В.М. Пестриков
А.Н. Маслобоев

Программирование на языке Object Pascal

Учебно-методическое пособие

Санкт-Петербург
2014

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ
РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ»

В.М. Пестриков, А.Н. Маслобоев

Программирование на языке Object Pascal

Учебно-методическое пособие

Санкт-Петербург
2014

УДК 618.3 (075)
М 316
ББК 32.97я7

Пестриков В.М., Маслобоев А.Н. Программирование на языке Object Pascal: учебно-методическое пособие / СПбГТУРП. — СПб., 2014. — 74 с.

В настоящем учебно-методическом пособии рассматриваются основные конструкции и операторы языка программирования Object Pascal, который используется в системе программирования Borland Delphi. Пособие содержит необходимый теоретический минимум, примеры программ с подробными комментариями к ним и задания для самостоятельной работы студентов.

Пособие предназначено для студентов факультета АСУТП, обучающихся по направлению 01.03.02 «Прикладная математика и информатика», изучающих дисциплину «Языки и методы программирования», а также студентов других направлений, изучающих дисциплину «Информационные технологии».

Рецензенты:

зав. кафедрой ИИТиСУ СПбГТУРП, профессор, д-р техн. наук Г.А. Кондрашкова;

доцент кафедры информатики СПбГУСЭ, канд. техн. наук С.В. Тихов.

Рекомендовано к изданию Редакционно-издательским советом университета в качестве учебно-методического пособия.

Оглавление

Введение

Современные интегрированные среды разработки программного обеспечения (включая **Delphi**) поддерживают технологию визуального проектирования, которая позволяет достаточно легко, быстро и эффективно создавать пользовательский интерфейс приложения, а также производить начальную настройку свойств объектов, имеющихся в приложении. Но, тем не менее, создание полноценной работоспособной программы по-прежнему немыслимо без написания программного кода, описывающего реакцию объектов приложения на различные программные события. В свою очередь, для разработки программного кода необходимо знание соответствующего языка программирования, который используется в данной среде разработки программного обеспечения.

Для системы **Delphi** таким базовым языком программирования является язык **Object Pascal**. В нем сохраняются все основные возможности классического языка Паскаль и широко распространенной его модификации **Turbo Pascal**, и в то же время появились многие дополнительные возможности. Это новые возможности включают динамическую типизацию, использование вариантного типа данных, поддержку визуального объектно-ориентированного программирования (в языке появилось понятие свойства и связанные с ним ключевые операторы), перегрузку операторов и др.

Следует отметить, что язык **Object Pascal** используется не только в системе **Delphi**. Этот язык имеет и ряд других реализаций, к которым относятся **TopSpeed Pascal, Virtal Pascal, PascalABC.NET, Free Pascal, GNU Pascal**.

В настоящем пособии рассматриваются базовые понятия языка **Object Pascal**, основные операторы языка, объясняются особенности работы в консольном режиме среды **Delphi**. Изучение языка рассматривается на конкретных примерах разработки консольных приложений. В конце каждой главы (кроме главы 1) приведены задания для самостоятельной работы, выполнение которых поможет закреплению пройденного материала и получению необходимых навыков для самостоятельной профессиональной деятельности в сфере информационных технологий.

Глава 1. Основы программирования в Object Pascal

1.1. Особенности языка Object Pascal

Как и любой другой язык программирования, **Object Pascal** имеет свой алфавит. Алфавитом языка программирования называется набор символов, который применяется для записи программ на этом языке. Алфавит языка **Object Pascal** содержит следующие 3 группы символов:

1. Латинские буквы от **A** до **Z**. К буквам относится также символ подчеркивания. **Object Pascal** не чувствителен к регистру, т.е. он не делает разницы между командами и именами переменных, написанными строчными и прописными буквами, в отличие от некоторых других языков программирования, например, языка Си.

2. Цифры от 0 до 9.

3. Специальные символы. К специальным относятся следующие символы:

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

К этой же категории можно отнести применяемые в Object Pascal сочетания из двух символов:

(* *) .. // := <= >= <>

Буквы кириллицы (русского алфавита) могут присутствовать только в строковых константах (понятие строковой константы будет рассмотрено далее в этой главе), а также в комментариях к программе.

Комментариями называются пояснения к тексту программы, которые не влияют на ее выполнение. Для того чтобы отделить комментарии от основного текста программы, их заключают в фигурные скобки.

В качестве альтернативных символов для выделения комментариев могут использоваться двойные символы – открывающая и закрывающая скобка со звездочкой.

Допускается еще один способ создания комментариев. В строке ставятся два символа «слэш» (косая черта), и все, что находится справа от черты, воспринимается программой как комментарий. Однако следует иметь в виду, что в последнем случае комментарий может быть только однострочным.

Ниже приведены примеры комментариев в **Object Pascal**:

```
{ Это комментарий к тексту программы }
(* Это еще один комментарий к программе *)
// Однострочный комментарий
```

Из символов языка **Object Pascal** складываются слова, которые отделяются друг от друга пробелами. Из отдельных слов состоят операторы

(команды) языка. Слова, используемые в **Object Pascal**, можно также разделить на 3 основные категории:

1. Ключевые слова.
2. Идентификаторы.
3. Значения.

Ключевыми словами называются такие слова, которые в языке Object **Pascal** имеют строго определенный смысл, заданный разработчиками языка. Эти слова не могут произвольным образом переопределяться пользователем. К таким ключевым словам относятся:

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

К базовым понятиям языка **Object Pascal** относятся также идентификаторы и значения.

Идентификаторами в **Object Pascal** называются имена констант, переменных, типов, функций, процедур, модулей и некоторых других элементов программы. В отличие от ключевых слов, пользователь может давать имена этим элементам по своему усмотрению.

При выборе имен элементов следует отдавать предпочтение осмысленным именам, отражающим значение элемента, что значительно облегчит процесс чтения, понимания и корректировки (в случае необходимости) программы, в первую очередь для самого разработчика. Максимальная длина идентификатора не должна превышать 255 символов (остальные символы сверх 255 компилятор будет игнорировать).

Идентификаторы могут содержать только латинские буквы и цифры, причем первым символом идентификатора всегда должна быть буква. Пробелы в идентификаторах недопустимы. Поэтому, если нужно создать идентификатор, состоящий из нескольких слов, между ними ставится не пробел, а символ подчеркивания, который, как нам уже известно, считается в **Object Pascal** буквой.

Значения – это величины, которые могут присваиваться переменным или константам. Чаще всего в программах используются числовые и строковые значения. Целочисленные значения задаются просто в виде последовательности цифр. При задании вещественных значений следует отделять целую часть числа от дробной точкой.

Строковые значения (часто называемые также строковыми константами) представляют собой любую последовательность символов, заключенную в апострофы. Эта последовательность может включать в себя как любые символы, входящие в алфавит языка **Object Pascal**, так и прописные и строчные буквы русского алфавита.

Для того чтобы пользователь мог успешно составлять программы на языке **Object Pascal**, недостаточно знать только алфавит и ключевые слова. Необходимо также представлять структуру такой программы. Стандартная структура программы на языке **Object Pascal** выглядит следующим образом:

1) заголовок программы;
2) раздел подключения модулей;
3) раздел описания констант и переменных;
4) раздел описания функций и процедур;
5) раздел операторов.

Подробнее каждый из этих разделов программы мы разберем в разделе 1.2 пособия, где рассматривается создание консольного приложения в Object Pascal.

1.2. Консольный режим. Разработка первой программы

Для того чтобы изучить основные операторы языка **Object Pascal** и возможности их применения, не отвлекаясь на вопросы, связанные с разработкой пользовательского интерфейса, можно использовать консольный режим, поддерживаемый средой программирования **Delphi**.

Консоль – это монитор (дисплей) и клавиатура компьютера, которые рассматриваются как единое целое. Консольным приложением называется программа, которая работает в режиме эмуляции операционной системы **MS DOS**. В этом режиме устройством ввода является клавиатура, а устройством вывода – монитор (дисплей), работающий в текстовом режиме.

Рассмотрим, как осуществляется переход в консольный режим. После запуска системы **Delphi** на экране компьютера появляется стандартный интерфейс **Delphi** в режиме разработчика, включающий строку меню.

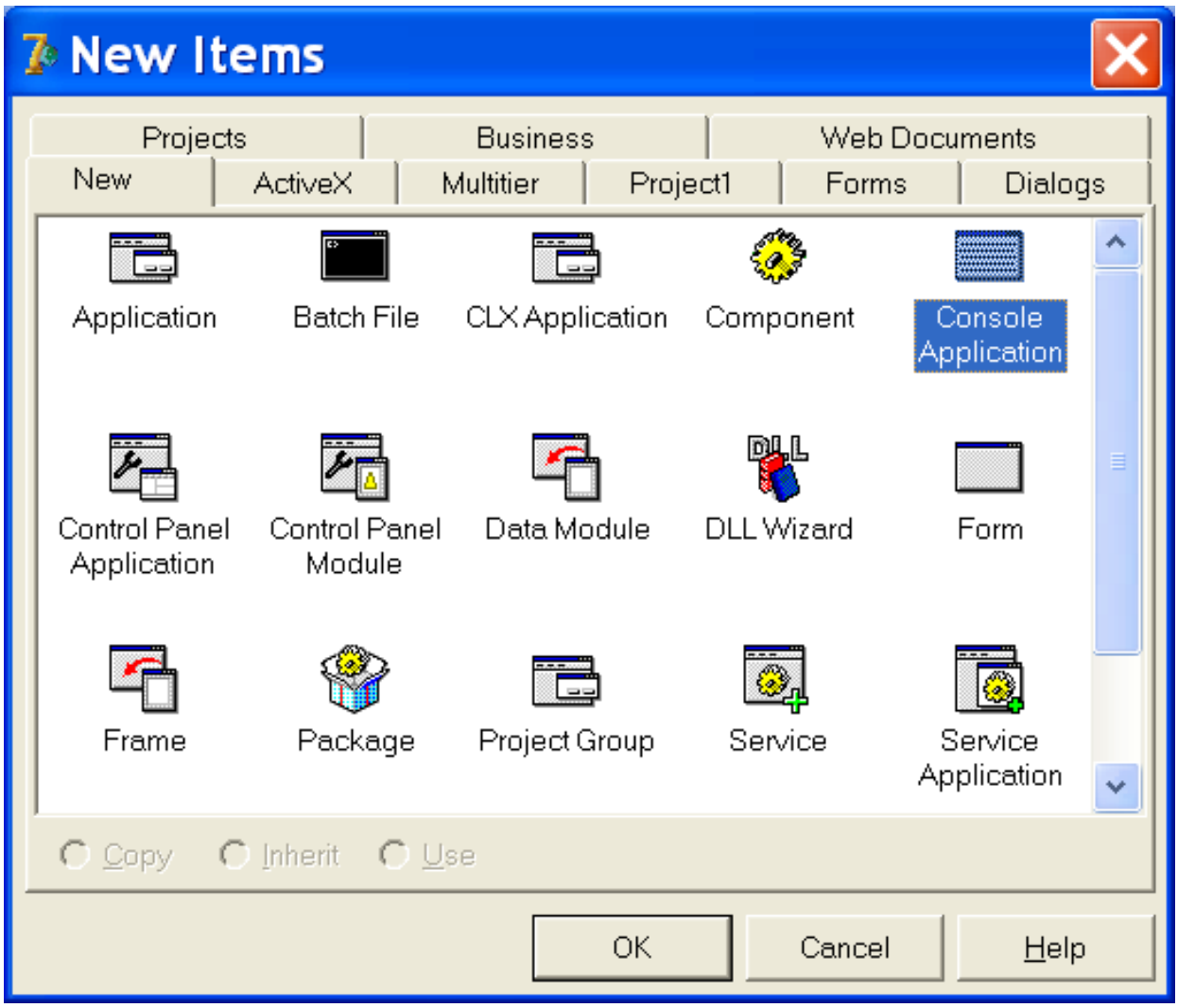

Рис.1.1. Диалоговое окно, позволяющее создать консольное приложение

В меню системы программирования следует открыть раздел **File**, а в нем найти пункт **New**. Справа от названия пункта находится треугольная стрелка, которая показывает, что данный пункт открывает вложенное подменю. В этом подменю нужно найти пункт **Other**. Справа от названия пункта стоит многоточие, говорящее о том, что после одиночного щелчка мыши на данном пункте открывается диалоговое окно. Внешний вид открывшегося диалогового окна **New Items** представлен на рис. 1.1.

В этом окне вместо предлагаемого по умолчанию варианта **Application** следует выбрать расположенный в правом верхнем углу вариант **Console Application**. После выбора вышеуказанного варианта на экране появляется окно **Project.dpr**, содержащее заготовку для создания текста новой программы - консольного приложения, показанное на рис.1.2.

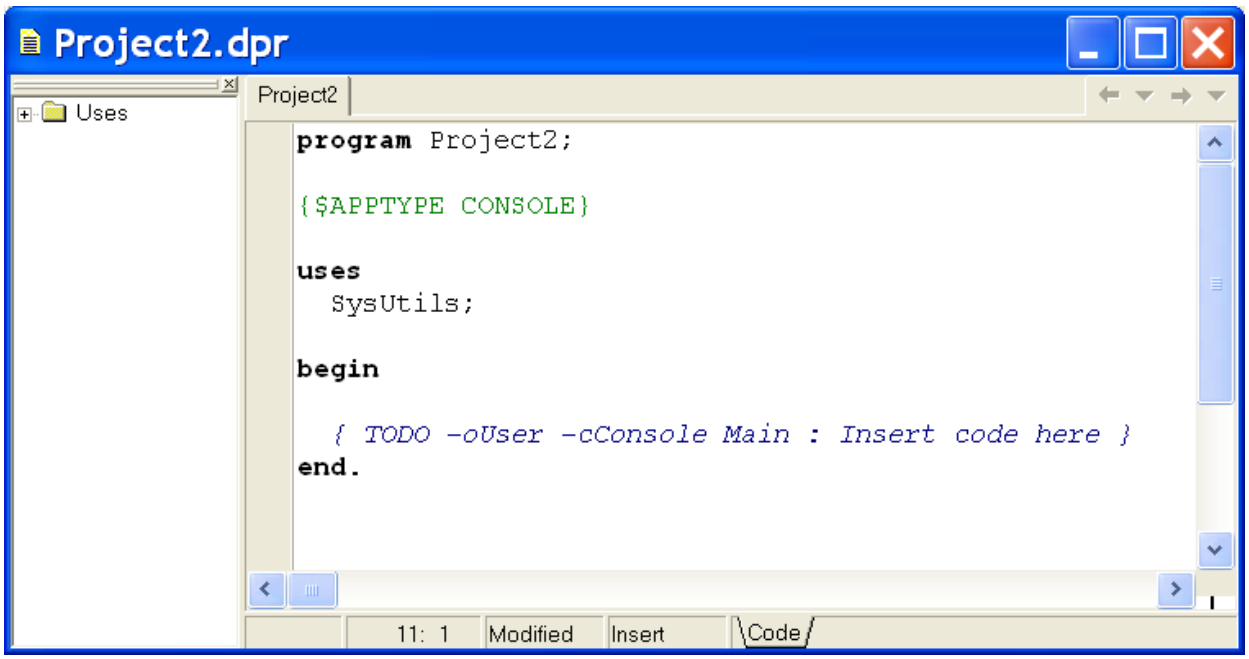

Рис.1.2. Окно, содержащее шаблон (заготовку) для создания консольного приложения

Первая строка заготовки представляет собой заголовок программы. Заголовок содержит служебное слово **program**, за которым (после пробела) указывается собственное имя программы. Заголовок всегда завершается точкой с запятой.

Для того чтобы дать программе собственное имя, программист вначале должен сохранить текущий проект. Проектом в данном случае называется программа, находящаяся в процессе разработки.

При сохранении проекта программист может заменить имя **Project2**, которое система предлагает по умолчанию, на то, которое он считает для данного проекта подходящим. Сохранение проекта производится следующим образом: открывается раздел меню системы программирования **File,** а в нем пункт **Save project as.** Этот пункт, в свою очередь открывает диалоговое окно, в нижней части которого и указывается имя для сохраняемого проекта. Указанное имя будет дано создающемуся на жестком диске компьютера файлу с расширением dpr. Это же имя автоматически получает и программа.

Например, проекту, который мы рассматриваем в этом разделе, дадим собственное имя **first.** Тогда на жестком диске компьютера будет создан файл проекта **first.dpr,** а строка заголовка программы будет выглядеть следующим образом:

```
program first;
```

Правила наименования для проектов (и программ) те же, что и для идентификаторов, т.е. собственное имя программы может содержать только латинские буквы и цифры, причем первым символом должна быть буква.

Естественно, что данное автором программы имя должно отражать содержание и смысл программы.

Помимо имени файла, в том же диалоговом окне можно выбрать и папку, в которой будет сохраняться данный файл.

На следующей (после строки заголовка) строке можно увидеть заключенную в фигурные скобки фразу:

```
{$APPTYPE CONSOLE}
```

Обычно текст на языке Паскаль, заключенный в фигурные скобки, является комментарием. Но в данном случае после открывающейся фигурной скобки мы видим символ доллара (официальное название этого символа – знак денежной единицы). Данный символ показывает, что следующий за ним текст является не комментарием, а управляющей директивой. Данная директива предназначена для компилятора. В соответствии с директивой компилятор на основе текста исходной программы создает консольное приложение.

Далее в программе располагается раздел подключения программных модулей. Модулем в **Object Pascal** называется набор функций и процедур, который содержится в отдельном библиотечном файле. Модули могут быть стандартными, т.е. входящими в состав системы программирования, и программируемыми пользователем. Подключение модуля к программе производится с помощью команды **uses.** Затем после пробела указывается собственное имя подключаемого модуля или имена модулей, перечисляемых через запятую (если их несколько). В данном приложении к основному содержанию программы по умолчанию подключается только один стандартный модуль **SysUtils.**

Далее в программе должен следовать раздел описаний констант и переменных. Так как в первой создаваемой нами программе никаких вычислений не производится, необходимости в использовании и описании именованных констант и переменных нет. Поэтому раздел описаний в данной программе можно пропустить.

Ниже в окне находятся служебные слова **begin** и **end**. Слово **begin** определяет начало основной части программы, **end** – конец программы. После слова **end** ставится точка, завершающая программу. Между этими словами можно увидеть следующий текст, заключенный в фигурные скобки:

```
{TODO –oUser –cConsole Main : Insert code here}
```

Поскольку знак денежной единицы после открывающейся скобки в данном случае отсутствует, то находящийся в фигурных скобках текст является комментарием (а не управляющий директивой). Часть текста, находящаяся после двоеточия означает, в переводе с английского: «Вводите (программный) код здесь». Именно здесь между словами **begin** и **end** вводится основной текст программы.

Первая программа, которую мы создадим, не будет выполнять никаких действий вычислительного характера. Эта программа просто выведет на экран компьютера в одну строку приветствие. Для этого между словами **begin** и **end** вводим следующие две строки:

```
writeln('This is my first program in Object Pascal');
readln;
```

Каждая из строк содержит оператор (т.е. команду) языка **Object Pascal**. В конце каждого оператора обязательно ставится точка с запятой.

Первый оператор является оператором вывода, который позволяет вывести определенную информацию на экран компьютера. В данном случае информация представляет собой текст на английском языке и, в переводе на русский язык, означает: «Это моя первая программа на Object Pascal». Оператор вывода начинается со служебного слова **writeln**. Этот оператор выполняет следующие два действия:

а) выводит на экран ПК информацию, заключенную в круглые скобки;

б) по завершении вывода информации переводит курсор, находящийся в окне вывода, на следующую строку.

В языке Паскаль существует и еще один оператора вывода – **write**. Отличие между операторами **write** и **writeln** заключается в том, что после вывода информации оператор **write** оставляет курсор в той же строке.

Информация, выводимая операторами **write** или **writeln,** должна обязательно быть заключена в круглые скобки. Если эта информация представляет собой текст, то, кроме круглых скобок, она должна быть заключена еще и в апострофы. Если выводится значение переменной, то имя переменной в апострофы не заключается.

Следующий оператор, расположенный строкой ниже, – это оператор ввода **readln**. Чаще всего этот оператор используется для присваивания значений переменным путем ввода этих значений с клавиатуры. Для этого после служебного слова **readln** в круглых скобках указывается имя соответствующей переменной. Когда при выполнении программы она доходит до оператора ввода, то следует ввести новое значение переменной в месте, обозначенном курсором, а затем подтвердить ввод нажатием клавиши **Enter**.

Отметим, что в нашей программе после служебного слова **readln** круглые скобки отсутствуют и никакого имени переменной не указывается. Такой оператор ввода просто приостанавливает работу программы до нажатия клавиши **Enter**. Это необходимо сделать, потому что в противном случае после запуска программы на выполнение и вывода текста окно с результатами работы программы сразу же закроется и пользователь не успеет ознакомиться с его содержанием. Теперь же благодаря использованию «пустого» (без параметров) оператора ввода пользователь может изучать

результаты выполнения программы столько, сколько необходимо, а для закрытия окна нажать **Enter**.

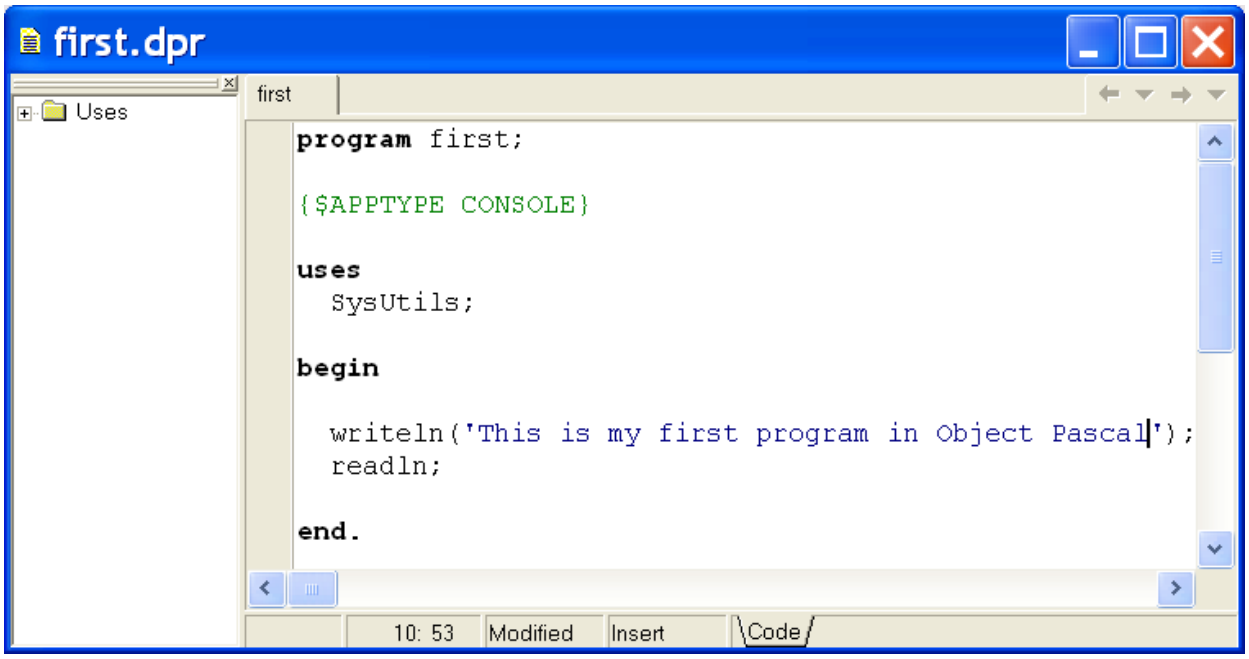

Рис. 1.3. Окно программного кода после записи операторов и сохранения файла программы

Комментарий, расположенный между **begin** и **end**, теперь можно удалить, а программу следует сохранить, указав при этом ее собственное имя и место для сохранения, как было сказано выше. После выполнения указанных действий окно программы приобретет вид, показанный на рис.1.3.

Следующим шагом является запуск программы на компиляцию и выполнение. Компиляция производится с помощью команды меню **Project** → **Compile** (стрелка в данном случае означает, что в начале открывается меню **Project**, а в нем пункт **Compile**) или комбинацией клавиш **Ctrl+F9**. Для запуска программы на выполнение нужно использовать команды меню **Run** → **Run** или клавишу **F9**. Также для запуска программы можно использовать кнопку, находящуюся на панели инструментов под строкой меню и имеющую вид треугольной стрелки зеленого цвета. Если в программе нет ошибок, то на экране компьютера появляется окно, в котором работает консольное приложение. Результат работы показан на рис.1.4.

Попробуем изменить рассмотренную выше программу таким образом, чтобы она выводила на экран приветствие на русском языке. Для этого изменим в программе текст, находящийся в скобках в операторе вывода. Теперь оператор вывода будет выглядеть следующим образом:

writeln('Это моя первая программа в Object Pascal');

Заново откомпилируем программу и снова запустим ее на выполнение. В итоге вместо текста на русском языке в рабочем окне консольного приложения получается бессмысленный набор символов, показанный на рис. 1.5. Правильно отображаются только та часть текста, которая написана латиницей.

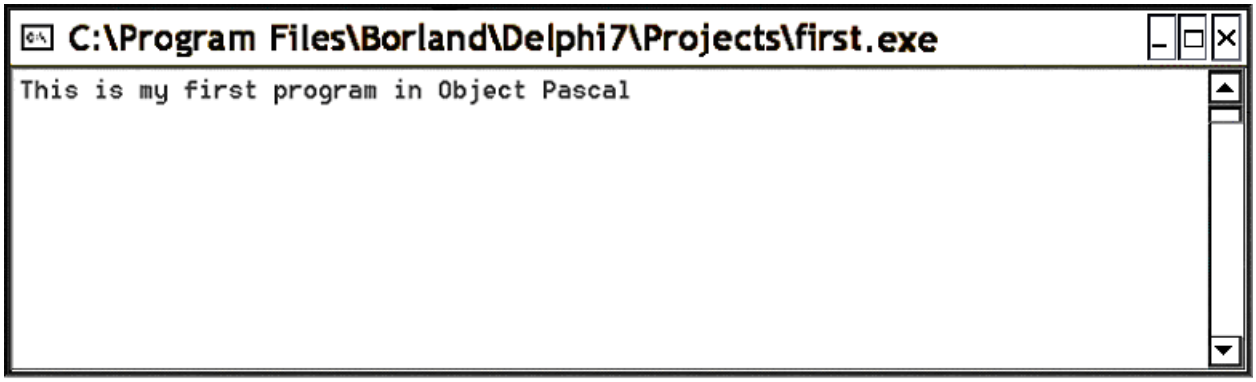

Рис. 1.4. Рабочее окно консольного приложения

Для того чтобы разобраться в причинах этого «загадочного» явления, следует обратить внимание на следующее обстоятельство. Консольное приложение создается в интегрированной среде, работающей под управлением операционной системы **Windows**, а выполняется как программа, работающая под управлением операционной системы **MS DOS**.

В этих операционных системах применяется разная кодировка для букв кириллицы (русского алфавита). В **Windows** применяется кодировка **ANSI**, а в **MS DOS** – кодировка **ASCII**. Поэтому при вводе исходного текста программы вводимые символы кодируются числами по таблице **ANSI**. При выводе же сообщения на экране появляются символы, имеющие тот же числовой код, что и в **ANSI**, но этим числовым кодам в кодовой таблице **ASCII** соответствуют совершенно другие символы.

Следовательно, для того, чтобы буквы кириллицы правильно отображались при выводе информации в окне консольного приложения, необходимо сразу после ввода текста в компьютер произвести его перекодировку из **ANSI** в **ASCII**.

К сожалению, в **Object Pascal** нет стандартной функции, которая бы выполняла данную операцию, но подобную функцию можно создать самостоятельно. Для этого функцию нужно предварительно описать в разделе, находящемся перед основной частью программы. Затем эту функцию можно вызвать из основной части программы.

Каким образом происходит процесс создания новой авторской функции в Object Pascal, будет подробно описано в главе 7 данного учебного пособия. Пока же ограничимся тем, что приведем готовый текст данной функции и разберемся, как ее использовать для правильного вывода сообщений на русском языке.

Назовем функцию перекодировки **`kir`**. Большинство функций имеет входное значение (аргумент). Для данной функции аргументом является строка, представленная в кодировке **ANSI**, используемой в **Windows**.

Выходным значением данной функции является та же строка, но уже преобразованная в кодировку **ASCII**, принятую в системе **MS DOS**.

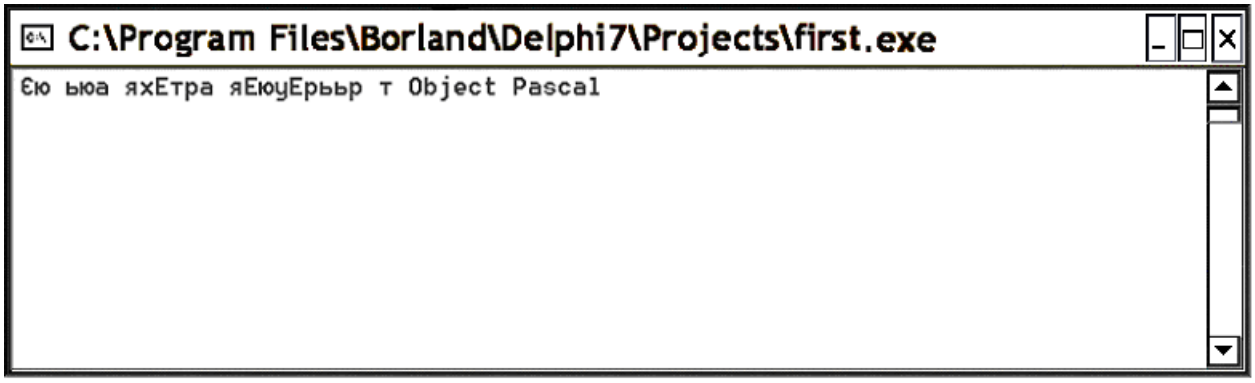

Рис. 1.5. «Загадочная» надпись, появляющаяся в рабочем окне консольного приложения при попытке вывести на экран текст на русском языке

Ниже приводится текст данной функции, которая должна быть размещена после команды подключения дополнительного модуля **uses**, но до основной части программы, т.е. до ключевого слова **begin**:

```
function kir(st:string):string;
 var i,k:integer;
begin
 for i:=1 to length(st) do
 begin
  k:=ord(st[i]);
  case st[i] of
  'A'..'п': st[i]:=Chr(k-64);
  'р'..'я': st[i]:=Chr(k-16);
  end;
 end;
 kir:=st;
end;
```

Для того чтобы функцию **kir** можно было использовать в основной части программы, достаточно в операторе вывода создать обращение к этой функции. Это обращение производится из оператора вывода **writeln**. Обращение к функции состоит из собственного имени функции и аргумента функции, заключенного в круглые скобки. В данном случае аргументом является выводимый на экран текст сообщения.

При использовании функции необходимо также помнить о том, что обращение к функции не является отдельным оператором, а входит составной частью в какой-либо оператор.

Измененный оператор вывода будет выглядеть следующим образом:

```
writeln(kir('Это моя первая программа в Object Pascal'));
```

Ниже приводится полный листинг программного модуля **first** с учетом произведенных в программе изменений и дополнений.

```
program first;

{$APPTYPE CONSOLE}

uses
SysUtils;

function kir(st:string):string;
 var i,k:integer;
begin
 for i:=1 to length(st) do
 begin
  k:=ord(st[i]);
  case st[i] of
  'A'..'п': st[i]:=Chr(k-64);
  'р'..'я': st[i]:=Chr(k-16);
  end;
 end;
 kir:=st;
end;

begin
writeln(kir('Это моя первая программа в Object Pascal'));
readln;
end.
```

Теперь проблема, показанная на рис.1.5, устранена и программа позволяет нормально вывести сообщение, написанное на русском языке, что наглядно продемонстрировано на рис.1.6.

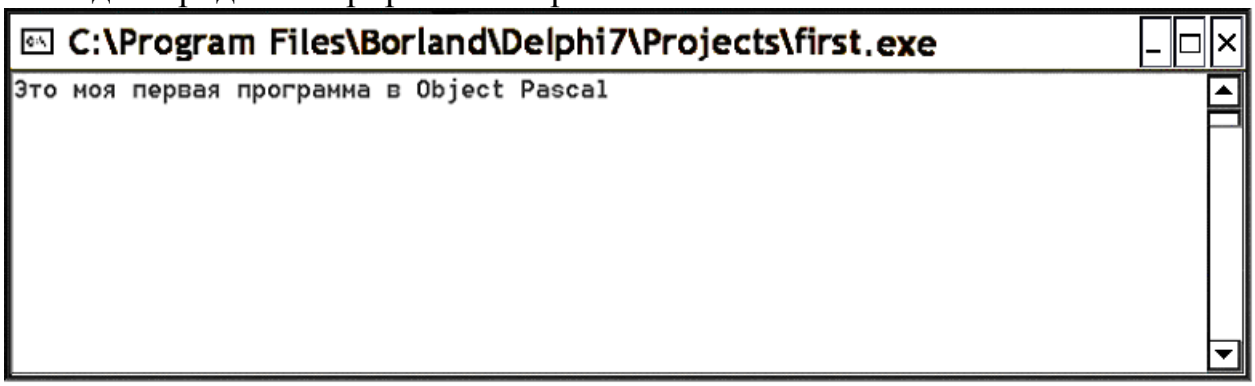

Рис.1.6. Результат работы программы **first** после внесенных в нее дополнений и изменений

Глава 2. Решение задач вычислительного характера

2.1. Общие правила описания и использования переменных

В первой программе, разработанной в среде Delphi на языке Object Pascal, которая была рассмотрена в предыдущей главе пособия, производился только вывод определенной информации на экран компьютера. Но, как правило, ввод или вывод информации при работе с компьютером не является самоцелью.

Подавляющее большинство задач, которые решаются на современных ПК, требуют выполнения различных вычислений на основе введенных исходных данных с последующим выводом полученного результата на экран или на принтер. Это было изначальной целью создания компьютеров, и само слово «компьютер», как известно, в переводе с английского, означает «вычислитель». Подход к решению вычислительных задач представлен в настоящей главе пособия.

В решении задач вычислительного характера важную роль играют переменные. Переменными называются величины, которые изменяют свое значение в процессе выполнения программы. Согласно правилам языка Object Pascal, все переменные, имеющиеся в программе, должны быть предварительно описаны. Раздел описания переменных в программе, как правило, находится после команды подключения модулей **Uses**, но до раздела описания подпрограмм (т.е. функций и процедур) и основной части программы (раздела операторов).

Раздел описания переменных начинается с ключевого слова **var**, за которым следует список используемых в программе переменных. Для каждой переменной должен быть указан ее тип. Между именем переменной и ее типом ставится двоеточие. После названия типа ставится точка с запятой, а затем описывается следующая переменная с ее типом и так до тех пор, пока не будет исчерпан весь список имеющихся в программе переменных. Если несколько переменных в программе относятся к одному и тому же типу, то эти переменные можно перечислить через запятую, а затем после двоеточия указать общий для них тип.

В языке Object Pascal существует большое количество стандартных типов, используемых для описания как числовых переменных, так и переменных других видов – логических, символьных, строковых. Но в этой главе при описании данных мы будем использовать только 2 наиболее часто употребляемых типа переменных. Это тип **integer**, применяемый для описания целочисленных переменных, и тип **real**, используемый для описания вещественных переменных, т.е. таких, которые содержат целую и дробную часть. Пример того, как может выглядеть в программе раздел описания переменных, приведен ниже:

```
var
      a,b:integer;
      c:real;
```

В данном случае описаны три переменные, из которых две (**a** и **b**) относятся к целочисленному типу, а одна (**c**) к вещественному.

Значения переменных в процессе работы программы могут изменяться двумя способами.

Первый способ заключается в том, что пользователь может изменять значение переменной путем ввода нового значения этой переменной с клавиатуры. Для этого применяются операторы ввода **readln** и **read**. С «пустым» оператором **readln** мы уже познакомились в предыдущей главе, а если необходимо с помощью оператора присвоить значение какой-либо переменной, то имя этой переменной следует указать после служебного слова **readln** или **read** в круглых скобках. Пример такого оператора:

```
readln(a);
```

Этот оператор присваивает новое значение переменной по имени **a**. Когда в ходе работы программы она доходит до оператора ввода, то на экране появляется курсор, обозначающий позицию для ввода нового значения переменной. Присваивание нового значения следует подтвердить нажатием клавиши **Enter**.

Различие между оператором **readln** и **read** заключается в следующем: если применяется **readln**, то после нажатия **Enter** курсор перемещается на следующую строку; в случае с оператором **read** курсор после нажатия **Enter** остается в той же строке. Таким образом, это различие аналогично тому, что существует между рассмотренными в предыдущей главе операторами вывода **writeln** и **write**.

В языке Object Pascal с помощью одного оператора ввода допускается присваивать значения сразу нескольким переменным. В этом случае имена переменных перечисляются в круглых скобках через запятую после служебного слова **readln** или **read**. Пример такого оператора приведен ниже:

```
readln(x,y,z);
```

С помощью этого оператора ввода новые значения присваиваются сразу трем переменным **x,y** и **z**. Когда при выполнении программы будет достигнут этот оператор, то в месте, обозначенном курсором, нужно ввести значение первой переменной **x**, затем ввести пробел. Далее вводится вторая переменная **y**, затем снова пробел, затем третья переменная **z**, и только затем нажимается клавиша **Enter**.

При использовании второго способа изменения значения переменной это значение может изменяться непосредственно самой программой без участия пользователя. Для этого в языке **Object Pascal** существует специальный оператор присваивания.

Общий вид оператора присваивания следующий: в левой части указывается имя переменной, которой присваивается новое значение, затем

ставится символ операции присваивания, состоящий из двоеточия со знаком равенства. В правой части стоит выражение, которое вычисляется при выполнении оператора, и вычисленное значение присваивается указанной в левой части переменной.

Выражение может включать в свой состав имена переменных, числа и знаки математических операций. В простейшем случае выражение может состоять из одного числа или одной переменной.

В Паскале существуют следующие правила обозначения математических операций:

- сложение и вычитание записываются так же, как и в математике, знаками «плюс» и «минус»;
- для операции умножения используются символ * («звездочка»). Никакие другие символы для обозначения умножения в Object Pascal использовать нельзя. Также не допускается пропускать знак умножения между сомножителями;
- для операции деления употребляется символ / («слэш»). Никакие другие символы для обозначения деления в Object Pascal использовать нельзя.

Ниже приводятся примеры операторов присваивания в языке Object Pascal, снабженные необходимыми комментариями (комментарии, как нам уже известно, заключаются в фигурные скобки):

```
x:=5; {переменной x присваивается значение, равное числу 5}

y:=z  {переменной y присваивается то же значение, которое имеет переменная z, значение z при этом остается неизменным}

i:=i+1; {берется значение переменной i, затем это значение увеличивается на единицу и присваивается той же переменной i}
```

При вычислении значения выражения должен соблюдаться определенный порядок выполнения действий, который определяется приоритетом (старшинством) операций. В первую очередь выполняются операции, обладающие более высоким приоритетом. К ним относятся умножение и деление. Сложение и вычитание обладают меньшим приоритетом, т.е. они выполняются во вторую очередь.

Операции, имеющие одинаковый приоритет, выполняются последовательно слева направо. Для изменения порядка выполнения операций могут использоваться круглые скобки. Часть выражения, заключенная в скобки, обладает наиболее высоким приоритетом и выполняется в первую очередь.

2.2. Целочисленные переменные

Рассмотрим использование переменных на примере следующей задачи: по заданной длине и ширине требуется вычислить площадь прямоугольника. В начале программы присваиваем программе собственное имя – **rectangle** (что по-английски означает «прямоугольник»).

Затем после управляющей директивы и команды подключения модуля описываем используемые в программе переменные. В программе используются 3 переменные. Переменные **a** и **b** необходимы для обозначения длины и ширины прямоугольника. Переменная **c** должна содержать значение вычисленной площади прямоугольника, т.е. она представляет результат работы программы. Все 3 переменные относятся к целому типу, поэтому в разделе описания они перечисляются через запятую, а затем после двоеточия указывается общий для них тип – **integer**.

Далее описывается уже знакомая нам по предыдущей программе пользовательская функция **kir**, необходимая для правильного отображения текста, написанного на русском языке.

В основной части программы (разделе операторов) с помощью 2 операторов ввода **readln** присваиваются значения переменным **a** и **b**, т.е. программа получает исходные данные – длину и ширину прямоугольника. Перед каждым оператором ввода стоит оператор вывода **writeln**, и тогда блок операторов, отвечающий за ввод данных, будет выглядеть так:

```
writeln(kir('Введите длину прямоугольника'));
readln(a);
writeln(kir('Введите ширину прямоугольника'));
readln(b);
```

Для чего в данном блоке нужны операторы вывода? Ответ заключается в следующем. Возможно, что с программой будет работать пользователь, которой сам ее не составлял и не в курсе всех ее деталей. Когда программа в ходе выполнения дойдет до оператора ввода, то в окне программы появится курсор, обозначающей позицию для ввода данных. Однако если не принять дополнительных мер, то непосвященному пользователю будет непонятно, какие именно данные следует вводить. Поэтому желательно перед каждым оператором ввода вставлять подсказку, которая поясняла бы пользователю, что он должен делать в текущей ситуации. Эта подсказка и создается в текстовом виде с помощью соответствующего оператора вывода. Таким образом, обеспечивается дружественный пользовательский интерфейс приложения, т.е. комфортный и удобный способ общения пользователя с программой.

В следующем операторе (операторе присваивания) мы встречаем выражение, значение которого вычисляется путем перемножения двух исходных величин и присваивается переменной **c**, находящейся в левой части оператора:

```
c:=a*b;
```

Таким образом, остается только вывести полученный результат (площадь прямоугольника **c**) на экран компьютера, что и выполняется с помощью оператора вывода **writeln**. Обратите внимание на одно отличие этого оператора от аналогичного оператора в предыдущей программе. В данном операторе на экран компьютера выводится не какое-либо одно значение, а список значений, перечисляемый через запятую. Элементами списка в данном случае являются текст, являющийся аргументом функции **kir** и заключенный в апострофы, а также значение переменной **c**, причем имя переменной в апострофы заключать не надо.

Ниже приводится полный текст рассмотренного нами программного модуля, а под ним на иллюстрации показано окно с результатами работы программы (рис. 2.1).

```
program rectangle;

{$APPTYPE CONSOLE}

uses  SysUtils;

var a,b,c:integer;

function kir(st:string):string;
 var i,k:integer;
begin
 for i:=1 to length(st) do
 begin
  k:=ord(st[i]);
  case st[i] of
   'A'..'п': st[i]:=Chr(k-64);
   'р'..'я': st[i]:=Chr(k-16);
  end;
 end;
kir:=st;
end;

begin
writeln(kir('Введите длину прямоугольника'));
readln(a);
writeln(kir('Введите ширину прямоугольника'));
readln(b);
c:=a*b;
writeln(kir('Площадь прямоугольника равна '),c);
readln;
end.
```

Рис. 2.1. Результат работы программы «Вычисление площади прямоугольника»

2.3. Вещественные переменные

Понятно, что далеко не всегда при вычислениях можно обойтись только целыми величинами. Например, результатом операции деления в Object Pascal всегда является вещественная величина, даже если делимое и делитель относятся к целым величинам. Также вещественным всегда является результат, если хотя бы одна из величин, участвующих в операции, является вещественной. Так вещественный результат получается при сложении целого и вещественного чисел, при умножении целого числа на вещественное, при вычитании из целого числа вещественного. Описание же вещественных переменных и вывод вещественных величин имеет определенные особенности, которые мы рассмотрим ниже.

Во-первых, как уже отмечалось, для описания вещественных величин необходимо использовать специальный тип данных **`real`**. Используемые в программе целые и вещественные величины следует описывать отдельно друг от друга, хотя они и располагаются в одном разделе программы. Эти описания должны быть отделены друг от друга точкой с запятой.

Во-вторых, при выводе вещественных величин следует учитывать, что вещественное число в Object Pascal может быть представлено в 2 различных видах. Эти виды называются: число с фиксированной точкой и число с плавающей точкой.

Представление числа с фиксированной точкой похоже на привычное представление вещественного числа в математике, только для отделения целой части числа от дробной используется не запятая, а точка. Этот способ представления вещественных чисел в большинстве случаев оказывается более удобным и наглядным.

Но по умолчанию Object Pascal выводит вещественные числа в виде с плавающей точкой. В этом случае вещественное число представляется по следующей формуле:

$$M*10^P,$$

где **M** – мантисса числа (т.е. значимая часть числа);

P – целое число со знаком, которое называется порядком**.**

Поскольку при работе в консольном режиме информация может выводиться на экран компьютера только в одну строку (без надстрочных и подстрочных индексов), то на экране запись числа с плавающей точкой будет выглядеть таким образом:

MEP

Буква **E** в этой записи заменяет число 10.

Поэтому, например, запись на экране компьютера **1.237E2** расшифровывается как $1{,}237*10^2$ или 123,7.

Если имеется необходимость преобразовать запись вещественного числа с плавающей точкой к виду с фиксированной точкой, следует задать формат выводимого значения. Для этого в операторе вывода **write** или **writeln** после имени выводимой вещественной переменной ставят двоеточие, затем указывают общее количество символов, которое отводится под данную переменную. Далее ставят второе двоеточие, после которого указывается количество символов, отводимое под дробную часть. Количество символов в целой части при задании формата указывать не надо. Это количество равно общему количеству символов за вычетом количества символов в дробной части и еще одного символа, который отводится под точку, отделяющую целую часть от дробной.

Например, если в программе мы видим следующий оператор вывода:

```
writeln(x:8:3);
```

то этот оператор осуществляет вывод на экран компьютера значения вещественной переменной **x** в виде с фиксированной точкой. Значение данной переменной занимает 8 позиций, из которых 3 резервируется под дробную часть. Одну позицию занимает точка и 4 позиции остается под целую часть числа. Если реальное количество цифр в целой части числа будет меньше, чем заданное при указании формата, то оставшиеся позиции будут заняты пробелами.

Рассмотрим использование вещественных переменных в Object Pascal на следующем примере. По заданному радиусу шара требуется вычислить объем шара и площадь его поверхности. В программе **sphere**, решающей поставленную задачу, в разделе описаний после ключевого слова **var** описываются 3 переменные: переменная **r** (радиус шара) является целочисленной, а переменные **s** (площадь шара) и **v** (объем шара) описываются как переменные вещественного типа. Результаты вычислений **s** и **v** могут относиться только к вещественному типу ввиду того, что используемые при вычислениях стереометрические формулы содержат число π, которое, как известно является бесконечной дробью.

В разделе операторов программы после ввода единственного исходного данного радиуса шара r производится вычисление объема и площади шара по следующим формулам:

$V = \frac{4}{3}\pi r^3$ — объем шара;

$S = 4\pi r^2$ — площадь шара.

В программе запись может производиться только в одну строку, поэтому операция деления, присутствующая в первой формуле, обозначается символом /. Поскольку операция деления должна быть выполнена в первую очередь, она заключается в круглые скобки. Число π, присутствующее в формуле, определяется с помощью стандартной функции **pi**, имеющейся в языке **Object Pascal**. Так как операция возведения в степень в **Object Pascal** отсутствует, ее приходится заменять множественным умножением. Следует также помнить, что знак умножения * между сомножителями пропускать нельзя. С учетом вышеизложенного формула вычисления объема шара будет записана в программе в виде следующего оператора:

```
v:=(4/3)*pi*r*r*r;
```

Формула вычисления площади шара, записанная по аналогичным правилам, выглядит в программе следующим образом:

```
s:=4*pi*r*r;
```

Операторы, выводящие результат вычислений на экран компьютера, аналогичны таким же операторам в предыдущей программе. Ниже приводится полный текст программного модуля, а под ним на рис. 2.2 показано окно с результатами вычислений:

```
program sphere;

{$APPTYPE CONSOLE}

uses
SysUtils;

var r:integer;
    s,v:real;

function kir(st:string):string;
 var i,k:integer;
begin
 for i:=1 to length(st) do
 begin
  k:=ord(st[i]);
  case st[i] of
```

```
    'A'..'п': st[i]:=Chr(k-64);
    'р'..'я': st[i]:=Chr(k-16);
   end;
  end;
kir:=st;
end;
begin
 writeln(kir('Введите радиус шара'));
 readln(r);
 s:=4*pi*r*r;
 v:=(4/3)*pi*r*r*r;
 writeln(kir('Площадь шара равна'),s);
 writeln(kir('Объем шара равен '),v);
 readln;
end.
```

Рис.2.2. Результаты выполнения программы вычисления площади и объема шара до форматирования выходных данных

На рис.2.2 видно, что площадь и объем шара по умолчанию выводятся как величины с плавающей точкой. Но, как уже отмечалось, этот способ представления вещественных величин в данном случае недостаточно нагляден. Поэтому изменим в программе два оператора, которые отвечают за вывод результатов.

Преобразуем значения переменных к виду с фиксированной точкой. Для каждой из выводимых переменных укажем формат 10:3, т.е. под значение каждой переменной будет отведено по 10 позиций. 3 позиции займет дробная часть, 1 – точка, отделяющая целую часть от дробной, и 6 – целая часть (если количество цифр в целой части будет меньше чем 6, то оставшиеся позиции будут заняты пробелами). В выводе текстовых констант изменять ничего не будем. Тогда операторы вывода в данной программе будут выглядеть следующим образом:

```
writeln(kir('Площадь шара равна'),s:10:3);
writeln(kir('Объем шара равен'),v:10:3);
```

На рис. 2.3. показаны результаты работы программы после преобразования выходных данных к виду с фиксированной точкой. Как

видно на рисунке, в этом случае представление данных действительно становится более удобным для восприятия.

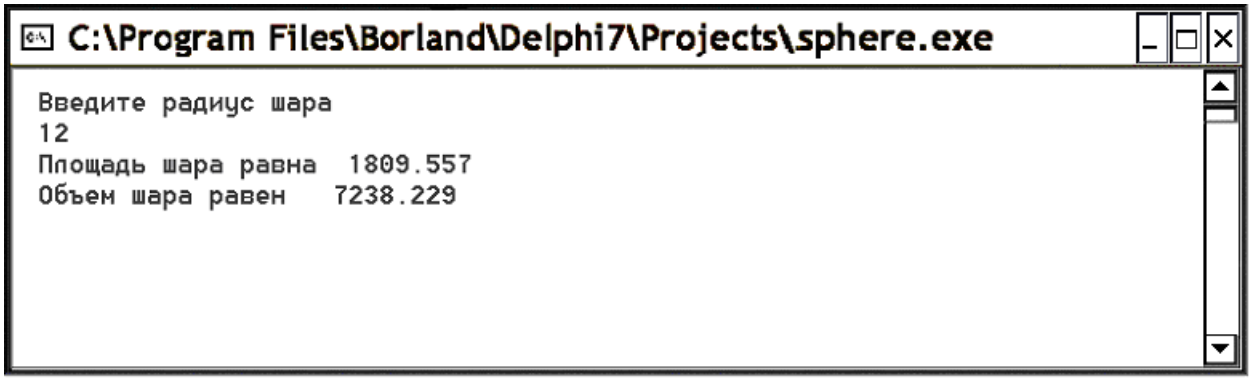

Рис. 2.3. Результаты выполнения программы вычисления площади и объема шара после форматирования выходных данных

Задания для самостоятельной работы

1. Составить программу, которая по заданной длине, ширине и высоте находит объем параллелепипеда и общую площадь его поверхности.
2. Составить программу, которая вычисляет среднее арифметическое трех чисел, введенных пользователем с клавиатуры.
3. Составить программу, которая по заданному радиусу основания и высоте находит объем цилиндра и общую площадь его поверхности.
4. Составить программу, которая пересчитывает размер диагонали экрана компьютера, выраженный в дюймах, в сантиметры. 1 дюйм равен 2,54 см.
5. Составить программу, которая пересчитывает температуру, выраженную по шкале Фаренгейта, в температуру по шкале Цельсия. Пересчет осуществляется по следующей формуле:

$$t_c=(t_f - 32)*5/9,$$

где t_c – температура по шкале Цельсия;

t_f – температура по шкале Фаренгейта.

Глава 3. Операторы выбора

Задачи вычислительного характера, рассмотренные в главе 2 данного пособия, относятся к программам линейной структуры. В таких программах все операторы выполняются последовательно друг за другом.

Но при решении многих поставленных перед программистом задач часто возникает ситуация, когда нужно выбирать один из двух или

нескольких возможных вариантов дальнейшего хода программы. В этом случае используется структура, называемая ветвлением. В программах с ветвлением в зависимости от значений вводимых исходных данных может работать одна из двух или нескольких ветвей.

Такая структура реализуется в программе посредством условных операторов. В заголовке условного оператора содержится некоторое условие, выполнение или невыполнение которого и определяет дальнейшую работу программы. Так как программист заранее не знает, какой из вариантов будет выполняться, выбор должен производить сам компьютер. В качестве условий, истинность или ложность которых проверяется, чаще всего используются операции сравнения различных величин. В языке Паскаль используется 6 таких операций:

< меньше > больше = равно

<= меньше или равно >= больше или равно <> не равно

Написание последних 3 операций отличается от принятого в математике, где соответственно используются обозначения $\leq$, $\geq$ и $\neq$, но так как клавиш с подобными обозначениями нет на клавиатуре компьютера, то используются вышеуказанные сочетания из двух символов.

3.1. Условный оператор

Рассмотрение операторов выбора начнем с того оператора, который осуществляет выбор одного из двух возможных вариантов. В случае истинности проверяемого соотношения выполняется один вариант действий, а в случае ложности – другой вариант. Такой выбор дальнейшего хода действий реализуется с помощью условного оператора **if**…**then.**

Общий вид данного оператора следующий:

```
if  условие then
вариант1
else
вариант2;
```

служебные слова **if, then** и **else** в переводе с английского означают соответственно: если, то, иначе.

Оператор действует следующим образом: сначала идет проверка, выполняется ли условие, находящееся после слова **if**. Если это условие выполняется (иначе говоря, является истинным), то осуществляется первый вариант действий, в противном случае, если условие не выполняется (является ложным) – второй, записанный после служебного слова **else**. В самом простом случае действие, осуществляемое в каждом из вариантов, состоит из одного оператора.

Тот вид условного оператора, который был описан нами выше, представляет собой полную форму условного оператора, но такая форма его записи не является единственно возможной. Наряду с ней в языке Паскаль используется и сокращенный условный оператор. Такой оператор имеет следующий общий вид:

```
if условие then
действие;
```

Сокращенный условный оператор работает следующим образом. Если условие, содержащееся после служебного слова **if**, – истинно, то выполняется действие, записанное после **then**, а если условие ложно, то в условном операторе не выполняется никаких действий, и программа переходит к выполнению следующего оператора, расположенного вслед за данным условным оператором.

Достаточно часто в процессе разработки программ возникает необходимость разместить в одной или обеих ветвях условного оператора не одно, а целый ряд действий. В таком случае в качестве вариантов действий, находящихся в ветвях условного оператора, используют не простые, а составные операторы, которые помещаются в программе после служебных слов **then** или **else**.

Составной оператор представляет собой группу операторов, размещенную между служебными словами **begin** и **end.** Слова **begin** и **end,** называемые в данном случае операторными скобками, обозначают здесь не начало и конец основной части программы, а начало и конец составного оператора. Между ними могут располагаться различные операторы: операторы ввода и вывода, присваивания, вложенные условные операторы и другие виды операторов. Вложенным условным оператором называется условный оператор, входящий в качестве составной части в другой условный оператор. Количество простых операторов, входящих в составной оператор, не ограничено.

Операторы, входящие в составной оператор, друг от друга отделяются точками с запятой. После слова **end** в данном случае точка с запятой не ставится (если только это слово не находится в конце всего условного оператора), так как **end** обозначает здесь лишь конец одного из вариантов, входящих в состав единого условного оператора. Таким образом, в общем виде структуру условного оператора с составными операторами в его ветвях можно представить следующим образом:

```
if условие then
  begin
  оператор 1;
  оператор 2;
  …
  оператор m
  end
```

```
else
  begin
 оператор 1;
 оператор 2;
 …
 оператор n
 end;
```

Составные операторы могут использоваться как в полных, так и в сокращенных условных операторах. Кроме условных операторов составные операторы могут использоваться и в других программных структурах, которые будут рассмотрены в главах 4 и 5.

В качестве примера использования условного оператора рассмотрим следующую задачу. Требуется составить программу решения стандартного квадратного уравнения вида $ax^2+bx+c=0$. Как известно, ход решения такого уравнения зависит от значения величины, которая называется дискриминантом. Дискриминант вычисляется по формуле

$$D=b^2-4ac.$$

В зависимости от значения дискриминанта возможны три варианта решения уравнения:

а) при положительном дискриминанте квадратное уравнение имеет два корня, которые вычисляются по следующим формулам:

$$x_1 = \frac{-b-\sqrt{D}}{2a};$$

$$x_2 = \frac{-b-\sqrt{D}}{2a};$$

б) при дискриминанте, равном нулю, уравнение имеет один корень, который вычисляется по формуле

$$x = \frac{-b}{2a};$$

в) при отрицательном дискриминанте уравнение не имеет решения в действительных числах.

Для того чтобы упростить решение задачи, условимся, что в случае нулевого дискриминанта уравнение также имеет два корня, которые равны между собой. В этом случае можно использовать при расчетах ту же формулу, что и для положительного дискриминанта.

Тогда решение поставленной задачи можно свести к двум вариантам:

а) при неотрицательном дискриминанте уравнение имеет два корня;

б) при отрицательном дискриминанте уравнение не имеет решения.

Для решения задачи в соответствии с указанной схемой можно использовать условный оператор **if..then**, что мы и сделаем в программе решения квадратного уравнения **quadr**.

В начале программы после заголовка и команды подключения модулей, как обычно, описываются переменные. Переменные **a,b** и **c,** соответствующие коэффициентам уравнения, будут описаны как целые числа. Дискриминант **d** также будет описан как целочисленная величина, поскольку при его вычислении используются только операции умножения и вычитания, а операндами являются целочисленные величины.

Переменные **x1** и **x2**, соответствующие корням квадратного уравнения, описаны как вещественные величины. При их вычислении используются операции деления и извлечения квадратного корня, которые всегда дают вещественный результат.

В основной части программы после ввода исходных данных – коэффициентов уравнения – вычисляется дискриминант. Затем вычисленное значение дискриминанта используется в условном операторе, который, в зависимости от этого значения, либо находит два корня уравнения и выводит их на экран компьютера, либо выводит на экран сообщение о том, что уравнение не имеет решения. Этот условный оператор является полным, так как имеет две ветви.

В заголовке оператора указано условие **d>=0**. Если это условие истинно, то выполняется вариант, находящийся после **then**. Этот вариант должен содержат три действия, каждому из которых соответствует отдельный оператор. Этими действиями являются вычисление двух корней уравнения (**x1** и **x2**) и вывод полученных значений на экран компьютера. Таким образом, этот вариант должен быть оформлен как составной оператор, содержащий три простых, и заключен в операторные скобки **begin** и **end**.

При вычислениях корней производится операция извлечения квадратного корня, которая выполняется с помощью стандартной функции **sqrt** (не путать с функцией **sqr** – возведение в квадрат). Аргументом данной функции может быть целое или вещественное число, а значением функции всегда является вещественная величина. При выводе полученных корней уравнения на экран компьютера применяется форматирование выходных значений с указанием общего количества символов и их количества в дробной части, подобно тому, как это было описано в главе 2 пособия.

Таким образом, данный составной оператор приобретает следующий вид:

```
begin
x1:=(-b-sqrt(d))/(2*a);
x2:=(-b+sqrt(d))/(2*a);
writeln(kir('Корни     уравнения     равны'),x1:8:2,
x2:8:2);
end
```

Этот составной оператор, в свою очередь, является частью условного оператора. Вторая часть условного оператора находится после слова **else**. Эта часть представляет собой простой оператор вывода, который выводит на экран сообщение о том, что уравнение не имеет решения. Эта вторая ветвь будет выполняться в том случае, если условие **d>=0**, записанное в заголовке условного оператора, окажется ложным.

Ниже приводится полный текст программы **quadr**.

```
program quadr;

{$APPTYPE CONSOLE}

uses
SysUtils;

var a,b,c,d:integer;
    x1,x2:real;

function kir(st:string):string;
 var i,k:integer;
begin
 for i:=1 to length(st) do
 begin
  k:=ord(st[i]);
  case st[i] of
  'A'..'п': st[i]:=Chr(k-64);
  'р'..'я': st[i]:=Chr(k-16);
  end;
 end;
kir:=st;
end;

begin
writeln(kir('Введите коэффициенты уравнения a, b и
c'));
readln(a,b,c);
d:=b*b-4*a*c;
if d>=0 then
begin
x1:=(-b-sqrt(d))/(2*a);
x2:=(-b+sqrt(d))/(2*a);
writeln(kir('Корни    уравнения    равны'),x1:8:2,
x2:8:2);
end
else
writeln (kir('Уравнение не имеет решения'));
readln;
end.
```

На рис. 3.1 и 3.2 показаны результаты работы программы при различных исходных данных. В первом случае дискриминант положителен, и поэтому на экран выводятся в виде с фиксированной точкой два вычисленных значения. Во втором случае дискриминант отрицателен, и программа выводит на экран сообщение о том, что в данном случае уравнение не имеет решения в действительных числах.

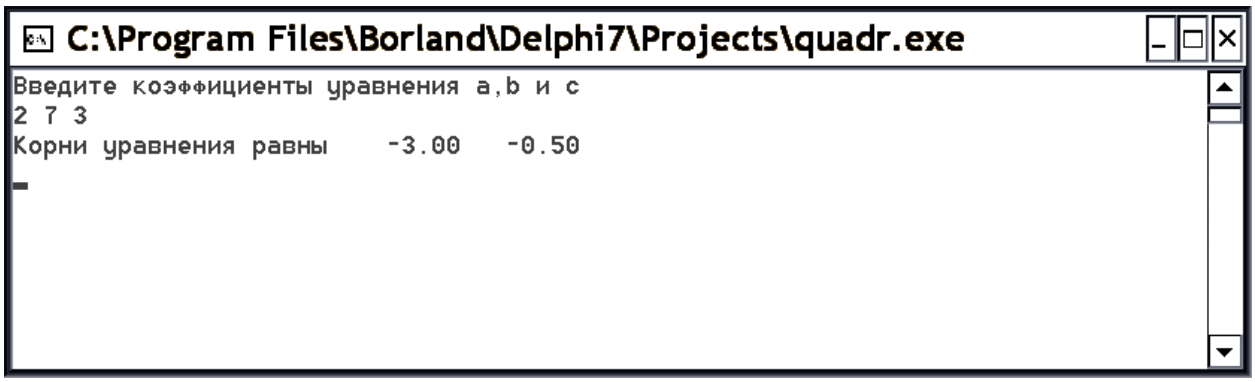

Рис. 3.1. Результаты работы программы решения квадратного уравнения при положительном дискриминанте

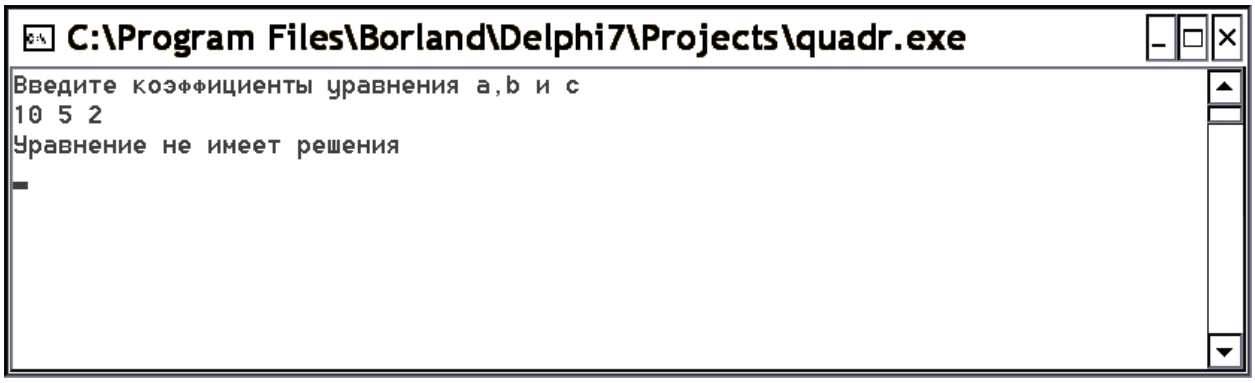

Рис. 3.2. Результаты работы программы решения квадратного уравнения при отрицательном дискриминанте

3.2. Оператор множественного выбора

При составлении программ часто возникает потребность в структуре, которая обеспечивала бы возможность рассмотреть не два, а большее количество возможных вариантов дальнейших действий. В некоторых случаях для этого используется оператор **`if`**. В одну или обе ветви оператора вставляется еще по одному условному оператору **`if`**, который позволяет разбить каждый вариант на два подварианта, т. е., как говорят, используют вложенные условные операторы. В каждый из вложенных условных операторов можно вставить следующий вложенный оператор и т.д.

Такой способ создания многовариантного ветвления делает, однако, структуру программы чересчур сложной, что, понятно, может привести к появлению в программе ошибок. Поэтому в языке Паскаль предусмотрена

другая конструкция, которая позволяет осуществлять выбор из множества возможных вариантов и в то же время является более простой и наглядной, чем вышеописанная. Эта конструкция реализуется с помощью оператора множественного выбора **case**.

Общий вид оператора **case** следующий:

```
case селектор of
Значение1: Вариант1;
Значение2: Вариант2;
.............................
Значение n: Вариант n;
else  Вариант n+1;
end;
```

где **case, of , else** – служебные слова **Object Pascal**.

Словосочетание **case**…**of**, находящееся в заголовке оператора, переводится на русский, как «в случае если». Между этими словами находится переменная-селектор. Селектором называется переменная целого или символьного типа. Эта переменная может принимать ряд различных значений.

После заголовка в операторе идет перечень возможных значений переменной-селектора. Каждому из этих значений соответствует определенный вариант действий, который реализуется в том случае, если в программе селектор принимает это значение. Этот вариант указывается после двоеточия, отделяющего его от значения, и представляет собой простой или составной оператор.

Если переменная-селектор не принимает ни одного из перечисленных в операторе **case** значений, то выполняется альтернативный вариант, который указан после служебного слова **else**. Обратите внимание на то, что в отличие от условного оператора **if**…**then** в операторе **case** перед вариантом с **else** ставится точка с запятой. Эта часть оператора (**else** с соответствующим ему вариантом) не является обязательной. Возможен сокращенный вариант оператора без этой части. Заканчивается оператор множественного выбора служебным словом **end**, после которого ставится точка с запятой.

Следует отметить, что каждому из вариантов, описанных в операторе множественного выбора, может соответствовать не одно значение переменной селектора, а целый ряд значений. Этот ряд значений может задаваться двумя способами:

а) в виде значений, перечисляемых перед соответствующим вариантом через запятую:

```
Значение1, Значение 2,… Значение k: вариант;
```

б) если значения образуют непрерывный диапазон, можно указать только начальное значение, две горизонтальные точки и конечное значение. Затем после двоеточия нужно указать общий для них вариант:

```
Значение 1 .. Значение k: вариант;
```

Используем оператор множественного выбора для решения следующей задачи. Требуется по введенному с клавиатуры номеру месяца определить соответствующее время года.

Программа, решающая поставленную задачу, использует в основной своей части всего одну переменную **n**. Это переменная целого типа используется в программе в качестве переменной-селектора для оператора множественного выбора. Внутри оператора описаны 4 возможных набора значений для переменной **n**.

Первый набор значений, соответствующий номерам зимних месяцев, не образует непрерывного диапазона (январь, февраль и декабрь имеют соответственно номера 1,2 и 12). Поэтому данные значения перечисляются через запятую, а затем для них указывается общий вариант – сообщение о том, что это зимний месяц. Соответствующий фрагмент оператора множественного выбора будет выглядеть таким образом:

```
1,2,12: writeln(kir('Это зимний месяц'));
```

Следующий набор значений – это весенние месяцы, номера которых образуют непрерывный диапазон значений (март, апрель и май имеют номера 3, 4 и 5). Поэтому в данном случае можно указать только начальный и конечный номера диапазона, а между ними поставить две горизонтальные точки. Фрагмент оператора множественного выбора для данного диапазона будет выглядеть так:

```
3..5: writeln(kir('Это весенний месяц'));
```

Аналогичным образом задаются непрерывные диапазоны значений для летних и осенних месяцев.

Осталось рассмотреть еще один последний вариант, при котором пользователь вводит несуществующий номер месяца. Ошибочным является любой номер месяца, который не принимает ни одного из допустимых значений (от 1 до 12). В этом случае для защиты программы от ошибочного ввода данных используется ветвь оператора множественного выбора, находящаяся после слова **else**. Эта ветвь представляет собой оператор, выдающий сообщение об ошибке.

Полный текст программы **month**, определяющей время года по номеру введенного месяца, приводится ниже. Под ним находится рис.3.3 с окном, содержащим результат работы программы.

```
program month;

{$APPTYPE CONSOLE}
```

```
uses
SysUtils;

var n:integer;

function kir(st:string):string;
 var i,k:integer;
begin
 for i:=1 to length(st) do
 begin
  k:=ord(st[i]);
  case st[i] of
  'A'..'п': st[i]:=Chr(k-64);
   'р'..'я': st[i]:=Chr(k-16);
  end;
 end;
kir:=st;
end;

begin
writeln(kir('Введите номер месяца'));
readln(n);
case n of
1,2,12: writeln(kir('Это зимний месяц'));
3..5: writeln(kir('Это весенний месяц'));
6..8: writeln(kir('Это летний месяц'));
9..11: writeln(kir('Это осенний месяц'));
else writeln(kir('Месяца с таким номером не существует'));
end;
readln;
end.
```

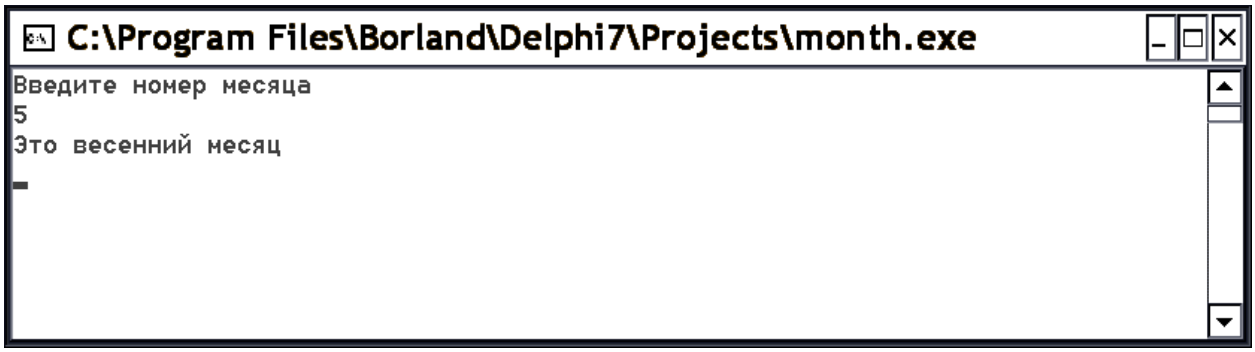

Рис. 3.3. Результаты работы программы, которая определяет время года по номеру месяца

Задания для самостоятельной работы

1. Составить программу, которая определяет большее из 3 чисел, введенных пользователем с клавиатуры.

2. Составить программу, которая определяет, является ли веденное с клавиатуры число положительным числом, нулем или отрицательным числом.

3. Составить программу, которая вводит с клавиатуры 3 числа, и проверяет, являются ли они пифагоровыми числами (пифагоровыми называются такие три числа, у которых сумма квадратов двух чисел равна квадрату третьего).

4. Составить программу, которая вводит с клавиатуры некоторое число, и вычисляет его куб, если оно положительное, и квадрат, если оно отрицательное.

5. Составить программу, которая вводит с клавиатуры 3 числа, и находит сумму только тех из них, которые являются положительными.

6. Составить программу, которая предлагает пользователю ввести два числа, а затем находит сумму этих чисел, сумму их квадратов или среднее арифметическое. Выбор действия, производимого над числами, производится с помощью меню, в котором для выбора нужно ввести числа 1, 2 или 3.

Глава 4. Циклические операторы

В ходе создания программ нередко возникает ситуация, когда программисту нужно многократно использовать один и тот же оператор или группу операторов, которые аналогичны друг другу и отличаются лишь значениями некоторых переменных или вводимых исходных данных.

Естественно, что возникает вопрос: нельзя ли использовать в таких случаях какую-либо структуру, которая бы позволяла облегчить и сделать менее трудоемким решение задачи путем автоматического повторения данного оператора или группы операторов с соответствующим изменением значений переменной или переменных. Такая структура в языке Паскаль существует и называется циклом.

В общем виде цикл состоит из двух основных блоков:

1. Заголовок цикла. В заголовке цикла содержится некоторое условие, которое определяет число повторений цикла.

2. Тело цикла. Телом цикла называется простой или составной оператор (как известно, составным оператором называется группа операторов, заключенная в операторные скобки), который может многократно повторяться в ходе работы цикла. В теле цикла могут содержаться операторы различных типов. Это могут быть операторы ввода и вывода, операторы присваивания, условные операторы, а также другие операторы цикла. Оператор цикла, находящийся в теле другого циклического оператора, называется вложенным.

При работе цикла постоянно производится проверка содержащегося в заголовке условия. В зависимости от истинности условия и вида цикла либо очередной раз выполняются операторы тела цикла, либо цикл завершается. Если выполняются операторы тела цикла, то по завершении этого процесса вновь проверяется условие в заголовке и все начинается сначала. Если же работа цикла завершается, то управление программой переходит к оператору, расположенному в программе вслед за циклом.

В правильно работающем цикле не должно происходить бесконечного повторения тела цикла, приводящего к зацикливанию программы. Поэтому условие, находящееся в заголовке, должно быть составлено таким образом, чтобы в какой-то момент в ходе выполнения программы оно приводило бы к завершению работы цикла.

Циклы могут использоваться для решения самых разнообразных задач, и, соответственно, структура самих циклов может несколько отличаться. Всего в языке Паскаль используется 3 разновидности циклов:

1) цикл с заранее заданным числом повторений. Его также называют циклом со счетчиком (цикл **`for`**…**`to`**);

2) цикл с предусловием (цикл **`while`**…**`do`**);

3) цикл с постусловием (цикл **`repeat...until`**).

Каждый из этих видов цикла имеет свои особенности и область применения, которые мы и рассмотрим ниже. Если количество повторений тела цикла заранее известно, то рекомендуется использовать оператор цикла

for…**to**. Если нужно, чтобы тело цикла выполнялось до тех пор, пока верно некоторое условие, то используется цикл **while**…**do**. При этом тело цикла может быть ни разу не выполнено. Если же нужно, чтобы тело цикла также выполнялось в зависимости от некоторого условия, но при этом гарантированно было выполнено хотя бы один раз — в этом случае используется цикл **repeat**…**until**. Далее перейдем к более подробному рассмотрению каждого из этих видов цикла.

4.1. Цикл с заранее заданным числом повторений

Общий вид данного оператора следующий:

```
for счетчик_цикла:=нач_значение to кон_значение do
тело цикла;
```

где **for, to** и **do** – служебные слова (**for** в данном случае означает «для», **to** – «до», **do** – «делать, выполнять»);

счетчик_цикла – управляющая переменная цикла, значение которой изменяется от начального до конечного. После того как переменная-счетчик цикла примет конечное значение, тело цикла выполнится последний раз и цикл будет завершен. Переменная-счетчик обычно относится к целочисленному типу, но допускаются и некоторые другие типы. Категорически нельзя использовать в качестве счетчика переменные вещественного типа;

нач_значение, кон_значение – начальное значение и конечное значение счетчика цикла. При этом конечное значение должно быть больше начального. Оба значения должны быть того же типа, что и счетчик цикла;

тело цикла - в качестве тела цикла выступает обычный оператор или составной оператор, в который входит несколько других.

Если переменная-счетчик имеет тип **integer**, то в ходе работы цикла ее значение при каждом выполнении тела цикла увеличивается на единицу и таким образом принимает все целочисленные значения от начального до конечного. Следует обратить внимание на то, что между заголовком цикла и телом цикла не ставится точка с запятой.

Приведенный вид записи цикла **for** не является единственным. Цикл может иметь и такой общий вид:

```
for счетчик_цикла:=нач_значение downto кон_значение do
тело цикла;
```

В этом случае при каждом повторении тела цикла значение счетчика уменьшается на единицу, и, следовательно, конечное значение счетчика цикла должно быть меньше начального.

Рассмотрим практическое применение оператора цикла с заранее заданным количеством повторений на следующем примере. Требуется составить программу, которая вычисляет факториал числа **n**. Факториалом

натурального числа, как известно, называется произведение всех натуральных чисел от единицы до самого этого числа включительно.

В программе **faktor**, используемой для вычисления факториала, в соответствующем разделе описываются 3 переменные. Исходная величина **n** и переменная цикла **i** описывается как переменные обычного целого типа **integer.**

Но для переменной **f**, содержащей результат вычислений, данный тип не подходит. Тип **integer** может использоваться только для переменных, которые могут принимать значения в диапазоне от -32768 до 32767. Но факториал числа 7 равен 5040, а факториал 8 уже равен 40320, т.е. это число уже выходит за границы диапазона, и правильного результата при вычислениях не получится.

Для описания переменной **f** будем использовать тип **longint.** Переменные этого типа могут принимать значение до 2 миллиардов и лучше подходят для поставленной задачи. Использование этого типа позволит правильно производить вычисления со значениями n от 1 до 12. Но для бóльших значений **n** не годится и этот тип, поэтому в программе необходимо установить ограничение на исходное значение **n.**

Это ограничение в программе реализовано в виде условного оператора **if**…**then**, который находится в основной части программы после оператора ввода. В заголовке этого оператора указаны 2 условия. Эти условия объединены с помощью логической операции **or**, что в переводе с английского означает «**или**». Если выполняется или одно условие (**n<1**), или другое условие (**n>12**), то значение исходной величины **n** выходит за допустимые для нее границы, и программа выводит сообщение об ошибке.

В том случае, если оба этих условия ложны, значение **n** находится в допустимых пределах, можно приступать к вычислению значения факториала. Вычислительные операции находятся в той ветви условного оператора, которая расположена после слова **else.** Эта ветвь представляет собой составной оператор, ограниченный операторными скобками **begin** и **end.**

В начале этого составного оператора находится простой оператор, который присваивает переменной **f** значение, равное единице. Это значение является наименьшим возможным значением факториала, так как наименьшим натуральным числом является 1, а значение факториала единице равно самому этому числу.

Следующим простым оператором, находящимся внутри той же ветви, является оператор цикла. Этот оператор для всех натуральных чисел в диапазоне от 1 до **n** выполняется одну и ту же операцию: умножает предыдущее значение факториала на текущее число. По завершении работы данного цикла мы получаем искомый результат. Этот результат остается только вывести на экран компьютера, что делает последний оператор, находящийся внутри этот ветви.

Полный текст программы **factor**:

```
program faktor;
{$APPTYPE CONSOLE}
uses
 SysUtils;
function kir(st:string):string;
 var i,k:integer;
begin
 for i:=1 to length(st) do
 begin
  k:=ord(st[i]);
  case st[i] of
   'A'..'п': st[i]:=Chr(k-64);
   'р'..'я': st[i]:=Chr(k-16);
  end;
 end;
 kir:=st;
end;
var i,n:integer;
    f:longint;
begin
  writeln(kir('Введите натуральное число от 1 до
12'));
  readln(n);
  if (n<1) or (n>12) then
      writeln(kir('Ошибка ввода данных'))
  else
     begin
     f:=1;
     for i:=1 to n do
     f:=f*i;
     writeln(kir('Факториал числа '),n,kir(' равен
'),f);
     end;
   readln;
end.
```

На рис. 4.1 и 4.2 показаны результаты работы программы **factor** при различных исходных данных. На первой иллюстрации показано, что при попытке ввести неверные исходные данные программа блокирует ввод и выдает сообщение об ошибке. На второй иллюстрации показан результат вычислений при правильном вводе исходных данных.

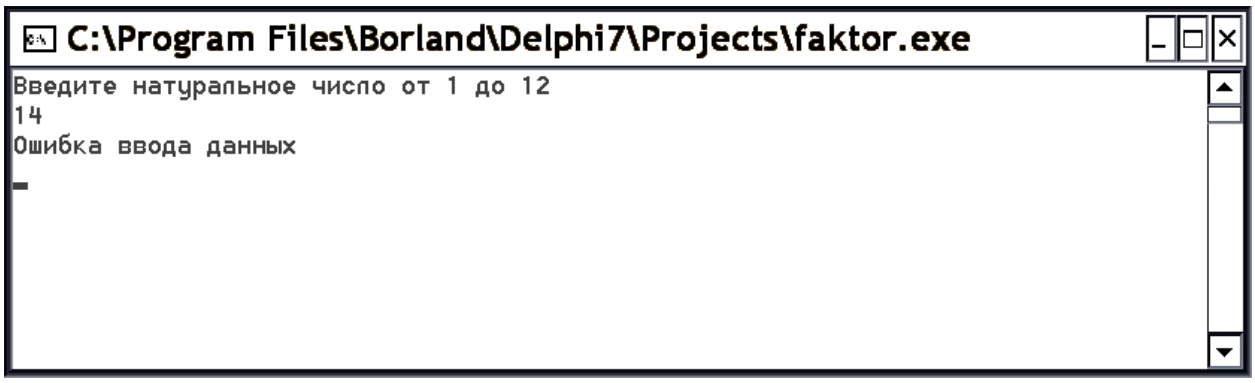

Рис. 4.1. Результаты работы программы вычисления факториала при ошибочном вводе данных

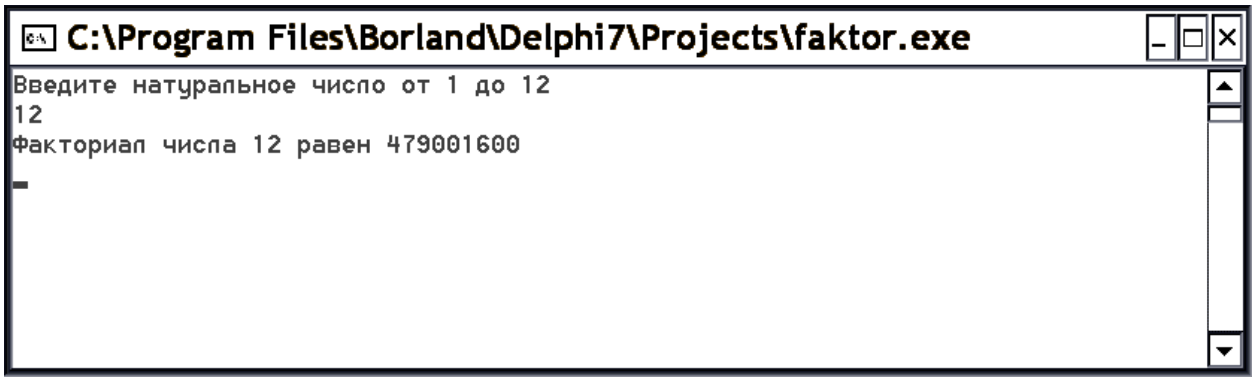

Рис. 4.2. Результаты работы программы вычисления факториала при правильном вводе данных

4.2. Циклы с заранее неопределенным количеством повторений

В том случае, если количество повторений цикла заранее определить невозможно в **Object Pascal** используют следующие виды циклов: цикл с постусловием и цикл с предусловием.

Общий вид оператора цикла с постусловием следующий:

```
repeat
тело цикла
until условие;
```

где **repeat** и **until** – служебные слова. Слово **repeat** в переводе с английского означает «повторять», **until** – «до тех пор». Цикл работает следующим образом: вначале выполняется оператор или группа операторов, входящих в тело цикла. Затем проверяют, выполняется ли условие, находящееся после **until.** Если условие истинно, то работа цикла на этом завершается, если условие ложно, то тело цикла выполняется снова, после чего выполняется очередная проверка и т.д. Таким образом, данный цикл будет работать до тех пор, пока условие находящееся после слова **until**, является ложным.

Такой цикл называют циклом с постусловием (латинская приставка «пост» означает «после»). Это название цикла связано с тем, что проверка условия, определяющего работу цикла, производится уже после его выполнения. По определению цикла понятно, что даже если условие вообще никогда не выполняется, то тело цикла будет выполнено хотя бы один раз.

Общий вид оператора цикла с предусловием следующий:

```
while   условие   do
тело цикла;
```

где **while** и **do** – служебные слова, **while** в переводе с английского означает «пока», **do** – «делать, выполнять».

Цикл работает следующим образом: первоначально проверяется условие, находящееся после слова **while**, если данное условие является ложным, то работа цикла на этом и завершается. Если условие является истинным, то выполняется оператор или группа операторов, входящая в тело цикла. Затем снова производится проверка условия, содержащегося в строке заголовка и т. д. Таким образом, тело цикла выполняется пока истинно условие, содержащееся в заголовке цикла. Цикл такого типа называют циклом с предусловием. Это объясняется тем, что перед первым же выполнением цикла производится проверка истинности находящегося в заголовке условия. Если условие изначально не выполняется, то тело цикла не будет выполнено ни разу.

Рассмотрим пример программы, в которой используется цикл с заранее неизвестным количеством повторений (в данном случае — это цикл с постусловием). Программа **sum_series** используется для решения следующей задачи. Требуется определить сумму элементов ряда. Элементами ряда являются натуральные числа. Число элементов ряда заранее неизвестно. Для завершения ввода элементов необходимо ввести ноль. Ноль является только признаком окончания ввода, а не элементом ряда, т.к. ноль – не натуральное число (как известно, наименьшим натуральным числом является единица). Требуется определить число элементов ряда, сумму элементов ряда и среднее арифметическое ряда.

В соответствующем разделе программы описываются используемые в программе переменные. Переменная **x** содержит значение очередного элемента ряда и относится к целочисленным переменным. Также целочисленной является переменная **k** – счетчик количества элементов ряда. В этом разделе описаны также 2 вещественные переменные. Это сумма элементов ряда **sum** и среднее арифметическое ряда **a.**

В основной части программы перед циклом находятся операторы, которые производят начальные присваивания переменным **k** и **sum.** Если в ячейках памяти компьютера, соответствующих указанным переменным, на момент начала работы цикла будут содержаться какие-либо значения, то эти значения могут исказить результаты вычислений, производимых в

программе. Поэтому для очистки этих ячеек переменные **k** и **sum** следует обнулить, что и делают операторы, выполняющие начальное присваивание.

Затем в программе организуется цикл, в котором происходит ввод данных – элементов ряда, подсчитывается количество этих элементов и вычисляется их сумма. Хотя тело цикла, в котором выполняются все эти действия, фактически является составным оператором, оно не требует операторных скобок **begin** и **end**. В данном случае роль операторных скобок выполняют ключевые слова **repeat** и **until**. Как только в цикле вводится очередной элемент, счетчик количества элементов **k** увеличивается на единицу, что видно в соответствующем операторе присваивания. Другой оператор присваивания прибавляет значение текущего элемента к общей сумме элементов, содержащейся в переменной **sum**.

Условием завершения работы цикла является равенство очередного вводимого числа нулю. Как только это равенство становится истинным, цикл завершается свою работу. При этом следует иметь в виду, что количество элементов ряда **k**, подсчитанное в цикле, оказывается на единицу больше действительного. Данное обстоятельство объясняется тем, что при последнем выполнении тела цикла счетчик элементов учитывает ноль, который на самом деле элементом не является. Этот ноль также приплюсовывается и к переменной **sum** – сумме элементов ряда, но здесь он не оказывает влияние на итоговое значение.

Далее в программе выводятся результаты вычислений. Количество элементов выводится как значение переменной **k**, уменьшенное на единицу. Затем выводится значение переменной **s** – сумма элементов ряда. Наконец, находится среднее арифметическое элементов ряда **a**, которое равно сумме элементов ряда, деленной на количество элементов. Значение переменной **a** также выводится на экран. На этом программа завершает работу. Ниже приводится полный текст программы **sum_series**:

```
program sum_series;

{$APPTYPE CONSOLE}
uses
  SysUtils;

function kir(st:string):string;
 var i,k:integer;
begin
 for i:=1 to length(st) do
 begin
  k:=ord(st[i]);
  case st[i] of
  'A'..'п': st[i]:=Chr(k-64);
  'р'..'я': st[i]:=Chr(k-16);
  end;
```

```
 end;
 kir:=st;
end;

var x,k:integer;
    sum,a:real;

begin
  k:=0;  sum:=0;
  writeln(kir('Ввод элементов ряда'));
  repeat
   writeln(kir('Введите очередной элемент ряда - натуральное число'));
   writeln(kir('Для завершения ввода введите ноль'));
   readln(x);
   k:=k+1;
   sum:=sum+x;
  until x=0;
  writeln(kir('Количество элементов ряда равно '),k-1);
  writeln(kir('Сумма элементов ряда равна '),sum:8:3);
  a:=sum/(k-1);
  writeln(kir('Среднее арифметическое ряда равно '),a:8:3);
  readln;
end.
```

На рис. 4.3 показан ввод данных, производимый в программе **sum_series**, и вывод результатов работы программы на экран.

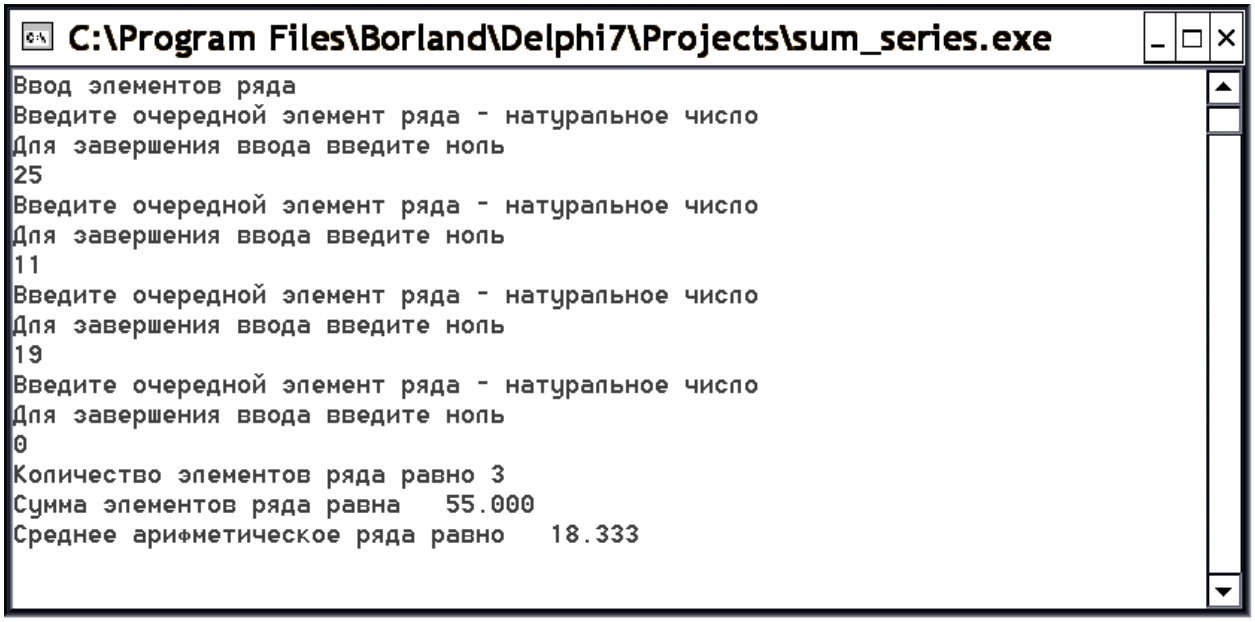

Рис. 4.3. Результаты работы программы, которая вычисляет сумму элементов ряда

Задания для самостоятельной работы

1. Составить программу, которая выводит на экран компьютера сумму квадратов всех натуральных чисел от 1 до **n**, где **n** – натуральное число, введенное пользователем с клавиатуры.

2. Составить программу, которая выводит на экран компьютера сумму квадратных корней всех натуральных чисел от 1 до **n**, где **n** – натуральное число, введенное пользователем с клавиатуры.

3. Клиент положил в банк сроком на **n** лет некоторую денежную сумму под определенный процент. Составить программу, которая бы определяла, какую величину составит сумма вклада по истечении срока хранения данного вклада, если по условиям договора о вкладе, заключенного между клиентом и банком, в течение всего срока хранения вклада банковский процент не должен изменяться. Сумма вклада, срок действия договора и процент по вкладу вводятся в компьютер с клавиатуры.

4. Определить средний балл, полученный студентами группы, состоящей из **n** человек, по итогам сдачи экзамена по какой-либо дисциплине. Данные о количестве студентов в группе и оценка каждого студента вводятся в компьютер с клавиатуры.

5. В университете проводится олимпиада по информатике. Принять участие в олимпиаде могут только студенты, которые сдали экзамен по информатике на оценку не ниже 4. Составить программу, которая подсчитывает, сколько студентов из одной группы могут принять участие в олимпиаде. Оценка каждого студента по информатике вводится в компьютер с клавиатуры. Общее количество студентов в группе заранее неизвестно, но известно, что признаком окончания ввода данных является ввод нуля.

Глава 5. Массивы

Массив представляет собой упорядоченную последовательность однородных элементов. Элементами массива могут быть различные величины, как числовые (целые и вещественные), так и других типов. Но при этом строго соблюдается следующее правило: все элементы каждого, отдельно взятого, массива должны относиться к одному и тому же типу, что называется однородностью массива. В массиве каждый элемент имеет свой порядковый номер, который называется индексом. Массивы в **Object Pascal** могут быть одномерными и двумерными.

Примером простейшего одномерного массива является список учеников одного школьного класса или студентов одной группы. В этом случае элементами массива будут фамилии учеников или студентов, а индексами – номера учеников или студентов в списке. Важной характеристикой массива является его диапазон, т. е. пределы, в которых может изменяться значение индекса массива.

В случае использовании массива в программе, он предварительно должен быть описан в разделе описания переменных. Но описывается массив иначе, чем обычная переменная. В общем виде описание массива, состоящего из элементов-переменных, выглядит следующим образом:

```
var  имя_массива: array[a..b] of тип_элементов;
```

где **var, array** и **of** – служебные слова. Слово **array** в переводе с английского означает «массив», предлог **of** в данном случае – «из»;

a и **b** – соответственно, нижняя и верхняя границы диапазона массива;

имя_массива – задается по тем правилам, что и имена переменных;

тип_элементов – любой из стандартных типов элементов, имеющихся в Object Pascal.

В программе можно работать не только со всем массивом целиком, но и с отдельными его элементами. Для того чтобы обратиться в программе к какому-либо элементу массива, нужно указать имя массива и индекс содержащегося в нем элемента. Например, если в программе имеется запись: **a[10]**, то она означает, что мы обращаемся к элементу массива **a** с порядковым номером 10.

Простейшим примером двумерного массива является таблица умножения, в которой результат умножения двух чисел определяется по номеру строки, соответствующей одному из сомножителей, и номеру столбца, соответствующему другому. Соответственно, в любом двумерном массиве элемент определяется по двум индексам.

В языке Object Pascal в общем виде двумерные массивы описываются следующим образом:

```
var  имя_массива: array[a..b,c..d]  of  тип_элементов;
```

где **a** и **b** – соответственно верхняя и нижняя границы диапазона значений для первого индекса; **c** и **d** - верхняя и нижняя границы диапазона значений для второго индекса.

5.1. Основные операции с массивами

Рассмотрим основные операции с массивами на следующем примере. Требуется создать программу, которая заполняет целочисленный массив из 20 элементов случайными числами. Элементы массива могут иметь значение от 1 до 50. Все элементы массива необходимо вывести на экран компьютера в одну строку. Затем следует подсчитать количество четных и нечетных элементов, имеющихся в данном массиве, и также вывести вычисленные значения на экран. Эта задача будет выполнена с помощью программы **array1.**

В начале программы, как обычно, описываются используемые переменные. Все переменные относятся к целочисленному типу. Переменная **n** является переменной цикла с заранее известным количеством повторений, с помощью которого заполняется массив. Переменной **a** присваивается значение очередного элемента массива. Переменные **even** и **uneven** являются, соответственно, счетчиками количества четных и нечетных элементов, имеющихся в массиве. В том же разделе программы, что и переменные, описывается одномерный целочисленный массив **m**, содержащий 20 элементов.

Для заполнения массива элементами в программе применяется генератор случайных чисел. Генератором случайных чисел называется специальная подпрограмма, входящая в состав языка **Object Pascal**, которая выдает случайным образом выбранные числа. Для того чтобы генератор случайных чисел можно было использовать, его следует предварительно запустить в действие или, как говорят инициализировать. Инициализация генератора случайных чисел производится процедурой **randomize**. Данная процедура представляет собой самостоятельный оператор **Object Pascal** и поэтому записывается в программе на отдельной строке:

```
randomize;
```

Следующее, что требуется сделать, – это взять некоторое число из созданного ряда случайных чисел. Это действие производится посредством стандартной функции **random(n),** где **n** – указываемое пользователем целое число или имя целочисленной переменной. Функция **random** будет выбирать числа, которые находятся в диапазоне от **0** до **n-1**. Команда **random** не является отдельным оператором, но может входить в качестве составной части, например, в оператор вывода. Эта функция также может быть частью математического выражения в операторе присваивания.

Если в программе будет, например, указана команда **random(10)**, то данная команда выдаст случайное число из ряда 0,1,2,…..9, т.е. первое число

ряда равно 0, а последнее будет на единицу меньше, чем указанное в круглых скобках.

В рассматриваемой программе в начале основной части производится инициализация генератора случайных чисел. Затем обнуляются счетчики четных и нечетных чисел **even** и **uneven.** Далее сообщается о том, что будет производиться вывод элементов массива.

Следующим оператором в программе является цикл с заранее заданным количеством повторений. Тело цикла представляет собой составной оператор. В теле цикла выполняются 3 задачи.

Во-первых, с помощью оператора присваивания заполняется очередной элемент массива **m[n]**. В правой части оператора находится выражение, содержащее функцию **random.** К значению, созданному функцией **random,** прибавляется единица, так как по условиям поставленной задачи необходимо, чтобы диапазон возможных значений элемента массива начинался не с нуля, а с единицы.

Во-вторых, выясняется, является ли очередной элемент массива четным или нечетным числом. Для этого значение элемента массива присваивается целочисленной переменной **a.** Затем переменная a используется в качестве аргумента стандартной функции **odd.** Данная функция относится к разряду логических.

Логической называется такая функция, которая может иметь только два значения: **true** или **false,** т.е. в переводе с английского, «**истина**» или «**ложь**». Аргументом функции может быть только целочисленная величина. Если аргумент функции представляет собой нечетное число, то значение функции будет равно **true**, если же аргумент нечетный, то значение будет равно **false.**

В рассматриваемой программе функция **odd** находится в заголовке условного оператора **if**... **then.** Аргументом функции **odd** не может быть элемент массива, поэтому значение элемента присваивается вспомогательной переменной **a**, а затем эта переменная подставляется в функцию. При значении функции **odd**, равном **true,** выполняется ветвь условного оператора, находящаяся после **then**, и прибавляется единица к счетчику нечетных чисел **uneven**. При значении функции, равном **false,** выполняется ветвь, находящаяся после **else**, и единица добавляется к счетчику четных чисел **even.**

В-третьих, необходимо вывести на экран компьютера очередной элемент массива. Для этой цели используется оператор **write** (а не **writeln**), так как вывод данных должен производиться в одну строку.

Для выводимого элемента массива указывается формат вывода, т.е. после имени элемента ставится двоеточие, и далее указывается общее количество символов выводимой величины (в данном случае это количество равно 3). Поскольку значения элементов массива являются однозначными или двузначными величинами, а под каждый элемент выделяются по 3

позиции, оставшиеся позиции будут заполнены пробелами, которые будут разделять между собой выводимые значения.

По завершении работы цикла на экран компьютера будет выведено подсчитанное в цикле количество четных и нечетных элементов массива. Ниже представлен полный текст программы **array1**:

```
program array1;

{$APPTYPE CONSOLE}

uses
SysUtils;

var a,n,even,unev:integer;
    m:array[1..20] of integer;

function kir(st:string):string;
 var i,k:integer;
begin
 for i:=1 to length(st) do
 begin
  k:=ord(st[i]);
  case st[i] of
  'A'..'п': st[i]:=Chr(k-64);
  'р'..'я': st[i]:=Chr(k-16);
  end;
 end;
 kir:=st;
end;

begin
  randomize;
  even:=0;
  unev:=0;
  writeln(kir('Вывод элементов массива'));
  for n:=1 to 20 do
  begin
  m[n]:=random(50)+1;
  a:=m[n];
  if odd(a) then
     unev:=unev+1
  else
     even:=even+1;
  write(m[n]:3);
  end;
  writeln;
```

```
      writeln(kir('Количество четных элементов массива
равно '),even);
      writeln(kir('Количество нечетных элементов
массива равно '),unev);
      readln;
    end.
```

Результаты работы программы **array1**, в которой представлены основные операции, производимые с массивами, показаны на рис. 5.1.

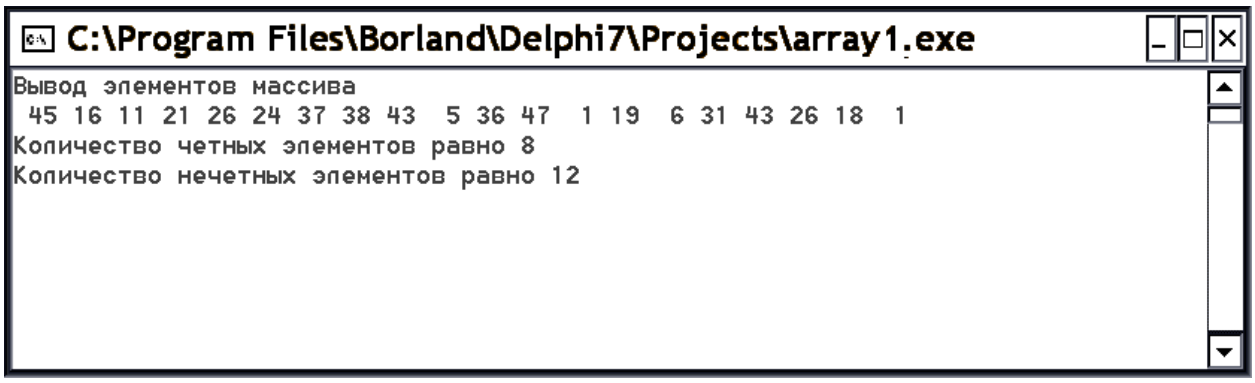

Рис. 5.1. Результаты работы программы, определяющей количество четных и нечетных элементов целочисленного массива

5.2. Сортировка элементов массива

Одной из наиболее распространенных задач (помимо описанных в предыдущем параграфе), с которыми приходится сталкиваться при обработке массивов, является сортировка массивов, т.е. расположение элементов массивов определенным упорядоченным образом. Можно встретить следующие основные способы сортировки массивов:

1. Сортировка по возрастанию. В этом случае элементы массива располагаются таким образом, что в начале его располагается наименьший элемент, затем следующий по величине элемент и так далее вплоть до наибольшего, т.е. каждый следующий элемент должен быть больше предыдущего.

2. Сортировка по убыванию. В данном случае после проведения сортировки в начале массива оказывается наибольший по величине элемент, затем идет следующий по величине и так далее вплоть до наименьшего, т. е. каждый следующий элемент должен быть меньше предыдущего.

Существует несколько различных способов сортировки массивов. Мы разберем в самый простой способ, который называется «сортировка методом простого выбора». Суть этого способа в том случае, если нужно упорядочить массив по возрастанию, сводится к следующему: в массиве нужно найти наибольший по величине элемент и поставить его на последнее место в массиве. В результате наибольший элемент массива займет подобающее ему место. Тот элемент, который ранее находился в массиве на последнем месте,

необходимо переместить на то место, которое ранее занимал в массиве наибольший элемент.

Далее работаем с группой элементов, которая начинается с первого элемента массива, а заканчивается предпоследним элементом. В этой группе также следует найти наибольший элемент и затем поставить его на последнее место в данной группе элементов, т. е. на предпоследнее место во всем массиве. Элемент, стоявший предпоследним, также помещается в массиве на освободившееся место.

Затем рассматривается группа элементов без последнего и предпоследнего. В этой группе также находится наибольший элемент, который помещается на положенное ему последнее в группе место. Такие операции будут продолжаться до тех пор, пока каждый элемент массива не займет подобающее ему место. Последняя операция сведется к выбору наибольшего в группе из двух оставшихся элементов. Описанный метод реализован в программе **array2**.

Поставленная задача выглядит следующим образом. Необходимо заполнить целочисленный массив из 20 элементов с помощью генератора случайных чисел. Элементы массива могут принимать значение от 1 до 99. Этот массив следует вывести на одну строку на экран компьютера. Затем элементы массива требуется упорядочить по возрастанию, и вывести на экран этот массив уже в упорядоченном виде.

В начале программы находится цикл, заполняющий массив элементами и выводящий этот массив поэлементно на экран. Эту часть программы разбирать не будем, так как аналогичные действия подробно описаны в разделе 5.1 пособия.

Далее приступаем к сортировке созданного в предыдущей части программы массива **m**. Процесс сортировки является однотипным, но при этом он производится в соответствии с вышеприведенным алгоритмом сначала для группы из десяти элементов, затем для девяти (без последнего элемента), потом для группы из восьми элементов и далее по убывающей.

Поэтому сортировка будет осуществляться в цикле с заранее заданным числом повторений и с убывающим значением переменной цикла **n**. Начальное значение данной переменной будет равно 10, а конечное – 2, так как на последнем этапе сортировки мы будем иметь дело с группой, состоящей только из двух элементов.

В самом теле цикла мы будем производить следующие действия. В начале вспомогательной переменной **max**, которая должна содержать значение максимального элемента в рассматриваемой группе, мы присваиваем значение первого по счету элемента массива **m[1]**. Одновременно мы присваиваем начальное значение и другой вспомогательной переменной **t**. Эта переменная должна содержать индекс наибольшего по значению элемента в рассматриваемой группе. Так как в начале поиска максимума мы условно считаем наибольшим первый элемент, то переменной **t** мы присваиваем значение 1.

Далее для нахождения действительно наибольшего элемента в группе мы используемый внутренний цикл, вложенный во внешний цикл с переменной **n**. Переменной внутреннего цикла будет переменная **r**, показывающая текущее значение индекса элемента. В процессе поиска максимального элемента значение переменной **r** будет изменяться от 2 до **n** – индекса последнего по счету элемента в группе.

Для каждого элемента группы, начиная со второго, мы сравниваем его значение с наибольшим. Если значение данного элемента окажется больше значения переменной **max**, то переменной **max** присваивается новое значение, которое равно значению данного элемента массива **m[r]**, а переменной **t** также присваивается новое значение, которое равно **r** – индексу данного элемента.

В итоге, по окончании работы внутреннего цикла после перебора всех элементов группы переменная **max** будет содержать значение действительно максимального элемента группы, а переменная **t** – индекс этого элемента.

Теперь в соответствии с алгоритмом осталось поставить максимальный элемент на последнее место в группе, а последний элемент – на то место, которое ранее занимал максимальный. Для этого элементу с номером **t** мы присваиваем значение последнего элемента массива **m[n],** а последнему элементу – найденное наибольшее значение **max**. Таким образом, в результате работы двух вложенных циклов мы получим исходный массив уже в упорядоченном виде.

Последний имеющийся в программе цикл с переменной **j** выводит на экран компьютера в одну строку элементы упорядоченного массива **m.** Так же, как и в предыдущей программе, для элементов массива задается формат вывода данных, который обеспечивает наличие пробелов между числами при выводе их в одну строку.

Ниже приводится полный текст программы **array2**:

```
program array2;
{$APPTYPE CONSOLE}

uses
 SysUtils;
var j,max,n,r,t:integer;
    m:array[1..20] of integer;
function kir(st:string):string;
 var i,k:integer;
begin
 for i:=1 to length(st) do
 begin
  k:=ord(st[i]);
  case st[i] of
  'A'..'п': st[i]:=Chr(k-64);
  'р'..'я': st[i]:=Chr(k-16);
```

```
  end;
 end;
 kir:=st;
end;

begin
 randomize;
 writeln(kir('Элементы неупорядоченного массива'));
 for j:=1 to 10 do
 begin
 m[j]:=random(99)+1;
 write(m[j]:3);
 end;
 {Начало сортировки массива}
 for n:=10 downto 2 do
  begin
  max:=m[1];
  t:=1;
  for r:=2 to n do
   begin
   if m[r]>max then
     begin
     max:=m[r];
     t:=r;
     end;
   end;
   m[t]:=m[n];
   m[n]:=max;
  end;
 {Завершение сортировки массива}
 writeln;
 writeln(kir(Элементы   массива,   упорядоченного   по
возрастанию'));
 for j:=1 to 10 do
 write(m[j]:3);
 readln;
end.
```

На рис. 5.2. показаны результаты работы программы **array2**, которая производит сортировку по возрастанию целочисленного массива из 10 элементов методом простого обмена.

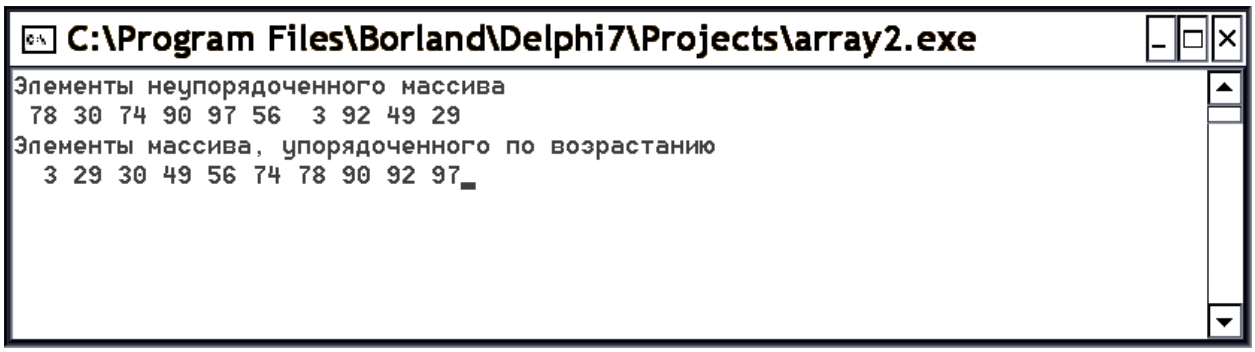

Рис. 5.2. Результаты работы программы, выполняющей сортировку массива по возрастанию

Задания для самостоятельной работы

1. Составить программу, которая заполняет массив из 10 элементов числами, введенными с клавиатуры. Требуется определить количество положительных и отрицательных элементов в массиве.

2. Составить программу, которая заполняет массив из 20 элементов с помощью генератора случайных чисел. Требуется вывести элементы массива на экран компьютера в одну строку, а затем определить сумму элементов массива и среднее арифметическое массива.

3. Составить программу, которая заполняет массив из 20 элементов с помощью генератора случайных чисел. Требуется вывести элементы массива на экран компьютера в одну строку. Затем нужно упорядочить элементы массива по убыванию и вывести упорядоченный таким образом массив на экран компьютера.

4. Составить программу, которая заполняет массив из 20 элементов с помощью генератора случайных чисел. Значения элементов массива должны находиться в диапазоне от 1 до 50. Требуется вывести элементы массива на экран компьютера в одну строку. Затем нужно подсчитать, сколько в массиве содержится элементов, значениями которых являются двузначные числа.

5. Составить программу, которая заполняет массив из 20 элементов с помощью генератора случайных чисел. Значения элементов массива должны находиться в диапазоне от 1 до 50. Требуется создать еще один массив, элементами которого являются только четные элементы из первого массива. Вывести оба массива на экран компьютера.

Глава 6. Символьные и строковые переменные

В предыдущих главах учебного пособия рассматривались программы, в которых производилась обработка только числовых данных. Но современный компьютер, представляет собой устройство, способное работать не только с числовой информацией, но и с другими ее видами (текстовой, графической, аудио и видеоинформацией). В языке **Object Pascal** также предусмотрена возможность обработки текстовых данных. Для работы с ними используются специальные типы **`char`** и **`string`**, которые, соответственно, применяются для работы с данными символьного и строкового типов.

6.1. Использование символьных переменных

В программах, написанных в Object Pascal, широко используются символьные константы и переменные. Значениями, как тех, так и других, являются символы. Символами называются все буквы и цифры, пробел, знаки препинания, знаки математических операций и другие, которые можно вводить с клавиатуры компьютера.

Центральный процессор компьютера может работать только с информацией, представленной в числовом виде. Любая вводимая в компьютер информация другого вида должна быть обязательно преобразована в числовую форму или, как говорят, закодирована с помощью соответствующих чисел. Так обстоит дело и при вводе в компьютер и дальнейшей обработке текстовой информации. Фактически компьютер работает не с буквами, пробелами, точками и запятыми, а соответствующими им числовыми кодами.

Когда пользователь при работе на компьютере использует готовые программы, такие как текстовые редакторы, электронные словари и переводчики, то ему не обязательно знать механизм обработки текстовых данных внутри компьютера. Программисту же необходимо представлять себе взаимосвязь между символом, отображаемым на экране компьютера, и хранящимся в памяти компьютера его числовым кодом.

Понятно, что если каждый программист будет изобретать свои собственные числовые коды для используемых в компьютере символов, то использовать созданные им программы для обработки текстов сможет только он сам. Поэтому возникла необходимость в создании единой, общепринятой системы представления символов числовыми кодами. Такая система была реализована в виде специальных кодовых таблиц. Существуют несколько различных кодовых таблиц, из которых в **Object Pascal** применяются две. Это кодовая таблица **ASCII**, используемая для операционной системы **MS-DOS**, и таблица **ANSI** для операционной системы **Windows.**

Кодовая таблица **ASCII** содержит 128 символов, к которым относятся прописные и строчные латинские буквы, цифры, знаки математических операций, знаки препинания и управляющие символы, которые не отображаются при вводе на экране компьютера, но выполняют определенные

действия (например, перевод строки). Расширенная версия **ASCII** состоит из 256 символов. Вторая половина расширенной таблицы содержит прописные и строчные буквы русского алфавита, а также псевдографические символы, которые могут применяться для создания рамок таблицы, несложных иллюстраций, схем, диаграмм.

Кодовая таблица **ANSI** также содержит 256 символов. Первая половина данной таблицы совпадает с первой частью таблицы **ASCII**, а вторая от нее отличается. Так как **Windows**, в которой используется кодировка **ANSI**, является полноценной графической операционной системой, то необходимость в псевдографических символах отпала, и вместо них во второй части таблицы появились другие специальные символы, а также буквы украинского и белорусского алфавитов.

Результатом этих изменений явилось то обстоятельство, что буквам русского алфавита в таблицах **ASCII** и **ANSI** соответствуют разные числовые коды. В таблице **ASCII** символы русского алфавита занимают два интервала: прописные буквы от «**А**» до «**Я**» и строчные буквы от «**а**» до «**п**» имеют десятичные числовые коды, начиная от 128 до 175; строчным буквам от «**р**» до «**я**» соответствуют числовые коды от 224 до 239. В таблице **ANSI** прописные и строчные русские буквы образуют непрерывный диапазон с десятичными кодами от 192 до 255.

Этим обстоятельством и объясняется необходимость перекодировки символов, представляющих собой буквы русского алфавита, о которой уже говорилось в главе 1 данного пособия. Именно для выполнения этой цели была разработана и используется в описываемых в пособии программах авторская функция **kir**. Но для того, чтобы разобраться в том, как устроена данная функция, необходимо рассмотреть общие принципы описания и использования символьных переменных, используемых в Object Pascal, а также ознакомиться со стандартными функциями Object Pascal, которые применяются для работы с переменными символьного типа.

Подобно переменным других типов, символьные переменные должны быть описаны перед их использованием в программе. Для описания символьных переменных используется тип **char** (сокращение от английского слова **character** – символ). В общем виде описание символьной переменной выглядит следующим образом:

```
var имя_переменной:char;
```

Для работы с символьными переменными в Object Pascal существуют специальные стандартные функции. Функция **chr** по известному числовому коду символа определяет сам этот символ. Общий вид функции:

```
c:=chr(k)
```

где **k** – аргумент функции: целая величина, представляющая собой десятичный код символа;
c - значение функции, представляющее собой символьную переменную.

Действие функции **ord** противоположно функции **chr**. Аргументом данной функции является символ, а значением – десятичный код данного символа. Общий вид функции:

```
k:=ord('s')
```

где **s** – аргумент функции: символ, код которого мы определяем (обратите внимание, что обрабатываемый функцией символ должен быть обязательно заключен в кавычки; если аргументом функции является переменная символьного типа, то заключать имя переменной в апострофы не надо);
k – значение функции, представляющее собой целую величину.

Подобно целочисленной переменной, символьную переменную можно использовать в качестве селектора в операторе множественного выбора. Принцип использования символьной переменной тот же, что и для целочисленной переменной, только вместо числовых значений перед описанными в операторе вариантами действий ставятся любые символы. Следует помнить, что когда в операторе множественного выбора перед одним из вариантов указывается соответствующее ему значение символьной переменной, то это значение также должно быть обязательно заключено в апострофы.

Рассмотрим использование символьных переменных на примере программы, которая должна вывести на экран компьютера фрагмент кодовой таблицы **ASCII**. Исходными данными, которые пользователь вводит с клавиатуры, являются начальный и конечный числовые коды фрагмента.

В начале программы описываются переменные **a,b** и **j**. Переменные **a** и **b** соответствуют начальному и конечному числовым кодам фрагмента, **j** является переменной цикла. Также описывается символьная переменная **c**, которой присваивается значение каждого очередного символа из фрагмента кодовой таблицы.

В основной части программы после ввода исходных данных используется цикл с заранее известным количеством повторений, с помощью которого производится вывод всех символов фрагмента кодовой таблицы на экран компьютера. Ниже приводится полный текст программы:

```
program fragment;

{$APPTYPE CONSOLE}

uses
SysUtils;

var a,b,j:integer; c:char;
function kir(st:string):string;
var i,k:integer;
begin
 for i:=1 to length(st) do
  begin
```

```
      k:=ord(st[i]);
      case st[i] of
     'A'..'п': st[i]:=chr(k-64);
     'р'..'я': st[i]:=chr(k-16);
      end;
      end;
    kir:=st;
  end;

  begin
    writeln(kir(Введите начальный код фрагмента'));
    readln(a);
    writeln(kir('Введите конечный код фрагмента'));
    readln(b);
    writeln(kir('Вывод фрагмента кодовой таблицы
ASCII'));
    for j:=a to b do
    begin
    c:=chr(j);
    write(c:2);
    end;
  readln;
  end.
```

На рис. 6.1 показан результат работы программы, выводящей фрагмент кодовой таблицы **ASCII**. В данном случае на экран выведен фрагмент второй части кодовой таблицы, содержащий псевдографические символы.

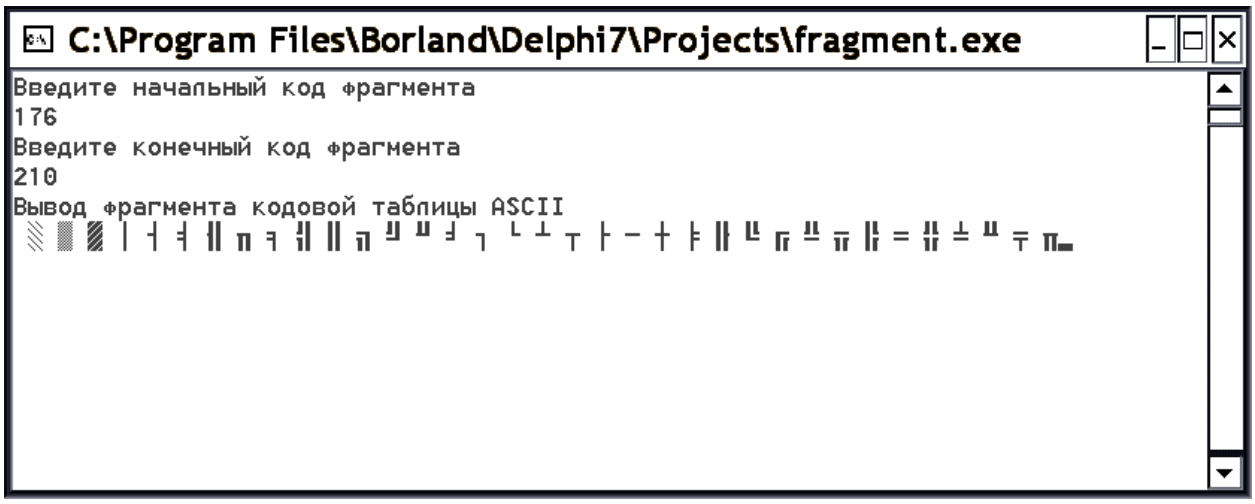

Рис. 6.1. Результаты работы программы, выводящей фрагмент кодовой таблицы ASCII

6.2. Использование строковых переменных

Для эффективной работы с текстом необходимо уметь обрабатывать не только отдельные символы, но и слова, фразы, предложения, т. е. группы

символов. Для работы с такими группами символов в **Object Pascal** используется строковый тип данных. Для описания символьных переменных используется служебное слово **string**. Описание символьной переменной в общем виде выглядит следующим образом:

```
var имя_переменной: string[длина];
```

после слова **string** указывается длина описываемой переменной, т. е. максимальное количество символов, которое может содержать данная переменная. Например, если в разделе описания переменных в программе мы видим следующее:

```
var i:integer;
    x:real
    stroka:string[25];
```

то в данной программе, наряду с переменными целого типа **i** и вещественного типа **x,** используется и переменная строкового типа **stroka**, которая может содержать до 25 символов. Параметр, заключенный в квадратные скобки, не является обязательным и используется в том случае, если существуют причины, по которым необходимо жестко ограничить максимальную длину строковой переменной. Если же после имени переменной ее длина не указывается, то по умолчанию она считается равной 255 символам. В этом случае описание символьной переменной будет выглядеть так:

```
var stroka:string;
```

Значения символьных переменных так же, как и уже знакомые нам символьные константы (к текстовым константам относятся, в частности, сообщения, выводимые оператором **writeln**), должны обязательно заключаться в апострофы. Таким образом, оператор присваивания строковой переменной **stroka** значения «**набор_символов**» будет выглядеть следующим образом:

```
stroka:= 'набор_символов';
```

Подобно числовым переменным, строковые переменные можно складывать. Результат сложения также можно присваивать какой-либо символьной переменной. Например, если значение переменной **slovo1** равно «**Object** », а переменной **slovo2** – «**Pascal**», то в результате операции

```
slovo:=slovo1+slovo2;
```

значение переменной **slovo** станет равным словосочетанию «**Object Pascal**».

Если есть необходимость работать не со всей переменной целиком, а с каким-либо отдельным содержащимся в ней символом, то к этому символу можно обратиться в программе непосредственно. Для этого нужно указать имя строковой переменной, содержащей символ, и порядковый номер символа в этой переменной, заключенный в квадратные скобки (подобно тому, как мы обращаемся к элементу массива). При обращении к элементу строковой переменной в скобках после ее имени может указываться не только числовая константа, но и имя переменной целого типа. В этом случае номер «вызываемого» символа будет равен значению данной целочисленной переменной.

Для обработки строковых переменных в **Object Pascal** существует несколько стандартных функций. Одной из наиболее часто употребляемых стандартных функций является функция **length**. Общий вид данной функции следующий:

```
l:=length(s);
```

где **s** – строковая переменная;

l – длина строковой переменной, выраженная в символах.

Рассмотрим использование строковых переменных в **Object Pascal** на примере программы, которая подсчитывает количество слов, содержащееся в предложении, введенном пользователем с клавиатуры, и среднюю длину слова. Словом будем считать любую последовательность символов, которая отделена от соседних последовательностей пробелом. Тогда количество слов в предложении будет равно количеству пробелов плюс единица. Следовательно, задача сводится к правильному подсчету количества пробелов, имеющихся в предложении.

В начале программы, как обычно, описываются используемые переменные. Переменные **i,k** и **n** являются целочисленными. Переменная **i** является переменной цикла, в котором проверяются все символы, входящие в строку; **k** – число символов, содержащееся в строке, **n** – счетчик количества пробелов. Переменной **l** вещественного типа будет присвоено значение, равное средней длине слова в предложении. Переменной **c** символьного типа поочередно присваиваются значения всех символов, имеющихся в предложении. Переменной **s** строкового типа присваивается значение строки, содержащей все предложение.

В основной части программы после ввода предложения обнуляется счетчик количества пробелов **n**. Затем с помощью стандартной функции **length** определяется количество символов **k,** содержащееся в введенном предложении.

Далее в программе организуется цикл с заранее заданным количеством повторений, в котором просматриваются все символы, имеющиеся в предложении. Переменная цикла **i** изменяет свое значение от 1 до конечного значения **k**, равного количеству символов во введенной фразе.

Каждый очередной элемент массива с индексом **i** является каким-либо символом. Каждый такой символ поочередно присваивается символьной переменной **c**. Затем с помощью стандартной функции **ord** определяется числовой код символа. Числовой код пробела равен 32. Поэтому если значение функции **ord** равно указанному числу, то в тексте предложения присутствует пробел, и значение счетчика пробелов **n** увеличивается на единицу. Данная проверка выполняется с помощью сокращенного условного оператора.

После того, как цикл завершит свою работу, нужно к подсчитанному в цикле количеству пробелов добавить единицу. Это и будет искомое количество слов в предложении, которое затем выводится на экран компьютера.

Осталось только подсчитать среднюю длину слова в предложении. Для этого необходимо вычесть из общей длины предложения **k** установленное в цикле количество пробелов **n**. Это вычитание даст общее количество значимых символов, имеющихся в предложении. Затем полученная величина делится на количество слов, содержащихся в предложении, равное **n+1**. Полученное при делении частное **l** и будет равно средней длине слова.

Поскольку при вычислении **l** используется операция деления, то данная величина должна относиться к вещественному типу. Для того чтобы обеспечить вывод этой величины в удобном для восприятия виде с фиксированной точкой, задаем обычный для вещественных величин формат вывода.

```
program words;
{$APPTYPE CONSOLE}
uses
 SysUtils;

var i,k,n:integer;
    l:real;
    c: char;
    s:string;

function kir(st:string):string;
var i,k:integer;
begin
 for i:=1 to length(st) do
  begin
  k:=ord(st[i]);
  case st[i] of
  'А'..'п': st[i]:=chr(k-64);
  'р'..'я': st[i]:=chr(k-16);
  end;
  end;
```

```
 kir:=st;
end;

begin
  writeln(kir('Введите предложение'));
  readln(s);
  k:=length(s);
  n:=0;
  for i:=1 to k do
  begin
  c:=s[i];
  if ord(c)=32 then n:=n+1;
  end;
  writeln;
  writeln(kir('Количество слов в предложении
равно'),n+1);
  l:=(k-n)/(n+1);
  writeln(kir('Средняя длина слова в предложении равна
'),l:7:2);
  readln;
end.
```

Результаты работы программы, производящей анализ введенного с клавиатуры предложения, приведены на рис. 6.2.

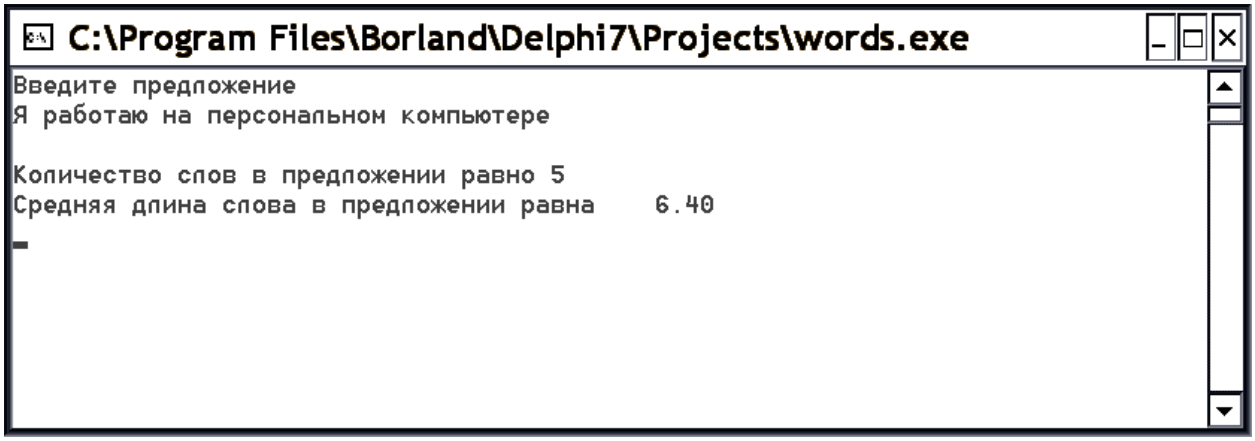

Рис. 6.2. Результаты работы программы, подсчитывающей количество слов в предложении и среднюю длину слова

Задания для самостоятельной работы

1. Составить программу, которая вводит с клавиатуры произвольный символ, вычисляет его десятичный код в кодовой таблице и определяет, является ли данный символ буквой русского алфавита, буквой латинского алфавита или цифрой.

2. Составит программу, которая выводит на экран компьютера все буквы русского алфавита (прописные и строчные) и для каждой буквы выводит ее десятичный числовой код в кодовой таблице.

3. Составить программу, которая вводит с клавиатуры произвольное предложение на русском языке и определяет, сколько гласных букв содержится в данном предложении.

4. Составить программу, которая проверяет, является ли введенная с клавиатуры последовательность символов целым числом, записанным в двоичной системе счисления, если известно, что в данной системе счисления для записи числа используются только две цифры - нуль и единица.

5. Составить программу, которая зашифровывает введенный с клавиатуры в компьютер текст. Процесс шифровки производится следующим образом: из десятичного кода каждого введенного с клавиатуры символа вычитается какое-либо число (например, 10). Получившаяся в результате вычитания величина интерпретируется как десятичный код некоторого другого символа, который и выводится на экран компьютера. Таким образом, получается внешне бессмысленный набор символов, который можно расшифровать, только зная вышеупомянутый принцип шифрования исходного сообщения.

Глава 7. Функции и процедуры

7.1. Стандартные и авторские функции

При составлении различных программ часто возникает необходимость в выполнении таких операций, как возведение числа в квадрат, извлечение квадратного корня, определение модуля числа, вычисление тригонометрических функций, округление дробного числа до ближайшего целого, определение четности или нечетности целого числа и многих других.

Для того чтобы облегчить процесс составления программ, в языке Object Pascal предусмотрены специальные функции, которые применяются для автоматического выполнения этих действий. Такие функции называются стандартными и являются неотъемлемой частью языка Object Pascal.

Функция по вводимым в нее исходным данным определяет некоторый результат, который далее используется в программе. Исходные данные называются аргументом функции, а вычисленный результат – ее значением. При обращении к функции аргумент указывается в круглых скобках. Есть небольшое количество функций, которые не имеют аргументов, но значение существует у любой функции.

Ниже приводится список наиболее часто используемых стандартных функций языка Object Pascal (некоторые из них уже знакомы по предыдущим главам данного пособия):

- **`abs(x)`** – используется для получения абсолютной величины числа;
- **`chr(k)`** – определяет символ по его коду;
- **`concat(s1,s2,…)`** – объединяет несколько строк в одну;
- **`copy(s,n,l)`** – создает копию части строки или части массива;
- **`cos(x)`** – вычисляет косинус числа;
- **`date`** – определяет текущую дату;
- **`exp(x)`** – вычисляет экспоненту числа;
- **`floattostr(x)`** – преобразует вещественное число в строку;
- **`inttostr(x)`** – преобразует целое число в строку;
- **`length(s)`** – определяет число элементов в массиве или строке;
- **`ln(x)`** – вычисляет натуральный логарифм числа;
- **`log10(x)`** – вычисляет десятичный логарифм числа;
- **`max(a,b)`** – определяет большее число из двух целых значений;
- **`min(a,b)`** – определяет меньшее число из двух целых значений;
- **`now`** – определяет текущую дату и время;
- **`odd(x)`** – проверяет, является ли целое число нечетным;
- **`ord(c)`** – определяет числовой код символа;
- **`pi`** – вычисляет значение числа Пи;
- **`random(x)`** – генерирует случайное целое число;

- **round(x)** – округляет вещественное число до целого числа;
- **sin(x)** – вычисляет синус числа;
- **sqr(x)** – вычисляет квадрат числа;
- **sqrt(x)** – вычисляет квадратный корень числа;
- **strToFloat(s)** – преобразует строку в вещественное число;
- **strToInt(s)** – преобразует строку в целое число;
- **tan(x)** – вычисляет тангенс числа;
- **time** – определяет текущее время;
- **trunc(x)** – определяет целую часть числа вещественного числа.

Язык **Object Pascal** имеет достаточно обширный набор стандартных функций (выше перечислена только некоторая часть этих функций). Но возможны ситуации, когда этого набора стандартных функций при разработке программы оказывается недостаточно. Например, в **Object Pascal** есть функция возведения числа в квадрат, но нет функции, которая бы возводила число в произвольную целую степень. Отсутствуют функции вычисления факториала натурального числа, перевода чисел из одной системы счисления в другую и ряд других полезных функций.

В этом случае программист должен самостоятельно разработать необходимую ему функцию. Такая вновь создаваемая функция называется авторской. Создание новой функции начинается с ее описания. Описание функции, как правило, находится в тексте программы после раздела описания констант и переменных и до начала ее основной части. Такую функцию часто называют функцией-подпрограммой, так как она представляет собой отдельный блок внутри основной программы, который по своей структуре напоминает основную программу. Структура описания создаваемой программистом функции выглядит следующим образом:

```
заголовок функции;
раздел описания  локальных переменных;
begin
раздел операторов функции
end;
```

Рассмотрим элементы этой структуры более детально.

Общий вид заголовка функции следующий:

```
function имя_функции(параметры функции:тип_функции;
```

где **function** – служебное слово, означающее «функция». Имена функциям даются по тем же правилам, что и имена переменных. В скобках указываются аргументы функции, называемые ее параметрами, причем для каждого параметра обязательно должен быть указан его тип. Если параметры относятся к одному типу, то они перечисляются через запятую, а после двоеточия указывается их общий тип, если же параметры относятся к разным типам, то они отделяются друг от друга точкой с запятой. После скобок

обязательно указывается тип значения самой функции.

Пример заголовка функции:

```
function beta (x,y:integer; z:real):real;
```

данная функция имеет имя **beta**, в ней используются 3 параметра: **x** и **y** – целого типа; **z** – вещественного, а значение самой функции является вещественным.

Следующей частью функции является раздел описания локальных переменных. Локальными называются переменные, которые могут использоваться только внутри данной функции. Те переменные, которые используются в основной части программы, называются глобальными. Раздел описания локальных переменных выглядит так же, как и для глобальных переменных. Он начинается с ключевого слова **var**, за которым следует список локальных переменных. Для каждой переменной нужно указать ее имя и после двоеточия – тип переменной. Если локальные переменные в функции не используются, то соответствующий раздел функции можно опустить.

Последним разделом описания функции является раздел операторов, который также называют телом функции. Тело функции представляет собой оператор или группу операторов, с помощью которых вычисляется значение функции. Последним оператором в теле функции является оператор, который присваивает функции новое значение. Тело функции обязательно заключается в операторные скобки, в качестве которых используются ключевые слова **begin** и **end**. После слова **end** ставится точка с запятой, так как это конец описания функции, а не конец программы.

Для того чтобы использовать авторскую функцию в основной части программы, необходимо произвести вызов функции (обращение к функции). Обращение к функции не является самостоятельным оператором, а входит в качестве составной части в какой-либо оператор.

Обращение к функции включает в себя имя функции и следующий за ним список параметров, заключенный в круглые скобки. Параметры, указанные в обращении к функции, называются фактическими параметрами, а параметры, указанные в описании функции – формальными. Фактические параметры должны быть того же типа, что и формальные. При обращении к функции формальные параметры заменяются на соответствующие им фактические параметры. Затем с использованием фактических параметров вычисляется значение функции, и это вычисленное значение возвращается в основную часть программы.

Рассмотрим описание и использование авторской функции на следующем примере. Требуется вычислить значение выражения $(a^x+b^y)^z$, где **a,b,x,y** и **z** – целые числа, введенные пользователем с клавиатуры. Для решения этой задачи была разработана программа под названием **express**.

В начале программы описываются глобальные переменные обычного целого типа **a**,**b**,**x**,**y**,**z**, которые соответствуют исходным данным. В

программе имеется еще ряд глобальных переменных, которые описываются как переменные «большого» целого типа **longint**. Это результат вычислений **r**, а также вспомогательные переменные **ax**, **bx** и **s**. Значение **ax** равно **a** в степени **x**, **by** равно **b** в степени **y**. Переменная **s** представляет собой сумму этих величин.

Затем в программе описываются две авторские функции. Первой из них является уже знакомая нам функция **kir**. Теперь, зная о принципах создания авторских функций, можно подробно рассмотреть ее конструкцию.

В заголовке функции указан тип значения функции – строковый, и тип значения входного параметра **st** – также строковый. В разделе описания локальных переменных описываются целочисленные переменные **i** и **k**.

Тело функции содержит цикл с заранее заданным количеством повторений. В этом цикле переменная цикла **i** изменяет свое значение от начального, равного единице, до конечного значения, равного длине строки. Длина строки определяется с помощью функции **length**. Таким образом, в цикле обрабатываются все элементы строки, т.е. содержащиеся в ней символы. При этом каждый символ в кодировке ANSI заменяется на аналогичный ему символ, но уже в кодировке ASCII.

Для этого вначале с помощью стандартной функции **ord** определяется **k** — десятичный числовой код очередного символа в кодировке ANSI. Затем этот числовой код пересчитывается в код ASCII для того же символа. Для 2 частей алфавита действуют разные правила пересчета. Для всех прописных букв русского алфавита и строчных букв от «**а**» до «**п**» новый код получается, если вычесть из старого число 64. Например, десятичный код буквы «**А**» в кодировке ANSI равен 192, а код той же буквы в кодировке ASCII равен 128. Для строчных букв от «**р**» до «**я**» используется другая формула: чтобы получить из кода ANSI код ASCII, нужно вычесть из кода ANSI число 16. Например, буква «**я**» в ANSI имеет код 255, а в ASCII – 239. После того, как код ASCII для символа определен, он преобразуется в сам этот символ с помощью стандартной функции **chr**.

Для того чтобы правильно выбрать формулу пересчета, используется оператор множественного выбора. В качестве переменной-селектора в этом операторе используется сам преобразуемый символ, являющийся i-m элементом строки. Последний оператор функции присваивает ей значение преобразованной с помощью вышеописанного цикла строки **st**.

Вторая авторская функция, используемая в программе – это функция **power**, которая возводит число в произвольную целую степень. Необходимость создания и использования данной функции обусловлена тем, что одна и та же операция возведения числа в степень 3 раза используется при вычислении заданного выражения. В заголовке функции указывается тип ее значения – «длинный» целый и тип значения параметров – обычный целый. Параметр **m** – это основание степени, а **n** – показатель степени.

В разделе описания локальных переменных описаны переменные **i** и **p**. Переменная **p** является вычисляемым значением степени, а **i** – это переменная цикла с заранее известным количеством повторений.

В теле функции переменной **p** присваивается начальное значение, равное единице. Как известно, операцию возведения числа в целую степень можно свести к многократному умножению числа само на себя. Именно этот вычислительный алгоритм реализован в цикле с заранее заданным количеством повторений. При каждом очередном повторении тела цикла предыдущее значение переменной **p** умножается на основание **m**. Такая операция повторяется в цикле **n** раз. Результатом работы цикла является вычисление значения степени **p**. В последнем операторе функции это значение присваивается самой функции **power**.

В основной части программы после ввода исходных данных производится обращение к функции **power** с фактическими параметрами **a** и **x**, которые заменяют формальные **m** и **n**. Результатом является вычисление значения переменной **ax**, т.е. **a** в степени **x**. При повторном обращении к функции **power** с фактическими параметрами **b** и **y** вычисляется значение переменной **by**. Значения **ax** и **by** складываются, и сумма присваивается переменной **s**. Затем в третий раз производится обращение к функции **power** с фактическими параметрами **s** и **z**, что позволяет получить искомый результат **r**, который затем выводится на экран компьютера. Ниже приводится полный текст программы.

```
program express;
{$APPTYPE CONSOLE}
uses
SysUtils;

var a,b,x,y,z:integer;
    ax,by,s,r:longint;

function kir(st:string):string;
 var i,k:integer;
begin
 for i:=1 to length(st) do
 begin
  k:=ord(st[i]);
  case st[i] of
  'A'..'п': st[i]:=Chr(k-64);
  'р'..'я': st[i]:=Chr(k-16);
  end;
 end;
 kir:=st;
end;
```

```
function power(m,n:integer):longint;
  var i,p:integer;
  begin
  p:=1;
  for i:=1 to n do
  p:=p*m;
  power:=p;
  end;

begin
  writeln(kir('Введите a,x,b,y,z'));
  readln(a,x,b,y,z);
  ax:=power(a,x);
  by:=power(b,y);
  s:=ax+by;
  r:=power(s,z);
  writeln(kir('Значение выражения равно'),r);
  readln;
end.
```

На рис. 6.2. представлен результат работы рассмотренной выше программы **express**.

Рис. 7.1. Результат работы программы, вычисляющей значение выражения $(a^x+b^y)^z$

7.2. Стандартные и авторские процедуры

Наряду с функциями в языке **Object Pascal** широко используется еще один вид подпрограмм – процедуры. В отличие от функции, для процедуры не существует такое понятие, как значение. Но, подобно функции, процедура может иметь различные параметры. В то же время существуют некоторые процедуры, у которых отсутствуют параметры, но которые могут выполнять определенные полезные действия.

Существует ряд стандартных процедур языка **Object Pascal**, которые являются частью языка и не требуют предварительного описания. Ниже

приводится список, содержащий названия и краткие характеристики некоторых из этих процедур:

dec(x,n) – уменьшает значение целочисленной переменной на **n**;
delay(t) – задерживает выполнение программы на **t** миллисекунд;
exit – производит выход из программы;
inc(x,n) – увеличивает значение целочисленной переменной на **n**;
randomize – активизирует генератор случайных чисел.

К процедурам относятся и многократно использовавшиеся в предыдущих программах операторы ввода **read** и **readln**, а также операторы вывода **write** и **writeln**.

Возможности языка Object Pasal не ограничиваются использованием только стандартных процедур. Программист, использующий этот язык, может создавать и свои собственные авторские процедуры. Для того чтобы такую процедуру можно было использовать в программе, ее, подобно авторской функциям, следует предварительно описать.

Описание процедуры, так же как и описание функции, должно содержаться в программе в разделе описаний после описания констант и переменных. Структура описания процедуры сходна со структурой основной программы, поэтому создаваемую программистом процедуру часто называют процедурой-подпрограммой. Описание включает в себя заголовок процедуры, раздел описаний и раздел операторов.

Общий вид описания процедуры похож на описание функции и имеет следующий вид:

```
заголовок процедуры;
раздел описания локальных переменных процедуры;
begin
раздел операторов процедуры
end;
```

Рассмотрим более подробно составные части данного описания.

Общий вид заголовка процедуры следующий:

```
procedure имя_процедуры (параметры процедуры);
```

где **procedure** – служебное слово, имя процедуры дается по тем же правилам, что и имена переменных в Object Pascal, параметры перечисляются в скобках через запятую с указанием их типа. Эти параметры являются формальными.

Все формальные параметры делятся на два вида. Если перед именем параметра в заголовке процедуры стоит служебное слово **var**, то это – параметр-переменная. Если служебное слово **var** перед именем переменной в заголовке отсутствует, то данный параметр является параметром-значением.

При обращении к процедуре формальному параметру-значению присваивается значение соответствующего ему фактического параметра, причем в качестве такого значения может выступать константа, переменная или выражение. Во время работы процедуры параметр-значение не может изменяться даже в том случае, если он является переменной.

При обращении же к процедуре, в которой имеются формальные параметры-переменные, соответствующие им фактические параметры могут быть только переменными (не константами и не выражениями). Вызываемая процедура получает доступ к ячейкам памяти, в которых хранятся эти фактические параметры, и может изменять значения этих параметров в ходе своей работы.

Пример заголовка процедуры:

```
procedure xyz(a,b:integer; var c,d:real);
```

где **xyz** – имя процедуры, **a,b,c,d** – имена формальных параметров, причем **a** и **b** являются параметрами-значениями, **c** и **d** – параметрами-переменными. Следует обратить внимание, что в отличие от функции, у процедуры нет значения, и поэтому не надо указывать ее тип.

Раздел описания локальных переменных сходен с таким же разделом в описании функции. Так же, как и для функции, локальными переменными процедуры называются те переменные, которые используются только внутри данной процедуры.

Раздел операторов процедуры также заключается в операторные скобки **begin** и **end**. После ключевого слова **end** ставится точка с запятой. Но если в теле функции последним должен быть оператор присваивания, присваивающий новое значение функции, то в теле процедуры последним может быть любой оператор, ввиду отсутствия у нее значения.

Вызов процедуры из основной части программы, в отличие от вызова функции, представляет собой отдельный оператор. Этот оператор содержит имя процедуры и указываемый в круглых скобках после имени список фактических параметров процедуры.

Рассмотрим работу с процедурами на следующем примере. Требуется найти сумму факториалов трех натуральных чисел, введенных пользователем с клавиатуры. Поскольку вычисление факториала выполняется в данной программе 3 раза, целесообразно создать отдельную процедуру, выполняющую эту операцию, а затем обращаться к этой процедуре по мере необходимости. Вышеописанная схема решения поставленной задачи реализована в программе **sum_fakt**.

В начале программы, как обычно описываются глобальные переменные. Исходные данные **a, b** и **c** описываются как переменные обычного целого типа. Переменные **fa, fb** и **fc** – это факториалы, соответственно **a,b** и **c**, а **sum** – сумма этих факториалов. Для описания этих 4 переменных используется «длинный» целый тип.

Далее в программе идет описание авторской функции **kir** и авторской процедуры **fakt.** Как видно из заголовка авторской процедуры **fakt**, она имеет 2 параметра. Параметр **n**, содержащий значение исходной величины – натурального числа, является параметром-значением. Параметр **f**, который содержит вычисляемое значение факториала, является параметром-переменной. Далее в процедуре описывается локальная вспомогательная переменная **i** целого типа. В теле процедуры содержатся операторы, вычисляющие значение факториала. При вычислении применяется цикл с заранее известным количеством повторений. Используемый при вычислении факториала алгоритм был описан в главе 4, посвященной циклам в Object Pascal, и поэтому детально рассматривать данный алгоритм в этой главе нет необходимости.

В основной части программы после ввода исходных данных трижды происходит обращение к авторской процедуре **fakt**. При первом обращении формальные параметры **n** и **f** заменяются на фактические **a** и **fa**. При помощи этих параметров вычисляется факториал числа **a**. Затем с использованием аналогичных фактических параметров вычисляются факториалы **b** и **c**. Остается сложить вычисленные факториалы и вывести полученную сумму на экран компьютера. Ниже приводится полный текст программы **sum_fakt**.

```
program sum_fakt;

{$APPTYPE CONSOLE}

uses
  SysUtils;

var a,b,c:integer;
    fa,fb,fc,sum:longint;

function kir(st:string):string;
var i,k:integer;

begin
 for i:=1 to length(st) do
  begin
  k:=ord(st[i]);
  case st[i] of
  'A'..'п': st[i]:=chr(k-64);
  'р'..'я': st[i]:=chr(k-16);
  end;
  end;
 kir:=st;
end;

procedure fakt(n:integer; var f:longint);
```

```
var i:integer;
begin
f:=1;
for i:=1 to n do
f:=f*i;
end;

begin
  writeln(kir('Введите 3 целых числа (каждое не больше
12)'));
  readln(a,b,c);
  fakt(a,fa);
  fakt(b,fb);
  fakt(c,fc);
  sum:=fa+fb+fc;
  writeln(kir('Сумма факториалов введенных чисел равна
'),sum);
  readln;
end.
```

На рис. 7.2. показаны результаты работы программы **sum_fakt**, которая вычисляет сумму факториалов 3 натуральных чисел.

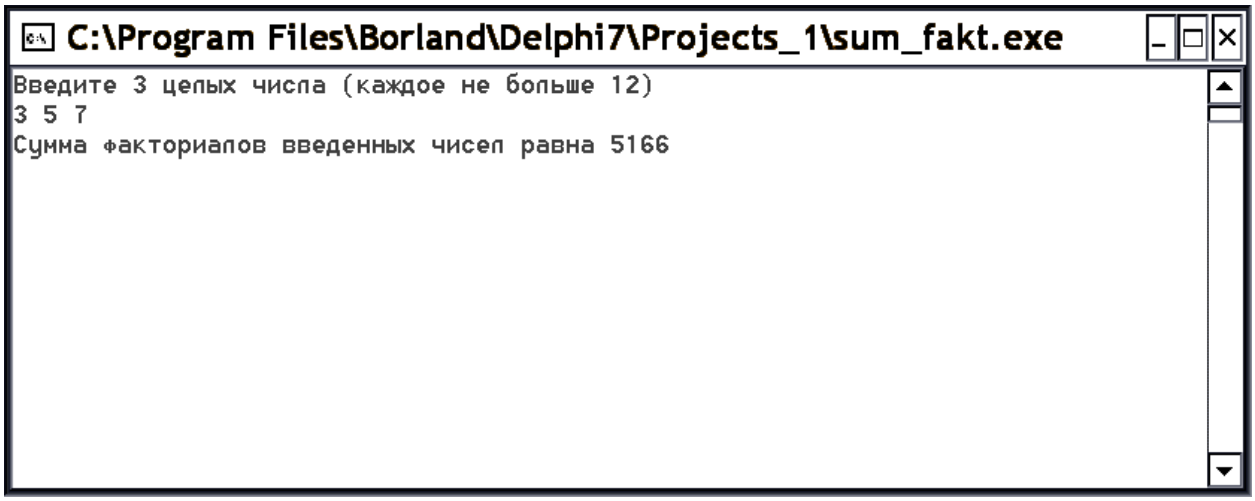

Рис. 7.2. Результаты работы программы, вычисляющей сумму факториалов 3 чисел

Задания для самостоятельной работы

1. Составить программу, которая вводит длину, ширину и высоту для двух параллелепипедов, а затем определяет, какой из двух параллелепипедов имеет большую площадь. Вычисление площади параллелепипеда оформить в виде отдельной функции.

2. Составить программу, которая вводит с клавиатуры целые числа **a,b,x** и **y**. Требуется определить, какое из двух выражений больше: $\mathbf{a^x}$ или $\mathbf{b^y}$. При решении задачи использовать авторскую функцию возведения числа в произвольную целую степень.

3. Составить программу, которая вычисляет число сочетаний из **n** по **m**. Число сочетаний определяется по формуле

$$\tilde{N}_n^m = \frac{n!}{m!(n-m)!}.$$

При составлении программы использовать авторскую процедуру вычисления факториала.

4. Составить программу, которая заполняет одномерный массив из 20 элементов случайными целыми числами от 1 до 99, а затем определяет, сколько в массиве имеется простых чисел (простым числом называется такое, которое делится только на единицу или само на себя). Процесс определения того, является ли число простым, оформить в виде отдельной процедуры.

5. Составить программу, которая определяет в интервале от 1 до 10000 все совершенные числа. Совершенным числом называется число, которое равно сумме всех своих делителей, включая единицу. Процесс определения того, является ли данное число совершенным, оформить в виде отдельной процедуры.

Библиографический список

Бобровский С.И. Delphi 7. Учебный курс. – СПб.: Питер, 2004.

Культин Н.Б. Delphi в задачах и примерах. – СПб.: БХВ, 2003.

Пестриков В.М., Маслобоев А.Н. Delphi на примерах. – СПб.: БХВ, 2005.

Пестриков В.М., Маслобоев А.Н. Основы программирования в системе Borland Delphi: учебное пособие/ СПбГТУРП. – СПб., 2004.

Пестриков В.М., Маслобоев А.Н. Решение математических задач в Turbo Pascal: учебное пособие/ СПбГТУРП. – СПб., 2009.

Пестриков В.М., Маслобоев А.Н. Turbo Pascal 7.0. Изучаем на примерах. – 2-е изд., перераб. и доп. – СПб.: Наука и техника, 2004.

Поган А.М., Царенко Ю.А. Программирование в Delphi. – М.: Эксмо, 2007.

Редактор и корректор Н.П. Новикова

Техн. редактор Л.Я. Титова Темплан 2014 г., поз. 83

Подп. к печати 02.07.2014 Формат 60х84/16. Бумага тип. № 1.

Печать офсетная. 4,75 уч.-изд. л. ; 4,75 усл. печ. л. Тираж 50 экз.

Изд. № 83. Цена «С». Заказ

Ризограф Санкт-Петербургского государственного технологического университета растительных полимеров, 198095, СПб., ул. Ивана Черных, 4.