий – методологии программирования).

Широкое распространение персональных компьютеров с операционной системой *Windows* и необходимость быстрой разработки программ для решения прикладных задач вызвало к жизни такой класс программного обеспечения как средства быстрой разработки интерфейса приложений (*Rapid Application Interface Development*, *RAD*). Приложениями (*applications*) в данном случае называются программы для решения каких-либо прикладных задач. Об основах работы с одним из таких программных продуктов – *Borland Delphi* – и пойдет речь ниже.

1. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Объектно-ориентированное программирование (**ООП**) – *это методика разработки программ, в основе которой лежит понятие объекта как некоторой структуры, описывающей объект реального мира, его поведение*. Задача, решаемая с использованием методики **ООП**, описывается в терминах объектов и операций над ними, а программа при таком подходе представляет собой набор объектов и связей между ними.

Обобщенное определение:

Объект ООП *– это совокупность переменных состояния и связанных с ними методов (операций). Эти методы определяют, как объект взаимодействует с окружающим миром.*

Далее рассмотрим коротко основные термины ООП:

Класс (*class*) – это группа данных и методов (функций) для работы с этими данными. Это своего рода шаблон. Объекты с одинаковыми свойствами, т.е. с одинаковыми наборами переменных состояния и методов, образуют класс.

Объект (*object*) – это конкретная реализация, экземпляр класса. В программировании отношения объекта и класса можно сравнить с описанием переменной, где сама переменная (объект) является экземпляром какого-либо типа данных (класса).

Обычно, если объекты соответствуют конкретным сущностям реального мира, то классы являются некими абстракциями, выступающими в роли понятий. Класс можно воспринимать как шаблон (чертеж) объекта. Для формирования какого-либо реального объекта необходимо иметь шаблон, на основании которого и строится создаваемый объект. При рассмотрении основ ООП часто смешивают понятие объекта и класса. Дело в том, что класс – это некоторое абстрактное понятие. Для проведения аналогий или приведения примеров оно не очень подходит. Намного проще приводить примеры, основываясь на объектах из реального мира, а не на абстрактных понятиях. Поэтому, говоря, например, про наследование, прежде всего имеется в виду наследование классов (шаблонов), а не объектов, хотя часто применяется и слово объект. Скажем так: объект – это физическая реализация класса (шаблона).

Методы (*methods*) – это функции (процедуры), принадлежащие классу. Методы класса (процедуры и функции, объявление которых включено в описание класса) выполняют действия над объектами класса. Чтобы метод был выполнен, надо указать имя объекта и имя метода, отделив одно имя от другого точкой. Например, инструкция *professor*.*Show*; вызывает применение метода *show* к объекту *professor*.

Отличительные черты объектно-ориентированного программирования – это:

- наследование;
- инкапсуляция;
- полиморфизм.

1. Наследование (***Inheritance***). Наследование – это процесс, посредством которого один объект может наследовать свойства другого объекта и добавлять к ним черты, характерные только для него, т.е. концепция объектно-ориентированного программирования предполагает возможность определять новые классы посредством добавления полей, свойств и методов к уже существующим классам.

2. Инкапсуляция (***Encapsulation***). Инкапсуляция – это механизм, который объединяет данные и методы, манипулирующие этими данными, и защищает и то и другое от внешнего вмешательства или неправильного использования. Когда методы и данные объединяются таким способом, создается объект. Переменные состояния объекта скрыты от внешнего мира. Изменение состояния объекта (его переменных) возможно ТОЛЬКО с помощью его методов (операций).

3. Полиморфизм (***Polymorphism***). **Полиморфизм** – это свойство, которое позволяет одно и тоже имя использовать для решения нескольких технически разных задач. Говоря иначе, полиморфизм – это возможность использовать одинаковые имена для методов, входящих в различные классы. Концепция полиморфизма обеспечивает при применении метода к объекту использование именно того метода, который соответствует классу объекта.

2. СРЕДА БЫСТРОЙ РАЗРАБОТКИ ИНТЕРФЕЙСА ПРИЛОЖЕНИЙ *DELPHI*

Разработка приложения проводится в две стадии:

1. Создание интерфейса (выбор и расположение на формах требуемых компонентов).
2. Обеспечение требуемой функциональности (определение процедур обработки событий).

Среда *Delphi* избавляет программиста от необходимости самому писать программный код, отвечающий за внешний вид интерфейса: отрисовку окон, изменение внешнего вида компонентов (например, кнопок в нажатом и не нажатом состояниях) и т.д. Вместо этого он просто выбирает нужные ему компоненты из соответствующей палитры, помещает их на форму и определяет требуемые свойства и события с помощью Инспектора объектов (*Object Inspector*). Более подробно интерфейс среды *Delphi* и порядок работы в ней описан ниже в лабораторной работе № 1.

3. ОСНОВЫ АЛГОРИТМИЧЕСКОГО ЯЗЫКА *OBJECT PASCAL*

Object Pascal – высокоуровневый, строго типизированный язык. Он характеризуется простотой чтения кода, быстрой компиляцией, и использует подключение модулей для модульного программирования.

Алфавит языка

Object Pascal использует *ASCII*-символы:

- латинские буквы: **A ... Z a ... z**
- арабские цифры: **0 ... 9**
- одиночные специальные символы

 # $ & ' ( ) * + , – . / : ; < = > @ [ ] ^ { }

- парные специальные символы **(* (. *) .) .. // := <= >= <>**

Символ [эквивалентен паре символов (.; символ] эквивалентен паре символов .); пары символов (* *) эквивалентны символам { }.

Буквы русского алфавита не входят в состав алфавита языка. Их использование допустимо только в строковых и символьных значениях.

Нет различий при использовании больших и малых букв в записи имен переменных, процедур, функций и меток. Их максимальная длина ограничена 126 символами.

В *Object Pascal* в качестве ограничителей комментария могут также использоваться пары символов **(***, ***)** и //. Скобки **(*...*)** используются подобно фигурным скобкам, т.е. комментарием считается находящийся в них фрагмент текста, а символы // указывают компилятору, что комментарий располагается за ними и продолжается до конца текущей строки:

{ Это комментарий }
(Это тоже комментарий *)*
// Все символы до конца этой строки – комментарий

Структура проекта

```
program Project1;
uses
 Forms,
 Unit1 in 'Unit1.pas' {Form1};
{$R *.RES}

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

Имя файла с кодом

Имя фор-

Строка 1 содержит заголовок программы. Объявление ***uses*** занимает строки с 3 по 6. В строке 8 находится директива компилятору, которая связывает файлы ресурсов проекта с программой. Строки 10 – 14 содержат блок описаний, который выполняется при запуске программы. Как и все файлы кода, так и проектный файл, заканчивается словом ***end*** с точкой.

Это обычный проектный файл. Проектные файлы обычно короткие, так как большинство логики в программе находится в unit-файлах. Проектные файлы генерируются и заполняются автоматически, редко возникает необходимость редактировать их вручную.

Заголовок программы содержит имя программы. Он состоит из зарезервированного слова ***program***, следующий за ним идентификатор заканчивается точкой с запятой. Идентификатор должен соответствовать имени файла проекта. В предыдущем примере программа называлась *Project1*, следовательно, проектный файл должен называться *Project1.dpr*.

Структура и синтаксис модуля

Модули – это программные единицы, предназначенные для размещений фрагментов программ. Следующий фрагмент программы является синтаксически правильным вариантом модуля:

```
unit Unit1;
interface
// Секция интерфейсных объявлений
implementation
// Секция реализации
end.
```

В секции интерфейсных объявлений описываются программные элементы (типы, классы, процедуры и функции), которые будут "видны" другим программным модулям, а в секции реализации раскрывается механизм работы этих элементов. Разделение модуля на две секции обеспечивает удобный механизм обмена алгоритмами между отдельными частями одной программы. Он также реализует средство обмена программными разработками между отдельными программистами. Получив откомпилированный "посторонний" модуль, программист получает доступ

только к его интерфейсной части, в которой, как уже говорилось, содержатся объявления элементов. Детали реализации объявленных процедур, функций, классов скрыты в секции реализации и недоступны другим модулям.

Элементы программы

Элементы программы – это минимальные неделимые ее части, еще несущие в себе определенную значимость для компилятора. К элементам относятся:

- зарезервированные слова;
- идентификаторы;
- типы;
- константы;
- переменные;
- метки;
- подпрограммы;
- комментарии.

Зарезервированные слова – это английские слова, указывающие компилятору на необходимость выполнения определенных действий. Зарезервированные слова не могут использоваться в программе ни для каких иных целей кроме тех, для которых они предназначены.

Зарезервированные слова

and	***except***	***label***	***resourcestring***
array	***exports***	***library***	***set***
as	***file***	***mod***	***shl***
asm	***finalization***	***nil***	***shr***
begin	***finally***	***not***	***string***
case	***for***	***object***	***then***
class	***function***	***of***	***threadvar***
const	***goto***	***or***	***to***
constructor	***if***	***out***	***try***
destructor	***implementation***	***packed***	***type***
dispinterface	***in***	***procedure***	***unit***
div	***inherited***	***program***	***until***
do	***initialization***	***property***	***uses***
downto	***inline***	***raise***	***var***
else	***interface***	***record***	***while***
end	***is***	***repeat***	***withxor***

Типы – это специальные конструкции языка, которые рассматриваются компилятором как образцы для создания других элементов программы, таких как переменные, константы и функции.

Основные типы данных

К основным типам данных языка *Delphi* относятся:

- целые числа (*integer*);
- дробные числа (*real*);
- символы (*char*);
- строки (*string*);
- логический тип (*boolean*).

Целые числа и числа с плавающей точкой могут быть представлены в различных форматах.

Массивы в *Object Pascal* во многом схожи с аналогичными типами данных в других языках программирования. Отличительная особенность массивов заключается в том, что все их компоненты по сути данные одного типа. Эти компоненты можно легко упорядочить и обеспечить доступ к любому из них простым указанием его порядкового номера.

Тип	Диапазон возможных значений	Размер памяти для хранения данных
Integer	–2147483648…2147483647	4 байта (32 бита)
Cardinal	0…4294967295	4 байта (32 бита)
Shortint	–128…127	1 байт (8 бит)
Smallint	–32768…32767	2 байта (16 бит)
Longint	–2147483648…2147483647	4 байта (32 бита)
Int64	$-2^{63}…2^{63}-1$	8 байт (64 бита)
Byte	0…255	1 байт (8 бит)
Word	0…65535	2 байта (16 бит)
Longword	0…4294967295	4 байта (32 бита)

Тип	Диапазон возможных значений	Максимальное количество цифр в числе	Размер в байтах
Real48	$2{,}9\cdot10^{-39}…1{,}7\cdot10^{38}$	11-12	6
Real	$5{,}0\cdot10^{-324}…1{,}7\cdot10^{308}$	15-16	8
Single	$1{,}5\cdot10^{-45}…1{,}7\cdot10^{38}$	7-8	4
Double	$5{,}0\cdot10^{-324}…1{,}7\cdot10^{308}$	15-16	8
Extended	$3{,}6\cdot10^{-4951}…1{,}1\cdot10^{4932}$	19-20	10
Comp	$-2^{63}+1…2^{63}$	19-20	8
Currency	–922337203685477,5808… 922337203685477,5807	19-20	8

Тип	Максимальная длина строки	Память, отводимая для хранения строки	Примечание
ShortString	255 символов	От 2 до 256 байт	
ANSIString	2^{31}	От 4 байт до 2 Гбайт	8-битовые
WideString	2^{30}	От 4 байт до 2 Гбайт	Unicode

Массивы

Описание типа массива задается следующим образом:

<имя типа> = array [<сп.инд.типов>] of <тип>

В качестве индексных типов в *Object Pascal* можно использовать любые порядковые типы, имеющие объем не более 2 Гбайт (т. е. кроме *LongWord* И *Int64*).

Определить переменную как массив можно и непосредственно при описании этой переменной, без предварительного описания типа массива, например:

```
Var
a,b : array [1..10] of Real;
```

Динамические массивы

С версии *Delphi* 4 впервые введены так называемые динамические массивы. При объявлении таких массивов в программе не следует указывать границы индексов:

```
Var
A: array of Integer;
B: array of array of Char;
C: array of array of array of Real
```

В этом примере динамический массив ***A*** имеет одно измерение, массив ***B*** – два и массив ***C*** – три измерения. Распределение памяти и указание границ индексов по каждому измерению динамических массивов осуществляется в ходе выполнения программы путем инициации массива с помощью функции ***SetLength***. В ходе выполнения такого оператора:

```
SetLength(A,3);
```

одномерный динамический массив **A** будет инициирован, т.е. получит память, достаточную для размещения трех целочисленных значений. Нижняя граница индексов по любому измерению динамического массива всегда равна 0, поэтому верхней границей индексов для **A** станет 2.

Основные операторы, функции и конструкции *Object Pascal* по своей сути подобны аналогичным в других языках программирования и отличаются от них только синтаксисом, поэтому в данных методических указаниях не рассматриваются.

4. КУЛЬТУРА ПРОГРАММИРОВАНИЯ

При написании программ часто забывают о так называемой культуре программирования, т.е. о следовании определенным принципам, призванным обеспечить, с одной стороны, понятность и удобочитаемость программы для программиста, а с другой – дружественность интерфейса для конечного пользователя.

Между тем, существуют несколько достаточно простых правил, следование которым поможет избежать описанных выше сложностей:

- имена переменным следует присваивать таким образом, чтобы по ним (именам) было понятно, что за данные в них хранятся. Например *StudentsCount*, *BarWidth*, а не *sc*, *a*, *b* и т.д.;
- имена компонентов следует выбирать так, чтобы первые буквы имени указывали на тип компонента, например, btnStart (кнопка *Button*), *edtVvod* (поле ввода *Edit*), *lblInfo* (метка *Label*), *imgPhoto* (картинка *Image*) и т.д.;
- в программу обязательно следует вставлять комментарии, облегчающие понимание логики ее работы, особенно если у программиста был длительный перерыв в работе с кодом данной программы.

При разработке интерфейса также желательно следовать некоторым правилам:

- интерфейс должен предоставлять необходимую пользователю информацию самым удобным для него образом;
- интерфейс должен легко осваиваться персоналом, быть прост в использовании, интуитивно понятен и устойчив к ошибкам;
- не следует перегружать интерфейс элементами. Если их оказывается слишком много на одной форме, можно использовать компонент *PageControl* и распределить компоненты по его страницам или использовать несколько форм;
- следует использовать согласованную терминологию и сокращения;
- необходимо предусматривать средства вывода пояснительных сообщений;
- следует использовать визуальное выделение пространства и цвет.

ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа 1 (2 часа)

ЗНАКОМСТВО С ИНТЕРФЕЙСОМ *DELPHI*

Цель работы: познакомиться с внешним видом и возможностями среды быстрой разработки приложений *Delphi*.

Порядок выполнения работы

1. Изучить интерфейс среды быстрой разработки интерфейса приложений *Delphi*.
2. Познакомиться с основными вкладками палитры компонентов.
3. Научиться размещать компоненты на форме.
4. Освоить работу с *Object Inspector*.
5. Изучить состав проекта *Delphi*.

Методические указания

Среда быстрой разработки интерфейса приложений (*RAD – Rapid Application Interface Development*) *Delphi* предназначена для ускорения разработки интерфейса приложений для

операционной системы *Windows* с тем, чтобы основное внимание уделить их функциональности.

Интерфейс среды *Delphi* представлен на рис. 1.

В верхней части окна расположено главное меню. Это стандартное меню в стиле *Windows*. Около большинства команд меню изображены пиктограммы. Некоторые из них расположены ниже, в левой части панели инструментов. Пользуясь ими, можно открывать и сохранять проекты, просматривать экранные формы и их программный код, включать в состав проекта новую форму, запускать приложение для выполнения.

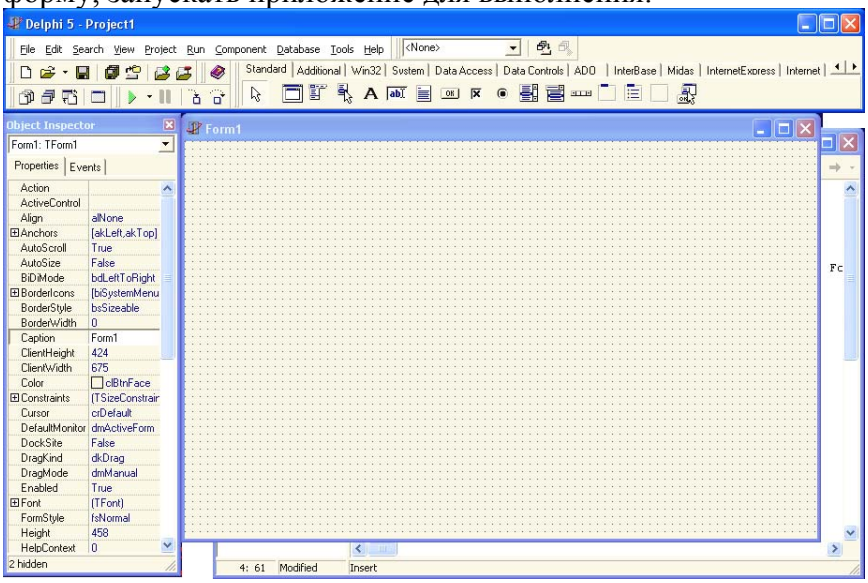


Рис. 1. Интерфейс Delphi

По умолчанию при старте *Delphi* сразу создается одна форма (заготовка для будущего окна создаваемого приложения).

На ней размещают необходимые компоненты. Последние сгруппированы по типам на разных вкладках палитры компонентов (рис. 2 – 6).

В левой нижней части расположен инспектор объектов (*Object Inspector*). Это диалоговое окно отображает списки всех свойств (***Properties***) и событий (***Events***) одного или более компонентов, выбранных в проектируемой экранной форме.

В правой нижней части (самой большой по площади) расположены окно проектирования экранной формы и окно редактора кода, переключаться между ними можно с помощью клавиши *F12* или соответствующей пиктограммы.

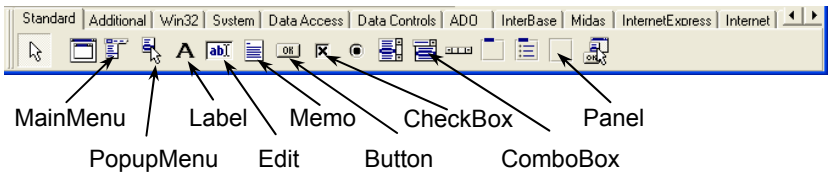


Рис. 2. Вкладка *"Standard"* ("Стандартная")

Проект в *Delphi* состоит из трех основных типов файлов:

- **.dpr* (в нем хранится код проекта);
- **.pas* (в каждом из них хранится код одной экранной формы);
- **.dfm* (в каждом из них хранится информация о расположении и визуальных настройках компонентов форм).

Ниже представлены наиболее часто используемые палитры и расположенные на них компоненты.

MainMenu	Главное меню – позволяет поместить главное меню в программу.
PopupMenu	Контекстное меню – вызывается по нажатии правой кнопки мыши.
Label	Метка – служит для отображения на экране однострочного текста.
Edit	Поле ввода – стандартный управляющий элемент *Windows* для ввода.
Memo	*Memo* – служит для ввода и отображения многострочного текста.

Button	Кнопка – позволяет выполнить какие-либо действия при нажатии на нее во время выполнения программы.
CheckBox	Флажок опции – предназначен для установки или сброса какой-либо настройки.
ComboBox	Выпадающий список – предназначен для показа прокручиваемого списка, позволяет также вводить информацию в маленьком поле ввода сверху.
Panel	Панель – управляющий элемент, используется в декоративных целях и как контейнер для других элементов.

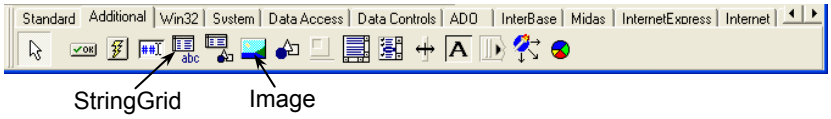


Рис. 3. Вкладка *"Additional"* ("Дополнительная")

StringGrid	Строковая решетка – используется при отображении информации в виде таблицы.
Image	Картинка – может отображать графический файл на форме или использоваться для "рисования" на нем во время выполнения программы.

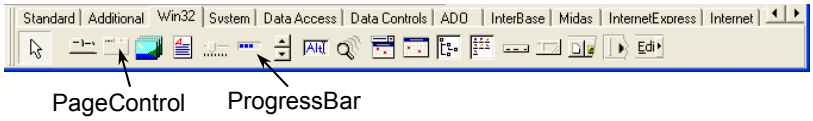


Рис. 4. Вкладка *"Win32"* ("Для 32*bit-Windows*")

PageControl	Окно со страницами-вкладками – используется для размещения многих элементов на небольшом пространстве (чтобы не перегружать интерфейс).
ProgressBar	Прогресс-индикатор – служит для отображения степени выполнения какого-либо процесса.

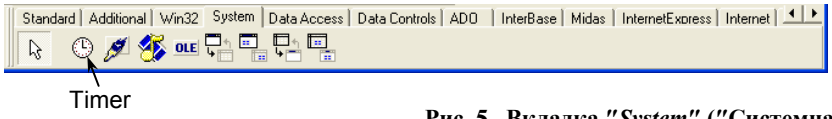


Рис. 5. Вкладка *"System"* ("Системная")

Timer	Таймер – используется для отсчета времени.

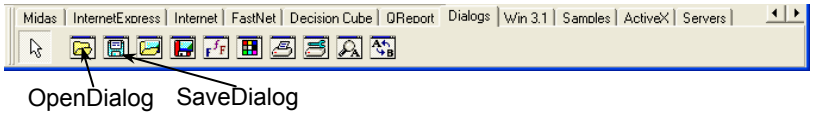


Рис. 6. Вкладка *"Dialogs"* ("Диалоги")

OpenDialog	Диалог выбора открываемого файла – активизирует стандартное окно выбора файла.
SaveDialog	Диалог сохранения – активизирует стандартное окно сохранения файла.

Содержание отчета

1. Название и цель работы.
2. Перечень типов файлов, из которых состоит проект *Delphi*.
3. Перечень основных вкладок палитры и расположенных на них часто используемых компонентов с описаниями последних.

Контрольные вопросы

- К какому классу программного обеспечения относится Delphi?
- Каковы основные элементы интерфейса *Delphi*?
- Что такое *Object Inspector* и для чего он используется?
- Какие часто используемые компоненты расположены на вкладке *"Standard"* палитры компонентов *Delphi*?
- Какие часто используемые компоненты расположены на вкладке *"Additional"* палитры компонентов *Delphi*?
- Какие часто используемые компоненты расположены на вкладке *"Win32"* палитры компонентов *Delphi*?

- Какие часто используемые компоненты расположены на вкладке *"System"* палитры компонентов *Delphi*?
- Какие часто используемые компоненты расположены на вкладке *"Dialogs"* палитры компонентов *Delphi*?

Лабораторная работа 2 (4 часа)

ОСНОВЫ РАЗРАБОТКИ ПРИЛОЖЕНИЙ В СРЕДЕ *DELPHI*

Цель работы: Познакомиться с основными настройками проекта *Delphi* и создать простое приложение, содержащее следующие компоненты: форму, поле ввода, кнопку и метки.

Порядок выполнения работы

1. Запустить среду *Delphi*, создать в ней новый проект и сохранить его в отдельной папке на жестком диске компьютера.
2. Разместить на новой форме указанные компоненты и изменить их свойства, перечисленные ниже.
3. Определить процедуру обработки нажатия кнопки для того, чтобы приложение выполняло необходимые действия.
4. Отладить приложение и сохранить проект.
5. Проверить функциональность приложения, запустив созданный *.exe*-файл.

Методические указания

Проект в *Delphi* состоит из трех основных типов файлов:

- **.dpr* (в нем хранится код проекта);
- **.pas* (в каждом из них хранится код одной экранной формы);
- **.dfm* (в каждом из них хранится информация о расположении и визуальных настройках компонентов форм).

Иными словами, проект *Delphi* – это целая группа файлов, поэтому для их хранения целесообразно создавать отдельный каталог.

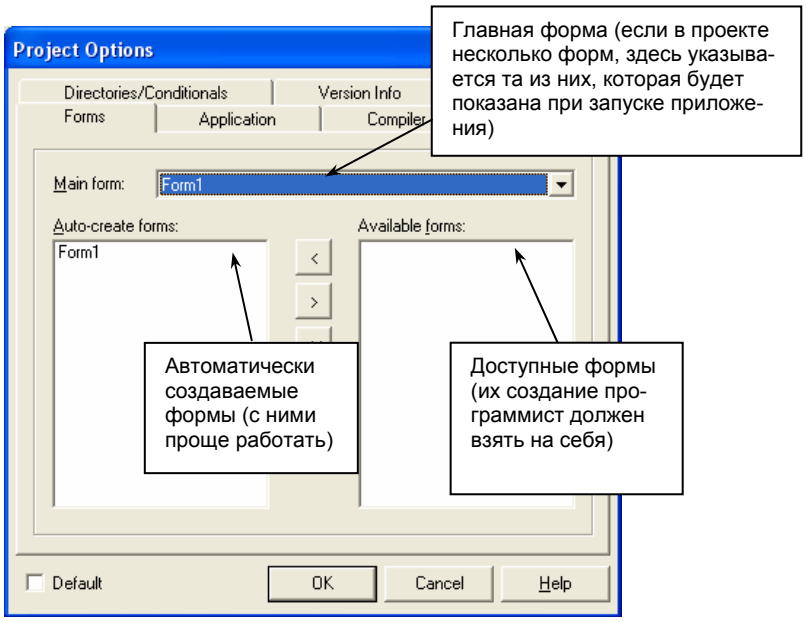


Рис. 1. Вкладка *Forms* настроек проекта *Delphi*

Каждый проект в *Delphi* имеет некоторые основные настройки, перейти к определению которых можно, например, вызвав пункты верхнего меню *Project* – *Options*… или нажав комбинацию клавиш *Shift-Ctrl-F11*. По умолчанию откроется вкладка *Forms* (Формы), изображенная на рис. 1. Здесь можно задавать **Главную форму** (*Main form*), автоматически соз-

даваемые формы и доступные формы (создаваемые программистом). На рис. 2 показана следующая вкладка окна настроек проекта – *Application* (Приложение).

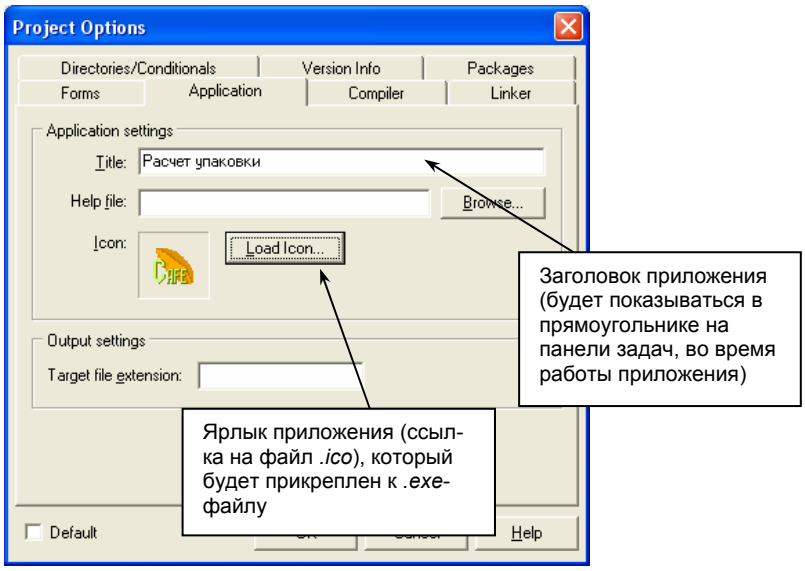


Рис. 2. Вкладка *Application* настроек проекта *Delphi*

После запуска среды *Delphi* автоматически создается заготовка нового проекта с одной пустой формой. Открывать и сохранять проекты, переключаться из окна формы в окно кода, запускать программу на выполнение и т.д. можно, пользуясь кнопками на панелях управления (рис.3).

При запуске проекта на выполнение (при нажатии кнопки с зеленой стрелкой (рис. 3) или *F*9) в папке с файлами проекта создается исполняемый файл с расширением *.exe*, который можно использовать самостоятельно, т.е. сам по себе.

Рассмотрим далее процесс создания простого приложения в среде *Delphi*, содержащего следующие компоненты: форму, поле ввода, кнопку и метки.

Форма (объект класса *TForm*) является основным компонентом *Delphi*, служащим для размещения на нем всех остальных объектов (так называемым контейнером).

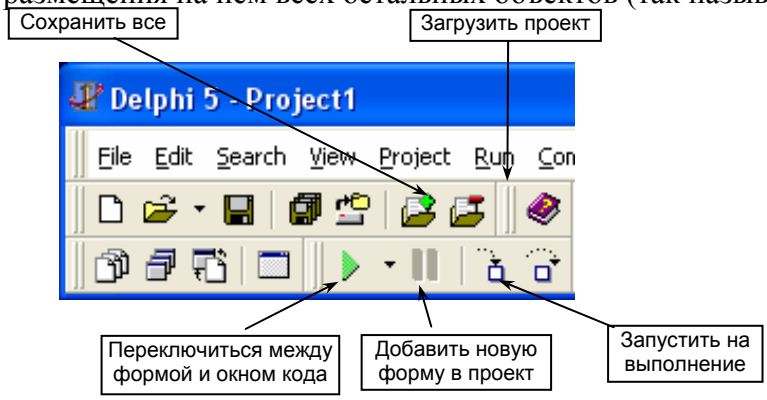


Рис. 3. Кнопки управления проектом *Delphi*

Основными свойствами формы являются: *Name*, *BorderStyle*, *Caption*, *Color*, *Position* и *Scaled*. Ниже в таблице приведено описание некоторых из них.

Caption	Заголовок, он отображается в заголовке формы (вверху)
Name	Имя, должно начинаться с *frm*
Position	Положение формы при запуске приложения, рекомендуется указывать *poScreenCenter* (в центре экрана)

Имя – это основное свойство любого компонента, так как это идентификатор, с помощью которого можно обратиться к конкретному компоненту и производить с ним различные действия.

Создайте в среде *Delphi* новое приложение и разместите на форме следующие компоненты: два поля ввода (*Tedit*), одну кнопку (*TButton*) и три метки (*TLabel*), как показано на рис. 4.

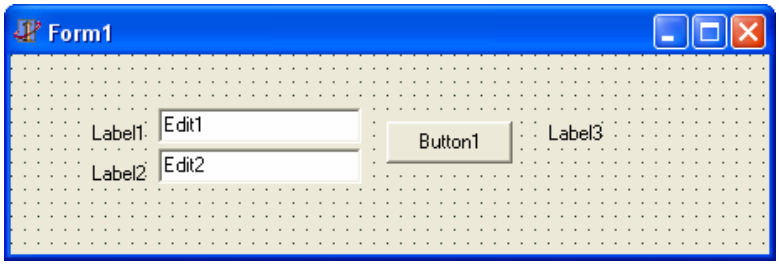


Рис. 4. Пример создания простого приложения

Цель создания данного приложения – по нажатию кнопки вычислять среднее арифметическое двух чисел, вводимых до этого в поля ввода, и отображать результат в третьей метке.

После размещения компонентов на форме надо прежде всего изменить их имена в соответствии с правилами культуры программирования. Присвойте имя *frmMain* форме, имена *lblInfo1* и *lblInfo2* первым двум меткам, имена *edtChislo1* и *edtChislo2* полям ввода, имя *btnCalc* кнопке и имя *lblResult* третьей метке. Первые две метки выполняют информационную функцию и нужны для того, чтобы сообщить пользователю, что именно нужно вводить в поля ввода. Чтобы эта информация отображалась в метках, надо изменить их свойства *Caption* в окне *Object Inspector*. (например, ″Число1:″ и ″Число2:″).

Текст, отображаемый в полях ввода, содержится в свойстве *Text* каждого из полей. Присвойте значениям этих свойств пустые строки, чтобы при старте приложения в полях ввода ничего не отображалось.

Текст, отображаемый на кнопке, также хранится в ее свойстве *Caption*. Присвойте ему значение ″Расчет″.

Наконец, свойству Caption третьей метки присвойте значение ″Результат″.

После этих действий форма примет вид, показанный на рис. 5.

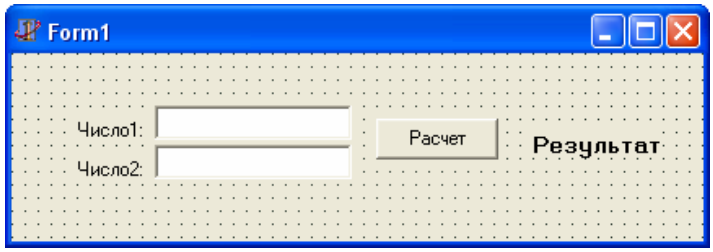


Рис. 5. Вид окна приложения после внесенных изменений

Затем надо определить процедуру, выполняющуюся по нажатию кнопки. Для этого надо выделить кнопку на форме, перейти на вкладку *Events* (События) в окне *Object Inspector* и дважды щелкнуть на пустом поле справа от события *OnClick* (По щелчку). После этого автоматически откроется окно кода:

```
procedure TfrmMain.btnCalcClick(Sender: TObject);
begin

end;
```

В нем можно видеть заготовку для процедуры, обрабатывающей нажатие кнопки. В общем виде идея выглядит так: надо считать из полей ввода введенные числа. Затем надо преобразовать их из строковой формы в числовую (эта операция нужна потому, что свойство *Text* поля ввода имеет именно строковый тип). После этого уже можно присвоить полученные значения определенным переменным, произвести необходимые расчеты и затем вывести полученный результат в метку, произведя обратное преобразование числа в строку символов. Все эти операции логично описать в процедуре, выполняющейся после нажатия кнопки *btnCalc*. Готовая процедура будет выглядеть так:

```
procedure TfrmMain.btnCalcClick(Sender: TObject);
{описание переменных}
var
 chislo1,chislo2,res: single;
begin
{преобразование содержимого Tedit в числовую форму}
```

```
 chislo1:=StrToFloat(edt1.text);
 chislo2:=StrToFloat(edt2.text);

{вычисление результата}
 res:=(chislo1+chislo2)/2;
{вывод результата в метку с именем lblResult}
 lblResult.caption:=FloatToStr(res);
end;
```

В приведенном выше коде использованы следующие встроенные функции: *StrToFloat* – для преобразования строки символов в число и *FloatToStr* – для обратного преобразования.

Содержание отчета

1. Название и цель работы.
2. Перечень типов файлов, из которых состоит проект *Delphi*.
3. Основные настройки проекта *Delphi*.
4. Последовательность создания нового приложения, описание использованных компонентов и их основных свойств.
5. Программный код процедуры, обрабатывающей нажатие кнопки с комментариями и список встроенных функций, использованных для преобразования строки символов в число и наоборот.

Контрольные вопросы

- Из файлов каких типов состоит проект *Delphi*?
- Каковы основные настройки проекта *Delphi*?
- Какие кнопки имеются на панели инструментов *Delphi* для управления проектом?
- Каковы основные свойства объектов *TForm*, *TLabel*, *TEdit*, *TButton*? Для чего и как они используются?
- Как определяются процедуры обработки событий в *Delphi*?
- Какие встроенные функции существуют в *Delphi* для преобразования строки символов в числовую форму и наоборот?

Лабораторная работа 3 (4 часа)

РАЗРАБОТКА ПРИЛОЖЕНИЯ В СРЕДЕ *DELPHI* С ИСПОЛЬЗОВАНИЕМ КОМПОНЕНТОВ *COMBOBOX*, *PANEL*, *IMAGE*

Цель работы: В среде *Delphi* создать простое приложение, содержащее следующие компоненты: форму, выпадающий список, панель, картинку и метку.

Порядок выполнения работы

1. Запустить среду *Delphi*, создать в ней новый проект и сохранить его в определенной папке на жестком диске компьютера.
2. Разместить на новой форме указанные компоненты и изменить их свойства, перечисленные ниже.
3. Определить процедуру обработки выбора пункта из выпадающего списка для того, чтобы приложение выполняло необходимые действия.
4. Отладить приложение и сохранить проект.
5. Проверить функциональность приложения, запустив созданный *.exe*-файл.

Методические указания

Рассмотрим далее процесс создания простого приложения в среде Delphi, содержащего следующие компоненты: форму, выпадающий список, панель, картинку и метку.

Выпадающий комбинированный список (объект класса *TComboBox*) используется для того, чтобы можно было выбрать пункт из списка и при необходимости отредактировать его. В данной работе возможность редактирования не рассматривается

Основными свойствами комбинированного списка являются: *Name*, *Items* и *Style*. Ниже в таблице приведено их описание.

Items	Пункты, с помощью этого свойства задаются элементы списка
Name	Имя, должно начинаться с *cb*
Style	Задайте *csDropDownList* для списка, раскрывающегося только для чтения

Создайте в среде *Delphi* новое приложение и разместите на форме следующие компоненты: метку (*TLabel*), выпадающий список (*TComboBox*), панель (*TPanel*) и картинку (*TImage*), как показано на рис. 1–2.

Цель создания данного приложения – после выбора одного из пунктов выпадающего списка в картинке отображается тот или иной графический файл.

После размещения компонентов на форме надо прежде всего изменить их имена в соответствии с правилами культуры программирования. Присвойте имя *frmMain* форме, имя *lblInfo* метке, имя *cbChoose* выпадающему списку, *pnlBase* панели и *imgPhoto* картинке.

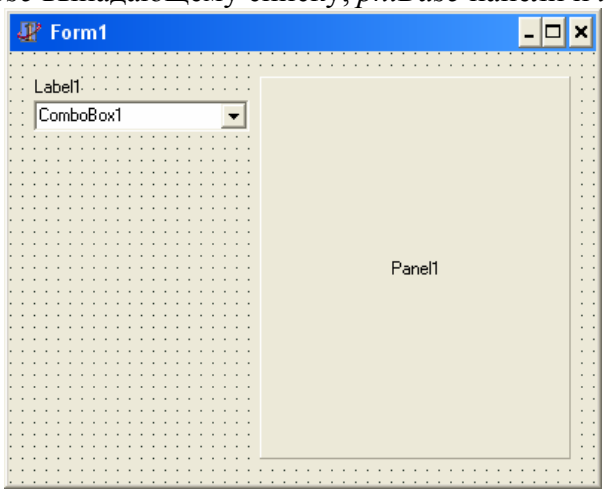


Рис. 1. Внешний вид приложения после размещения компонентов *ComboBox* и *Panel*

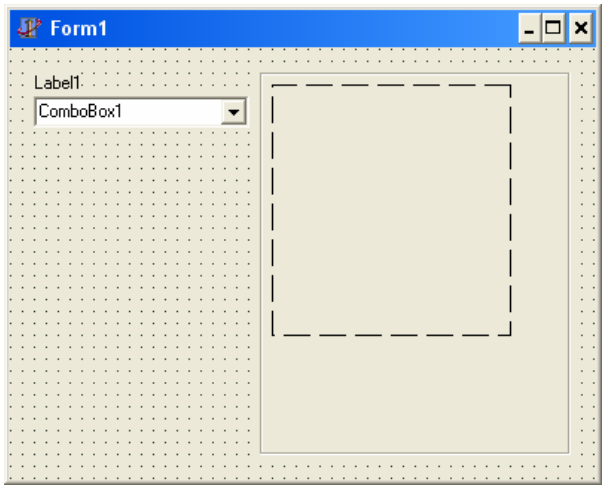


Рис. 2. Внешний вид приложения после изменения свойств *Panel* и размещения компонента *Image*

Для сохранения файлов проекта используйте папку ″D:\Users\Студен-ты\Delphi″.

Метка выполняет информационную функцию и нужна для того, чтобы сообщить пользователю, что именно нужно делать с выпадающим списком, поэтому в свойстве *Caption* этой метки напишите ″Выберите картинку:″.

Измените свойство *Style* компоненты *ComboBox*, чтобы оно стало равным *csDropDownList*.Чтобы указать, какие пункты будут в списке во время выполнения приложения,

выберите свойство *Items*, затем щелкните на маленькой кнопке с тремя точками. Появится простой текстовый редактор, в котором пропишите две строчки: ″Фото-1″ и ″Фото-2″.

Панель используется обычно как контейнер для других компонентов, чтобы логически сгруппировать их. В данном же случае она применена еще и для улучшения внешнего вида интерфейса. У панели измените значения свойств *Caption* (должна быть пустая строка), а также *BevelInner* и *BevelOuter* (внутренний и внешний края – соответственно *bvRaised* и *bvLowered*).

Проследите, чтобы размеры панели, измеряемые в точках (высота *Height* и ширина *Width*) были не меньше, чем соответственно 250 и 220.

После этого разместите на панели компонент Картинка (*TImage*). Так как пока мы не указали, какое изображение будет в этом компоненте, он имеет вид пустой рамки, очерченной пунктирной линией. Чтобы картинка занимала всю площадь панели, измените ее свойство *Align* (выравнивание), чтобы оно стало равным *alClient*.

Затем надо определить процедуру, выполняющуюся при выборе из выпадающего списка того или иного пункта. Для этого надо выделить *ComboBox* на форме, перейти на вкладку *Events* (События) в окне *Object Inspector* и дважды щелкнуть на пустом поле справа от события *OnChange* (По изменению). После этого автоматически откроется окно кода:

```
procedure TfrmMain.cbChooseChange(Sender: TObject);
begin

end;
```

Готовая процедура будет выглядеть так:

```
procedure TfrmMain.cbChooseChange(Sender: TObject);
begin
{выбор файла с нужной картинкой}
 case cbChoose.ItemIndex of
 0: imgPhoto.Picture.LoadFromFile('photo1.jpg');
 1: imgPhoto.Picture.LoadFromFile('photo2.jpg');
 end;
end;
```

Чтобы приложение корректно работало, надо в самой верхней части окна кода модуля в разделе **uses** добавить упоминание модуля *Jpeg* (работа с графическими файлами этого формата):

```
unit Unit1;
interface
uses
 Windows, Messages, SysUtils, Classes, Graphics,
 Controls, Forms, Dialogs, ExtCtrls, StdCtrls,
 Jpeg; <- добавить
```

В приведенном выше коде использован метод *LoadFromFile* (загрузить из файла) компонента класса *TImage*.

Содержание отчета

1. Название и цель работы.
2. Последовательность создания нового приложения, описание использованных компонентов и их основных свойств.
3. Программный код процедуры, обрабатывающей выбор пункта из выпадающего списка с комментариями.

Контрольные вопросы

- Каковы основные свойства объектов *TComboBox, TPanel* и *TImage*? Для чего и как они используются?
- Как определяются процедуры обработки события *OnChange* для компонента *ComboBox*?

Лабораторная работа 4 (4 часа)

РАЗРАБОТКА ПРИЛОЖЕНИЯ В СРЕДЕ *DELPHI* С ИСПОЛЬЗОВАНИЕМ КОМПОНЕНТОВ *STRINGGRID* И *CHECKBOX*

Цель работы: В среде *Delphi* создать простое приложение, содержащее следующие компоненты: форму, сетку строк, панель, кнопку и флажок опции.

Порядок выполнения работы

1. Запустить среду *Delphi*, создать в ней новый проект и сохранить его в определенной папке на жестком диске компьютера.
2. Разместить на новой форме указанные компоненты и изменить их свойства, перечисленные ниже.
3. Определить процедуру нажатия кнопки для того, чтобы приложение выполняло необходимые действия.
4. Отладить приложение и сохранить проект.
5. Выполнить задание для самостоятельной работы (см. ниже).

Методические указания

Рассмотрим далее процесс создания простого приложения в среде *Delphi*, содержащего следующие компоненты: форму, сетку строк, панель, кнопку и флажок опции.

Сетка строк (объект класса *TStringGrid*) используется для того, чтобы визуально упорядочить представляемые данные в виде таблицы с целью удобства работы с ними.

Основными свойствами сетки строк являются: *Name*, *Cells*, *ColCount*, *RowCount*, *DefaultColWidth, DefaultRowHeight, FixedCols, FixedRows*. Ниже в таблице приведено их описание.

Cells[i,j]	Обеспечивает доступ к данным для каждой ячейки. При этом i – номер столбца, j – номер строки. В каждой ячейке хранится строковое значение.
Name	Имя, должно начинаться с *sg*
ColCount	Количество столбцов (Columns) в сетке
RowCount	Количество строк в (Rows) в сетке
DefaultColWidth	Ширина столбца по умолчанию (в пикселах)
DefaultRowHwight	Высота строки по умолчанию (в пикселах)
FixedCols	Количество фиксированных столбцов
FixedRows	Количество фиксированных строк

Флажок опции (объект класса *TCheckBox*) используется для того, чтобы можно было производить какие-либо действия в зависимости от того, отмечен этот компонент или нет (выбор из двух вариантов). Основными его свойствами являются: *Checked* (Отмечено) и *Caption* (Заголовок). Свойство *Checked* может принимать только два значения: *true* (истина) или *false* (ложь).

Создайте в среде *Delphi* новое приложение и разместите на форме следующие компоненты: кнопку (*TButton*), панель (*TPanel*), на панели – сетку строк (*TStringGrid*) и флажок опции (*TCheckBox*).

Панель в данном случае используется как контейнер для компонента *StringGrid* с целью улучшения внешнего вида интерфейса. У панели измените значения свойств *Caption* (должна быть пустая строка), а также *BevelInner* и *BevelOuter* (внутренний и внешний края – соответственно *bvRaised* и *bvLowered*).

Задайте размеры панели, измеряемые в точках (высота *Height* и ширина *Width*): соответственно 214 и 350.

После этого разместите на панели компонент ″сетка строк″ (*TStringGrid*). Затем измените ее свойство *Align* (выравнивание), чтобы оно стало равным *alClient* (чтобы сетка строк занимала всю площадь панели). Затем у сетки строк измените следующие свойства: *ColCount* (5), *RowCount* (21), *FixedCols* (0), *FixedRows* (1), *DefaultRowHeight* (20), *Options → goThumbTrack-*

ing (*true*). Последнее необходимо для того, чтобы содержимое таблицы смещалось одновременно при перемещении движка вертикальной полосы прокрутки.

Свойству *Caption* кнопки присвойте значение ″Расчет″.

Свойству *Caption* флажка опции присвойте значение ″С заголовком″.

Присвойте имя *frmMain* форме, имя *sgData* сетке строк, имя *btnCalc* кнопке, *pnlBase* панели и *cbxTitle* флажку опции.

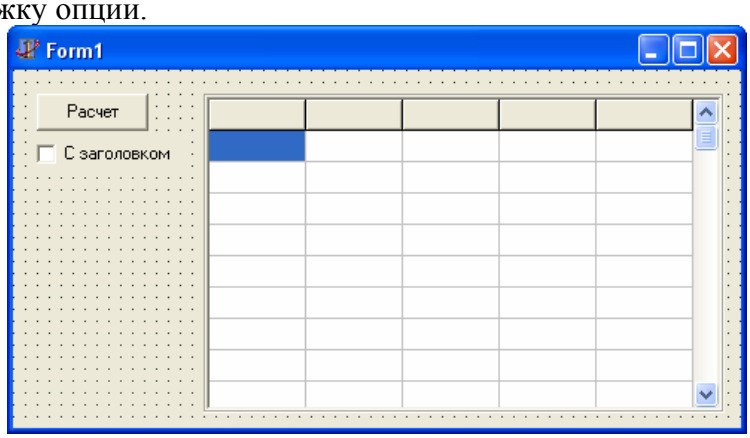


Рис. 1. Внешний вид приложения после размещения компонентов и изменения их свойств

После этих действий форма приложения будет выглядеть, как показано на рис. 1.

Для сохранения файлов проекта используйте папку ″D:\Users\Студен-ты\Delphi″.

Цель создания данного приложения – после нажатия кнопки происходит расчет по некоторому алгоритму и в сетке строк отображаются данные этого расчета. Если компонент *CheckBox* отмечен, в первой строке сетки прописываются обозначения столбцов.

Далее надо определить процедуру, выполняющуюся при нажатии кнопки *btnCalc*. Готовая процедура будет выглядеть так:

```
procedure TForm1.btnCalcClick(Sender: TObject);
var
 i: byte;
begin
 // заголовки столбцов
 if cbxTitle.Checked then
 begin
 sgData.Cells[0,0]:='Число';
 sgData.Cells[1,0]:='Квадрат';
 sgData.Cells[2,0]:='Корень';
 end
 else
 begin
 sgData.Cells[0,0]:='';
 sgData.Cells[1,0]:='';
 sgData.Cells[2,0]:='';
 end;
 // расчет
 for i:=1 to 20 do
 begin
 sgData.Cells[0,i]:=IntToStr(i);
 sgData.Cells[1,i]:=IntToStr(Sqr(i));
 sgData.Cells[2,i]:=FloatToStrF(Sqrt(i),ffGeneral,5,7);
 end;
end;
```

В данном случае алгоритм расчета заключается в следующем: в первом столбце выводятся целые числа от 1 до 20, во втором столбце – квадраты этих чисел, а в третьем – квадратные корни из них.

В приведенном выше коде использованы встроенные функции:

IntToStr – преобразует целое число в строку;

Sqr – возводит число в квадрат;

Sqrt – вычисляет квадратный корень из числа;

FloatToStrF – преобразует десятичное число в строку в соответствии с указанным форматом.

► Задание для самостоятельной работы:

Добавьте к интерфейсу приложения еще одну кнопку, нажатие которой должно вызывать очистку сетки строк.

Содержание отчета

1. Название и цель работы.
2. Последовательность создания нового приложения, описание использованных компонентов и их основных свойств.
3. Программный код процедуры, обрабатывающей нажатие кнопки (с комментариями).

Контрольные вопросы

- Каковы основные свойства компонентов *TStringGrid, TPanel* и *TCheckBox*? Для чего и как используются эти компоненты?

Лабораторная работа 5 (4 часа)

РАЗРАБОТКА ПРИЛОЖЕНИЯ В СРЕДЕ *DELPHI* С ИСПОЛЬЗОВАНИЕМ КОМПОНЕНТОВ *OPENDIALOG* И *STRINGGRID*

Цель работы: В среде *Delphi* создать простое приложение, содержащее следующие компоненты: форму, сетку строк, панель, кнопку и диалог открытия файла.

Порядок выполнения работы

1. Запустить среду *Delphi*, создать в ней новый проект и сохранить его в определенной папке на жестком диске компьютера.
2. Разместить на новой форме указанные компоненты и изменить их свойства, перечисленные ниже.
3. Определить процедуру нажатия кнопки для того, чтобы приложение выполняло необходимые действия.
4. Отладить приложение и сохранить проект.
5. Проверить функциональность приложения, запустив созданный .*exe*-файл.

Методические указания

Рассмотрим далее процесс создания простого приложения в среде Delphi, содержащего следующие компоненты: форму, сетку строк, панель, кнопку и диалог открытия файла.

Диалог открытия файла (объект класса *TOpenDialog*) – это невизуальный компонент (он не виден в *Runtime*) и предназначен для работы со стандартным окном диалога открытия файла в *Windows*.

Основными его свойствами являются: *FileName* (Имя файла), *Filter* (Фильтр) и *FilterIndex* (Индекс фильтра). Свойство *Filter* используется для того, чтобы определить, файлы каких типов (в соответствии с их расширениями) будут показываться во время работы с диалогом. Для задания фильтра в *DesignTime* используется специальный редактор (рис. 1).

Создайте в среде *Delphi* новое приложение и разместите на форме следующие компоненты: кнопку (*TButton*), диалог открытия файла (*TOpenDialog*), панель (*TPanel*) и на панели – сетку строк (*TStringGrid*).

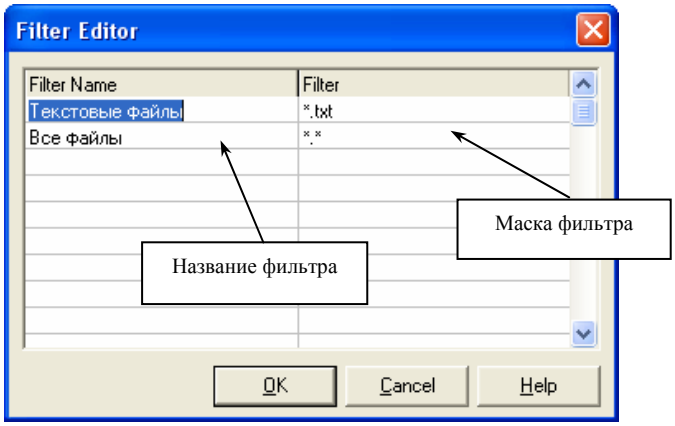


Рис. 1. Редактор фильтра компонента *OpenDialog*

Панель в данном случае используется как контейнер для компонента *StringGrid* с целью улучшения внешнего вида интерфейса. У панели измените значения свойств *Caption* (должна быть пустая строка), а также *BevelInner* и *BevelOuter* (внутренний и внешний края – соответственно *bvRaised* и *bvLowered*).

После этого разместите на панели компонент Сетка строк (*TStringGrid*). Затем измените ее свойство *Align* (выравнивание), чтобы оно стало равным *alClient* (чтобы сетка строк занимала всю площадь панели). Затем у сетки строк измените следующие свойства: *ColCount* (4), *RowCount* (10), *FixedCols* (1), *FixedRows* (0), *DefaultRowHeight* (20).

Свойству *Caption* кнопки присвойте значение ″Загрузить данные″ и измените ее размер так, чтобы отображалась вся фраза.

Присвойте имя *frmMain* форме, имя *sgData* сетке строк, имя *btnLoad* кнопке, *pnlBase* панели и *odLoad* диалогу открытия файла.

После этих действий форма приложения будет выглядеть, как показано на рис. 2.

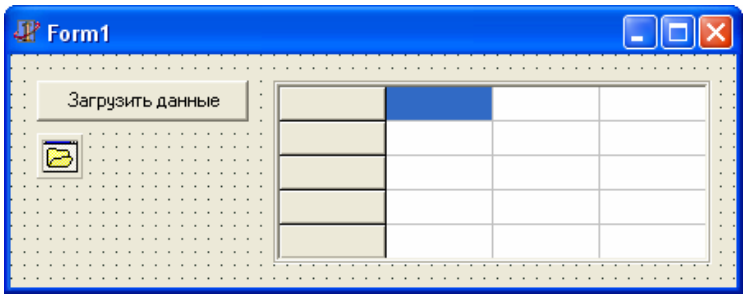


Рис. 2. Внешний вид приложения после размещения компонентов и изменения их свойств

Для сохранения файлов проекта используйте папку ″D:\Users\Студен-ты\Delphi″.

Цель создания данного приложения – после нажатия кнопки запускается стандартный диалог *Windows* для открытия файла и в сетку строк загружаются данные из выбранного с помощью диалога текстового файла с именем *″data.txt″*.

Файл *″data.txt″* необходимо создать самостоятельно перед запуском приложения. Для этого можно использовать, например, программу Блокнот из стандартного набора *Windows* или встроенный редактор оболочки *FAR*. В последнем случае надо запустить *FAR*, нажать комбинацию клавиш *Shift+F*4, ввести имя создаваемого файла (*″data.txt″*), нажать *Enter*, после чего откроется окно редактора. Запишите в упомянутый файл 12 чисел (каждое в отдельной строке) и нажмите *F*2, чтобы сохранить файл на диске.

При загрузке данных из файла должна использоваться следующая схема: числа считываются блоками по три, при этом первое число из блока записывается во второй столбец, второе – в третий, третье – в четвертый столбец одной строки начиная с первой; следующие три числа записываются во вторую строку и т.д. (рис. 3). В первый столбец выводятся строки ″Ряд 1″, ″Ряд 2″, ″Ряд 3″ и т.д.

В данном случае нам известно, сколько всего чисел записано в файл и сколько строк потребуется для их вывода, однако такая ситуация имеет место далеко не всегда, поэтому может потребоваться составлять код процедуры для загрузки данных с учетом того, что количество чисел в файле заранее не известно.

Далее надо определить процедуру, выполняющуюся при нажатии кнопки *btnLoadData*. Ниже приведены два варианта готовых процедур (вначале процедура для заранее известного количества чисел, затем – для произвольного).

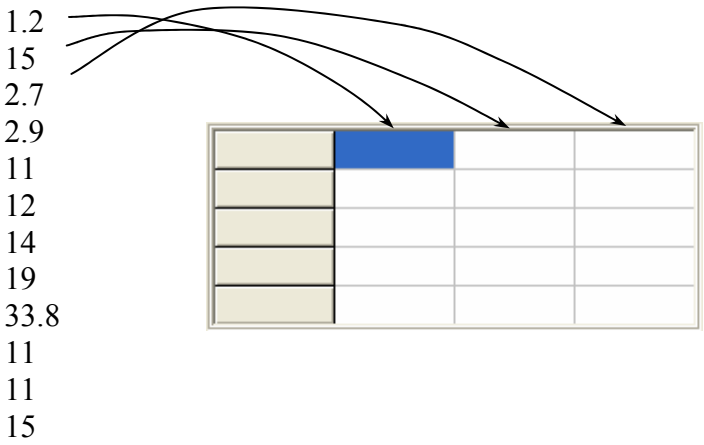


Рис. 3. Схема записи чисел из файла в сетку строк

Вариант *1: Процедура с заранее известным количеством*

```
procedure TfrmMain.btnLoadClick(Sender: TObject);
var
 fdata: textfile;
 st: string;
 i,j: byte;
begin
 if odLoad.execute then
 begin
 assignfile(fdata,odLoad.filename);
 reset(fdata);
 for i:=1 to 4 do
 begin
 sgData.Cells[0,i-1]:='Ряд '+IntToStr(i);
 for j:=1 to 3 do
 begin
 readln(fdata,st);
 sgData.Cells[j,i-1]:=st;
 end;
 end;
 closefile(fdata);
 end;
end;
```

Вариант *2: Процедура с неизвестным заранее количеством*

```
procedure TfrmMain.btnLoadDataClick(Sender: TObject);
var
 fdata: textfile;
 st: string;
 j: byte;
 count: word; // счетчик
begin
 if odLoad.execute then
 begin
 assignfile(fdata,odLoad.filename);
 reset(fdata);
 count:=0;
 while not eof(fdata) do
```

```
 begin
 count:=count+1;
 sgData.Cells[0,count-1]:='Ðÿä '+IntToStr(count);
 for j:=1 to 3 do
 begin
 readln(fdata,st);
 sgData.Cells[j,count-1]:=st;
 end;
 end;
 closefile(fdata);
 end;
end;
```

Содержание отчета

1. Название и цель работы.
2. Последовательность создания нового приложения, описание использованных компонентов и их основных свойств.
3. Программный код процедуры, обрабатывающей нажатие кнопки (с комментариями).

Контрольные вопросы

- Каковы основные свойства объектов *TOpenDialog* и *TStringGrid*? Для чего и как они используются?
- Продемонстрируйте приемы практической работы с этими компонентами.

Лабораторная работа 6 (6 часов)

РАЗРАБОТКА ПРИЛОЖЕНИЯ В СРЕДЕ *DELPHI* ДЛЯ ОТОБРАЖЕНИЯ ГРАФИЧЕСКОЙ ИНФОРМАЦИИ С ИСПОЛЬЗОВАНИЕМ КОМПОНЕНТА *IMAGE*

Цель работы: В среде *Delphi* создать приложение для построения графика функции, содержащее компоненты: форму, картинку, две кнопки и панель.

Порядок выполнения работы

1. Запустить среду *Delphi*, создать в ней новый проект и сохранить его в определенной папке на жестком диске компьютера.
2. Разместить на новой форме указанные компоненты и изменить их свойства, перечисленные ниже.
3. Определить процедуру нажатия кнопки для того, чтобы приложение выполняло необходимые действия.
4. Отладить приложение и сохранить проект.
5. Проверить функциональность приложения, запустив созданный *.exe*-файл.

Методические указания

Картинка (объект класса *TImage*) может использоваться не только для демонстрации изображений из графических файлов (см. лабораторную работу № 3).

Помимо этого, используя имеющееся у него свойство *Canvas* (Холст), можно создавать собственные изображения, состоящие, например, из текста и простых геометрических фигур, в частности, графики.

В соответствии с принципами ООП свойство *Canvas* также имеет свои свойства и методы. Ниже в таблице приведено описание основных свойств и методов *Canvas*:

Свойства Canvas	
pen	цвет переднего плана
brush	цвет фона
font	шрифт, для задания цвета шрифта используйте его свойство font.color
pixels[x,y]	определяет цвет отдельных пикселов (точек)
Методы Canvas	
rectangle	рисует на холсте прямоугольник
ellipse	рисует на холсте эллипс
MoveTo	перемещает перо к точке с указанными координатами
LineTo	проводит прямую линию из текущей точки до точки с указанными координатами
TextOut	выводит текст в точку с указанными координатами

Создайте в среде *Delphi* новое приложение и разместите на форме следующие компоненты: две кнопки (*TButton*), панель (*TPanel*) и на панели – картинку (*TImage*).

У панели измените значения свойств *Caption* (должна быть пустая строка), а также *BevelInner* и *BevelOuter* (внутренний и внешний края – соответственно *bvRaised* и *bvLowered*).

Задайте размеры панели, измеряемые в точках (высота *Height* и ширина *Width*): примерно 250 и 300.

После этого разместите на панели компонент Картинка (*TImage*). Затем измените ее свойство *Align* (выравнивание), чтобы оно стало равным *alClient* (чтобы картинка занимала всю площадь панели).

Свойству *Caption* первой кнопки присвойте значение ″Сетка″, а второй – ″График″. Также измените это же свойство, но у формы на ″Построение графика функции″

Присвойте имя *frmMain* форме, имя *imgGraph* картинке, имя *btnGrid* первой кнопке и *btnGraph* – второй, а также *pnlBase* – панели.

После этих действий форма приложения будет выглядеть, как показано на рис. 1.

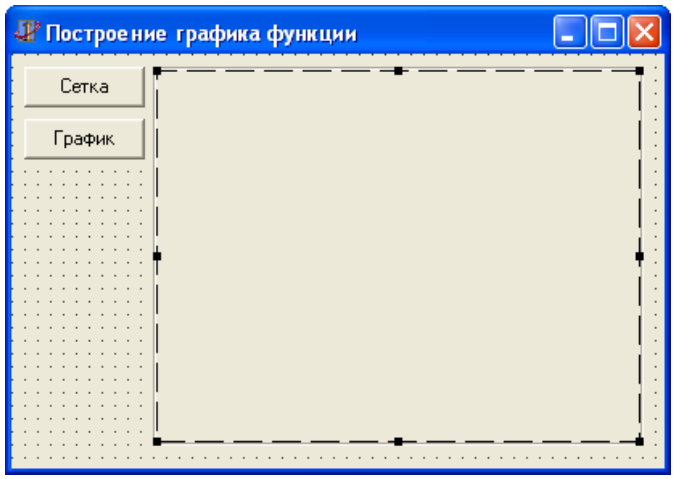


Рис. 1. Внешний вид приложения после размещения компонентов и изменения их свойств

Для сохранения файлов проекта используйте папку ″D:\Users\Студен-ты\Delphi″.

Цель создания данного приложения – после нажатия первой кнопки на картинке рисуется координатная сетка; после нажатия второй кнопки поверх сетки выводится график функции $y = f(x)$.

Процедуру вывода графика можно разбить на несколько стадий:

- отображение координатной сетки (с помощью методов *Rectangle* (внешняя рамка) и *MoveTo*, *LineTo* (в цикле, горизонтальные и вертикальные линии));
- нанесение обозначений координатных осей и делений (метод *OutText*);
- собственно изображение графика функции (изменяя свойство *pixels*[*x*, *y*] или с помощью методов *Rectangle* или *Ellipse*).

Для того, чтобы изобразить график некоторой функции на холсте компонента *Image*, будем исходить из следующих соображений. В реальных условиях задача обычно ставится следующим образом: построить график некоторой функции $y = f(x)$, при этом x принадлежит некоторому отрезку $[x_{\min}, x_{\max}]$, соответственно, y будет лежать в пределах $[y_{\min}, y_{\max}]$. Начало координат находится при этом в средней части изображения. Горизонтальный и вертикальный размер изображения в реальных координатах при этом соответственно равны $\Delta x^{(p)} = x_{\max}^{(p)} - x_{\min}^{(p)}$, $\Delta y^{(p)} = y_{\max}^{(p)} - y_{\min}^{(p)}$.

Размер холста картинки измеряется в пикселах (точках), количества которых являются целыми числами. Кроме того, как это принято в программировании, начало координат (0, 0) находится в левом верхнем углу. Координата $X^{(э)}$ увеличивается слева направо, а координата $Y^{(э)}$ – сверху вниз.

Если мы хотим изобразить на холсте график, мы должны решить две задачи:

1. растянуть или сжать реальный график до размеров картинки;
2. сместить начало координат из левого верхнего угла картинки в центральную область.

Сам график, очевидно, следует строить по отдельным точкам.

Таким образом, построение графика также можно разбить на несколько стадий:

- расчет коэффициентов масштабирования *kx* и *ky*;
- расчет смещения центра координат;
- расчет значений текущих реальных координат для каждой выводимой точки графика;
- переход от реальных (в общем случае дробных) координат к экранным (целочисленным) и вывод точек с этими координатами на холст.

Формулы для расчета приведены ниже в приложении.

Далее надо определить процедуры, выполняющиеся при нажатии кнопок *btnGrid* и *btnGraph*. Готовые процедуры будет выглядеть так:

```
procedure TfrmMain.btnGridClick(Sender: TObject);
var
 i: byte;
 Dx,Dy: single;
begin
 a:=25; b:=25;
 Nx:=10; Ny:=10;
 with imgGraph.Canvas do
 begin
 // черный фон для графика
 brush.color:=clBlack;
 rectangle(0,0,imgGraph.width,imgGraph.height);
 // внешняя рамка координатной сетки
 pen.color:=clGray;
 rectangle(a,b,imgGraph.width-a,imgGraph.height-b);
 // вертикальные линии
 Dx:=(imgGraph.width-2*a)/Nx;
 for i:=1 to Nx-1 do
 begin
 MoveTo(round(a+i*Dx),b);
 LineTo(round(a+i*Dx),imgGraph.height-b);
 end;
 // горизонтальные линии
 Dy:=(imgGraph.height-2*b)/Ny;
 for i:=1 to Ny-1 do
 begin
 MoveTo(a,round(b+i*Dy));
 LineTo(imgGraph.width-a,round(b+i*Dy));
 end;
 end;
end;

procedure TfrmMain.btnGraphClick(Sender: TObject);
var
 i: word;
 Offset_x, Offset_y: word;
 Npix: word;
 xmin,xmax,ymin,ymax,dx: single;
```

```
 xreal,yreal: single;
 Xscreen,Yscreen: word;
begin
 Npix:=200; // количество точек
 xmin:=-3.14; xmax:=3.14;
 ymin:=-1; ymax:=1;
 // коэффициенты масштабирования по осям
 kx:=(imgGraph.width-2*a)/(xmax-xmin);
 ky:=(imgGraph.height-2*b)/(ymax-ymin);
 Offset_x:=round(imgGraph.width/2);
 Offset_y:=round(imgGraph.height/2);
 dx:=(xmax-xmin)/Npix;
 for i:=1 to Npix do
 begin
 xreal:=xmin+dx*(i-1);
 yreal:=sin(xreal);
 Xscreen:=round(kx*xreal+Offset_x);
 Yscreen:=round(-ky*yreal+Offset_y);
 imgGraph.canvas.pixels[Xscreen,Yscreen]:=clLime;
 end;
end;
```

Обратите внимание: в приведенных выше процедурах использованы некоторые переменные, которые должны быть объявлены не в теле процедуры, а в разделе ***public*** в коде формы:

```
private
 { Private declarations }
public
 { Public declarations }
 a,b: byte; // отступы
 Nx,Ny: byte; // количество разбиений
 kx,ky: single; // коэффициенты масштабирования
end;
```

Это сделано для того, чтобы они (и их значения) действовали для всех процедур формы (т.е. это – глобальные переменные для этой формы). Также в приведенном выше коде использована функция ***round***, применяющаяся для округления до целого числа вычисленной координаты.

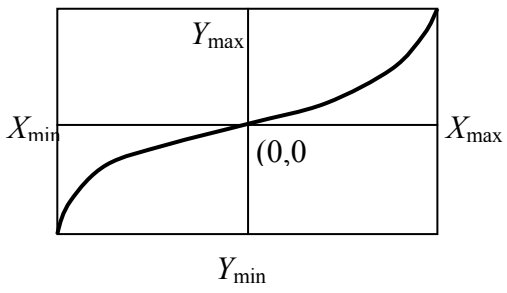
Содержание отчета

1. Название и цель работы.
2. Последовательность создания нового приложения, описание использованных компонентов и их основных свойств.
3. Программный код процедуры, обрабатывающей нажатие кнопок (с комментариями).

Контрольные вопросы

- Каковы основные свойства и методы *Canvas*? Для чего и как они используются?
- На какие стадии можно разбить построение графика функции с помощью компонента *Image*?

ПРИЛОЖЕНИЕ



$$kx = \frac{\Delta X^{(э)}}{\Delta X^{(p)}},\ ky = \frac{\Delta Y^{(э)}}{\Delta Y^{(p)}}$$

$$\Delta X^{(э)} = img.width - 2a$$

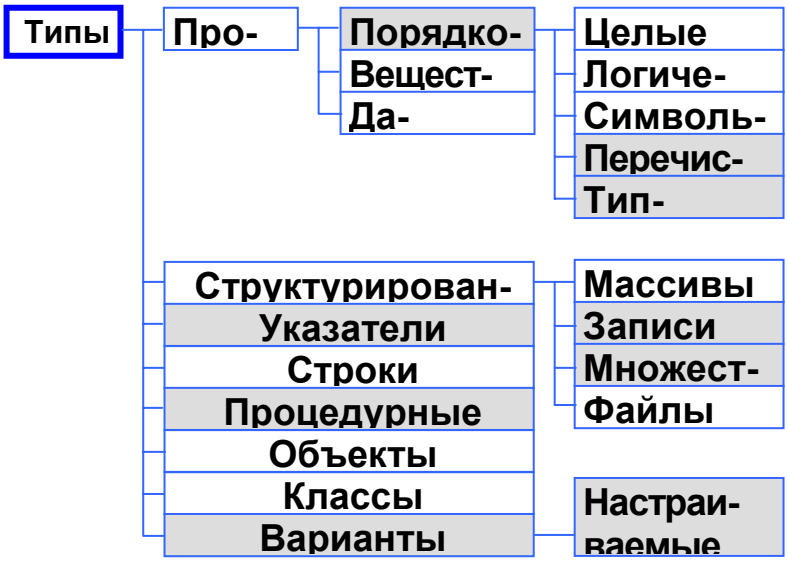
$$\Delta Y^{(э)} = img.height - 2b$$

$$X^{(э)} = kx \cdot X^{(p)} + Offset\_x$$

$$Y^{(э)} = -ky \cdot Y^{(p)} + Offset\_y$$

СПИСОК ЛИТЕРАТУРЫ

1. Сван, Том. Delphi 4. Библия разработчика / Т. Сван ; пер. с англ. – СПб. : Диалектика, 1998. – 672 с., ил.
2. Озеров, В. Delphi. Советы программистов / В. Озеров. – СПб. : Символ-Плюс, 2003. – 976 с., ил.
3. Осипов, Д. Delphi. Профессиональное программирование / Д. Осипов. – СПб. : Символ-Плюс, 2006. – 1056 с., ил.



Тип	Диапазон возможных значений	Размер памяти для хранения данных
Integer	-2147483648…2147483647	4 байта (32 бита)
Cardinal	0…4294967295	4 байта (32 бита)
Shortint	-128…127	1 байт (8 бит)
Smallint	-32768…32767	2 байта (16 бит)
Longint	-2147483648…2147483647	4 байта (32 бита)
Int64	$-2^{63}…2^{63}-1$	8 байт (64 бита)
Byte	0…255	1 байт (8 бит)
Word	0…65535	2 байта (16 бит)
Longword	0…4294967295	4 байта (32 бита)

Тип	Диапазон возможных значений	Максимальное количество цифр в числе	Размер в байтах
Real48	$2,9 \cdot 10^{-39} \ldots 1,7 \cdot 10^{38}$	11-12	6
Real	$5,0 \cdot 10^{-324} \ldots 1,7 \cdot 10^{308}$	15-16	8
Single	$1,5 \cdot 10^{-45} \ldots 1,7 \cdot 10^{38}$	7-8	4
Double	$5,0 \cdot 10^{-324} \ldots 1,7 \cdot 10^{308}$	15-16	8
Extended	$3,6 \cdot 10^{-4951} \ldots 1,1 \cdot 10^{4932}$	19-20	10
Comp	$-2^{63}+1 \ldots 2^{63}$	19-20	8
Currency	-922337203685477,5808… 922337203685477,5807	19-20	8

Тип	Максимальная длина строки	Память, отводимая для хранения **строки**	Примечание
ShortString	255 символов	От 2 до 256 байт	
ANSIString	2^{31}	От 4 байт до 2 Гбайт	8-битовые
WideString	2^{30}	От 4 байт до 2 Гбайт	Unicode