Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования

Владимирский государственный университет

Кафедра приборостроения и
информационно-измерительных технологий

ПРОГРАММИРОВАНИЕ В СРЕДАХ TURBO PASCAL и DELPHI

Методические указания
к лабораторным работам
по дисциплине «Алгоритмические языки»

Составитель
Ю.С. КЛИМЕНКОВ

Владимир 2010

УДК 004.438
ББК 32.97
П78

Рецензент
Доктор технических наук, профессор
кафедры радиотехники и радиосистем
Владимирского государственного университета
А.Г. Самойлов

Печатается по решению редакционного совета
Владимирского государственного университета

П78 Программирование в средах Turbo Pascal и Delphi : метод. указания к лаб. работам по дисциплине «Алгоритмические языки» / Владим. гос. ун-т; сост.: Ю. С. Клименков. – Владимир : Изд-во Владим. гос. ун-та, 2010. – 82 с.

Приведены методики выполнения лабораторных работ по программированию в среде Turbo Pascal и Delphi по дисциплине «Алгоритмические языки».

Предназначено для студентов 3-го курса дневного отделения, обучающихся по специальностям 200101 – приборостроение и 200106 – информационная техника и технологии.

Табл. 2. Ил. 22. Библиогр.: 5 назв.

УДК 004.438
ББК 32.97

ВВЕДЕНИЕ

Данный курс лабораторных работ позволит студентам получить навыки программирования и научиться решать инженерные задачи с применением ЭВМ.

Настоящее пособие состоит из двух частей. В первой части предусмотрено шесть лабораторных работ по три задания в каждой, где студенту предлагается самостоятельно с использованием изложенного теоретического материала разработать программы для решения широкого круга типовых задач на языке высокого уровня Turbo Pascal, а также составить диаграммы алгоритмов их решения. В основу заданий первой части пособия положены принципы организации циклических структур, условных переходов, операторов ввода-вывода информации, процедур, функций, массивов.

Вторая часть методического пособия также состоит из шести лабораторных работ, в которых изложен объектно-ориентированный подход к программированию. Лабораторная работа № 1 предполагает ознакомление со средой визуального программирования Delphi, № 2 – разработку первой программы (решение квадратного уравнения). В лабораторной работе № 3 исследуются графические возможности среды Delphi. В работе № 4 изучается процесс создания стандартных диалоговых окон на примере разработки простейшего текстового редактора. Лабораторная работа № 5 посвящается работе с файлами. В работе № 6 изложены принципы построения баз данных с их последующим просмотром и выбором искомой информации.

Методические указания предназначены для студентов специальностей 200101 – приборостроение и 200106 – информационно-измерительная техника и технологии очного обучения.

Часть I. ОЗНАКОМЛЕНИЕ С ЯЗЫКОМ ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ TURBO PASCAL 7.0

Лабораторная работа № 1
РЕШЕНИЕ ПРОСТЕЙШИХ ЗАДАЧ

Цель работы: разработка программ на языке Turbo Pascal для решения поставленной задачи с прохождением всех основных этапов разработки программы.

Оборудование: дисплейный класс, интегрированная среда Turbo Pascal версии 7.0 или выше.

1.1. Общие сведения

Все задачи, решаемые с помощью ЭВМ, имеют ряд общих этапов. При решении одних задач некоторые этапы могут быть вырожденными. В других задачах какие-то этапы могут оказаться крайне трудоемкими, а иногда и неразрешимыми. Какие-то этапы могут разбиваться на более мелкие части и т. д.

Обычно при решении отдельных задач или комплекса взаимосвязанных задач на ЭВМ выделяются следующие этапы:

1) разработка математической модели решаемой задачи;

2) разработка методики решения и определение ограничений на решаемую задачу;

3) разработка алгоритма и запись его на некотором языке;

4) программирование решения задачи на одном из языков программирования;

5) тестирование и отладка программы или комплекса программ;

6) решение задачи на ЭВМ.

Разработка математической модели решаемой задачи – это первый и самый существенный этап. Если математической модели нет, то решать задачу на ЭВМ бессмысленно. Математическая модель может быть крайне простой или очень сложной. Когда математическая модель построена, можно переходить к разработке алгоритма, но часто случается так, что существует прекрасная математическая модель и изящный метод решения задачи, но они страдают одним недостатком – не могут быть реализованы на ЭВМ. Пригодность или

непригодность избранных методов в некоторых случаях может быть обнаружена лишь на последующих этапах, что заставляет возвратиться к самому началу решения задачи.

Разработка алгоритма решения зависит от методики решения задачи. Иногда алгоритмы известны, иногда их приходится разрабатывать самому. Под алгоритмом понимается набор предписаний, определяющих процесс преобразования исходных данных в искомый результат и обладающий свойствами определенности, результативности и массовости. Алгоритм может быть записан на естественном языке в виде пронумерованной последовательности действий или же в некоторой системе обозначений, например в виде блок-схем, специальных диаграмм или специальной нотации. При этом важно учитывать особенности ЭВМ, например диапазон представления чисел, их точность, необходимость обмена с внешними устройствами ввода/вывода и др.

Программирование решения задачи заключается в записи разработанного алгоритма на одном из языков программирования.

Программы, написанные на языке Паскаль, перед выполнением на ЭВМ должны транслироваться в эквивалентные программы, написанные на машинном коде. Для этих целей используются компиляторы. Компилирование программы включает в себя два действия: анализ (определение правильности записи программы в соответствии с правилами языка Паскаль) и синтез (генерирование эквивалентной программы в машинном коде). В процессе анализа компилятор находит синтаксические ошибки в программе и выдает соответствующие диагностические сообщения. Если в исходной программе на языке Паскаль синтаксических ошибок нет, то компилятор вырабатывает эквивалентный исходной программе код на машинном языке, который можно выполнять. Отсутствие синтаксических ошибок не означает отсутствия ошибок в программе. Как правило, ошибки в программе всегда имеются, и цель этапа тестирования и отладки – выявить их и исправить.

Тестирование программы заключается в составлении набора тестов как правильных, так и неправильных, с единственной целью – найти как можно больше ошибок в программе. Тест, приводящий к отказу программы, гораздо более полезен, чем тест, демонстрирующий работоспособность программы. Если в программе найдены ошибки, то необходимо локализовать места их нахождения и испра-

вить их. Этот процесс называется отладкой. Необходимо помнить, что при исправлении одних ошибок в программу могут вноситься другие. Поэтому исправленную программу снова подвергают тестированию и т. д. В результате большинство ошибок удалено из программы (но наверняка не все) и начинается этап решения задачи на ЭВМ. При этом готовятся исходные данные для программы, программа запускается, производит необходимые действия и выдает результаты. Результаты работы программы и представляют собой решение задачи на ЭВМ.

Общая структура программы на языке Паскаль имеет следующий вид:

Program < имя программы >;
Label < раздел меток >;
Const < раздел констант >;
Type < раздел типов >;
Var < раздел переменных >;
Procedure, Function < раздел процедур, функций >;
Begin
< раздел операторов >;
End.

Последовательность операций, которые должны выполняться на ЭВМ, заключается в операторные скобки begin…end и носит название «тело программы». Остальная её часть носит описательный характер. Последовательность операторов и описаний разделяется знаком «;», заканчивается программа словом «end» с обязательной точкой после него. В любом месте программы могут включаться комментарии, которые заключаются в фигурные скобки {*} и не меняют её смысл.

Все величины, которыми оперирует программа, должны быть объявлены в разделе описаний с указанием их типа. Величины делятся на постоянные и переменные. Переменной называется величина, значение которой меняется в процессе выполнения алгоритма, постоянной – значение которой не изменяется.

Пример описания переменной величины:

Var A: Real,

где Var – служебное слово, означающее, что данная величина является переменной; A – имя переменной; Real – тип переменной.

Типы переменной и диапазоны допустимых значений, принимаемых величинами этого типа, кратко отражены в табл. 1.1.

Таблица 1.1

Тип	Диапазон допустимых значений
Integer (целый тип)	-32768…32767
Real (вещественный тип)	$2{,}9 \cdot 10^{-39} ... 1{,}7 \cdot 10^{38}$
Boolean (логический тип)	True, false
Char (символьный тип)	Символы кода ASCII

Оператор присваивания

Оператор присваивания – один из самых простых и наиболее часто используемых операторов в любом языке программирования. Оператор присваивания имеет вид:

Имя переменной := выражение (A:=5+7).

Оператор присваивания вычисляет значение выражения, записанное справа, и записывает это значение в переменную, обозначенную именем.

Операторы ввода и вывода данных

Оператор, который в режиме диалога с клавиатуры присваивает значение для переменной величины, называется оператором ввода.

В языке Паскаль этот оператор выглядит следующим образом:

Read (A, B, C).

Как только в программе встречается вызов процедуры Read, ЭВМ приостанавливает выполнение этой программы и ждет, пока пользователь введет с клавиатуры соответствующие значения, которые по очереди будут присваиваться переменным, перечисленным в списке ввода. Значения вводимых данных одновременно отображаются на экране дисплея. После нажатия клавиши **Enter**, когда все переменные примут свои значения из входного набора данных, определенного пользователем, выполнение программы продолжается с оператора, следующего за Read.

В списке ввода значения разделяются между собой пробелом. Присваивание значений из входного потока выполняется слева направо в соответствии с порядком следования переменных в операторе

Read. Оператор Readln похож на Read. Разница лишь в том, что Readln реагирует на конец строки, и в случае его обнаружения происходит сразу переход к следующей строке.

Оператор, который выводит содержимое переменных на экран, называется оператором вывода.

В языке Паскаль этот оператор выглядит следующим образом:

Write (A, B, C).

В списке вывода этих операторов может быть либо одно выражение, либо последовательность таких выражений, разделённых между собой запятыми.

Текст, который необходимо вывести на экран заключают в апострофы.

Например: Writeln (‘Корнем уравнения является:’, X).

1.2. Выполнение работы

С использованием теоретического материала и материалов лекций по заданию преподавателя самостоятельно решить поставленные задачи. При выполнении данной лабораторной работы самостоятельно определить исходные требования, предъявляемые к программе, выбрать метод решения, составить алгоритм и написать программу на языке Паскаль.

Задание 1. Скорость первого автомобиля V_1 км/ч, второго V_2 км/ч, расстояние между ними S км. Написать программу, рассчитывающую расстояние между ними через время t ч, если автомобили движутся в разные стороны (в одну сторону).

Задание 2. Написать программу вычисления объёма шара. Формула для определения объёма шара: $V = 4/3\,\pi R^3$. В качестве исходных данных служит радиус шара.

Задание 3. Написать программу, которая печатает – если число e^{π} больше числа π^e и false – trve, в противном случае. Числа e и π с точностью 10^{-5} описывать как константы, e^{π} и π^e описывать формулами: exp(pi), exp(e*ln(pi)). В написании программы **не** использовать оператор условия.

1.3. Содержание отчета

Отчет оформляется каждым студентом самостоятельно. Защита происходит в начале каждого следующего занятия с демонстрацией ра-

боты программы на ЭВМ. Студент, не подготовивший или не защитивший отчет по работе, к следующей лабораторной работе не допускается.

Содержание отчета:

a) титульный лист;
b) цель работы;
c) задание на работу и исходные данные;
d) алгоритм решения поставленной задачи, записанный в виде блок-схемы и в виде последовательности действий на естественном языке;
e) исходный код программы, написанный на языке Turbo Pascal;
f) набор тестов для проверки работоспособности разработанной программы;
g) выводы по работе.

1.4. Контрольные вопросы

1. Назовите основные этапы решения задачи на ЭВМ.
2. Дайте понятие алгоритма, свойства алгоритма, представление алгоритма.
3. Опишите общий вид программы на языке Turbo Pascal.
4. Назовите числовые типы переменных в Turbo Pascal.
5. Охарактеризуйте операторы write и read, операторы сложения, вычитания, умножения, деления, возведения в степень и извлечения корня. Опишите оператор присваивания.

Лабораторная работа № 2
РЕШЕНИЕ ЗАДАЧ С ИСПОЛЬЗОВАНИЕМ ОПЕРАТОРА УСЛОВИЯ IF

Цель работы: разработка программы на языке Turbo Pascal для решения поставленной задачи, с прохождением всех основных этапов разработки программы.

Оборудование: дисплейный класс, интегрированная среда Turbo Pascal версии 7.0 или выше.

2.1. Общие сведения

Условный оператор используется в тех случаях, когда вычисления могут пойти по различным путям, в зависимости от выполнения

или невыполнения определённых условий. В языке Паскаль оператор условия имеет следующий вид: IF < логическое выражение > Then < оператор1 > Else < оператор2 >.

Здесь IF (если), THEN (тогда), ELSE (иначе) – служебные слова; оператор1 и оператор2 – простые или составные операторы. Если логическое выражение истинно, тогда выполняется оператор1, если логическое выражение ложно, тогда оператор2. Блок схема условного оператора IF в полной и сокращённой форе представлена на рис. 2.1.

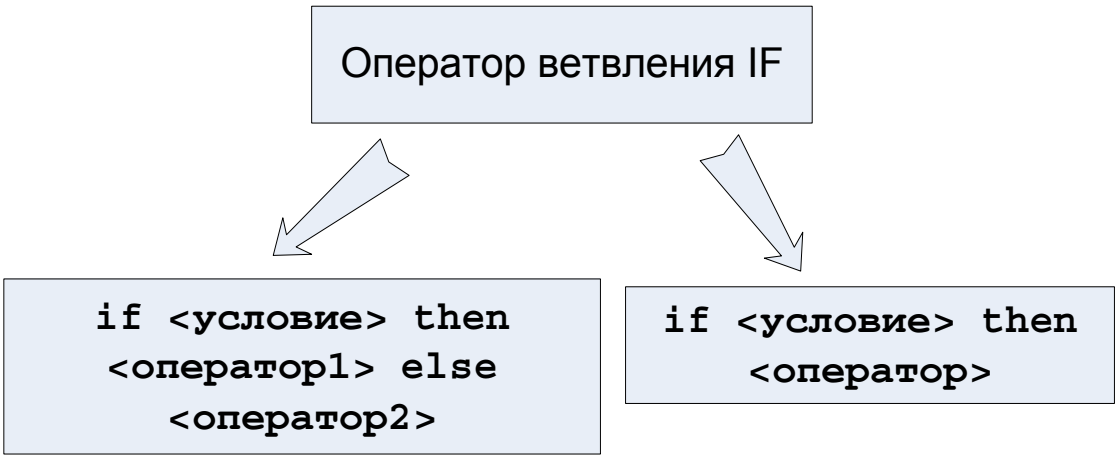

Рис. 2.1. Блок схема условного оператора IF

Следует обратить внимание, что при записи составных операторов используются операторные скобки begin ... end. Перед служебным словом ELSE, знак точка с запятой не ставится. Пример записи составного оператора:

```
IF A>B Then
        Begin
        X:=5;
        Y:=A+B
        End
  Else
 Begin
X:=2*A;
Y:=A/B
        End.
```

Кроме операторов условного выполнения в языке Паскаль имеется оператор выбора, который используется в тех случаях, когда в

зависимости от значения какого-либо выражения необходимо выполнить один из нескольких последовательных операторов. Оператор выбора имеет следующую форму записи:

CASE < выражение > OF
Константа 1: оператор 1;
Константа 2: оператор 2;
Константа n: оператор n
END

Здесь CASE (выбор), OF (из), END (конец) – служебные слова. Если значение выражения равно одной из констант, то выполняется соответствующий ей оператор. Затем управление передаётся за пределы оператора выбора. Если значение выражения не совпадает ни с одной константой, то управление передаётся за пределы группы.

Выражение может быть любым стандартным типом, кроме действительного (Real). В соответствии с этим и константа не может быть действительного типа. Тип константы должен совпадать с типом выражения.

2.2. Выполнение работы

По заданию преподавателя самостоятельно решить поставленную задачу с применением знаний и навыков, полученных в ходе первой лабораторной работы.

При выполнении данной лабораторной работы самостоятельно определить исходные требования, предъявляемые к программе, выбрать метод решения, составить алгоритм и написать программу на языке Паскаль. Составить необходимый набор тестовых данных для тестирования программы.

Задание 1. Написать программу вычисления значения функции:

$$y = \begin{cases} \cos^2 x, & \text{если } 0 \le x \le 2; \\ 1 - \sin x^2 & \text{в остальных случаях.} \end{cases}$$

Задание 2. Написать программу выбора наибольшего из трех чисел.

Задание 3. Написать программу для решения квадратного уравнения $AX^2 + BX + C = 0$.

При составлении программы необходимо предусмотреть следующие возможности:

• Если D > 0, то уравнение имеет два положительных корня $X_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$.

• Если D = 0, то уравнение имеет один положительный корень $X = -\frac{B}{2C}$.

• Если D < 0, то уравнение не имеет положительных корней.

Исходные значения A, B, C вводятся с клавиатуры в ходе выполнения программы, результат выполнения программы отображается на экране монитора.

Для проверки выполнения различных условий использовать оператор «IF THEN ELSE». Для ввода и вывода данных – операторы «WRITE, WRITELN, READ, READLN». Если нужно – применить операторные скобки «BEGIN ... END».

2.3. Содержание отчета

Отчет оформляется каждым студентом самостоятельно. Защита происходит в начале каждого следующего занятия с демонстрацией работы программы на ЭВМ. Студент, не подготовивший или не защитивший отчет по работе, к следующей лабораторной работе не допускается.

Содержание отчета:

a) титульный лист;

b) цель работы;

c) задание на работу и исходные данные;

d) алгоритм решения поставленной задачи ,записанный в виде блок-схемы и в виде последовательности действий на естественном языке;

e) исходный код программы, написанный на языке Turbo Pascal;

f) набор тестов для проверки работоспособности разработанной программы;

g) выводы по работе.

2.4. Контрольные вопросы

1. Какова форма записи условного оператора IF?
2. В каких случаях используют операторные скобки begin ... end?
3. Какова форма записи оператора выбора CASE?

Лабораторная работа № 3
РЕШЕНИЕ ЗАДАЧ С ПОМОЩЬЮ ОПЕРАТОРОВ ЦИКЛОВ

Цель работы: разработка программы на языке Turbo Pascal для решения поставленной задачи с применением всех изученных операторов циклов.

Оборудование: дисплейный класс, интегрированная среда Turbo Pascal версии 7.0 или выше.

3.1. Общие сведения

Командой повторения или *циклом* называется такая форма организации действий, при которой одна и та же последовательность действий повторяется до тех пор, пока сохраняется значение некоторого логического выражения. При изменении значения логического выражения на противоположное, повторения прекращаются (цикл завершается).

Для организации цикла необходимо выполнить следующие действия:

- перед началом цикла задать начальное значение параметра;
- внутри цикла изменять параметр цикла с помощью оператора присваивания;
- проверять условие повторения или окончания цикла;
- управлять циклом, т. е. переходить к его началу, если он не закончен, или выходить из цикла в противном случае.

Различают циклы с известным числом повторений (*цикл с параметром*) и *итерационные* (с пред- и постусловием).

В цикле с известным числом повторений параметр изменяется в заданном диапазоне.

Если в цикле изменяется простая переменная, то она является параметром цикла, если в цикле изменяется переменная с индексом, то индекс этой переменной является параметром цикла.

Для организации цикла с известным числом повторений в языке Pascal используется оператор for.

Структура цикла, организованного с помощью этого оператора, имеет вид:

For I:= A To B Do Begin < операторы > End;

или

For I:= A DownTo B Do Begin < операторы > End;

Здесь I – параметр, изменяющийся в цикле; A, B – выражения порядкового типа, обозначающие начальное, конечное значение параметра цикла. Шаг изменения номера параметра цикла равен 1, если в заголовке цикла стоит To (т. е. реально следующее значение параметра цикла вычисляется с помощью функции succ); и -1 – при DownTo (вычисление производится с помощью функции pred).

Порядок выполнения цикла с шагом 1 следующий: вычисляются значения начального и конечного значений параметра цикла; если I принимает начальное значение; если I меньше или равно конечному значению, исполняется тело цикла; значение параметра цикла увеличивается, т. е. I:= I+1 (для DOWNTO уменьшается – I:=I-1); проверяется условие I ≤ B (для отрицательного шага условие I ≥ B) и при его выполнении цикл повторяется. Выход из цикла осуществляется, если I >B (I<B для H = -1), и выполняется оператор, следующий за оператором цикла. Если A > B (или A < B для H = -1), то цикл не исполняется ни разу.

Если в операторе цикла с параметром начальное или конечное значение параметра заданы переменными или выражениями, то значения этих переменных должны быть определены в программе до оператора цикла. Не следует внутри цикла изменять параметр цикла, его начальное и конечное значения с помощью операторов присваивания или ввода.

Достаточно часто цикл с параметром используется при разработке программ обработки массивов.

Примечание. Из сказанного выше следует, что область применения цикла с параметром в языке Pascal значительно ограничена: ограничения связаны с шагом изменения параметра цикла, с типом параметра цикла, его начального и конечного значения. В некоторых языках, например в Basic, таких ограничений не существует.

По сравнению с циклом с параметром итерационные циклы являются универсальными. Для организации итерационных циклов используются операторы цикла с предусловием while и цикла с постусловием until.

Эти операторы не задают закон изменения параметра цикла, поэтому необходимо перед циклом задавать начальное значение параметра с помощью оператора присваивания, а внутри цикла изменять текущее значение этого параметра.

Соответствующие структуры циклов:

while B Do Begin < операторы > End;

Repeat < операторы > Until C;

Здесь B, C – логические выражения.

Для оператора цикла с предусловием проверяется значение логического выражения, если оно имеет значение True (истинно), то операторы, входящие в цикл, выполняются, в противном случае осуществляется выполнение оператора, следующего за циклом.

Цикл с постусловием выполняется хотя бы один раз. Затем проверяется значение логического выражения, если оно False (ложно), то операторы, входящие в цикл, выполняются, в противном случае осуществляется выход из цикла.

Входить в цикл можно только через его начало, т. е. нельзя входить внутрь цикла с помощью управляющего оператора, т. к. в этом случае параметр цикла не определен.

Внутрь одного цикла может входить еще один или несколько других. При этом охватывающий цикл называется внешним, а вложенные циклы – внутренними. Правила организации как внешнего, так и внутренних циклов такие же, как и простого цикла.

3.2. Выполнение работы

Разработать три программы для решения поставленных задач с применением операторов циклов.

Задание 1. Дано трехзначное число. Назовем это число «счастливым», если у него сумма трех цифр равна 13. Подсчитать количество всех «счастливых» трехзначных чисел, у которых суммы трех цифр равны 13. Использовать цикл FOR.

Задание 2. Пусть даны два числа A и B, (A>1) и надо получить все члены бесконечной последовательности A, A^2, A^3, … меньшие числа B. Использовать цикл WHILE.

Задание 3. Написать программу подсчёта суммы положительных чисел введённых с клавиатуры. Как только будет введено отрицательное число, программа завершает свою работу. Использовать цикл REPEAT.

3.3. Содержание отчета

Отчет оформляется каждым студентом самостоятельно. Защита происходит в начале каждого следующего занятия с демонстрацией работы программы на ЭВМ. Студент, не подготовивший или не защитивший отчет по работе, к следующей лабораторной работе не допускается.

Содержание отчета:

a) титульный лист;
b) цель работы;
c) задание на работу и исходные данные;
d) алгоритм решения поставленной задачи, записанный в виде блок-схемы или в виде последовательности действий на естественном языке;
e) исходный код программы, написанный на языке Turbo Pascal;
f) набор тестов для проверки работоспособности разработанной программы;
g) выводы по работе.

3.4. Контрольные вопросы

1. Назовите отличия итерационных циклов от циклов с параметрами.
2. Какова структура оператора цикла с параметром? Как выполняется цикл с параметром?
3. Какого типа должен быть параметр цикла, его начальное и конечное значения в цикле с параметром на языке Pascal?
4. Могут ли параметр цикла, его начальное и конечное значения в цикле с параметром в языке Pascal быть разных типов? Обоснуйте ответ.
5. Может ли один цикл быть вложен внутрь другого? Если да, то какова глубина этой вложенности?
6. Какова структура циклов с пред- и постусловием? Как выполняются эти циклы?
7. Каково минимальное и максимальное количество исполнений циклов с пред- и постусловием? С чем это связано?
8. Сколько раз исполнится фрагмент программы?

```
For i:= 1 to -1 Do k:=k·i;
```

9. Сколько раз исполнится фрагмент программы?
   For i:= -1 to 1 Do k:=k·i;
10. Сколько раз исполнится фрагмент программы?
   For i:= 1 downto -1 Do k:=k·i;
11. Сколько раз исполнится фрагмент программы?
   M:= 123; While M <> 0 Do M:= M Mod 10;
12. Для цикла с параметром запишите его полный эквивалент с помощью циклов с пред- и постусловием.
13. Для цикла с предусловием запишите его полный эквивалент с помощью цикла с постусловием.
14. Для цикла с постусловием запишите его полный эквивалент с помощью цикла с предусловием.

Лабораторная работа № 4
ПРОЦЕДУРЫ И ФУНКЦИИ

Цель работы: разработка программы на языке Turbo Pascal для решения поставленной задачи с применением процедур и функций.

Оборудование: дисплейный класс, интегрированная среда Turbo Pascal версии 7.0 или выше.

4.1. Общие сведения

Описание каждой процедуры начинается с заголовка, в котором задаются имя процедуры и список формальных параметров с указанием их типов. Процедура может быть и без параметров, тогда в заголовке указывается только ее имя. С помощью параметров осуществляется передача исходных данных в процедуру, а также передача результатов работы обратно в вызвавшую ее программу.

Общая форма записи заголовка процедуры:

PROCEDURE < имя > (< список формальных параметров >);

Список формальных параметров может включать в себя параметры-значения, параметры-переменные (перед ними должно стоять служебное слово VAR), параметры-процедуры (перед ними должно стоять служебное слово PROCEDURE) и параметры-функции (перед ними должно стоять служебное слово FUNCTION). После заголовка процедуры следуют разделы в том же порядке, что и в программе.

Вызов и выполнение процедуры осуществляются при помощи оператора процедуры:

< имя процедуры >(< список фактических параметров >);

Между формальными и фактическими параметрами должно быть полное соответствие, т. е. формальных и фактических параметров должно быть одинаковое количество, порядок следования фактических и формальных параметров должен быть один и тот же, тип каждого фактического параметра должен совпадать с типом соответствующего ему формального параметра.

При вызове процедуры сначала передаются параметры, при этом параметры-значения передаются по значению, а параметры-переменные – по ссылке. Основное отличие этих способов передачи параметров заключается в том, что присваивания значений параметру-переменной внутри процедуры одновременно выполняются и для соответствующего аргумента (фактического параметра). Таким образом, параметры, в которые записываются результаты работы процедуры, должны передаваться только по ссылке. Параметры, через которые в процедуру передаются исходные данные, передаются по значению.

Например, ниже приведена программа, которая в первой вводимой с терминала строке подсчитывает количество точек, а во второй – количество букв 'А'. Подсчет символов реализован в процедуре ПОДСЧЕТ.

```
PROGRAM ПОД;
CONST ТОЧКА=’.’;
VAR S : INTEGER;
PROCEDURE ПОДСЧЕТ (СИМ: CHAR; VAR КОЛ : INTEGER);
VAR C : CHAR;
BEGIN КОЛ :=0; WRITELN( ‘ВВОДИ СТРОКУ=');
REPEAT  READ(C);
IF C=СИМ THEN КОЛ:=КОЛ+1
 UNTIL EOLN
 END;
BEGIN READLN;
ПОДСЧЕТ (ТОЧКА,S);
WRITELN('КОЛИЧЕСТВО ТОЧЕК=',S:3);
ПОДСЧЕТ ( 'A' , S);
 WRITELN ('КОЛИЧЕСТВО БУКВ=A' , S :3)
 END.
```

Процедура ПОДСЧЕТ имеет два формальных параметра: СИМ – входной параметр (параметр-значение) определяет символ, который подсчитывается в строке, и КОЛ – выходной параметр (параметр-переменная), через который передается в основную программу количество подсчитанных символов. Для того чтобы результат работы процедуры ПОДСЧЕТ был доступен в программе, он передается по ссылке. EOLN – ожидание в строке символов клавиши enter.

Фактический параметр, соответствующий формальному параметру СИМ, при первом вызове процедуры задается именем константы, определенной в основной программе, а при втором вызове – в явном виде. Обе формы задания правильны, так как в качестве фактического параметра, передаваемого по значению, может использоваться произвольное выражение соответствующего типа.

Процедуры возвращают результат в основную программу не только при помощи параметров-переменных, но и непосредственно изменяя глобальные переменные. Переменные, описанные в основной программе, являются глобальными по отношению к внутренним процедурам и функциям. *Переменные, описанные внутри процедур и функций, называются локальными.* Они порождаются при каждом входе в процедуру и уничтожаются при выходе из этой процедуры, т. е. локальные переменные существуют только при выполнении процедуры и недоступны в основной программе. Например, переменная С символьного типа, описанная в процедуре ПОДСЧЕТ, является локальной, а переменная S целого типа, описанная в основной программе, является глобальной.

Изменим процедуру ПОДСЧЕТ таким образом, чтобы она передавала результат своей работы через глобальную переменную S.

```
PROGRAM ПОД 1;
CONST ТОЧКА='.';
VAR S : INTEGER;
PROCEDURE ПОДСЧЕТ (СИМ:CHAR);
VAR C: CHAR;
BEGIN S:=0; WRITELN (' ВВОДИ СТРОКУ=');
 REPEAT READ(C);
IF C=СИМ THEN S:=S+l
UNTIL EOLN
END;
```

```
 BEGIN READLN;
ПОДСЧЕТ (ТОЧКА);
WRITELN('КОЛИЧЕСТВО ТОЧЕК=',S:3 );
 ПОДСЧЕТ ('А'); .
WRITELN(' КОЛИЧЕСТВО БУКВ А=',S:3)
END.
```

Программы ПОД и ПОД 1 работают одинаково. Для каждой конкретной задачи программист может выбирать тот или иной способ передачи результатов работы процедуры в вызвавшую ее программу. Однако, в сложных программных комплексах не рекомендуется использование глобальных переменных, так как это ухудшает структурированность программ.

В языке Pascal допускается любой уровень вложенности процедур и функций. Например, процедура, описанная в основной программе, в свою очередь, имеет описания внутренних процедур или функций и т. д. Для таких сложных программ имеются правила локализации имен, определяющие область действия для любого имени.

1. Любое имя (константы, типы, переменной, процедуры или функции) определено только в пределах той процедуры или функции, в которой оно описано. Область действия распространяется на все внутренние процедуры или функции.

2. Одно и то же имя может быть определено в каждой отдельной процедуре (функции) или в программе. При этом областью действия этого имени является процедура (функция) или вся программа, в которой описан объект с данным именем, за исключением внутренних процедур, содержащих описание объекта с таким же именем.

Описание функции. Указатель функции

Описание функции в основном аналогично описанию процедуры. Однако имеются некоторые отличия. Результатом работы функции является одно скалярное значение или одно значение ссылочного типа. Тип результата задается в заголовке функции, общий вид которого

FUNCTION < имя функции > (< список формальных параметров >): < тип результата >

Если функция изменяет значения формальных параметров-переменных или значения глобальных по отношению к данной функции переменных, то говорят, что функция имеет побочный эффект.

Применение функций с побочным эффектом нарушает структурированность программы, поэтому их использование нежелательно.

Этот оператор и определяет значение, вырабатываемое функцией.

Вызов и выполнение функции производятся при вычислении значения указателя функции, который входит в некоторое выражение. После выполнения функции выработанный ею результат используется в качестве значения указателя функции в том выражении, в которое входит этот указатель. При вызове функции передача фактических параметров производится так же, как и при вызове процедуры.

4.2. Выполнение работы

Задание 1. Вычислить значение F= (m!-k!)·l! Программа должна запрашивать значения m, k, l. Вычисление факториала оформить в виде функции FACT.

Задание 2. Оформить в виде процедуры «SumPR» вычисление суммы 1+2+3+…N и произведения 1·2·3·….N целых чисел.

Задание 3. Даны отрезки a, b, c, d. Для каждой тройки этих отрезков, из которых можно построить треугольник, напечатать площадь данного треугольника. Определить процедуру AREA(x, y, z), печатающую площадь треугольника со сторонами x, y, z, если такой треугольник существует (сумма двух его сторон больше третьей).

4.3. Содержание отчета

Отчет оформляется каждым студентом самостоятельно. Защита происходит в начале каждого следующего занятия с демонстрацией работы программы на ЭВМ. Студент, не подготовивший или не защитивший отчет по работе, к следующей лабораторной работе не допускается.

Содержание отчета:

a) титульный лист;
b) цель работы;
c) задание на работу и исходные данные;
d) алгоритм решения поставленной задачи, записанный в виде блок-схемы или в виде последовательности действий на естественном языке;

e) исходный код программы, написанный на языке Turbo Pascal;
f) набор тестов для проверки работоспособности разработанной программы;
g) выводы по работе.

4.4. Контрольные вопросы

1. Для чего предназначены процедуры?
2. Что включает в себя заголовок процедуры?
3. Чем отличаются формальные и фактические параметры?
4. Чем отличаются локальные и глобальные переменные?
5. Для чего предназначены функции?
6. Что включает в себя заголовок функции?
7. Какая разница между процедурой и функцией?

Лабораторная работа № 5
МАССИВЫ

Цель работы: разработка программы на языке Turbo Pascal с применением массивов.

Оборудование: дисплейный класс, интегрированная среда Turbo Pascal версии 7.0 или выше.

5.1. Общие сведения

Массив – это пронумерованная последовательность величин одинакового типа, обозначаемая одним именем. Элементы массива располагаются в последовательных ячейках памяти, обозначаются именем массива и индексом. Каждое из значений, составляющих массив, называется его компонентой (или элементом массива).

Массив данных в программе рассматривается как переменная структурированного типа. Массиву присваивается имя, посредством которого можно ссылаться как на массив данных в целом, так и на любую из его компонент.

Переменные, представляющие компоненты массивов, называются переменными с индексами в отличие от простых переменных,

представляющих в программе элементарные данные. Индекс в обозначении компонент массивов может быть константой, переменной или выражением порядкового типа.

Если за каждым элементом массива закреплен только один его порядковый номер, то такой массив называется линейным. Вообще количество индексов элементов массива определяет размерность массива. По этому признаку массивы делятся на одномерные (векторы), двумерные (матрицы), трёхмерные и т. д.

Пример: числовая последовательность четных натуральных чисел 2, 4, 6, ..., N представляет собой линейный массив, элементы которого можно обозначить A[1]=2, A[2]=4, A[3]=6, ..., A[K]=2*(K+1), где K – номер элемента, а 2, 4, 6, ..., N – значения. Индекс (порядковый номер элемента) записывается в квадратных скобках после имени массива.

Например, A[7] – седьмой элемент массива A; D[6] – шестой элемент массива D.

Для размещения массива в памяти ЭВМ отводится поле памяти, размер которого определяется типом, длиной и количеством компонент массива. В языке Pascal эта информация задается в разделе описаний. Массив описывается так: < имя массива > : Array [< начальное значение индекса .. конечное значение индекса >] Of < базовый тип >;

Например:

```
Var B : Array [1..5] Of Real;
    R : Array [1..34] Of Char;
```

– описывается массив B, состоящий из 5 элементов, и символьный массив R, состоящий из 34 элементов. Для массива B будет выделено 5·6=30 байт памяти, для массива R – 1·34=34 байта памяти.

Базовый тип элементов массива может быть любым, за исключением файлового.

Способы заполнения массива

Первый способ: с помощью оператора присваивания. Этот способ заполнения элементов массива особенно удобен, когда между элементами существует какая-либо зависимость, например, арифметическая или геометрическая прогрессии, или элементы связаны между собой реккурентным соотношением.

Пример 1. Заполнить одномерный массив элементами, отвечающими следующему соотношению:

$$a_1=1;\ a_2=1;\ a_i=a_{i-2}+a_{i-1}\ (i = 3, 4, ..., n).$$

```
    Read(N); {Ввод количества элементов}
    A[1]:= 1;
    A[2]:= 1;
    FOR I := 3 TO N DO  A[I] := A[I - 1] + A[I - 2];
```

Другой вариант присваивания значений элементам массива – заполнение значениями, полученными с помощью генератора случайных чисел.

Пример 2. Заполнить одномерный массив с помощью датчика случайных чисел таким образом, чтобы все его элементы были различны.

```
Program Create;
Type Mas = Array[1..100] Of Integer;
Var A : Mas;
    I, J, N : Byte;
    Log : Boolean;
Begin
  Write('Кол-во элементов массива: '); ReadLn(N);
  randomize;  {сброс генератора случайных чисел}
  A[1] := -32768 + random(65535);{получение случайных чисел}
  For I := 2 To N Do
    Begin
     Log := True;
     Repeat
       A[i] := -32768 + random(65535);
       J := 1;
       While Log and (j <= i - 1) Do
          begin
          Log := a[i] <> a[j];
          j := j + 1
          End
     Until Log
  End;
 For i := 1 to N Do Write(a[i]:7);
End.
```

Второй способ: ввод значений элементов массива с клавиатуры. Используется обычно тогда, когда между элементами не наблюдается никакой зависимости. Например, последовательность чисел 1, 2, -5, 6, -111, 0 может быть введена в память следующим образом:

```
Program Vvod;
 Var N, I : Integer;
      A : Array [1..20] Of Integer;
Begin
  Write('Введите количество элементов массива ');  ReadLn(N);
  FOR I := 1 TO N DO
    Begin
    Write('Введите A[', I, '] '); ReadLn(A[I])
    End;
End.
```

Над элементами массива чаще всего выполняются такие действия, как

а) поиск значений;

б) сортировка элементов в порядке возрастания или убывания;

в) подсчет элементов в массиве, удовлетворяющих заданному условию.

Сумму элементов массива можно подсчитать по формуле S = S + A[I], первоначально задав S = 0. Количество элементов массива можно подсчитать по формуле K = K + 1, первоначально задав K = 0. Произведение элементов массива можно подсчитать по формуле P = P · A[I], первоначально задав P = 1.

Если два массива являются массивами эквивалентных типов, то возможно присваивание одного массива другому. При этом все компоненты присваиваемого массива копируются в тот массив, которому присваивается значение. Типы массивов будут эквивалентными, если эти массивы описываются совместно или описываются идентификатором одного и того же типа. Например, в описании

```
Type Massiv = Array[1..10] Of Real;
Var A, B : Massiv;
     C, D : Array[1..10] Of Integer;
     E : Array[1..10] Of Real;
```

типы переменных A, B эквивалентны, и поэтому данные переменные совместимы по присваиванию; тип переменных C, D также один и тот же, и поэтому данные переменные также совместны по присваиванию. Но тип переменных C, D не эквивалентен типам переменных A, B, E, поэтому, например, A и D не совместимы по присваиванию. Эти особенности необходимо учитывать при работе с массивами.

При решении практических задач часто приходится иметь дело с различными таблицами данных, математическим эквивалентом которых служат матрицы. Такой способ организации данных, при котором каждый элемент определяется номером строки и номером столбца, на пересечении которых он расположен, называется *двумерным массивом* или *таблицей*.

Их можно занести в память компьютера, используя понятие двумерного массива. Положение элемента в массиве определяется двумя индексами. Они показывают номер строки и номер столбца. Индексы разделяются запятой. Например: A[7, 6], D[56, 47].

Заполняется двумерный массив аналогично одномерному с клавиатуры с помощью оператора присваивания. Например, в результате выполнения программы:

```
Program Vvod2;
 Var I, J : Integer;
     A : Array [1..20, 1..20] Of Integer;
 Begin
  FOR I := 1 TO 3 DO
      FOR J := 1 TO 2 DO A[I, J] := 456 + I;
 End.
```

элементы массива примут значения A[1, 1] = 457; A[1, 2] = 457; A[2, 1] = 458; A[2, 2] = 458; A[3, 1] = 459; A[3, 2] = 459.

При описании массива задается требуемый объем памяти под двумерный массив, указываются имя массива и в квадратных скобках диапазоны изменения индексов.

При выполнении инженерных и математических расчетов часто используются переменные более чем с двумя индексами. При решении задач на ЭВМ такие переменные представляются как компоненты соответственно трех-, четырехмерных массивов и т. д.

5.2. Выполнение работы

Задание 1. Дан линейный массив целых чисел. Подсчитать, сколько в нем различных чисел. Размер массива задан изначально и содержит не менее 10 чисел. Числа, входящие в массив, получить либо путем ввода с клавиатуры, либо с помощью генератора случайных чисел.

Для решения поставленной задачи можно завести вспомогательный массив, элементами которого являются логические величины (False – если элемент уже встречался ранее, True – иначе).

Задание 2. Ввести строку символов длиной не меньше 15. Признаком окончания строки является нажатие клавиши Enter. Подсчитать общее количество введённых символов строки и сколько раз встречается знак «+», начиная с 5-го по 15-й. Строку символов рассмотреть как массив с именем SIM. Знак «+» обозначить именем PLUS. В программе организовать два цикла: один для ввода строки символов и подсчёта их количества, другой – для подсчёта знака «+».

Примечание: EOLN – функция, определяющая достигнут ли в файле конец строки.

Задание 3. Подсчитать сумму и произведение всех элементов двумерного массива целых чисел B[I, J], где I = 5, J = 10.

5.3. Содержание отчета

Отчет оформляется каждым студентом самостоятельно. Защита происходит в начале каждого следующего занятия с демонстрацией работы программы на ЭВМ. Студент, не подготовивший или не защитивший отчет по работе, к следующей лабораторной работе не допускается.

Содержание отчета:

a) титульный лист;
b) цель работы;
c) задание на работу и исходные данные;
d) алгоритм решения поставленной задачи, записанный в виде блок-схемы или в виде последовательности действий на естественном языке;
e) исходный код программы, написанный на языке Turbo Pascal;
f) набор тестов для проверки работоспособности разработанной программы;
g) выводы по работе.

5.4. Контрольные вопросы

1. Что такое массив?
2. Почему массив является структурированным типом данных?

3. Что такое размерность массива? Существуют ли ограничения на размерность массива?

4. Какого типа могут быть элементы массива?

5. Какого типа могут быть индексы элементов массива?

6. Какие простые типы данных относятся к порядковым?

7. Какими способами может быть заполнен массив? Приведите примеры.

8. Как определить минимальный объём памяти, отводимой под массив?

9. Какие действия выполняют обычно над элементами массива?

10. Может ли массив быть элементом массива?

11. В каком случае массивы совместны по присваиванию?

Лабораторная работа № 6
МАССИВЫ. СОРТИРОВКА И ПОИСК

Цель работы: научиться оперировать массивами, производить сортировку массива и поиск в массиве.

Оборудование: дисплейный класс, интегрированная среда Turbo Pascal версии 7.0 или выше.

6.1. Общие сведения

Существует много методов (алгоритмов) сортировки массивов. Два широко известных из них: метод прямого выбора и метод прямого обмена.

Алгоритм сортировки массива по возрастанию (убыванию) методом прямого выбора может быть представлен так:

1. Просматривая массив от первого элемента, найти минимальный (максимальный) элемент и поместить на место первого элемента, а первый – на место минимального (максимального).

2. Просматривая массив от второго элемента, найти минимальный (максимальный) элемент и поместить его на место второго элемента, а второй – на место минимального (максимального).

3. И так далее до предпоследнего элемента.

В основе алгоритма сортировки методом прямого обмена лежит метод обмена соседних элементов массива. Каждый элемент массива,

начиная с первого, сравнивается со следующим, и если он больше следующего, то элементы меняются местами. Таким образом, элементы с меньшим значением продвигаются к началу массива (всплывают), а элементы с большим значением – к концу массива (тонут), поэтому этот метод иногда называют методом «пузырька».

При решении многих задач возникает необходимость установить, содержит массив определенную информацию или нет. Задачи такого типа называются поиском в массиве.

Для организации поиска в массиве могут быть использованы различные алгоритмы. Наиболее простой – это алгоритм простого перебора. Поиск осуществляется последовательным сравнением элементов массива с образцом до тех пор, пока не будет найден элемент, равный образцу, или не будут проверены все элементы. Алгоритм простого перебора применяется, если элементы массива не упорядочены.

Очевидно, что чем больше элементов в массиве и чем дальше расположен нужный элемент от начала массива, тем дольше будет программа искать нужный элемент.

На практике довольно часто проводится поиск в массиве, элементы которого упорядочены по некоторому критерию. Для поиска в упорядоченных массивах применяют другие, более эффективные по сравнению с методом простого перебора, алгоритмы, один из которых – метод бинарного поиска.

Суть метода бинарного поиска заключается в следующем. Выбирается средний (по номеру) элемент упорядоченного, например, по возрастанию массива, и с этим элементом сравнивается образец.

Если средний элемент равен образцу, то задача решена.

Если средний элемент меньше образца, то искомый элемент расположен выше среднего элемента. Если средний элемент больше образца, то искомый элемент расположен ниже среднего.

После того как определена часть массива, в которой может располагаться искомый элемент, поиск производят в этой части, выделяя новый средний элемент.

6.2. Выполнение работы

Задание 1. Написать программу для сортировки массива целых чисел по возрастанию. Числа в массив получать с клавиатуры или с

помощью генератора случайных чисел. Размер массива не менее 10 элементов. Перед началом сортировки и после окончания сортировки вывести содержимое массива на экран. Сортировку производить методом прямого обмена или методом прямого выбора.

Задание 2. Написать программу для поиска заданного элемента в массиве. Заданный элемент вводить с клавиатуры. Массив заполнять с клавиатуры или с помощью генератора случайных чисел. Размер массива не менее 10 элементов. Поиск производить методом простого перебора. В результате выполнения программы вывести на экран исходный массив, индекс заданного для поиска элемента или сообщение о том, что заданный элемент в массиве отсутствует. Также сосчитать количество произведенных сравнений элементов массива с образцом, прежде чем будет найден искомый элемент и вывести полученное значение на экран.

Задание 3. Написать программу для поиска заданного элемента в массиве. Заданный элемент вводить с клавиатуры. Массив заполнять с клавиатуры или с помощью генератора случайных чисел. Размер массива не менее 10 элементов. Поиск производить методом бинарного поиска. Для этого перед началом поиска произвести упорядочивание массива по возрастанию. В результате выполнения программы вывести на экран исходный массив, индекс заданного для поиска элемента или сообщение о том, что заданный элемент в массиве отсутствует. Также сосчитать количество произведенных сравнений элементов массива с образцом, прежде чем будет найден искомый элемент, и вывести полученное значение на экран.

6.3. Содержание отчета

Отчет оформляется каждым студентом самостоятельно. Защита происходит в начале каждого следующего занятия с демонстрацией работы программы на ЭВМ. Студент, не подготовивший или не защитивший отчет по работе, к следующей лабораторной работе не допускается.

Содержание отчета:

a) титульный лист;
b) цель работы;
c) задание на работу и исходные данные;

d) алгоритм решения поставленной задачи, записанный в виде блок-схемы или в виде последовательности действий на естественном языке;
e) исходный код программы, написанный на языке Turbo Pascal;
f) набор тестов для проверки работоспособности разработанной программы;
g) выводы по работе.

6.4. Контрольные вопросы

1. Что такое сортировка массива?
2. Поясните алгоритм сортировки массива методом прямого выбора.
3. Поясните алгоритм сортировки массива методом прямого обмена.
4. Что такое поиск заданного элемента массива?
5. Поясните поиск заданного элемента массива методом простого перебора.
6. Поясните поиск заданного элемента массива методом бинарного поиска.
7. Нарисуйте блок-схему алгоритма бинарного поиска.

Часть II. ОЗНАКОМЛЕНИЕ СО СРЕДОЙ ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ DELPHI 7.0

Лабораторная работа № 1
ОЗНАКОМЛЕНИЕ СО СРЕДОЙ ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ DELPHI

Цель работы: ознакомление со средой визуального программирования Delphi 7.0.

Оборудование: дисплейный класс, интегрированная среда Delphi версии 7.0 или выше.

1.1. Общие положения

Delphi – это среда разработки программ, ориентированных на работу в Windows. В основе идеологии Delphi лежат технология визуального проектирования и методология объектно-ориентированного программирования. Для представления программ в Delphi используется разработанный Borland язык Object Pascal, в основе которого лежит ставший классическим Turbo Pascal. Слово «Object» особо подчеркивает, что язык поддерживает концепцию объектно-ориентированного программирования.

1.2. Выполнение работы

Запуск Delphi

Если среда разработки Delphi установлена на компьютере, то для ее запуска надо в главном меню Windows нажать кнопку ПУСК, далее выбрать ПРОГРАММЫ, и затем в появившемся списке строку BORLAND DELPHI 7. В следующем списке следует выбрать строку DELPHI 7 и щелкнуть левой кнопкой мыши.

Начало работы в Delphi

Вид экрана после запуска Delphi представлен на рис. 1.1. Вместо одного окна на экране появляются пять: главное окно (Delphi7 – Project1), окно форм (Form1), дерево просмотра объектов (Object TreeView) окно инспектора объектов (Object Inspector) окно редактора кода (Unit1.pas), которое почти полностью закрыто окном формы.

В главном окне Delphi находятся меню команд Delphi, панель инструментов и палитра компонентов (рис. 1.2).

Окно формы **From1** представляет собой заготовку (макет) окна разрабатываемого приложения.

Окно инспектора объектов **Object Inspector** позволяет изменять свойства (характеристики) объектов: формы, командных кнопок, полей ввода и т. д. После запуска Delphi в диалоговом окне **Object Inspector** (инспектор объектов) находятся свойства формы **Form1**.

Окно редактора кода, которое можно увидеть, отодвинув в сторону окно формы или нажав клавишу «F12», содержит сформированный Delphi шаблон текста (кода) программы.

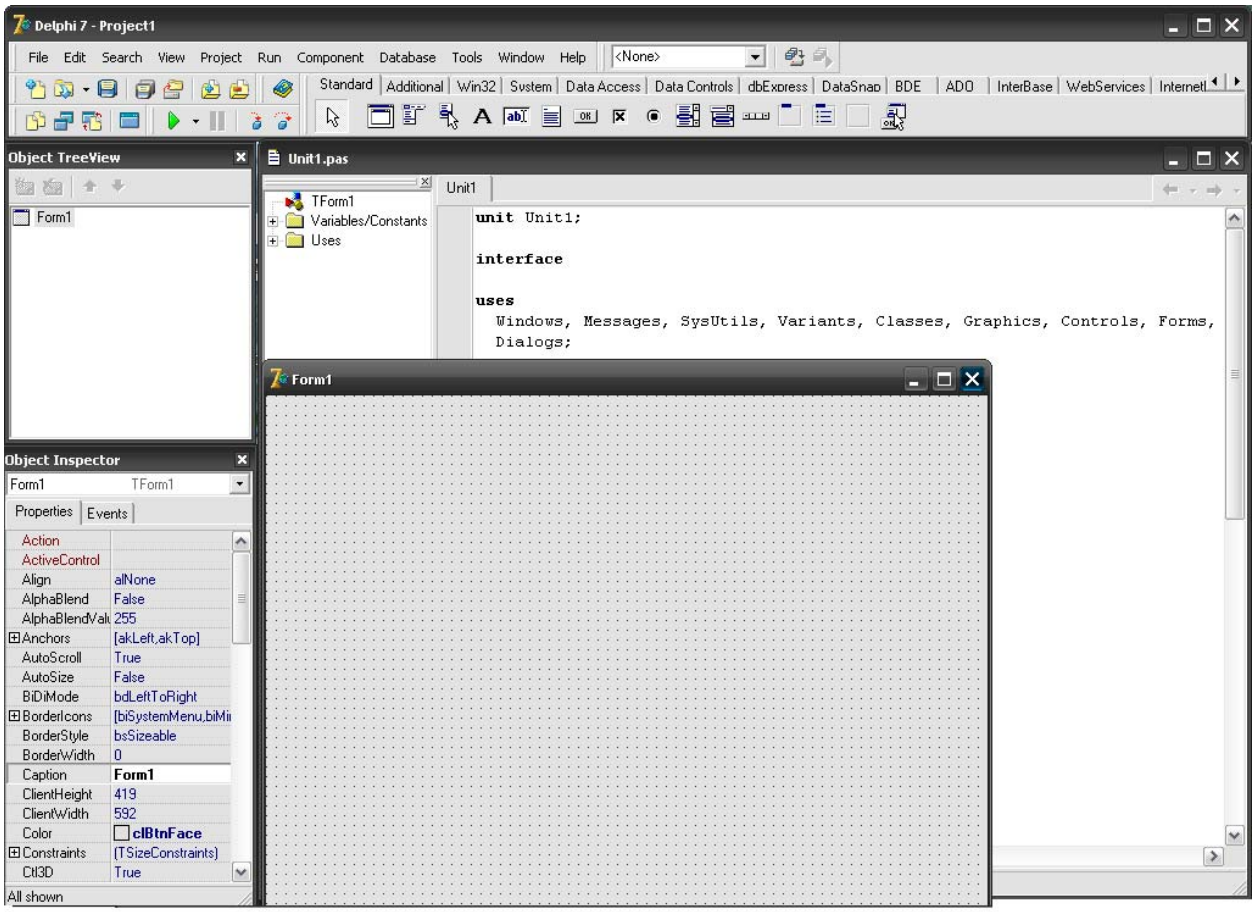

Рис. 1.1. Вид экрана после запуска Delphi

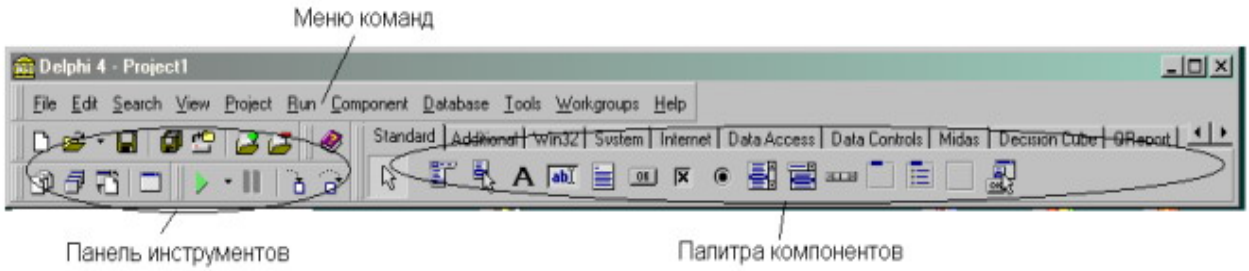

Рис. 1.2. Главное окно Delphi 7.0

Создание первой программы

Сначала создайте на диске пустой каталог, где будет находиться ваша первая программа на Delphi. Из меню главного окна Delphi выберите пункт **File -> New Application (**Файл -> Новое приложение). Всегда выполняйте данное действие, когда желаете перейти к созданию новой программы. Если в данный момент в Delphi уже была создана какая-то программа, появится окно с предложением сохранить старый проект.

Далее пользуясь верхним меню главного окна Delphi, выполните команду **File -> Save All** (файл -> сохранить все). На экране появится окно с заголовком «Save Unit1 As» (сохранить модуль 1 как), по умолчанию для него будет предложено имя Unit1.pas, выберите папку, в которой вы будете сохранять проект, и нажмите кнопку «Сохранить». Далее появится окно «Save Project1 As» (сохранить проект 1 как), по умолчанию имя проекта будет предложено «Project1.dpr», нажмите кнопку «Сохранить».

Работа над новым проектом (так в Delphi называется разрабатываемое приложение) начинается с создания стартовой формы – окна, которое появляется при запуске приложения.

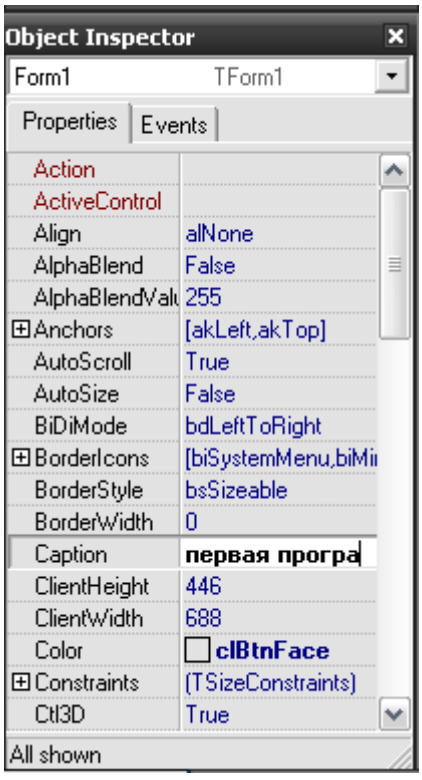

Рис. 1.3. Изменение заголовка окна программы

Стартовая форма создается путем изменения свойств (характеристик) формы **Form1**. Свойства формы определяют ее внешний вид: размер, положение на экране, текст заголовка, вид рамки. Свойства перечислены на вкладке **Properties** (свойства) диалогового окна **Object Inspector** (инспектор объектов) (рис. 1.3). В левой колонке находятся имена свойств, а в правой – их значения.

При создании формы изменим значение свойства Caption (заголовок). Назовем нашу программу «Первая программа». Аналогичным образом можно, например, изменить ширину и высоту формы (Height и Width) и, например, цвет формы (Color).

Теперь создадим исполняемый EXE файл, т. е. нашу первую созданную в Delphi программу. Для этого нужно скомпилировать наш проект.

Для этого, пользуясь верхним меню, выполните команду **Project -> Build All** (проект -> перестроить все). Всегда пользуйтесь этой командой, когда вам надо перекомпилировать весь проект. В результате в той папке, где мы сохранили наш проект, появиться EXE файл, который можно запустить на исполнение (имя EXE файла будет совпадать с именем проекта).

Запустить созданную программу можно прямо из Delphi, выбрав из меню пункт **Run -> Run** (запуск -> запустить). Ваша программа запустится и появится на экране. Все описанные выше действия можно также выполнять с помощью горячих клавиш, например «Ctrl+F9» – скомпилировать проект, «F9» – скомпилировать проект и запустить программу и др. По завершению компиляции в диалоговом окне Compiling отражается её результат (рис. 1.4). Если в программе нет синтаксических ошибок, то в поле Done выводится сообщение Compiled. В противном случае поле Errors содержит количество ошибок, обнаруженных компилятором, количество предупреждений (Warnings) и количество подсказок (Hints).

Для того чтобы в процессе компиляции на экран выводилось диалоговое окно Compiling (Компиляция), необходимо из меню Tools (Инструменты) выбрать команду Environment Options (Настройка), во вкладке Preferences (Установки) установить флажок Show compiler progress (Показать ход выполнения компиляции).

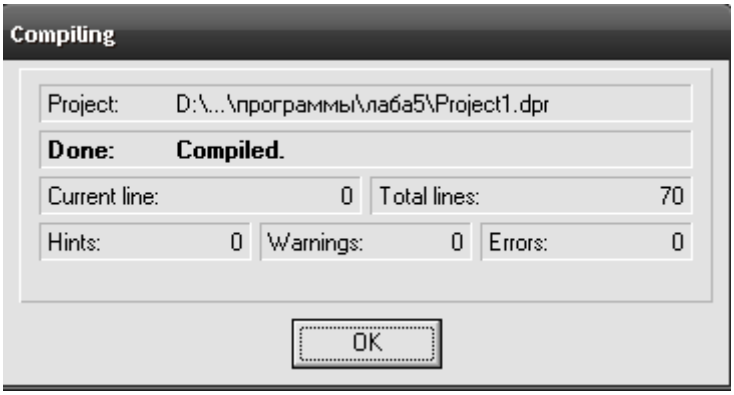

Рис. 1.4. Сообщение о результатах компиляции

Если посмотреть на окно программы, то оно будет иметь созданный нами заголовок, выбранные нами ширину и высоту, и цвет

фона. Его, как и окна других программ Windows, можно растянуть на весь экран, перетаскивать по экрану, сворачивать.

Обратите внимание! У нас уже есть готовая программа для Windows, которую можно запустить, хотя еще не было написано и строчки кода. Среда визуального проектирования Delphi сама написала за нас весь необходимый код.

Созданный Delphi код можно посмотреть. Для этого сверните окно с формой. По умолчанию под ним находится окно с редактором кода (рис. 1.5) (между окном формы и окном редактора кода можно переключаться с помощью клавиши «F12»). Этот код для нашего случая сохраняется в файле «unit1.pas». Изменять текущей текст для созданной программы крайне не рекомендуется, так как это может привести к неизвестным последствиям.

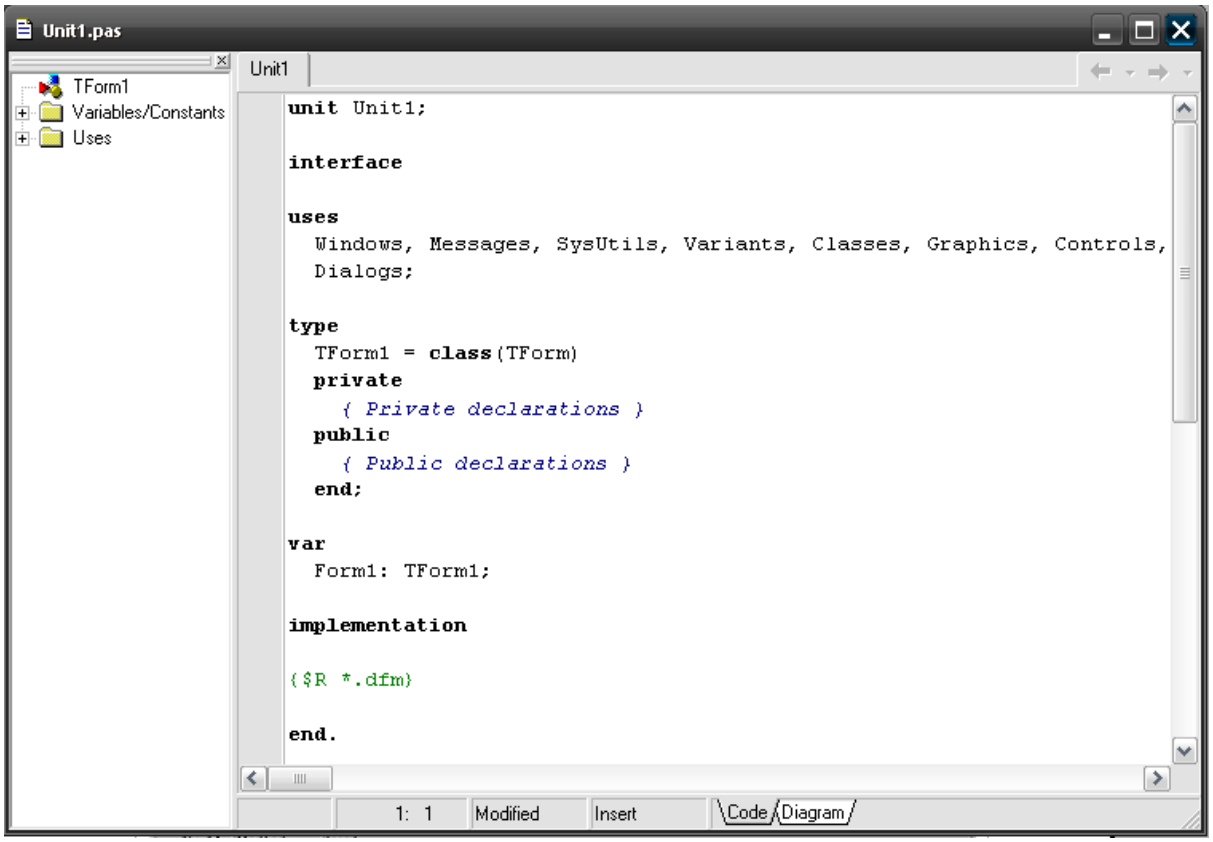

Рис. 1.5. Окно редактора кода программы

Теперь усложним нашу программу. Сделаем так, чтобы в окне нашей формы находилась кнопка, при нажатии на которую будет появляться окно и выводить указанное нами сообщение.

Для того чтобы добавить на форму кнопку следует на палитре компонентов перейти на вкладку «стандартная» (рис. 1.6) и щелкнуть один раз левой кнопкой мышки на пиктограмму ОК.

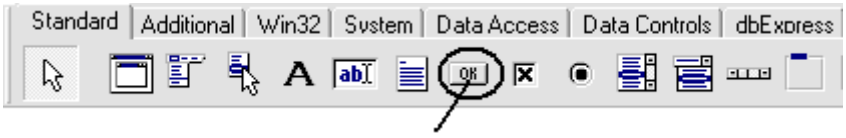

Рис. 1.6. Добавление кнопки на форму

Затем щелкните в любом месте формы. На вашей форме появится кнопка **Button1**. Как вы это делали для формы, так теперь для кнопки в инспекторе объектов измените ее размеры и название. Изменим размеры формы и сделаем это так, как показано на рис.1.7.

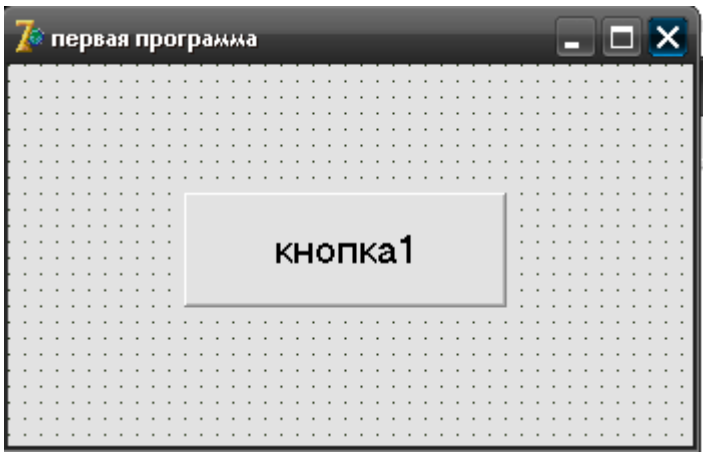

Рис. 1.7. Форма приложения

Теперь напишем для кнопки код, который будет выполняться при нажатии на нее. Для этого дважды щелкните левой кнопкой мышки по кнопке, расположенной на форме. В результате этого вы окажетесь в окне редактора кода, где будет создана пустая процедура (рис. 1.8).

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

end;
```

Рис. 1.8. Редактор кода с пустой процедурой, которая вызывается при нажатии на кнопку

Теперь между строчками **begin** и **end;** напишем код, который будет выводить в новом окне заданное нами сообщение. Для этого между указанными строками кода добавьте: «**ShowMessage('Наше сообщение')»** (рис. 1.9).

Здесь команда ShowMessage – выводит на экран новое окно, в котором отображается текст, написанный в скобах между кавычками.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
ShowMessage('Наше сообщение');
end;
```

Рис. 1.9. Редактор кода с процедурой,
которая вызывается при нажатии на кнопку

1.3. Содержание отчета

Студенты, выполняющие данную лабораторную работу, должны получить навыки работы в среде визуального проектирования Delphi версии 7.0, изучить основные приемы работы и горячие клавиши. Научиться создавать новый проект, открывать сохраненный проект. Знать, что означают понятия: окно формы, редактор кода, инспектор объектов, палитра компонентов. Уметь компилировать и запускать на выполнение разрабатываемую в Delphi программу, уметь прерывать работающую или завершившуюся в нештатном режиме программу и т. д.

1.4. Контрольные вопросы

1. Что такое Delphi?
2. Что такое Object Pascal?
3. Для каких операционных систем можно разрабатывать программы в Delphi?
4. Что такое инспектор объектов и зачем он нужен?
5. Что такое окно формы?
6. Что такое редактор кода?
7. Как назначать события для различных элементов формы?
8. Какие из элементов палитры компонентов вы знаете?

Лабораторная работа № 2
РАЗРАБОТКА ПРОГРАММЫ В DELPHI ДЛЯ РЕШЕНИЯ КВАДРАТНОГО УРАВНЕНИЯ

Цель работы: разработать программу для решения квадратного уравнения в среде визуального программирования Delphi 7.0.

Оборудование: дисплейный класс, интегрированная среда Delphi версии 7.0 или выше.

2.1. Общие положения

Текст программы пишется в окне редактора кода Unit1.pas для кнопки «решить уравнение» (рис. 2.1) и выполняется при непосредственном нажатии на неё. Код программы имеет синтаксис и структуру языка Pascal, подробно описанные в первой части методических указаний.

Ввод и вывод данных в Delphi

Ввод данных. Программа может получить исходные данные из окна ввода, поля ввода, диалогового окна или из файла.

Окно ввода – это стандартное диалоговое окно, которое появляется на экране в результате вызова функции InputBox. Значение функции InputBox – строка, которую ввел пользователь.

В общем виде инструкция ввода данных с использованием функции InputBox выглядит так:

Переменная:= InputBox (*Заголовок, Подсказка, Значение),*

где *Переменная* – переменная строкового типа, значение которой должно быть получено от пользователя; *Заголовок* – текст заголовка окна ввода; *Подсказка* – текст поясняющего сообщения; *Значение* – текст, который будет находиться в поле ввода, когда окно ввода появится на экране.

Ниже в качестве примера приведена инструкция, используя которую можно получить исходные данные для программы «Решение квадратного уравнения».

```
s:=InputBox('Решение квадратного уравнения', 'Введите значение A, '0');
```

Поле ввода – это компонент Edit. Ввод данных из поля ввода осуществляется обращением к свойству Text.

Инструкция ввода данных в этом случае будет иметь вид:

A:= StrToFloat(Editl.Text);

Вывод данных:

Наиболее просто программа может вывести результат своей работы в окно сообщения или в поле вывода (компонент Label) диалогового окна.

Окна сообщений используются для привлечения внимания пользователя. При помощи окна сообщения программа может, к примеру, проинформировать об ошибке в исходных данных или запросить подтверждение выполнения необратимой операции, например, удаления файла.

Вывести на экран окно с сообщением можно при помощи процедуры: ShowMessage или функции MessageDlg. Процедура ShowMessage выводит на экран окно с текстом и командной кнопкой **ОК.** В общем виде инструкция вызова процедуры ShowMessage выглядит так:

ShowMessage(Сообщение), где сообщение – текст, который будет выведен в окне.

Например: ShowMessage ('Уравнение не имеет корней').

Функция MessageDig более универсальная. Она позволяет поместить в окно с сообщением один из стандартных значков, например "Внимание", задать количество и тип командных кнопок и определить, какую из кнопок нажал пользователь.

Часть диалогового окна, предназначенная для вывода информации, называется полем вывода, или полем метки. Поле вывода – это компонент Label.

Содержимое поля вывода определяется значением свойства Caption. Изменить значение свойства Caption, как и большинства свойств других компонентов, можно как во время разработки формы приложения, так и во время работы программы.

Для того чтобы во время работы программы изменить содержимое поля вывода, например вывести в поле результат работы программы, нужно присвоить свойству новое значение.

Свойство Caption символьного типа. Поэтому для того, чтобы во время работы программы вывести в поле метки числовое значение, нужно преобразовать число в строку (см. функции преобразования типов) при помощи функции FloatToStr или IntToStr.

Label2.Caption:= ‘Корни уравнения:’ + #13+ – переход на новую строку

‘x1=’ +FloatToStr(x1)+……

Задать цвет поля вывода можно: Label2.color:= ClRed – красный цвет.

Цвет шрифта: Label2.font.color:= ClBlack – чёрный цвет.

Функции преобразования типов

Функции преобразования наиболее часто используются в инструкциях, обеспечивающих ввод и вывод информации. Например, для того чтобы вывести в поле вывода (компонент Label) диалогового окна значение переменной типа real, необходимо преобразовать число в строку символов, изображающую данное число. Это можно сделать при помощи функции FloatToStr, которая возвращает строковое представление значения выражения, указанного в качестве параметра функции.

Например, инструкция Label1.caption:= FloatTostr(x) выводит значение переменной x в поле Label1.

В табл. 2.1 перечислены наиболее часто используемые функции преобразования типов.

Таблица 2.1

Функции	Значение
Chr(n)	Символ ANSI с номером n
IntToStr(n)	Строка, являющаяся изображением значения целого n
FloatToStr(n)	Строка, являющаяся изображением значения вещественного n
FloatToStrF(n,f,l,m)	Строка, являющаяся изображением значения вещественного n. При вызове функции указываются: f – формат (способ изображения); l – точность (нужное общее количество цифр); m – количество цифр после десятичной точки
StrToInt (s)	Целое, изображением которого является строка s
StrToFloat (s)	Вещественное, изображением которого является строка s.

2.2. Выполнение работы

В данной лабораторной работе требуется разработать программу под Windows для решения квадратного уравнения вида $Ax^2+Bx+C=0$.

Для решения поставленной задачи нужно выполнить два этапа:

1) разработать интерфейс будущей программы;
2) написать код для решения поставленной задачи.

При разработке интерфейса программы сделать его похожим, на интерфейс, который приведен на рис. 2.1.

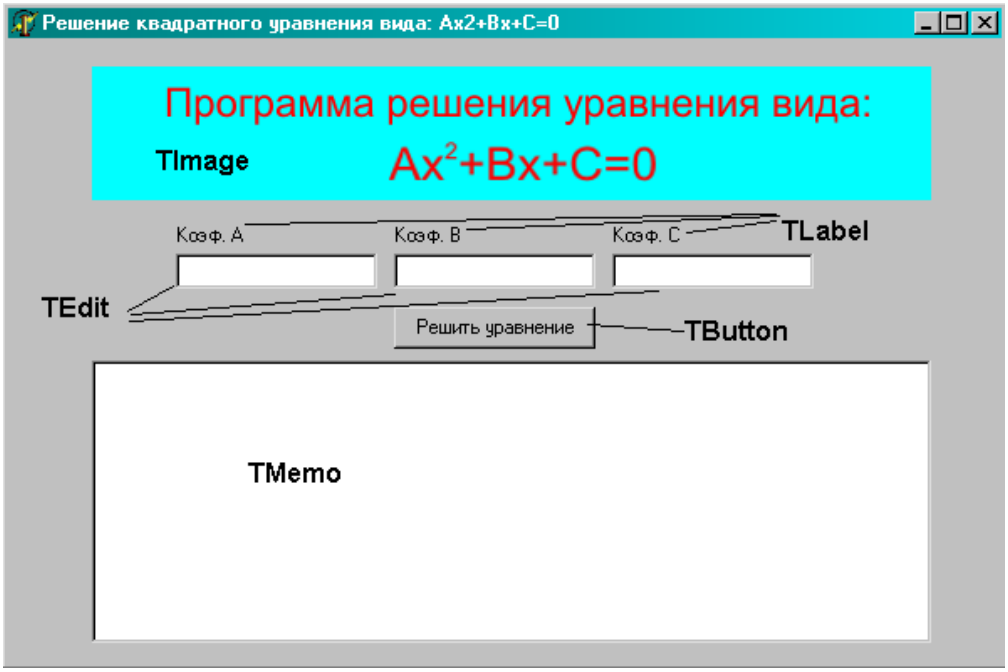

Рис. 2.1. Интерфейс формы разрабатываемого приложения

В элементы ввода TEdit пользователь вводит значения коэффициентов уравнения A, B и C.

При нажатии на кнопку TButton происходит решение уравнения, и результат выдается в компонент TMemo.

На экране формы должны присутствовать пояснения, которые указывают пользователю, для чего нужен тот или иной компонент формы (TLabel).

Также на форме присутствует компонент TImage, в который вставлен рисунок, дающий пояснение о назначении данной программы.

2.3. Содержание отчета

Отчет оформляется каждым студентом самостоятельно. Защита происходит в начале каждого следующего занятия с демонстрацией работы программы на ЭВМ. Студент, не подготовивший или не защитивший отчет по работе, к следующей лабораторной работе не допускается.

Содержание отчета:

a) титульный лист;
b) цель работы;
c) задание на работу и исходные данные;
d) разработка интерфейса пользователя (описание всех компонентов используемых на форме и их назначения, рисунок формы);
e) исходный код тех процедур и событий, которые непосредственно решают поставленную задачу;
f) выводы по работе.

2.4. Контрольные вопросы

1. Перечислите те компоненты Delphi, которые вы использовали при разработке данного приложения.

2. При возникновении какого события выполняется код программы, решающий поставленную задачу?

3. Как сделать так, чтобы пользователь в компоненты ввода коэффициентов уравнения TEdit мог вводить только числа и не мог вводить остальные символы?

4. Перечислите способы ввода и вывода данных в программу. Напишите соответствующие инструкции.

5. Что такое функции преобразования типов и для чего они используются? Перечислите все известные вам.

Лабораторная работа № 3
GDI – ГРАФИКА В DELPHI

Цель работы: изучение базовых графических возможностей Delphi.

Оборудование: дисплейный класс, интегрированная среда Delphi версии 7.0 или выше.

3.1. Общие сведения

GDI расшифровывается как Graphics Device Interface, и представляет собой интерфейс, который Windows использует для рисования 2D графики. Также это самый медленный способ отображения графики из существующих, однако самый простой для понимания основ. Итак, для начала поговорим об основных понятиях и терминах в GDI.

Начнём с того, что GDI обычно не используют для создания сложных графических эффектов, для этого есть DirectX, OpenGL или любые графические библиотеки (такие как: DelphiX, FastLib, DIBUltra, Graphics32 и т. д.). Однако, для создания простых эффектов с минимальными усилиями, возможно использование GDI.

GDI включает в себя еще один интерфейс – DC ("Device Context" – контекст устройства). Это то, на чём мы рисуем, и в Delphi контекст устройства представлен как TCanvas. Идея контекста устройства заключается в том, что это универсальное устройство вывода, поэтому можно использовать одинаковые функции как для экрана, так и для принтера.

Все графические функции в Delphi являются надстройками над стандартными GDI функциями Windows. Позже мы поговорим об этих функциях. А теперь самое время приступить к рассмотрению GDI. Ниже представлены некоторые важные классы.

Pen. Перо используется для рисования простых линий. Обычно применяется для функции LineTo или при рисовании рамки для определённой фигуры (например, для функции Rectangle).

Brush. Кисть используется для заполнения области определённым цветом. Применяется в функциях Rectangle, FillRect или FloodFill.

Font. Используется для задания шрифта, которым будет нарисован текст. Можно указать имя шрифта, размер и т. д.

Region. Позволяет задать замкнутое пространство, которым может быть круг, квадрат или произвольная фигура. Позволяет также делать полости в фигурах.

Рисование линий

Необходимо чётко уяснить, что координата (0,0) – это верхний левый угол экрана. То есть значения по оси *y* увеличиваются вниз эк-

рана. Соответственно, координата (0, 50) означает, что мы просто отступили на 50 пикселей от верха экрана.

Самое главное, что надо знать при рисовании линий и фигур, это различие между пером (Pen) и кистью (Brush). Всё очень просто: перо (Pen) используется при рисовании линий или рамок, а кисть (Brush) – для заполнения фигуры.

Ниже приведены две функции, которые используются для рисования линии, и обе принадлежат TCanvas:

MoveTo. Перемещает точку начала рисования линии в указанные координаты x и y: Canvas.MoveTo (50, 100);

LineTo. Рисует линию, начиная с текущей позиции (см. MoveTo) до указанных координат x и y: Canvas.LineTo (50, 100);

Эффект перемещения точки начала рисования линии также достигается при помощи установки свойства PenPos в «канвасе», например: Canvas.PenPos.x:=20;","Canvas.PenPos.y:=50", или Canvas.PenPos:= Point(20,50)".

По умолчанию точка начала рисования установлена в (0,0), то есть, если сразу вызвать "Canvas.LineTo (100,100);", то будет нарисована линия из точки (0,0) в точку (100, 100). Точка начала рисования автоматически переместится в (100, 100), то есть, если выполнить команду "Canvas.LineTo(200, 100);", то следующая линия будет нарисована из точки (100, 100) в (200, 100). Поэтому, если мы хотим рисовать линии несоединённые друг с другом, то придётся воспользоваться методом MoveTo.

Линия, нарисованная при помощи LineTo, использует текущее перо «канваса» (типа TPen). Основные свойства пера – это ширина "Canvas.Pen.Width:= 4;" (при помощи которого можно задавать различную ширину линий) и цвет "Canvas.Pen.Color:=clLime;".

Взглянем на простой пример беспорядочного рисования разноцветных линий:

```
procedure TForm1.N1Click(Sender: TObject);
const  NUM_LINES = 300;
var
  i: Integer;
begin
  for i := 0 to NUM_LINES - 1 do
  begin
```

```
    Image1.Canvas.Pen.Color:=RGB (Random(256), Random(256),
  Random(256));
    Image1.Canvas.LineTo (Random(ClientWidth), Random (Clien-
tHeight));
   end;
  end;
```

Процедура N1Click вызывается из обработчика кнопки OnClick. Количество линий задаётся в константе NUM_LINES. Функция RGB составляет цвет каждой линии из трёх основных составляющих: красного, зелёного и синего (значения от 0 до 255) и возвращает нам цвет в виде TColor.

Рисование фигур

Для рисования фигур в TCanvas предусмотрены следующие функции:

Ellipse. Рисует эллипс, вписанный в невидимый квадрат с координатами верхнего левого угла и правого нижнего. Если координаты *x* и *y* у углов будут совпадать, то получится круг. Canvas.Ellipse (0,0,50,50);

FillRect. Заполняет прямоугольник цветом текущей кисти (brush), но никак не за его пределами. Canvas.FillRect (Bounds(0,0,100,100));

FloodFill. Заполняет данную область цветом текущей кисти, до тех пор, пока не будет достигнут край. Canvas.FloodFill(10, 10, clBlack, fsBorder);

Rectangle. Рисует прямоугольник (или квадрат), заполненный цветом текущей кисти и обрамлённый цветом текущего пера. Canvas.Rectangle(Bounds(20, 20, 50, 50));

RoundRect. То же, что и Rectangle, но с закруглёнными углами. Canvas.RoundRect(20, 20, 50, 50, 3, 3);

Ниже представлен пример, который рисует случайным образом различные фигуры:

```
procedure TForm1.N2Click(Sender: TObject);
const   NUM_SHAPES = 200;
var i, ShapeLeft, ShapeTop: Integer;
begin
  for i := 0 to NUM_SHAPES - 1 do
```

```
  begin
    Image2.Canvas.Brush.Color:= RGB (Random(256),Random(256),
Random (256));
    ShapeLeft := Random(ClientWidth);
    ShapeTop := Random(ClientHeight);
    // теперь, случайным образом, решаем что рисовать
    case Random(2) of
      0: Image2.Canvas.Rectangle(ShapeLeft,
         ShapeTop,
         ShapeLeft + Random(50),
         ShapeTop + Random(50));
      1: Image2.Canvas.Ellipse(ShapeLeft,
         ShapeTop,
         ShapeLeft + Random(50),
         ShapeTop + Random(50));

    end;
  end;
end;
```

Как вы уже успели заметить, некоторые фигурки имеют цвет рамки, отличающийся от того цвета, которым заполнена фигура. Это как раз тот момент, когда кистью мы заполняем объекты, а пером обрамляем. Если цвет кисти (brush) меняется случайным образом, то цвет пера (pen) остаётся постоянным. Из-за этого и получается такая картина.

Печать текста

Функция **TextOut** позволяет рисовать текст, используя шрифт, заданный в «канвасе»:

TextOut. Рисует данную строку на «канвасе», начиная с координат (x,y), фон текста заполняется текущим цветом кисти. Canvas.TextOut(x, y, ‘СОЧИ-2014!');

Функция позволяет рисовать текст, не заполняя его фон. Если вам необходимо изменить шрифт, используемый в TextOut, то необходимо изменить свойство Font «канваса» (это свойство имеет тип TFont), например: "Canvas.Font.Name:='Verdana';", "Canvas.Font.Size:=24;" или "Canvas.Font.Color:=clRed;".

Очистка «канваса» TImage

Для очистки «канваса» TImage можно использовать следующий код:

```
with form1 do
Image1.Canvas.Brush.Color:=ClWhite;
Image1.Canvas.FillRect(Rect(0,0,ClientWidth, ClientHeight));
```

Данный код закрашивает «канвас» (Image1(2,3)) белым цветом.

3.2. Выполнение работы

Создайте в Delphi программу, которая случайным образом рисует линии разного цвета, геометрические фигуры и текст разного цвета и размера. Окно разработанной программы должно иметь вид, представленный на рис. 3.1.

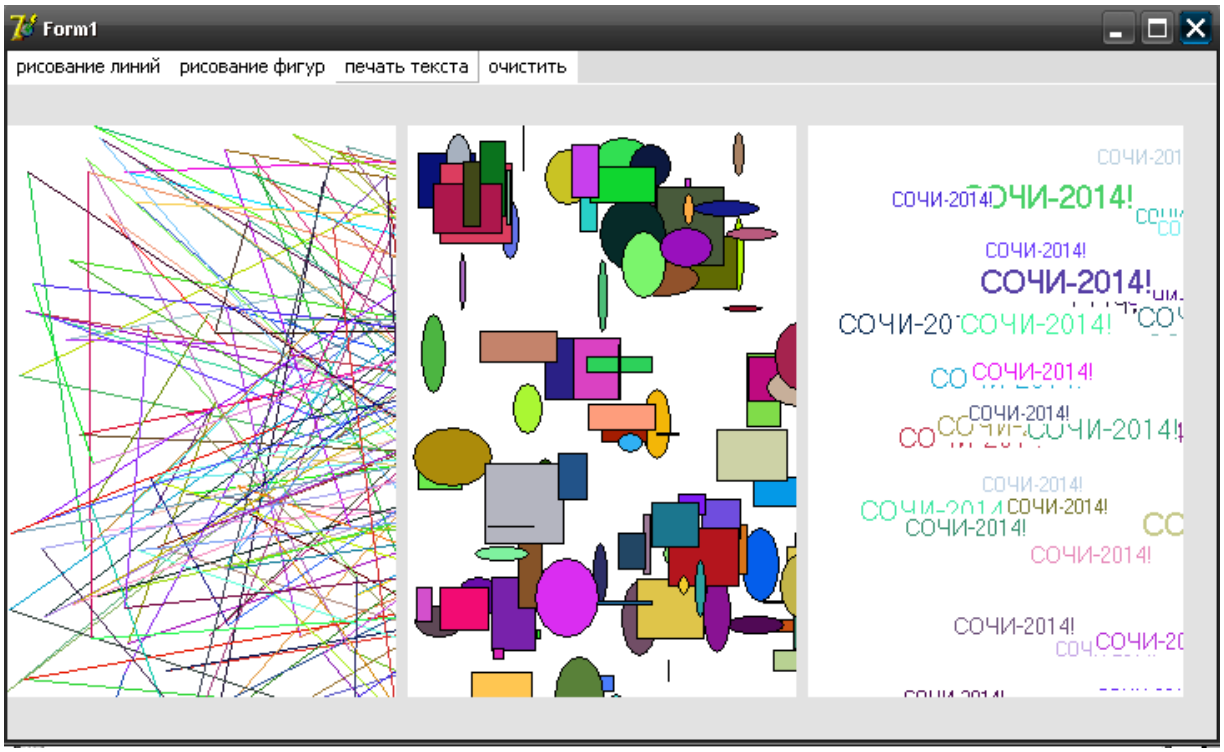

Рис. 3.1. Окно разработанной программы

Линии, геометрические фигуры и текст необходимо рисовать на «канвасе» TImage. Программа должна реагировать на событие onclick соответствующего пункта меню. Добавьте компонент

TMainMenu на форму разрабатываемого приложения, озаглавьте пункты меню как показано на рис. 3.2

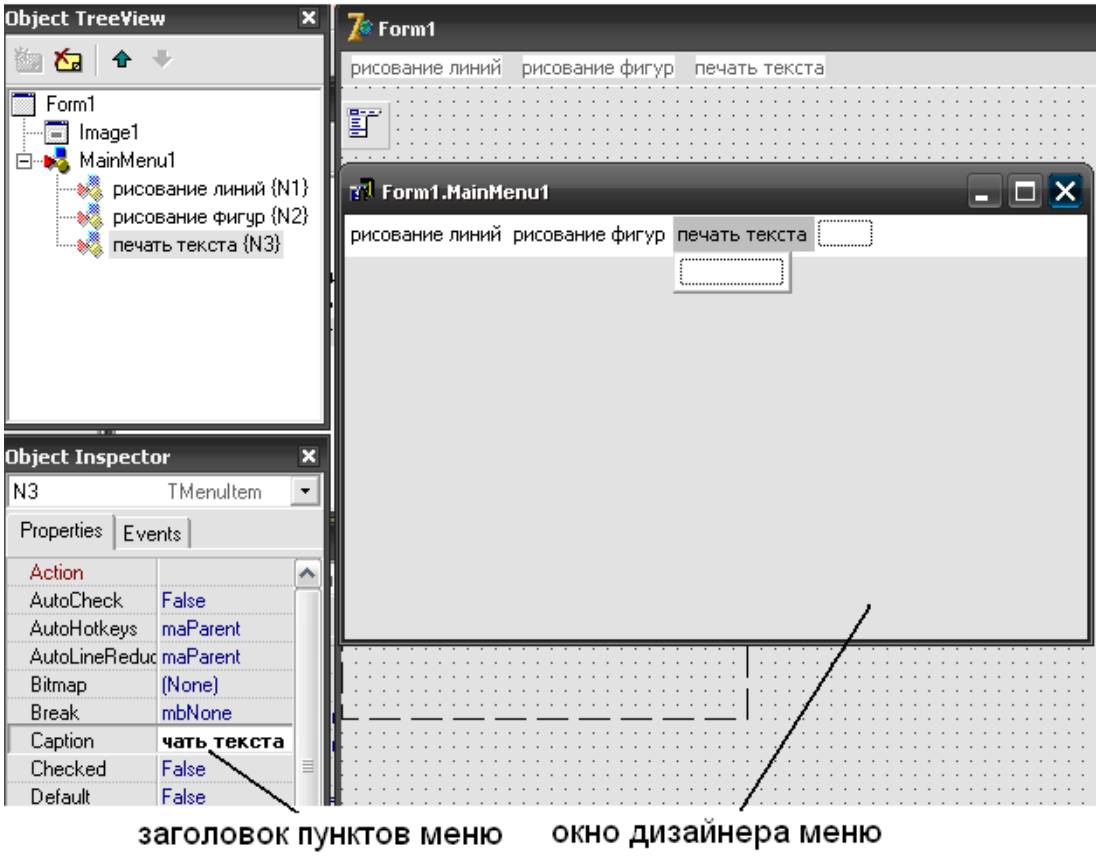

Рис. 3.2. Создание главного меню

В данном проекте обязательно используйте изученные выше функции.

3.3. Содержание отчета

Отчет оформляется каждым студентом самостоятельно. Защита происходит в начале каждого следующего занятия с демонстрацией работы программы на ЭВМ. Студент, не подготовивший или не защитивший отчет по работе, к следующей лабораторной работе не допускается.

Содержание отчета:

a) титульный лист;
b) цель работы;
c) задание на работу и исходные данные;

d) описание всех используемых графических функций;
e) исходный код программы;
f) выводы по работе.

3.4. Контрольные вопросы

1. Какие графические функции вы изучили в данной работе?
2. Что такое кисть, перо, зачем они нужны и каковы их отличия друг от друга?
3. С помощью каких свойств можно определить толщину и цвет вычерченной линии?
4. Напишите и расшифруйте код для беспорядочного рисования различных геометрических фигур.
5. Напишите инструкцию для вывода на поверхность графического объекта текста. Как изменить его цвет и шрифт?
6. Каким образом создаётся на форме главное меню?

Лабораторная работа № 4
СТАНДАРТНЫЕ ДИАЛОГОВЫЕ ОКНА

Цель работы: изучение базовых диалоговых компонентов TOpenDialog и TSaveDialog.

Оборудование: дисплейный класс, интегрированная среда Delphi версии 7.0 или выше.

4.1. Общие сведения

В Windows, начиная с версии 3.1, появилась библиотека стандартных диалоговых окон COMMDLG.DLL, которая позволила заменить вызовом функций программирование рутинных операций в типовых случаях взаимодействия с пользователем. К этим случаям относятся выбор имени файла для чтения и записи, выбор цвета или шрифта, поиск и замена текста, настройка принтера и параметров печати. В Delphi предусмотрены компоненты, представляющие собой оболочку для этих стандартных диалогов.

Стандартные диалоговые окошки можно найти на панели компонентов в закладке Dialogs. Компоненты стандартных диалогов являются невидимыми, поэтому невозможно изменить дизайн такого диалога во время разработки приложения. Поместив компоненты-диалоги на форму, вы освобождаетесь от необходимости вызывать их конструкторы и деструкторы. Все использование этих компонентов укладывается в трехэтапную схему:

1. Настройка параметров диалога, во время которой устанавливаются те или иные возможности. У всех диалогов для этого предусмотрено свойство Options, но у некоторых есть и дополнительные свойства.

2. Вызов метода Execute, который показывает диалоговое окно на экране и инициирует взаимодействие с пользователем. Execute является функцией: он возвращает True, если пользователь подтвердил ввод значения (т. е. нажал кнопку ОК на экране или клавишу <Enter> на клавиатуре), и False, если он отказался от выбора (т. е. нажал кнопку Cancel или клавишу <Esc>).

3. В случае положительного ответа – чтение установленных значений в представляющем их свойстве (свойствах).

TOpenDialog и TSaveDialog

Диалоговые окошки File Open и File Save имеют несколько общих свойств. File Open в основном используется для выбора и открытия файлов, в то время, как диалог File Save (также используется как диалоговое окошко Save As) используется для получения от пользователя имени файла, чтобы сохранить файл. Далее будут рассмотрены некоторые важные свойства TOpenDialog и TSaveDialog.

У двух диалогов имеется большой набор опций. Часть из них является общей, часть – играет роль только для одного из диалогов:

```
Property Options: TOpenOptions;
TOpenOption = (ofReadOnly, ofOverwritePrompt, ofHideReadOnly, ofNoChangeDir, ofShowHelp, ofNoValidate, ofAllowMuItiSelect, ofExtensionDifferent, ofPathMustExist, ofFileMustExist, ofCreatePrompt, ofShareAware, ofNoReadOnlyReturn, ofNoTestFileCreate);
TOpenOptions = set of TOpenOption;
```

Три опции отвечают за работу с файлами со статусом "только для чтения":

ofReadOnly – делает флажок "Read only" помеченным при появлении;
ofHideReadOnly – прячет этот флажок в появляющемся диалоге;

ofNoReadOnlyReturn – запрещает выбор файлов "только для чтения", извещая о необходимости выбрать другой файл при нажатии ОК.

Опции ограничивающие ввод имен для новых (несуществующих) файлов:

ofPathMustExist – указывает на то, что файл может находиться только в одном из существующих каталогов. В случае ввода несуществующего пути к файлу пользователь извещается об ошибке;

ofFileMustExist – аналогичным образом указывает на то, что может быть выбран только один из существующих файлов;

ofCreatePrompt – опция устанавливает реакцию на предыдущую ситуацию. Если она установлена, то вместо сообщения об ошибке выводится запрос на создание нового файла.

ofOverwritePrompt – запрашивает подтверждение, если пользователь выбрал для записи уже существующий файл;

ofNoChangeDir – запрещает изменение начального каталога, с которым диалог будет проинициализирован. Если она установлена, диалог каждый раз появляется с тем каталогом, который был установлен при первом запуске;

ofShowHelp – включает в состав диалога кнопку Help;

ofNoValidate – выключает проверку введенного имени файла на наличие в нем недопустимых символов;

ofAllowMultiSelect – позволяет выбирать несколько файлов одновременно;

ofShareAware – отключает проверку на возможность совместного доступа к выбранному файлу. В случае отсутствия этой опции файл нельзя выбрать, если он открыт другим приложением;

ofNoTestFileCreate – эта опция применяется только для файлов на тех узлах локальной сети, которым разрешено создание, но не модификация файлов. Если она установлена, диалог не проверяет возможность записи на выбранном устройстве.

Наконец, одна опция – ofExtensionDifTereiit – является выходной. Она устанавливается после завершения диалога в том случае, если расширение у введенного имени файла отличается от того, которое определено по умолчанию (в свойстве DefaultExt). Например, при помощи следующего кода:

```
with OpenDialog1 do
 Options := Options +[ofAllowMultiSelect, ofFileMustExist];
```

мы позволим пользователю выбирать несколько файлов, а также заставим генерироваться сообщение об ошибке, если пользователь выберет несуществующий файл.

Свойство **InitialDir** используется для указания директории, которая будет показана при создании диалога. Следующий код установит начальную директорию, из которой было запущено приложение:

```
SaveDialog1.InitialDir:=ExtractFilePath(Application.ExeName);
```

Свойство **Filter** содержит список типов файлов, которые сможет выбирать пользователь. Когда пользователь выберет тип файлов, то в диалоговом окне будут отображаться только файлы данного расширения. Фильтр можно легко установить на стадии создания приложения при помощи диалога редактора фильтра (рис. 4.1).

Также фильтр можно задать программно. Строка фильтра должна содержать описание и расширение для данного типа файлов, разделённые вертикальной чертой:

```
OpenDialog1.Filter:= 'Text files (*.txt)|*.txt|All files (*.*)|*.*';
```

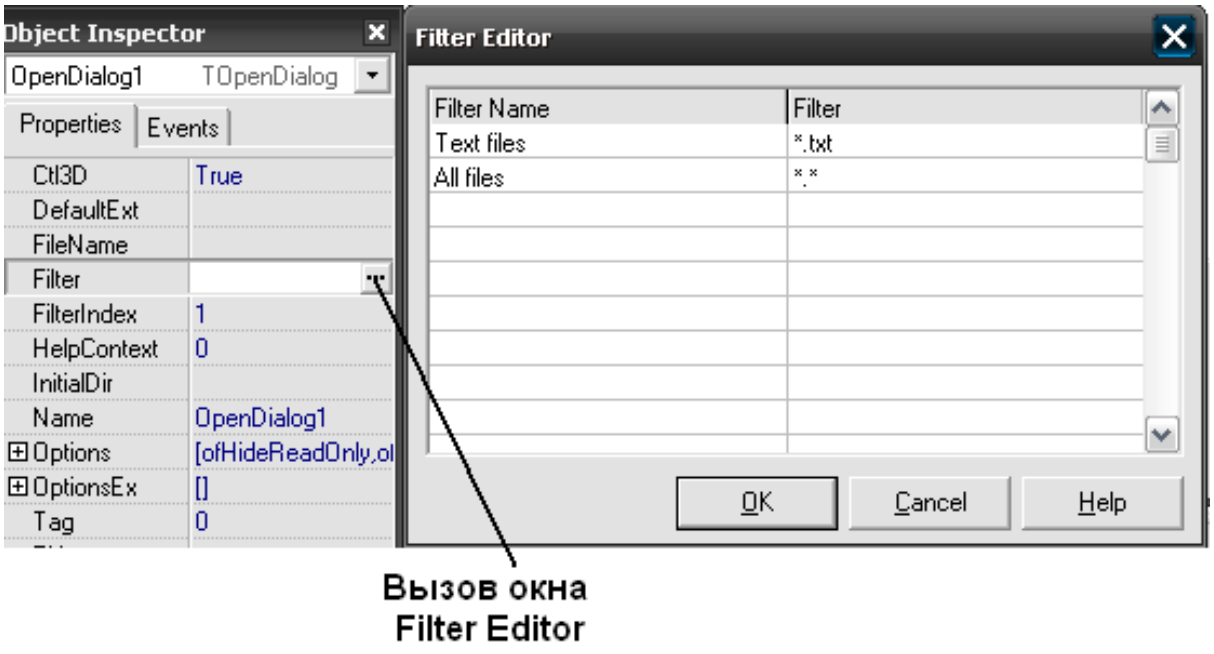

Рис. 4.1. Установка свойства Filter

Свойство **FileName**. Когда пользователь нажмёт на диалоге кнопку OK, то это свойство будет содержать полный путь и имя выбранного файла.

Вызов диалогового окошка

Для создания и отображения стандартного диалога необходимо выполнить метод Execute для нужного диалога. За исключением диалогов **TFindDialog** и **TReplaceDialog**, все остальные диалоги отображаются модально.

Все стандартные диалоговые окошки позволяют определить, нажал ли пользователь кнопку "Отмена" (Cancel) (или нажал ESC). Если метод Execute вернул True, значит пользователь нажал OK или сделал двойной щелчок по файлу либо нажал Enter на клавиатуре, иначе, если была нажата кнопка Cancel, клавиша Esc или Alt-F4, будет возвращено значение False.

```
if OpenDialog1.Execute then
 ShowMessage(OpenDialog1.FileName);
```

Этот код показывает диалог File Open и, если пользователь нажал "Открыть" (Open), то будет показано имя выбранного файла.

Чтобы работать диалогом Open (или любым другим), не помещая при этом на форму компонент OpenDialog, можно воспользоваться следующим кодом:

```
procedure TForm1.btnFromCodeClick(Sender: TObject);
var OpenDlg : TOpenDialog;
begin OpenDlg := TOpenDialog.Create(Self);
 {здесь устанавливаем опции...}
 if OpenDlg.Execute then begin
  Form1.Caption := OpenDialog1.FileName;
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
 end;
 OpenDlg.Free;
end;
```

Обратите внимание, что перед вызовом Execute можно установить различные свойства компонента OpenDialog.

TOpenPictureDialog и TSavePictureDialog

Эти два диалога представляют собой обычные File Open и File Save с дополнительной возможностью предварительного просмотра выбранной картинки.

4.2. Выполнение работы

Напишите программу простейшего блокнота с использованием диалоговых окошек Open и Save.

Для создания блокнота проделайте следующее:

- Запустите Delphi и выберите в меню File-New Application.
- Поместите на форму компоненты Memo, OpenDialog, SaveDialog и две кнопки.
- Для компонентов OpenDialog и SaveDialog настройте необходимые опции и свойства (InitialDir, Filter).
- Переименуйте командные кнопки Button1 в OpenFiles, а Button2 в SaveFiles.
- Для следующих командных кнопок напишите текст процедуры обработки события onclick, открывающих и сохраняющих текстовый файл соответственно.
- Поместите полосу прокрутки на компонент Memo можно с помощью следующей инструкции:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 Memo1.ScrollBars:=ssBoth;
end;
```

4.3. Содержание отчёта

Отчет оформляется каждым студентом самостоятельно. Защита происходит в начале каждого следующего занятия с демонстрацией работы программы на ЭВМ. Студент, не подготовивший или не защитивший отчет по работе, к следующей лабораторной работе не допускается.

Содержание отчета:

a) титульный лист;
b) цель работы;
c) задание на работу и исходные данные;
d) разработка интерфейса пользователя (описание всех компонентов, используемых на форме, и их назначение, рисунок формы);
e) исходный код тех процедур и событий, которые непосредственно решают поставленную задачу;
f) выводы по работе.

4.4. Контрольные вопросы

1. Что такое стандартные диалоговые окна?
2. Назовите основные опции диалогов TOpenDialog и TSaveDialog.
3. Для чего используются следующие свойства (InitialDir, Filter, FileName)?

Лабораторная работа № 5
РАБОТА С ФАЙЛАМИ В СРЕДЕ DELPHI

Цель работы: получить навыки разработки программ, которые выводят результаты своей работы в файл.

Оборудование: дисплейный класс, интегрированная среда Delphi версии 7.0 или выше.

5.1. Общие сведения

Файл – это именованная структура данных, представляющих собой последовательность элементов данных одного типа, причём количество элементов последовательности практически не ограничено.

Как и любая структура данных программы (переменная, массив), файл должен быть объявлен в разделе описания переменных. При объявлении файла указывается тип его элементов.

Например:

Res: **file of** char; {файл символов}.

Файл, компонентами которого являются данные символьного типа, называется символьным или текстовым. Описание текстового файла в общем виде выглядит так:

Имя: TextFile,

где *Имя* – имя файловой переменной; *TextFile* – обозначение типа, показывающее, что *имя* – это файловая переменная, представляющая текстовый файл.

Для того чтобы программа могла выводить данные в файл или считывать данные из файла, необходимо указать конкретный файл, т. е. задать имя файла. Имя файла задаётся вызовом процедуры *AssignFile*, связывающей файловую переменную с конкретным файлом. Примеры вызова процедуры *AssignFile*:

AssignFile (f, ‘c:\students.db’);

Вывод в файл

Непосредственно вывод в текстовый файл осуществляется при помощи инструкции *write* или *writeln*, например:

Writeln (f, edit1.text, ‘ФИО студента’, edit2.text…);

где f – переменная, идентифицирующая файл в который выполняется вывод; edit1.text – означает, что в файл выведется значение, введённой в поле компонента edit1.

Перед выводом в файл его необходимо открыть. Возможны следующие варианты открытия файла для записи в него данных:

- перезапись (запись нового файла поверх существующего или создание нового файла). Инструкция: *Rewrite(f)*;
- добавление в существующий файл. Инструкция: *Append (f)*.

Ниже приведён пример программного кода, сохраняющего в файл C:\sochi.txt три строки следующего содержания: «СОЧИ-2014!!!»

```
procedure TForm1.Button1Click(Sender: TObject);
var
f: TextFile;
i: integer;
begin
AssignFile (f, 'C:\ sochi.txt');
Append (f);
for i:=1 to 3 do
writeln (f, СОЧИ-2014!!!');
Closefile (f);
end;
end.
```

Ошибки открытия файла

Попытка открыть файл может вызвать ошибку, в случае отсутствия файла (неправильно задан путь к файлу, диск защищён от записи и т. д.). Программа может взять на себя задачу контроля за результатом выполнения инструкции открытия файла. Функция IOResult (Input-Output Result – результат ввода-вывода) возвращает 0, если операция ввода-вывода завершилась успешно. В противном случае – не ноль.

Для того чтобы программа смогла проверить результат выполнения операции ввода-вывода, нужно перед инструкцией вызова процедуры открытия файла поместить директиву – строку {$I-}, которая запре-

щает автоматическую обработку ошибок ввода-вывода (эта директива сообщает компилятору, что программа берёт на себя контроль ошибок). После инструкции открытия файла следует поместить директиву {$I+}, восстанавливающую режим обработки ошибок ввода вывода.

Пример:

```
AssignFile (f, filename);
{$I-}
Append (f) // открыть для добавления
{$I+}
If IOResult <> 0 // ошибка открытия
Then Rewrite (f); // открыть для записи.
```

Перед завершением работы программа должна закрыть все открытые файлы. Инструкция: *Close (f)*.

Ввод из файла

Открытие файла для ввода (чтения) выполняется вызовом процедуры *Reset*, имеющей один параметр – файловую переменную. Перед вызовом процедуры *Reset* с помощью функции *AssignFile* файловая переменная должна быть связана с конкретным файлом.

Например, следующие инструкции открывают файл для ввода:

```
AssignFile(f, 'c:\data.txt'); Reset(f);
```

Чтение из файла выполняется при помощи инструкций readln, которая в общем виде записываются следующим образом:

readln (*ФайловаяПеременная, СписокПеременных),*

где *ФайловаяПеременная* – переменная типа TextFile; *СписокПеременных* – имена переменных, разделенные запятыми.

Например, если текстовый файл C:\data.txt содержит следующие строки:

```
23 15 45 28 56 71
```

то в результате выполнения инструкций:

```
AssignFile (f, 'C:\data.txt');
Reset(f); // открыть для чтения
read(f, a); read(f, b, c); read(f, d);
```

значения переменных будут следующими: a = 23, b = 15, c = 45, d = 28.

Чтение строк

В программе строковая переменная может быть объявлена с указанием длины или без нее. Например:

```
stroka1:string[10]; stroka2:string;
```

При чтении из файла значения строковой переменной, длина которой явно задана в ее объявлении, считывается столько символов, сколько указано в объявлении, но не больше, чем в текущей строке.

Определение конца файла

Для определения конца файла можно воспользоваться функцией EOF (End of File – конец файла). У функции EOF один параметр – файловая переменная. Значение функции EOF равно False, если прочитанный элемент данных не является последним в файле, т. е. возможно дальнейшее чтение. Если прочитанный элемент данных является последним, то значение EOF равно True.

5.2. Выполнение работы

Задание 1. Разработать программу, которая записывает в файл c:\students.db информацию о студентах. Окно разрабатываемой программы должно выглядеть, как показано на рис. 5.1.

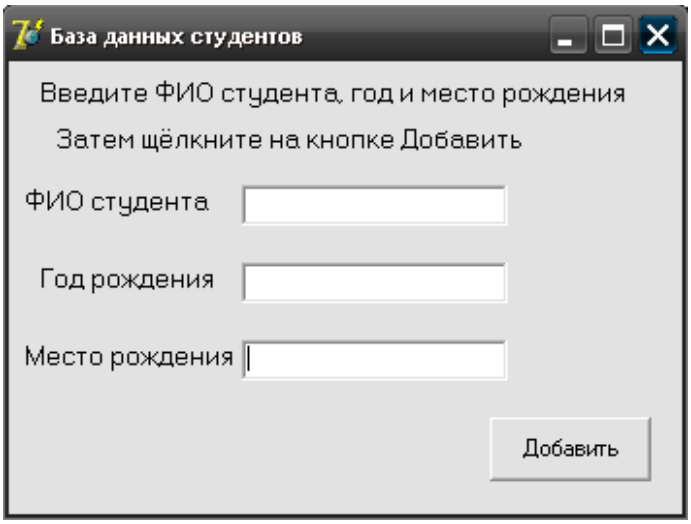

Рис. 5.1. Окно разрабатываемой программы

Примечания:

- Базу данных студентов открывает процедура TFormActivate (окно Object Inspector → закладка Events (события)→ onActivate), которая запускается автоматически при активизации формы приложения.
- Необходимо выполнить контроль результата открытия файла (IOResult=0). Если операция открытия файла завершается успешно, то процедура делает доступной кнопку «Добавить»

(Button1.Enabled:=true), в противном случае кнопка «Добавить» недоступна и выводится сообщение об ошибке создания файла.

- В процедуру Button1.Click пишем инструкции для ввода данных в файл (writeln). Если поля ввода edit не заполнены (length(edit1.text)=0), то должно выводиться сообщение об ошибке.
- Закрывает базу данных процедура TForm1.Formclose, которая обрабатывает событие onclose, возникающее при закрытии пользователем формы приложения. (Инструкция CloseFile (f)).
- Результатом выполнения программы служит создание файла данных c:\students.db, который можно открыть с помощью блокнота (notepad).

Задание 2. Разработать программу, которая читает строки из файла, имя которого ввел пользователь во время работы программы, и выводит эти строки в поле Memo. Окно разрабатываемой программы должно выглядеть, как показано на рис. 5.2.

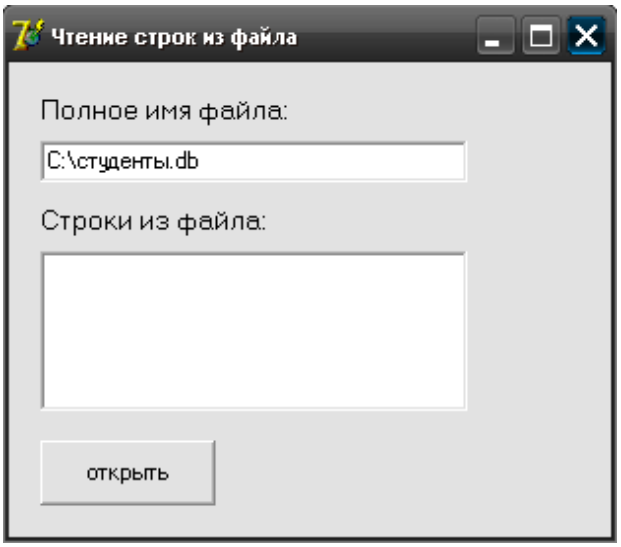

Рис. 5.2. Окно разрабатываемой программы

Примечания:

- Инструкции для чтения строк из файла пишем для события onclick компонента TButton (кнопка «Открыть»).
- Вводим строковые переменные.
- Выполняем контроль результата открытия файла: If IOResult<>0 then выводим сообщение об ошибке.

– Воспользуемся циклом While not EOF(f) do. Пока не закончится файл выполнять:

1. Прочитать строку из файла с помощью инструкции readln (f, buf).
2. Добавить строку в поле Memo1 с помощью инструкции – Memo1.Lines.Add(buf),

где f – файловая переменная, buf – строковая переменная.

5.3. Содержание отчёта

Отчет оформляется каждым студентом самостоятельно. Защита происходит в начале каждого следующего занятия с демонстрацией работы программы на ЭВМ. Студент, не подготовивший или не защитивший отчет по работе, к следующей лабораторной работе не допускается.

Содержание отчета:

a) титульный лист;
b) цель работы;
c) задание на работу и исходные данные;
d) разработка интерфейса пользователя (описание всех компонентов, используемых на форме, и их назначение, рисунок формы);
e) исходный код тех процедур и событий, которые непосредственно решают поставленную задачу;
f) выводы по работе.

5.4. Контрольные вопросы

1. Что такое файл и как он описывается? Описание текстового файла.
2. С помощью, какой процедуры можно связать файловую переменную с конкретным файлом?
3. Напишите инструкции вывода данных в файл.
4. Какими программными методами осуществляется контроль за результатом выполнения открытия файла?
5. Как осуществляется ввод данных из файла?
6. Назовите функцию определения конца файла и её параметры.

Лабораторная работа № 6
РАБОТА С БАЗАМИ ДАННЫХ В СРЕДЕ DELPHI

Цель работы: получить навыки разработки программ управления базами данных на примере СУБД Dbase.

Оборудование: дисплейный класс, интегрированная среда Delphi версии 7.0 или выше.

6.1. Общие сведения

База данных – это программа, которая обеспечивает работу с информацией. При запуске такой программы на экране, как правило, появляется таблица, просматривая которую пользователь может найти интересующие его сведения. Если система позволяет, то он может внести изменения в базу данных: добавить новую информацию или удалить ненужную.

В настоящее время существует достаточно большое количество программных систем, позволяющих создавать и использовать локальные (dBASE, FoxPro, Access, Paradox) и удаленные (Interbase, Oracle, Sysbase, Infomix, Microsoft SQL Server) базы данных.

В состав Delphi входят компоненты, позволяющие создавать программы работы с файлами данных, созданными различными системами: от dBASE до Infomix и Oracle. Delphi также позволяет программисту, используя утилиту Borland Database Desktop, создавать файлы баз данных в различных форматах.

Создание базы данных

Процесс создания базы данных может быть представлен как последовательность следующих шагов:

1) создание каталога;
2) создание псевдонима;
3) создание таблиц.

Псевдоним базы данных

В Delphi проблема передачи в программу информации о месте нахождения файлов базы данных решается путем использования *псевдонима* базы данных. Псевдоним (Alias) – это короткое имя, поставленное в соответствие реальному полному имени каталога базы данных. Например, псевдонимом каталога C:\data\students может быть

имя Students. Программа работы с базой данных для доступа к данным использует не реальное имя, а псевдоним.

Псевдоним базы данных может быть создан (зарегистрирован) при помощи утилиты BDE Administrator (рис. 6.1).

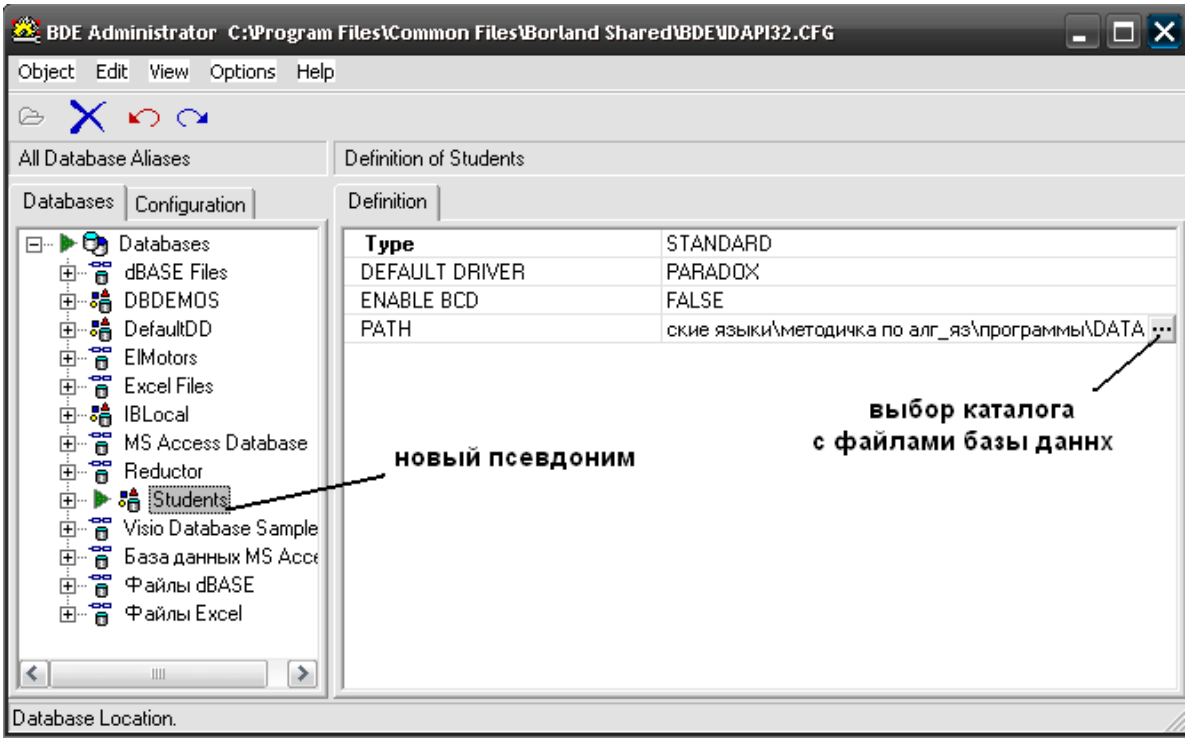

Рис. 6.1. Регистрация нового псевдонима в BDE Administrator

Эта же утилита позволяет изменить каталог, связанный с псевдонимом. Для того чтобы создать новый псевдоним, необходимо из меню **Object** выбрать команду **New**. Затем в открывшемся диалоговом окне **New Database Alias** (Новый псевдоним базы данных) из списка **Database Driver Name,** в котором перечислены зарегистрированные в системе драйверы доступа к базам данных, нужно выбрать драйвер для создаваемой базы данных, т. е. фактически выбрать тип создаваемой базы данных.

При создании псевдонима по умолчанию предлагается драйвер **STANDARD** (default driver), который обеспечивает доступ к таблицам в формате Paradox. После выбора драйвера и щелчка на кнопке ОК в список псевдонимов будет добавлен новый элемент.

После этого нужно изменить автоматически созданное администратором имя псевдонима и задать путь к файлам базы данных, для которой создается псевдоним.

Имя псевдонима можно изменить обычным для Windows способом: щелкнуть правой кнопкой мыши на имени псевдонима (на вкладке **Databases),** в появившемся контекстном меню выбрать команду **Rename** (Переименовать) и в открывшемся диалоговом окне ввести новое имя.

Путь к файлам базы данных можно ввести на вкладке **Definition** в поле **Path** с клавиатуры или воспользоваться стандартным диалоговым окном **Select Directory** (Выбор каталога), которое открывается щелчком на кнопке с тремя точками, находящейся в конце поля **Path.**

Для того чтобы созданный псевдоним был зарегистрирован в файле конфигурации (Idapi.cfg), необходимо в меню **Object** выбрать команду **Apply** (Применить). В открывшемся диалоговом окне **Confirm** следует подтвердить необходимость сохранения изменений в файле конфигурации.

Создание таблиц

После того как определена структура записей базы данных, можно приступить непосредственно к созданию таблицы. Таблицы создаются при помощи входящей в состав Delphi утилиты Database Desktop.

Утилита Database Desktop позволяет выполнять все необходимые при работе с базами данных действия. Она обеспечивает создание, просмотр и модификацию таблиц баз данных различных форматов (Paradox, dBASE, Microsoft Access). Кроме того, утилита позволяет выполнять выборку информации путем создания запросов.

Для того чтобы создать новую таблицу, нужно выбором из меню Tools команды **Database Desktop** запустить Database Desktop. Затем в появившемся окне утилиты Database Desktop надо из меню **File** выбрать команду New и в появившемся списке выбрать тип создаваемого файла – **Table**. Затем в открывшемся диалоговом окне **Create Table** следует выбрать тип создаваемой таблицы (значением по умолчанию является тип Paradox 7).

В результате открывается диалоговое окно **Create Paradox 7 Table**, в котором можно определить структуру записей таблицы (рис. 6.2).

Для каждого поля таблицы необходимо задать имя, тип и, если нужно, размер поля. Имя поля используется для доступа к данным. В качестве имени поля, которое вводится в колонку **Field Name,** можно использовать последовательность из букв латинского алфавита и

цифр длиной не более 25 символов. Тип поля определяет тип данных, которые могут быть помещены в поле. Тип задается вводом в колонку Type символьной константы. Одно или несколько полей можно пометить как *ключевые*. Ключевое поле определяет *логический* порядок следования записей в таблице. Для того чтобы пометить поле как ключевое, необходимо выполнить двойной щелчок в колонке **Key.** Следует обратить внимание на то, что ключевые поля должны быть сгруппированы в верхней части таблицы.

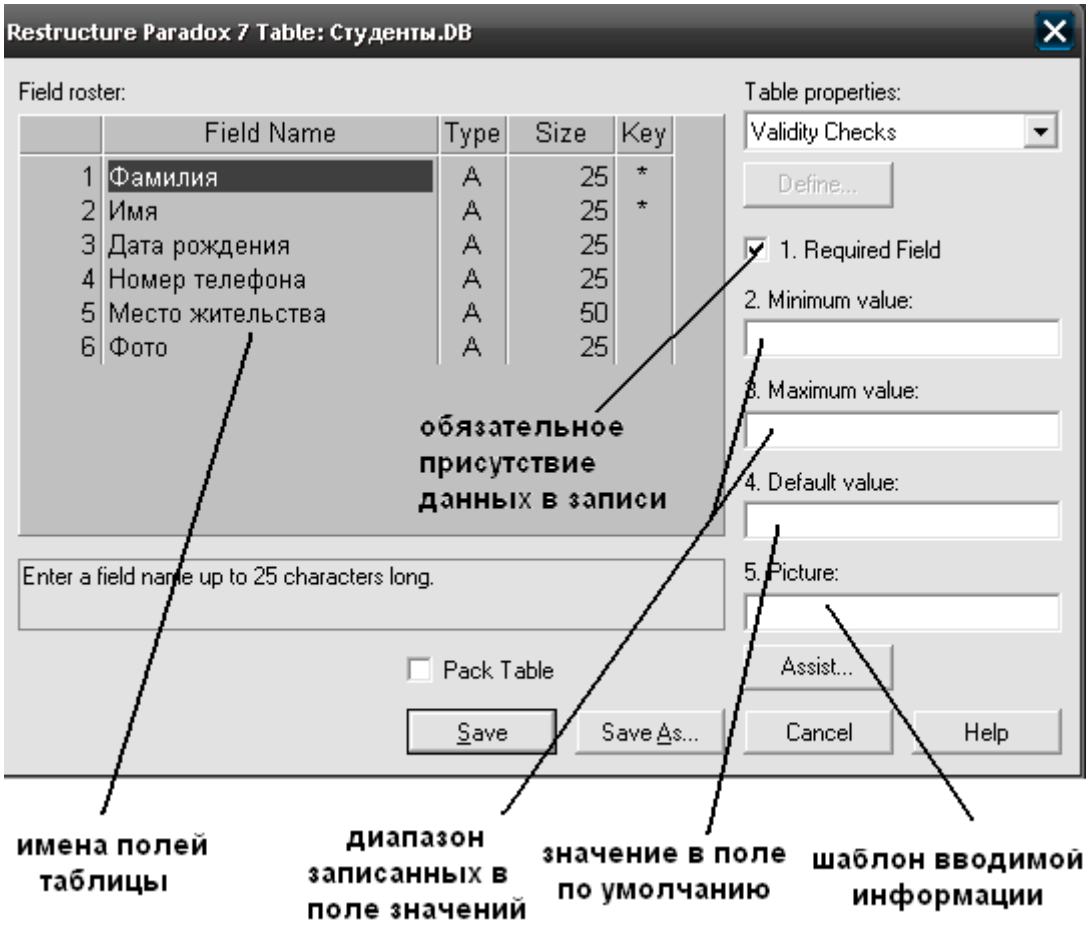

Рис. 6.2. Диалоговое окно Create Paradox 7 Table

После того как будет определена структура таблицы, таблицу следует сохранить. Для этого необходимо нажать кнопку **Save As**. В результате открывается окно **Save Table As**. В этом окне из списка **Alias** нужно выбрать псевдоним базы данных, частью которой является созданная таблица, а в поле **Имя файла** ввести имя файла, в котором нужно сохранить созданную таблицу.

Разработка программы управления базой данных

Методика разработки программы работы с базой данных ничем не отличается от методики создания обычной программы: к форме добавляются необходимые компоненты, устанавливаются значения свойств компонентов, разрабатываются необходимые процедуры обработки событий.

Приложение работы с базой данных должно содержать компоненты, обеспечивающие доступ к данным, возможность просмотра и редактирования содержимого полей. Компоненты доступа к данным находятся на вкладке **Data Access** палитры компонентов, а компоненты отображения данных – на вкладке **Data Controls.**

Доступ к базе данных (таблице)

Доступ к базе данных обеспечивают компоненты Database, Table, Query и DataSource, значки которых находятся на вкладках **Data Access** и **BDE** палитры компонентов (рис. 6.3).

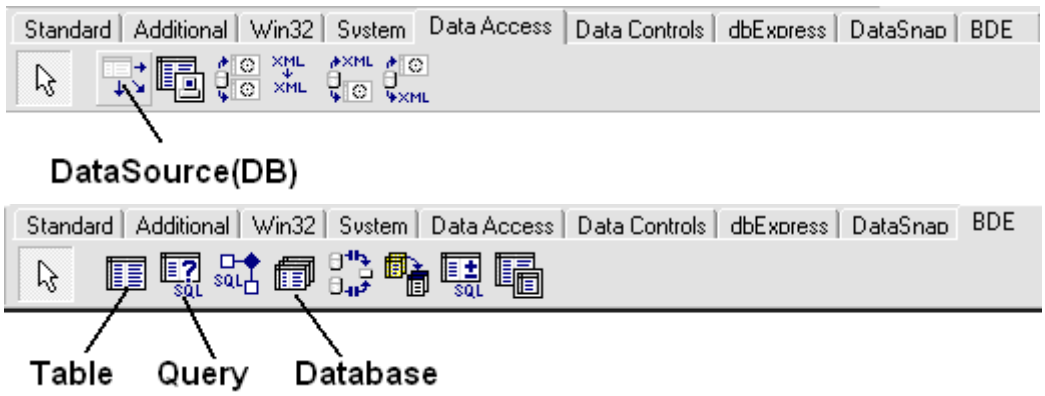

Рис. 6.3. Компоненты вкладок DataAccess и BDE для обеспечения доступа к данным

Компонент Database представляет базу данных как единое целое, т. е. совокупность таблиц, а компонент Table – одну из таблиц базы данных. Компонент DataSource (источник данных) обеспечивает связь компонента отображения-редактирования данных (например, компонента DBGrid) и источника данных, в качестве которого может выступать таблица (компонент Table) или результат выполнения SQL-запроса к таблице (компонент Query). Компонент DataSource позволяет оперативно выбирать источник данных, использовать один и тот

же компонент, например DBGrid, для отображения данных из таблицы или результата выполнения SQL-запроса к этой таблице.

Просмотр базы данных

Пользователь может просматривать базу данных в режиме формы или в режиме таблицы. В режиме формы можно видеть только одну запись, а в режиме таблицы – несколько записей одновременно. Довольно часто эти два режима комбинируют.

Компоненты, обеспечивающие просмотр и редактирование содержимого полей базы данных, находятся на вкладке **Data Controls** (рис. 6.4).

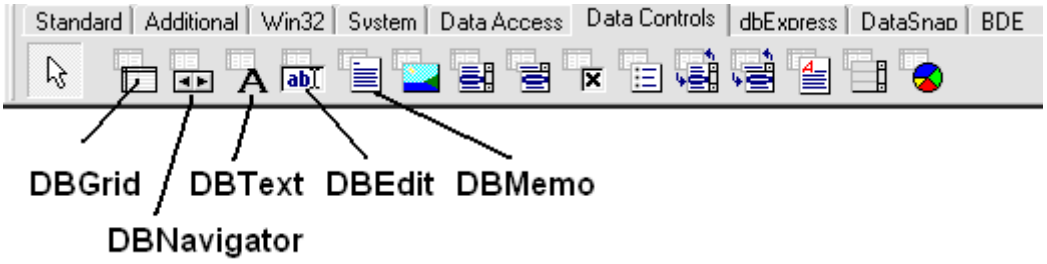

Рис. 6.4. Компоненты вкладки DataControls для просмотра и редактирования полей базы данных

Для того чтобы обеспечить просмотр базы данных в режиме формы, в форму приложения нужно добавить компоненты, обеспечивающие просмотр и, если нужно, редактирование содержимого полей записи, причем по одному компоненту для каждого поля.

Компонент DBText позволяет только просматривать содержимое поля, а компоненты DBEdit и DBMemo – просматривать и редактировать.

Для того чтобы иметь возможность просматривать другие записи файла данных, в форму приложения нужно добавить компонент DBNavigator. Компонент DBNavigator (рис. 6.4) представляет собой набор кнопок, при щелчках на которых во время работы программы происходит перемещение указателя текущей записи к следующей, предыдущей, первой или последней записи базы данных, а также добавление к файлу данных новой записи, удаление текущей записи.

Следует обратить внимание на свойство visibieButtons. Оно позволяет скрыть некоторые кнопки компонента DBNavigator и тем самым запретить выполнение соответствующих операций над фай-

лом данных. Например, присвоив значение False свойству VisibieButtons.nbDelete можно скрыть кнопку nbDelete и тем самым запретить удаление записей.

Для обеспечения просмотра и редактирования данных в режиме таблицы в форму приложения надо добавить компонент DBGrid. Свойства компонента DBGridl определяют вид таблицы и действия, которые могут быть выполнены над данными во время работы программы. Обратите внимание на то, что сначала в форму разрабатываемого приложения нужно добавить компоненты Table и DataSource, которые обеспечивают доступ к файлу данных, и установить значения их свойств.

Выбор информации из базы данных

Большинство систем управления базами данных позволяют произвести выборку нужной информации путем выполнения *запросов.* Пользователь в соответствии с определенными правилами формулирует запрос, указывая, каким критериям должна удовлетворять интересующая его информация, а система выводит записи, удовлетворяющие запросу.

Для выборки из базы данных записей, удовлетворяющих некоторому критерию, предназначен компонент Query.

Для того чтобы во время разработки программы задать, какая информация будет выделена из базы данных в результате выполнения запроса, свойство SQL должно содержать представленный на языке SQL запрос на выборку данных.

В общем виде запрос на выборку из таблицы данных выглядит так:

SELECT *СписокПолей*
FROM *Таблица*
WHERE *(Критерий)*
ORDER BY *СписокПолей,*

где SELECT – команда выбора записей из таблицы и вывода содержимого полей, имена которых указаны в списке; FROM – параметр команды, который определяет имя таблицы, из которой нужно сделать выборку; WHERE – параметр, который задает критерий выбора. В простейшем случае критерий – это инструкция проверки содержимого поля; ORDER BY – параметр, который задает условие, в соответствии с которым будут упорядочены записи, удовлетворяющие критерию запроса.

6.2. Выполнение работы

Разработать локальную базу данных, содержащую фамилию, имя, дату рождения, номер телефона и место жительства каждого студента своей группы.

Порядок выполнения работы:

1) создать и зарегистрировать псевдоним (Alias) разрабатываемой базы данных с помощью утилиты BDE Administrator;

2) создать таблицу с помощью утилиты Database Desktop;

3) разработать программу управления базой данных, обеспечивающую её просмотр в режиме формы (рис. 6.5);

4) Разработать программу, обеспечивающую просмотр базы данных в режиме таблицы (рис. 6.6);

5) Выполнить запрос, удовлетворяющий конкретной фамилии студента (компонент TQuery). Текст запроса формируется записью в свойство SQL и активизируется присвоением значения True свойству Active.

Примечание: Процедура TForm1.Button1Click запускается нажатием кнопки **«Запрос»** и принимает от пользователя строку (Фамилию).

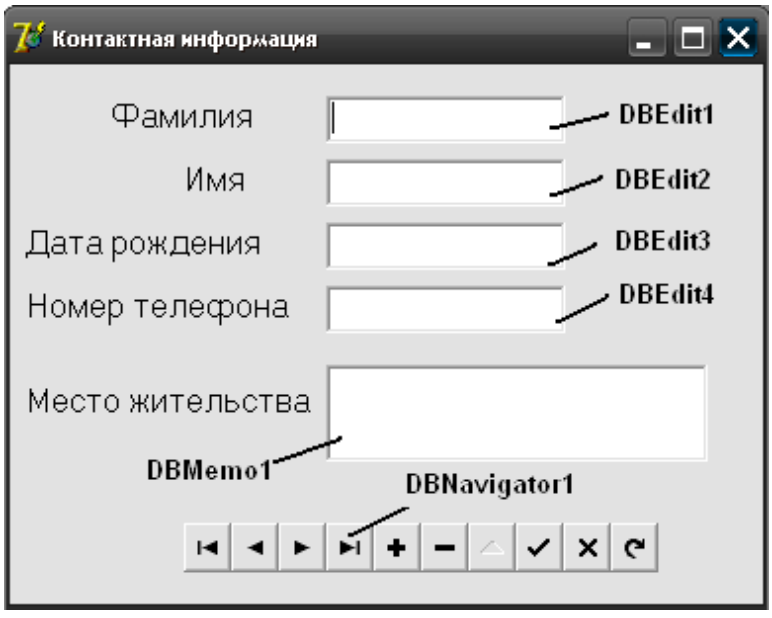

Рис. 6.5. Программа управления базой данных
с просмотром в режиме формы

Рис. 6.6. Программа управления базой данных с просмотром в режиме таблицы

6.3. Содержание отчёта

Отчет оформляется каждым студентом самостоятельно. Защита происходит в начале каждого следующего занятия с демонстрацией работы программы на ЭВМ. Студент, не подготовивший или не защитивший отчет по работе, к следующей лабораторной работе не допускается.

Содержание отчета:

a) титульный лист;
b) цель работы;
c) задание на работу и исходные данные;
d) разработка интерфейса пользователя (описание всех компонентов, используемых на форме, и их назначение, рисунок формы);
e) исходный код тех процедур и событий, которые непосредственно решают поставленную задачу;
f) выводы по работе.

6.4. Контрольные вопросы

1. Что такое база данных?
2. Опишите этапы создания базы данных в Delphi.
3. Что такое псевдоним базы данных и как его зарегистрировать?
4. Опишите процесс создания таблицы. Что такое ключевое поле таблицы?
5. Какие компоненты обеспечивают доступ к базе данных?
6. Каково назначение компонента DBNavigator?
7. Как выглядит SQL запрос на выборку информации из базы данных? Опишите его основные команды.

Приложение

Коды ошибок в TurboPascal 7.0

№ ошибки	Название
1	Out of memory (Выход за границы памяти).
2	Identifier expected (Не указан идентификатор).
3	Unknown identifier (Неизвестный идентификатор).
4	Duplicate identifier (Двойной идентификатор).
5	Syntax error (Синтаксическая ошибка).
6	Error in real constant (Ошибка в вещественной константе).
7	Error in integer constant (Ошибка в целой константе).
8	String constant exceeds line (Строковая константа превышает допустимые размеры).
9	Too many nested files (Слишком много вложенных файлов).
10	Unexpected end of file (Не найден конец файла).
11	Line too long (Слишком длинная строка).
12	Type identifier expected (Здесь нужен идентификатор типа).
13	Too many open files (Слишком много открытых файлов).
14	Invalid file name (Неверное имя файла).
15	File not found (Файл не найден).
16	Disk full (Диск заполнен).
17	Invalid compiler directive (Неправильная директива компилятора).
18	Too many files (Слишком много файлов).
19	Undefined type in pointer definition (Неопределенный тип в объявлении указателя).
20	Variable identifier expected (Отсутствует идентификатор переменной).
21	Error in type (Ошибка в объявлении типа).
22	Structure too large (Слишком большая структура).
23	Set base type of range (Базовый тип множества нарушает границы).

24 File components may not be files (Компонентами файла не могут быть файлы).
25 Invalid string length (Неверная длина строки).
26 Type mismatch (Несоответствие типов).
27 Invalid subrange base type (Неправильный базовый тип для типа-диапазона).
28 Lower bound greater than upper bound (Нижняя граница больше верхней).
29 Ordinal type expected (Нужен порядковый тип).
30 Integer constant expected (Нужна целая константа).
31 Constant expected (Нужна константа).
32 Integer or real constant expected (Нужна целая или вещественная константа).
33 Type identifier expected (Нужен идентификатор типа).
34 Invalid function result type (Неправильный тип результата функции).
35 Label identifier expected (Нужен идентификатор метки).
36 BEGIN expected (Нужен BEGIN).
37 END expected (Нужен END).
38 Integer expression expected (Нужно выражение типа INTEGER).
39 Ordinal expression expected (Нужно выражение перечисляемого типа).
40 Boolean expression expected (Нужно выражение типа BOOLEAN).
41 Operand types do not match operator (Типы операндов не соответствуют операции).
42 Error in expression (Ошибка в выражении).
43 Illegal assignment (Неверное присваивание).
44 Field identifier expected (Нужен идентификатор поля).
45 Object file too large (Объектный файл слишком большой).
46 Undefined external (Неопределенная внешняя процедура).
47 Invalid object file record (Неправильная запись объектного файла).
48 Code segment too large (Сегмент кода слишком большой).

49 Data segment too large (Сегмент данных слишком велик).
50 DO expected (Нужен оператор DO).
51 Invalid PUBLIC definition (Неверное PUBLIC-определение).
52 Invalid EXTRN definition (Неправильное EXTRN-определение).
53 Too many EXTRN definition (Слишком много EXTRN-определений).
54 OF expected (Требуется OF).
55 INTERFACE expected (Требуется интерфейсная секция).
56 Invalid relocatable reference (Неправильная перемещаемая ссылка).
57 THEN expected (Требуется THEN).
58 TO or DOWNTO expected (Требуется TO или DOWNTO).
59 Undefined forward (Неопределенное опережающее описание).
60 Too many procedures (Слишком много процедур).
61 Invalid typecast (Неверное преобразование типа).
62 Division by zero (Деление на ноль).
63 Invalid file type (Неверный файловый тип).
64 Cannot Read or Write variables of this type (Нет возможности считать или записать переменные данного типа).
65 Pointer variable expected (Нужно использовать переменную-указатель).
66 String variable expected (Нужна строковая переменная).
67 String expression expected (Нужно выражение строкового типа).
68 Circular unit reference (Перекрестная ссылка модулей).
69 Unit name mismatch (Несоответствие имен программных модулей).
70 Unit version mismatch (Несоответствие версий модулей).
71 Duplicate unit name (Повторное имя программного модуля).
72 Unit file format error (Ошибка формата файла модуля).
73 IMPLEMENTATION expected (Отсутствует исполняемая часть модуля).

74 Constant and case types do not match (Типы констант и тип выражения оператора CASE не соответствуют друг другу).

75 Record variable expected (Нужна переменная типа запись).

76 Constant out of range (Константа нарушает границы).

77 File variable expected (Нужна файловая переменная).

78 Pointer expression expected (Нужно выражение типа указатель).

79 Integer or real expression expected (Нужно выражение вещественного или целого типа).

80 Label not within current block (Метка не находится внутри текущего блока).

81 Label already defined (Метка уже определена).

82 Undefined label in processing statement part (Неопределенная метка в предшествующем разделе операторов).

83 Invalid @ argument (Неправильный аргумент операции @).

84 Unit expected (Нужно кодовое слово UNIT).

85 ”;” expected (Нужно указать”;”).

86 ”:” expected (Нужно указать”:”).

87 ”,”expected (Нужно указать”,”).

88 ”(” expected (Нужно указать ”(”).

89 ”)” expected (Нужно указать”)”).

90 ”=” expected (Нужно указать”=”).

91 ”:=” expected (Нужно указать”:=”).

92 ”[” or ”(.”expected (Нужно указать ”[” или ”(.”).

93 ”]” or ”.)” expected (Нужно указать”]” или ”.)”).

94 ”.” expected (Нужно указать”.”).

95 ”..” expected (Нужно указать”..”).

96 Too many variables (Слишком много переменных).

97 Invalid FOR control variable (Неправильный параметр цикла оператора FOR).

98 Integer variable expected (Нужна переменная целого типа).

99 File and procedure types are not allowed here (Здесь не могут использоваться файлы или процедурные типы).

100 String length mismatch (Несоответствие длины строки).

101 Invalid ordering of fields (Неверный порядок полей).

102 String constant expected (Нужна константа строкового типа).

103 Integer or real variable expected (Нужна переменная типа INTEGER или REAL).

104 Ordinal variable expected (Нужна переменная порядкового типа).

105 INLINE error (Ошибка в операторе INLINE).

106 Character expression expected (Предшествующее выражение должно иметь символьный тип).

107 Too many relocation items (Слишком много перемещаемых элементов).

108 Overflow in arithmetic operator (Переполнение при выполнении арифметического оператора).

109 No enclosing FOR, WHILE or REPEAT statement (Нет операторов, заканчивающих операторы FOR, WHILE или REPEAT).

110 Debug information table overflow (Переполнение информационной таблицы отладки).

111 N/A

112 CASE constant out of range (Константа CASE нарушает допустимые границы).

113 Error in statement (Ошибка в операторе).

114 Cannot call an interrupt procedure (Невозможно вызвать процедуру прерывания).

115 N/A

116 Must be in 8087 mode to compile this (Для компиляции необходим режим 8087).

117 Target address not found (Указанный адрес не найден).

118 Include files are not allowed here (Здесь не допускаются включаемые файлы).

119 No inherited methods are accessible here (В этом месте программы нет унаследованных методов).

120 N/A

121 Invalid qualifier (Неверный квалификатор).

122 Invalid variable reference (Недействительная ссылка на переменную).

123 Too many symbols (Слишком много символов).

124 Statement part too large (Слишком большой раздел операторов).

125 N/A

126 Files must be var parameters (Файлы должны передаваться как параметры-переменные).

127 Too many conditional symbols (Слишком много условных символов).

128 Misplaced conditional directive (Пропущена условная директива).

129 ENDIF directive missing (Пропущена директива ENDIF).

130 Error in initial conditional defines (Ошибка в условных определениях).

131 Header does not match previous definition (Заголовок не соответствует предыдущему определению).

132 Critical disk error (Критическая ошибка диска).

133 Cannot evaluate this expression (Нельзя вычислить данное выражение).

134 Expression incorrectly germinated (Некорректное завершение выражения).

135 Invalid format specifier (Неверный спецификатор формата).

136 Invalid indirect reference (Недопустимая косвенная ссылка).

137 Structured variable are not allowed here (Здесь нельзя использовать переменную структурного типа).

138 Cannot evaluate without System unit (Нельзя вычислить выражение без модуля SYSTEM).

139 Cannot access this symbol (Нет доступа к данному символу).

140 Invalid floating-point operation (Недопустимая операция с плавающей запятой).

141 Cannot compile overlay to memory (Нельзя выполнить компиляцию оверлейных модулей в память).

142 Procedure or function variable expected (Должна использоваться переменная процедурного типа).

143 Invalid procedure or function reference (Недопустимая ссылка на процедуру или функцию).

144 Cannot overlay this unit (Этот модуль не может использоваться в качестве оверлейного).

145 Too many nested scopes (Слишком много вложений).

146 File access denied (Отказано в доступе к файлу).

147 Object type expected (Здесь должен быть тип OBJECT).

148 Object types are not allowed (Нельзя объявлять локальные объекты).

149 VIRTUAL expected (Пропущено слово VIRTUAL).

150 Method identifier expected (Пропущен идентификатор инкапсулированного правила).

151 Virtual constructor are not allowed (Конструктор не может быть виртуальным).

153 Destructor identifier expected (Пропущен идентификатор деструктора).

154 Fail only allowed within constructor (Обращение к стандартной процедуре FAIL может содержаться только в конструкторе).

155 Invalid combination of opcode and operands (Недопустимая комбинация кода команды и операндов).

156 Memory reference expected (Отсутствует адрес).

157 Cannot add or subtract relocatable symbols (Нельзя складывать или вычитать перемещаемые символы).

158 Invalid register combination (Недопустимая комбинация регистров).

159 286/287 instructions are not enabled (Недоступен набор команд микропроцессоров 286/287).

160 Invalid symbol reference (Недопустимая ссылка на символ).

161 Code generation error (Ошибка генерации кода).

162 ASM expected (Отсутствует зарезервированное слово ASM).

Ошибки, возникающие во время выполнения программ, обнаруживаемые ДОС

№ ошибки	Название
1	Invalid function number (Неверный номер функции).
2	File not found (Не найден файл).
3	Path not found (Путь не найден).
4	Too many open files (Слишком много открытых файлов).
5	File access defined (Отказано в доступе к файлу).
6	Invalid file handle (Недопустимый файловый канал).
12	Invalid file access code (Недействительный код доступа к файлам)
15	Invalid drive number (Недопустимый номер дисковода).
16	Cannot remove current directory (Нельзя удалить текущий каталог).
17	Cannot rename across drives (Нельзя при переименовании указывать разные дисководы).

Ошибки ввода-вывода

№ ошибки	Название
100	Disk read error (Ошибка чтения с диска).
101	Disk write error (Ошибка записи на диск).
102	File not assigned (Файлу не присвоено имя).
103	File not open (Файл не открыт).
104	File not open for input (Файл не открыт для ввода).
105	File not open for output (Файл не открыт для вывода)
106	Invalid numeric format (Неверный числовой формат)

Критические ошибки

№ ошибки	Название
150	Disk is write protected (Диск защищен от записи).
151	Unknown unit (Неизвестный модуль).
152	Drive not ready (Дисковод находится в состоянии «не готов»).
153	Unknown command (Неопознанная команда).
154	CRC error in data (Ошибка в исходных данных).

155	Bad drive request structure length (При обращении к диску указана неверная длина структуры).
156	Disk seek error (Ошибка при операции установки головок на диске).
157	Unknown media type (Неизвестный тип носителя).
158	Sector not found (Сектор не найден).
159	Printer out of paper (Кончилась бумага на принтере).
160	Device write fault (Ошибка при записи на устройство).
161	Device read fault (Ошибка при чтении с устройства).
162	Hardware failure (Сбой аппаратуры).

Фатальные ошибки

Эти ошибки всегда приводят к немедленной остановке программы.

№ ошибки	Название
200	Division by zero (Деление на ноль).
201	Range check error (Ошибка при проверке границ).
202	Stack overflow error (Переполнение стека).
203	Heap overflow error (Переполнение кучи).
204	Invalid pointer operation (Недействительная операция с указателем).
205	Floating point overflow (Переполнение при операции с плавающей запятой).
206	Floating point underflow (Исчезновение порядка при операции с плавающей запятой).
207	Invalid floating point operation (Недопустимая операция с плавающей запятой).
208	Overlay manager not installed (Не установлена подсистема управления оверлеем).
209	Overlay file read error (Ошибка чтения оверлейного файла).
210	Object not initialized (Не инициирован объект).
211	Call to abstract method (Вызов абстрактного правила).
212	Stream registration error (Ошибка в регистрируемом потоке).
213	Collection index out of range (Набираемый индекс выходит из границ диапазона).
214	Collection overflow error (Переполнение коллекции).

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Культин, Н.Б. Программирорвание на Object Pascal в Delphi 5 / Н.Б. Культин. – СПб.: БХВ – Санкт-Петербург, 1999. – 464 с. – ISBN 5-8206-0079-7.
2. Ставровский, А.Б. Турбо Паскаль 7.0: учебник / А.Б. Ставровский. – Киев.: Издательская группа BHV, 2000. – 400 с. – ISBN 966-552-070-9.
3. Пильщиков, В.Н. Сборник упражнений по языку Паскаль: учеб. пособие для вузов / В.Н. Пильщиков. – М.: Наука, 1989. – 160 с. – ISBN 5-02-013995-5.
4. Шаньгин, В.Ф. Программное обеспечение микроЭВМ. В 11 кн. Кн.7. Программирование на языке Паскаль: учеб. пособие для ПТУ / В.Ф. Шаньгин, Л.М. Поддубная. – 2-е изд., перераб. и доп. – М.: Высш. шк., 1991. – 142 с. – ISBN 5-06-001788-5.
5. Васюкова, Н.Д. Практикум по основам программирования. Язык Паскаль: учеб. пособие для учащихся сред. спец. учеб. заведений / Н.Д. Васюкова, В.В. Тюляева. – М.: Высш. шк., 1991. – 160 с. – ISBN 5-06-00750-2.

ОГЛАВЛЕНИЕ

ПРОГРАММИРОВАНИЕ В СРЕДАХ TURBO PASCAL И DELPHI

Методические указания к лабораторным работам
по дисциплине «Алгоритмические языки»

Составитель
КЛИМЕНКОВ Юрий Сергеевич

Ответственный за выпуск – зав. кафедрой профессор В.П. Легаев

Подписано в печать 15.03.10.
Формат 60x84/16. Усл. печ. л. 5,12. Тираж 100 экз.
Заказ
Издательство
Владимирского государственного университета.
600000, Владимир, ул. Горького, 87.