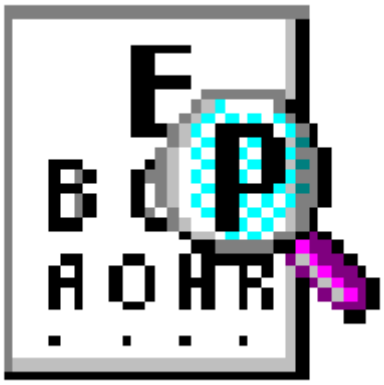
А.Н. Маслобоев

Практикум по алгоритмическому программированию в системе Turbo Pascal часть I

Учебно-методическое пособие



Санкт-Петербург
2010

Министерство образования и науки

Российской Федерации

Государственное образовательное учреждение
высшего профессионального образования

«Санкт-Петербургский государственный технологический университет растительных полимеров»

А.Н. Маслобоев

Практикум по алгоритмическому программированию в системе Turbo Pascal

Часть I

Учебно-методическое пособие

Санкт-Петербург
2010

ББК 32.97я7
П 286
УДК 681.3 (075)

Маслобоев А.Н. Практикум по алгоритмическому программированию в системе Turbo Pascal: учебно-методическое пособие /ГОУВПО СПб ГТУ РП. — СПб., 2010. ч.I. — 57 с.

В настоящем учебно-методическом пособии рассматриваются вопросы алгоритмического программирования на языке Паскаль в системе Turbo Pascal 7.0. Пособие содержит необходимый теоретический минимум, примеры программ с подробными комментариями к ним и задания для самостоятельной работы студентов.

Пособие предназначено для студентов факультета МАП специальностей 150405 (170400) “Машины и оборудование лесного комплекса” и 240801 (170500) “Машины и аппараты химических производств”, изучающих дисциплину «Информатика» по полной форме обучения: I курс (I и II семестры) и по сокращенной форме обучения: I курс (II семестр) и II курс (III семестр).

Рецензенты:
доктор технических наук, профессор В.М. Пестриков (СПбГТУ РП);
кандидат технических наук С.Г. Рыбаков (СКБ «Турбина»).

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

Оглавление

Введение

Язык Паскаль был создан в 1970 г. швейцарским ученым Н. Виртом как средство для обучения студентов основам алгоритмического программирования. В дальнейшем этот язык получил широкое распространение во всем мире не только для учебных целей, но и для решения многих профессиональных задач из области информационных технологий. В 80-х гг. XX в. на основе языка Паскаль американской фирмой Borland была разработана система программирования Turbo Pascal. Эта система программирования, созданная в свое время на платформе операционной системы MS DOS, в дальнейшем показала свою работоспособность и под управлением операционной системы Windows.

В настоящее время для профессионального программирования широко используется инструментальная система Borland Delphi. Эта система программирования является логическим развитием системы Turbo Pascal на платформе Windows. Языком программирования, применяемым в Borland Delphi, является тот же язык Паскаль, но в новой его версии, называемой Object Pascal.

В то же время, классический язык Паскаль, представленный в системе Turbo Pascal, сохраняет свое значение в качестве базового языка для изучения основ программирования и информационных технологий. Практика преподавания информатики показывает, что студенты, овладевшие основными принципами алгоритмического и объектно-ориентированного программирования, изучая классический язык Паскаль, в дальнейшем лучше и эффективнее осваивают современные системы программирования, такие как Borland Delphi, Visual Basic, VBA и другие.

Следует при этом иметь в виду, что для получения реальных навыков разработки приложений на Паскале обучаемому недостаточно изучить даже самый подробный и обстоятельный учебник по языку Паскаль. Для этого студенту необходимо на конкретных примерах разобраться в том, как решаются в Паскале те или иные задачи, а затем самостоятельно выполнить ряд практических работ.

Данное учебно-методическое пособие было написано для того, чтобы помочь студентам в приобретении практического опыта в программировании в системе Turbo Pascal. В первой части пособия рассматриваются основные понятия и структуры алгоритмического программирования. В каждой главе вначале кратко сообщаются необходимые теоретические сведения по изучаемой теме, затем приводятся примеры программ на Паскале с подробным их разбором и в конце главы даются задания для самостоятельной работы студентов. Изучение материалов, приведенных в пособии, и выполнение заданий должно создать у студентов необходимые предпосылки для формирования способности к успешному применению полученных знаний в своей будущей профессиональной деятельности.

Глава 1. Начальные сведения о языке Паскаль

1.1. Алфавит языка Паскаль

Любая программа на любом языке программирования записывается в виде символов. Символы эти не могут быть произвольными. Подобно естественным языкам, таким, как русский, английский, французский и другие, которые люди используют при общении друг с другом, языки программирования, на которых человек общается с компьютером, имеют свой алфавит. В данном случае алфавит – это набор букв, цифр и других символов, используемых при написании программ.

Алфавит языка Паскаль включает в себя:

1. Буквы латинского алфавита от **A** до **Z**. Буквы могут быть как прописными, так и строчными, так как компилятор языка Паскаль при обработке программ не делает различия между теми и другими. К буквам в Паскале относится также символ подчеркивания: **_** . Буквы используются для формирования имен переменных и ключевых слов.

2. Цифры от **0** до **9**. Цифры используются для записи чисел и имен переменных.

3. Специальные символы. К этой категории относятся знаки математических операций, знаки препинания и другие. Специальные символы подразделяются на одиночные и парные:

а) одиночные

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

б) парные

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

Особое место в ряду символов языка занимает такой символ, как пробел. Он используется для отделения в программе друг от друга различных синтаксических единиц (т. е. имен переменных, констант, команд языка и других).

Наряду с цифрами, буквами и специальными символами Паскаль содержит ряд служебных слов, значения которых заранее определены и не могут изменяться пользователем. К таким зарезервированным словам относятся:

```
and           div           function
asm           do            goto
array         downto        if
begin         else          implementation
case          end           in
const         exports       inherited
constructor   file          inline
destructor    for           interface
```

label	**procedure**	**to**
library	**program**	**type**
mod	**record**	**unit**
nil	**repeat**	**until**
not	**set**	**uses**
object	**shl**	**var**
of	**shr**	**while**
or	**string**	**with**
packed	**then**	**xor**

Из отдельных символов и служебных слов в Паскале складываются ***операторы***. Оператором называется выражение, обозначающее и описывающее какую-либо операцию, осуществляемую в программе.

Использование букв русского алфавита в программе допускается только в комментариях (пояснениях к программе, не влияющих на ход ее выполнения) и в качестве значений строковых констант и переменных. Что такое константы и переменные, мы рассмотрим ниже. Комментарии отделяются от основного текста программы с помощью фигурных скобок или парных символов – круглых скобок со звездочкой.

1.2. Структура программы на Паскале

Компьютерная программа представляет собой последовательность действий, выполняемых в процессе решения поставленной задачи. Эти действия описываются в виде операторов языка Паскаль, которые и являются основными элементами, из которых складывается программа. При этом операторы работают с различными величинами: постоянными (константами) и переменными.

Константой называется величина, которая в процессе выполнения программы остается неизменной. Константа может быть либо числом (числовая константа), либо некоторым произвольным набором символов (в таком случае она называется текстовой константой). В ряде случаев для удобства работы константам дают имена, состоящие из букв латинского алфавита и цифр.

Переменной называется величина, которая может изменяться в ходе выполнения программы. Каждая переменная должна иметь собственное имя, значение и тип. Имя переменной обозначается подобно имени константы латинскими буквами и цифрами, причем начинаться имя переменной обязательно должно с буквы. Каждая отдельная переменная может принимать значения только определенного типа. Это могут быть, например, целые или вещественные числа. В первом случае переменная называется целочисленной, во втором – переменной вещественного типа, или вещественной. Значениями переменной могут быть не только числа, но и отдельные символы. В таком случае переменная называется символьной. Если же значением переменной является не отдельный символ, а группа

символов (такая группа называется в Паскале строкой), то соответствующая переменная будет именоваться строковой. В языке Паскаль существуют и другие типы переменных.

В Паскале тип каждой используемой в программе переменной обязательно должен быть описан в соответствующем разделе программы. Тип переменной определяет не только область значений переменной, но и набор операций, которые можно производить над переменной. Такой набор является специфическим для каждого типа переменной. Например, над числовыми переменными можно производить операции умножения и деления, а для строковой переменной существует операция определения ее длины. Под каждую переменную в памяти компьютера отводится некоторая область, в которой хранится ее значение. Это значение сохраняется в неизменном виде до тех пор, пока переменная не получит новое значение. Операция изменения значения переменной называется операцией присваивания.

Операторы, а также обрабатываемые ими величины не могут располагаться в программе произвольным образом. Элементы программы должны находиться в определенных разделах программы. В общем виде программа на языке Паскаль состоит из следующих разделов:

1. Заголовок программы, который включает служебное слово **`program`** и название программы, отделенные друг от друга пробелом. В названии программы не должно быть пробелов; в случае необходимости пробел можно заменить знаком подчеркивания. Также в названии программы не должно быть точек, запятых, точек с запятыми – такие имена будут восприниматься системой программирования как ошибочные. В имени программы нельзя использовать также круглые скобки, вопросительный и восклицательный знаки.

2. Команда подключения модуля или модулей системы программирования, которая начинается со служебного слова **`Uses`** и содержит имена подключаемых модуля или модулей. Подробнее о модулях и их использовании будет сказано в разделе 1.3 данного пособия.

3. Раздел объявлений, в котором описываются константы и переменные. Описание констант начинается со служебного слова **`const`**, переменных – со слова **`var`**. После описания группы переменных, относящихся к одному типу, в программе ставится точка с запятой. Наряду с вышеуказанными в разделе могут присутствовать описания других элементов программы: функций – соответствующий раздел начинается со слова **`function`**, процедур – раздел начинается со слова **`procedure`**, меток – раздел начинается со слова **`label`**, типов данных – раздел начинается с **`type`**. В самых простых программах, где не используются никакие из вышеперечисленных элементов, раздел описаний может отсутствовать.

4. **`begin`** – служебное слово, предваряющее основную часть программы. После слова **`begin`** никаких знаков препинания не ставится.

5. Основная часть программы (тело программы), в которой находятся операторы, обеспечивающие ее выполнение. Операторы отделяются друг от друга точками с запятой, причем в одной строке может находиться несколько операторов, а если оператор не помещается в одной строке, то его можно переносить на следующую строку. После последнего в программе оператора точку с запятой ставить необязательно.

6. **end** – служебное слово, которым завершается программа (после **end** обязательно ставится точка).

1.3. Вывод информации на экран компьютера

Начнем рассмотрение программ на Паскале с простейших, которые не выполняют никаких вычислений или других способов преобразования информации, а только выводят какую-либо информацию на экран компьютера.

Вывод данных производится с помощью операторов **write** или **writeln**. Общий вид оператора следующий:

```
write(список вывода);
```

Список вывода может включать имена переменных и констант, перечисляемые через запятую. В простейшем случае список может состоять из одной переменной или одной константы. Если константа является текстовой, то она должна быть заключена в апострофы (одиночные кавычки). Общий вид оператора writeln такой же, как и оператора write.

```
writeln (список вывода);
```

Различие между этими операторами заключается в следующем. После вывода данным оператором **write** курсор остается в той же строке, что и выводимые данные. После вывода данных с помощью оператора **writeln** курсор перемещается на следующую строку. Разница между механизмом работы этих операторов может быть наглядно продемонстрирована на следующем примере. Пусть в программе нам необходимо вывести текст: «Мы изучаем программирование».

Если этот текст вывести тремя операторами **write**:

```
write('Мы');
write(' изучаем ');
write('программирование');
```

то при выполнении программы вся фраза будет выведена в одну строку:

```
Мы изучаем программирование
```

Если тот же текст вывести тремя операторами **writeln**:

```
writeln('Мы');
writeln(' изучаем ');
```

```
writeln('программирование');
```

то каждое слово фразы будет начинаться с новой строки, и при выполнении программы на экране мы увидим следующее:

```
Мы
изучаем
программирование
```

Ниже приводится текст программы, которая выводит на экран компьютера следующий текст: «Я программирую в системе Turbo Pascal».

```
program first;
uses crt;
begin
clrscr;
writeln('Я программирую в системе Turbo Pascal');
readln
end.
```

Рассмотрим более подробно данную программу. В начале программы в заголовке после ключевого слова **program** указывается собственное имя программы. При сохранении программы на жестком диске компьютера необходимо придерживаться следующего правила: файл, в котором сохраняется программа, необходимо называть тем же именем, что используется для наименования программы, т.е. в данном случае программу следует сохранить в файле с именем **first.pas**. Тогда пользователь будет совершенно определенно знать, что у него находится в каждом файле.

Во второй строке программы идет команда подключения модуля **Crt**. Для того чтобы понять, зачем нужно подключать данный модуль, необходимо знать, что система Turbo Pascal вообще имеет модульную структуру. При запуске данной системы программирования в оперативную память компьютера автоматически загружается только основной системный модуль. Если же требуется придать системе дополнительные функциональные возможности, то необходимо подключить к основному системному модулю дополнительный модуль (или модули) с помощью специальной команды. Эта команда включает ключевое слово **Uses** (использует) и название модуля.

В данной программе используется модуль **Crt**, который расширяет возможности системы Turbo Pascal при работе в текстовом режиме. Этот модуль позволяет производить очистку экрана в начале выполнения программы, использовать различные цвета при выводе информации, создавать цветной фон для выводимых данных, создавать на экране несколько окон, используемых для ввода и вывода информации, позиционировать курсор (т. е. помещать его в заданную позицию на экране

компьютера), применять аудиоэффекты. В приведенной выше программе мы используем только первую из перечисленных возможностей.

Помимо модуля **Crt,** в системе Turbo Pascal существуют и другие стандартные модули. Например, модуль **Graph** применяется для программирования в графическом режиме работы компьютера. Кроме использования существующих стандартных модулей, пользователь может сам создавать и компилировать собственные модули, на которые можно ссылаться из различных программ.

Тело программы открывается служебным словом **begin**, после которого идет команда очистки экрана **Clrscr** (для того чтобы ее можно было использовать, и понадобилось подключение модуля **Crt)**. Необходимость использования этой команды в данной программе (как и в других) объясняется следующим обстоятельством. Во время одного сеанса работы пользователь может неоднократно запускать на выполнение одну и ту же программу (или разные программы). Выведенные данные сохраняются на экране компьютера (если не выполнить очистку экрана), и пользователю может быть непонятно, какие именно данные относятся к последнему запуску программы. Решить эту проблему позволяет применение команды очистки экрана.

Далее в программе идет оператор **writeln**, который и обеспечивает вывод текста, заключенного в апострофы, на экран. Обратите внимание на следующее: когда при наборе текста возникает необходимость переключиться на русскую раскладку клавиатуры, то при этом могут использоваться комбинации клавиш, отличные от тех, которые применяются в Windows. Как правило, для переключения на русскую раскладку применяется комбинация клавиш – **правый Ctrl плюс правый Shift**, а для переключения на английскую раскладку – **левый Ctrl плюс левый Shift**. Эта разница в комбинациях клавиш связана с тем, что система Turbo Pascal является приложением операционной системы MS DOS, поэтому в ней используется другой драйвер клавиатуры.

Далее в программе стоит оператор ввода **readln,** не содержащий более никаких параметров. Подробно операторы ввода будут рассмотрены в разделе 2.1 данного пособия, а сейчас необходимо только указать, что в данной программе назначение **readln** заключается в том, чтобы приостанавливать выполнение программы до нажатия клавиши **Enter**. После запуска программы на выполнение данные будут оставаться на экране компьютера до нажатия указанной клавиши, что удобнее, чем открывать потом экран с результатами работы программы через меню системы Turbo Pascal.

Завершается программа служебным словом **end** с точкой. Точка в программе на языке Паскаль может быть только одна.

По умолчанию данные в программе выводятся белыми символами на черном фоне. Но подключение модуля **Crt** позволяет изменять как цвет символов, так и цвет фона, на котором они выводятся. Цвет символов

задается с помощью команды **TextColor**. После служебного слова **TextColor** в круглых скобках указывается цвет символов. Всего в Турбо Паскале используется 16 стандартных цветов. Вот их названия:

Black – черный
Blue – синий
LightBlue – светло-синий, голубой
Brown – коричневый
Cyan – бирюзовый
LightCyan – светло-бирюзовый
DarkGray – темно-серый
LightGray – светло-серый
Green – зеленый
LightGreen – светло-зеленый
Magenta – фиолетовый
LightMagenta – светло-фиолетовый
Red - красный
LightRed – светло-красный
White – белый
Yellow – желтый

Для задания цвета фона используется команда **TextBackground**. Формат ее аналогичен команде **TextColor**, но эта команда позволяет использовать только 8 цветов:

Black	**Red**	**Blue**	**Magenta**
Green	**Brown**	**Cyan**	**LightGray**

Рассмотрим пример программы, в которой используются цветовые эффекты. В этой программе на светло-сером фоне различными цветами выводится следующий текст:

Эта программа представляет собой пример
Использования цветовой палитры
системы программирования
Turbo Pascal 7.0

Первая строка этого текста должна быть выведена красным цветом, вторая – зеленым, третья – синим, а четвертая – желтым. Текст программы приводится ниже:

```
program democolor;
Uses Crt;
begin
TextBackground(LightGray);
ClrScr;
TextColor(Red);
```

```
writeln('Эта программа представляет собой пример ');
TextColor(Green);
writeln('использования цветовой палитры ');
TextColor(Blue);
writeln('системы программирования');
TextColor(Yellow);
writeln('Turbo Pascal 7.0 ');
readln
end.
```

Рассматривая текст этой программы, необходимо обратить внимание на следующие моменты.

Во-первых, команда, задающая цвет фона, ставится в данном случае перед командой очистки экрана **ClrScr**. Тогда после запуска программы на выполнение цвет всего экрана будет таким, какой указан в операторе **TextBackground** (в данном случае – светло-серым). Если поставить команду **TextBackground** после команды очистки экрана, то указанным цветом будут закрашены только те строки экрана, в которых выводится текст.

Во-вторых, команда **TextColor,** определяющая цвет выводимого текста, пишется перед оператором **writeln**, который выводит этот текст. Обратите внимание также на то, что точка с запятой ставится в конце каждого оператора, кроме последнего перед словом **end**.

Цвета в командах **TextColor** и **TextBackground** можно обозначать не только словами, как это сделано в приведенной выше программе, но и числами. В некоторых случаях использование чисел даже удобнее. Для кодирования цветов используются следующие числа:

0 – черный
1 - синий
2 – зеленый
3 - бирюзовый
4 - красный
5 - фиолетовый
6 – коричневый
7 – светло-серый
8 – темно-серый
9 - голубой
10 – светло-зеленый
11 – светло-бирюзовый
12 – светло-красный
13 – светло-фиолетовый
14 – желтый
15 – белый
128 – мерцание

Поясним последний числовой код. Он не используется самостоятельно, а является дополнительным, то есть употребляется вместе с каким-либо числовым значением цвета для создания эффекта мерцания. Для этого число 128 добавляется к основному числовому значению. Поясним вышеизложенное следующими примерами: если в тексте программы имеется команда

```
Textcolor(lightgreen);
```

то следующий за ней текст будет выводиться светло-зеленым цветом. Аналогичный результат обеспечит и команда

```
Textcolor(10);
```

если же мы используем команду

```
Textcolor(10+128);
```

то текст будет выводиться светло-зеленым цветом, и при этом будет мерцать.

Задания для самостоятельной работы к главе 1

1. Составить программу, которая бы выводила на экран компьютера на светло-сером фоне фиолетовыми буквами в пять строк следующий текст:

Существуют следующие основные виды
программного обеспечения персональных компьютеров:
- системные программы
- прикладные программы
- инструментальные системы или системы программирования

2. Составить программу, которая бы выводила на экран компьютера на черном фоне треугольник зеленого цвета, составленный из символов *. Вершину треугольника должна составлять 1 звездочка, на следующей строке должно быть 3 символа *, на третьей 5 и т. д., вплоть до основания треугольника, составленного из 15 звездочек.

3. Составить программу, которая на экране компьютера на черном фоне выводит следующий текст:

Состав персонального компьютера:
системный блок
монитор или дисплей
клавиатура
мышь или другое координатное устройство

Первая строка текста должна быть выведена белым цветом, вторая – светло-красным, третья – светло-бирюзовым, четвертая – желтым, пятая – светло-зеленым.

4. Составить программу, которая на экране компьютера на белом фоне выводит следующий текст:

Основные виды окон ОС Windows:
окна папок
окна приложений
диалоговые окна
окна справочной системы

Первая строка текста должна быть выведена черным цветом, вторая – красным, третья – синим, четвертая – фиолетовым, пятая – бирюзовым.

5. Составить программу, которая на экране компьютера на светло-сером фоне выводит следующий текст:

В Турбо Паскале
для оформления программы
применяются различные
визуальные эффекты

Первая строка текста должна быть выведена красным цветом, вторая – желтым, третья – зеленым, четвертая – синим. Во второй, третьей и четвертой строках создать эффект мерцания.

Глава 2. Работа с переменными

2.1. Переменные, оператор присваивания и выражения

Программы, рассмотренные в разделе 1.3, осуществляли вывод информации на экран компьютера. Теперь мы перейдем к задачам, в которых осуществляется не только ввод или вывод информации, но и ее ***обработка.*** Простейшим примером обработки информации является изменение значений переменных. Это изменение может производиться двумя способами:

а) переменной можно присвоить новое значение, введя его с клавиатуры;

б) значение переменной можно изменить непосредственно в самой программе с помощью оператора присваивания.

Ввод нового значения переменной пользователем с клавиатуры компьютера производится операторами **read** и **readln**. Общий вид такого оператора:

```
readln(список ввода);
```

Список ввода может состоять из одного элемента или нескольких, перечисленных через запятую. При этом, в отличие от операторов вывода, элементами списка в операторах ввода могут быть только имена переменных. Разница же между **read** и **readln** заключается в том, что при использовании первого после ввода данных курсор остается в той же строке, а при использовании второго курсор перемещается на следующую строку.

Примеры операторов ввода:

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

Первый из этих операторов присваивает значение, введенное пользователем с клавиатуры, переменной **x**. Значение вводится с клавиатуры и завершается нажатием клавиши **Enter**. Второй оператор присваивает значения сразу трем переменным **x**, **y** и **z**. Ввод значений переменных следует производить через пробел и только после ввода последнего значения нажимать клавишу **Enter.**

Присваивать новое значение переменной может не только пользователь, но и операторы в самой программе. Общий вид оператора присваивания в языке Паскаль

```
var_name:=e;
```

где **var_name** – имя переменной, **e –** присваиваемая величина, которая может быть числом, значением переменной или математическим

выражением. Число или значение переменной присваивается непосредственно, а значение арифметического выражения в процессе выполнения программы вычисляется и присваивается стоящей в левой части оператора переменной.

После выполнения операции присваивания в ячейке компьютерной памяти, отведенной под переменную, будет занесено значение, записанное справа от знака равенства. Это значение будет оставаться в неизменном виде вплоть до выполнения в программе следующей операции присваивания.

В операторе присваивания после имени переменной обязательно должно стоять двоеточие. Если пропустить в записи оператора двоеточие, то получится не оператор присваивания, а операция сравнения.

Приведем примеры операторов присваивания.

```
x:=5;
```

Этот оператор присваивает переменной **x** новое значение, которое равно числу 5. Можно присваивать одной переменной значение, равное значению другой переменной.

```
x:=y;
```

Этот оператор присваивает переменной **x** то же значение, что было до этой операции у переменной **y**, т. е., если до выполнения операции присваивания значение переменной **x** было равно 5, а значение **y** – 10, то после выполнения этой операции значения обеих переменных станут равными числу 10.

Возможен и такой вид оператора присваивания:

```
x:=x+5;
```

Подобная запись означает, что переменной **x** присваивается новое значение, которое больше старого значения на 5, т. е. если предыдущее значение переменной **x** было равно 10, то новое станет равным 15.

Выражение, находящееся в операторе присваивания справа от знака равенства, может включать имена переменных, числа, знаки математических операций и некоторые другие элементы.

В языке Паскаль применяются 4 основных математических операции и две дополнительные, которые применяются только по отношению к целочисленным величинам. Приведем перечень этих операций:

+ сложение
- вычитание
***** умножение
/ деление
div целочисленное деление
mod нахождение остатка от целочисленного деления

Первые 4 операции выполняются так же, как в обычной арифметике. Единственное, на что следует обратить внимание: знаки умножения и деления отличаются по написанию. Так, вместо принятых в математике знаков умножения × или · используется *****, вместо символа деления **:** используется **/ (**часто называемый «слэш»**)**. Кроме того, знак умножения между сомножителями нельзя опускать, как это часто делается в математике. Запись **ab**, которая в математике может обозначать произведение двух чисел **a** и **b**, компилятором языка Паскаль будет воспринята как имя переменной, состоящее из двух букв, что приведет к ошибке в работе программы.

Что же касается целочисленного деления, то оно выполняется как обычное, но затем отбрасывается дробная часть получившегося при делении числа. Операции **div** и **mod** выполняются только над переменными и константами целого типа.

Пример использования действий **div** и **mod**:

значение выражения 91 **div** 8 равно 11
значение выражения 91 **mod** 8 равно 3

При вычислении значения арифметического выражения вначале выполняются действия, обладающие более высоким ***приоритетом***. Слово «приоритет» в переводе с латыни означает первенство, и, применительно к программированию, имеется в виду первенство при выполнении математических операций. Умножение, деление, целочисленное деление и нахождение остатка как раз и относятся к действиям с высоким приоритетом. Сложение и вычитание имеют более низкий приоритет.

Если в выражении имеются действия с одинаковым приоритетом, то они выполняются слева направо по ходу выражения. Если необходимо изменить порядок действий в выражении, то следует использовать круглые скобки. Действия, заключенные в скобки обладают наиболее высоким приоритетом, т. е. выполняются в первую очередь.

Для лучшего понимания того, как действуют приоритеты, приведем следующий пример. Пусть в программе требуется вычислить значение арифметического выражения, записанного таким образом:

```
5 + 3*(7+2) – 40 div 4
```

Прежде всего, вычисляется выражение, находящееся в скобках, – его значение равно 9. Затем слева направо выполняются следующие действия: умножение выражения в скобках на 3 (получаем 27) и целочисленное деление (получаем 10). На завершающем этапе складываем 5 и 27 и вычитаем из суммы 10. В итоге получаем конечный результат – 22.

Рассмотрим применение операторов, изменяющих значение переменной, на примере следующей программы. Составим программу, которая вводит два числа с клавиатуры, складывает эти два числа и выводит их сумму на экран компьютера. Для того чтобы сложить эти числа,

необходимо вначале присвоить их значения каким-либо переменным (например, **a** и **b**), а эти переменные предварительно описать. Затем следует присвоить результат арифметической операции сложения третьей переменной (**c**) и вывести результат на экран компьютера. Текст программы приводится ниже:

```
program summa;
Uses Crt;
var
   a,b,c:integer;
begin
ClrScr;
writeln ('введите 2 числа');
writeln ('после ввода каждого числа нажимайте клавишу
Enter');
readln(a);
readln(b);
c:=a+b;
writeln('сумма 2 чисел равна ',c);
readln
end.
```

Рассмотрим подробно содержание данной программы. В начале ее после заголовка идет команда подключения модуля **Crt**. Хотя в этой программе не применяются цветовые эффекты, но подключение данного модуля требуется для того, чтобы производить очистку экрана после запуска программы на выполнение.

Затем идет раздел описания переменных, который обязательно должен начинаться со служебного слова **var** (сокращение от английского ***variable*** – переменная). После служебного слова через запятую перечисляются сами переменные, используемые в программе, и после двоеточия указывается тип этих переменных. Обратите внимание на то, что в этом разделе в обязательном порядке должны быть описаны ***все*** используемые в данной программе переменные. Если мы забудем описать какую-либо из использованных в программе переменных, то при запуске этой программы на выполнение мы увидим сообщение об ошибке, и программа не будет работать до тех пор, пока Вы не исправите ошибку, описав недостающую переменную в разделе описаний.

В данной программе для упрощения задачи будем складывать только целые числа, поэтому все переменные будут относиться к одному типу – целочисленному. Этот тип обозначается ключевым словом **integer**. Этот тип применяется для описания переменных, значения которых находятся в диапазоне от -32768 до 32767. Поскольку все переменные в этой программе одного типа, то имена переменных перечисляются через запятую, а затем после двоеточия указывается общий для них тип.

Следует отметить, что тип **integer** является не единственным, который применяется для описания целочисленных переменных. Наряду с ним применяются тип **word**, который применяется для целочисленных переменных, имеющих диапазон значений от 0 до 65535, тип **longint** (для него значения могут находиться в диапазоне от -2147483648 до 2147483647) и другие типы.

Далее идет тело программы, в котором после служебного слова **begin** идет команда очистки экрана **Clrscr**. За этой командой следуют операторы, обеспечивающие ввод исходных данных. Для этой цели в данной программе будут использованы два оператора **readln**, каждый из которых в списке ввода содержит по одной переменной:

```
readln(a);
readln(b);
```

С помощью операторов ввода переменным **a** и **b** будут присвоены численные значения. После запуска программы на выполнение на экране появится курсор, обозначающий позицию для ввода исходных данных. При этом нужно помнить о том, что вводимые числа по типу должны соответствовать переменным, т. е. быть целыми. При попытке ввести дробное число программа прекратит свою работу.

Для того чтобы пользователю было понятно, что от него требуется в процессе работы программы, перед операторами ввода в тексте программы обычно ставятся операторы вывода, содержащие поясняющую информацию. В данной программе это следующие операторы:

```
writeln ('введите 2 числа');
writeln ('после ввода каждого числа нажимайте клавишу Enter');
```

В этом случае программа не только правильно работает, но и обеспечивает для пользователя ***дружественный интерфейс***, т. е. удобный способ общения человека с программой.

После ввода исходных данных значения переменных **a** и **b** складываются, и сумма присваивается переменной **c**. Для этой цели используется оператор присваивания.

Для того чтобы вывести значение переменной **c** на экран компьютера, используем оператор вывода **writeln**. В данном случае этот оператор содержит список вывода, состоящий из двух элементов. Первым элементом списка вывода является текстовая константа, заключенная в апострофы. Вторым элементом является переменная **c**, значение которой выводится на экран. Имя переменной в апострофы не заключается.

Завершается данная программа так же, как и предыдущая, «пустым» оператором **readln** и командой **end** с точкой.

2.2. Специальные операции, выполняемые с целыми числами

Для решения ряда задач, в которых используются только целые числа, удобно бывает использовать действия **div** и **mod**. В качестве примера составим программу, обслуживающую работу банкомата. Банкомат должен выдавать клиенту требуемую сумму, но при этом расходовать наименьшее количество купюр. Необходимо также, чтобы на экране банкомата выводилось сообщение о том, какими именно купюрами и в каком количестве выдавалась запрошенная клиентом сумма.

Пусть в банкомате имеются купюры по 500, 100, 50 и 10 руб. Тогда алгоритм решения данной задачи сводится к следующему. В начале программа предлагает клиенту ввести требуемую сумму. (Так как банкомат не содержит мелочи, то попросим пользователя ввести сумму кратную 10). Затем эта сумма делится на 500. Результат целочисленного деления и будет минимальным количеством банкнот по 500 руб. Остаток от деления разделим на 100 и получим соответственно минимальное требуемое количество банкнот по 100 руб. Аналогичным образом найдем количество банкнот по 50 руб. Наконец, получившийся остаток при делении на 50 разделим на 10 и получим количество банкнот по 10 руб.

Если, например, нам нужно определить, каким минимальным количеством купюр можно выдать сумму 1470 руб., то вначале делим эту сумму на 500 и находим остаток от деления:

```
1470 div 500 =2;  1470 mod 500=470
```

теперь нам известно, что минимальное количество купюр по 500 руб. равно 2, а оставшаяся сумма составляет 470 руб. Эту сумму делим уже на 100:

```
470 div 100=4;    470 mod 100 =70
```

следовательно, нам понадобится как минимум 4 купюры по 100 руб., а сумма, которую нужно будет выдать 50- и 10-рублевыми купюрами, составит 70 руб. Используя аналогичные вычисления, мы получим, что эту сумму можно выдать одной 50-рублевой и двумя 10-рублевыми купюрами.

При составлении программы будут использоваться следующие переменные: **sum** – вводимая клиентом сумма; **k500** – количество купюр по 500 руб.; **k100** – количество купюр по 100 руб.; **k50** – количество купюр по 50 руб.; **k10** – количество купюр по 10 руб.; **vspom** – вспомогательная переменная, в которой содержится остаток, получающийся при делении. Программа с помощью вышеописанных операций находит значения переменных **k500**, **k100, k50** и **k10**, а затем выводит эти значения на экран компьютера. При этом переменная **vspom** в процессе выполнения программы несколько раз меняет значение. Вначале ей присваивается значение остатка от деления на 500, затем на 100, и наконец на 50. Ниже приводится текст данной программы.

```
program bankomat;
  Uses Crt;
  var
      sum, k10, k50, k100, k500, vspom:integer;
begin
{команда очистки экрана}
  ClrScr;
  writeln('Введите сумму, кратную 10');
{ввод исходной величины}
  readln(sum);
{вычисление количества купюр по 500 руб.}
  k500:=sum div 500;
  vspom:=sum mod 500;
{вычисление количества купюр по 100 руб.}
  k100:=vspom div 100;
  vspom:=vspom mod 100;
{вычисление количества купюр по 50 руб.}
  k50:=vspom div 50;
  vspom:=vspom mod 50;
{вычисление количества купюр по 10 руб.}
  k10:=vspom div 10;
  writeln('количество купюр по 500 руб. - ',k500);
{вывод}
  writeln('количество купюр по 100 руб. - ',k100);
{результатов}
  writeln('количество купюр по 50 руб. - ',k50);
{на экран}
  writeln('количество купюр по 10 руб. - ',k10);
{компьютера}
  readln
end.
```

Для лучшего понимания данной программы она снабжена комментариями. Мы уже упоминали о том, что комментарии представляют собой пояснения к тексту программы, а теперь рассмотрим их использование более подробно. Это необходимо сделать ввиду того, что комментарии являются неотъемлемой частью любой достаточно большой по объему и сложной программы. При их отсутствии в такой программе бывает непросто разобраться даже опытному программисту.

Правила использования комментариев в программе достаточно либеральны. Они могут располагаться в любом месте программы. Комментарии могут находиться на той же строке, что и основной текст программы, либо занимать отдельную строку или несколько строчек в программе. Вне зависимости от места и способа расположения комментариев при запуске программы на выполнение компилятор игнорирует комментарии.

Комментарии, таким образом, можно добавлять в программу там, где это удобно программисту, но при этом нужно помнить о том, что любой комментарий должен быть обязательно ограничен фигурными скобками или символами (* и *) с обеих сторон. При просмотре текста ранее составленной программы комментарии легко определить визуально, так как в отличие от основного текста программы они выделены серым цветом.

2.3. Использование вещественных чисел при вычислениях

Естественно, что задачи, решаемые с помощью Турбо Паскаля, не ограничиваются такими, в которых при расчетах используются только целые числа. Для того чтобы можно было использовать переменные, которые могут принимать вещественные значения (т. е. значения в виде десятичных дробей), следует в разделе описания переменных описать их с помощью типа **`real`** **(**вещественный).

Вещественные числа в Паскале можно записывать двумя способами – в виде с фиксированной точкой и в виде с плавающей точкой. Запись с фиксированной точкой похожа на обычную запись десятичной дроби в математике, только целая часть от дробной отделяется не запятой, а точкой. Например, число 10,23 в виде с фиксированной точкой записывается как 10.23.

Другим способом представления вещественных чисел является запись с плавающей точкой. В этом случае число представляется как произведение двух сомножителей - $\mathbf{m \cdot 10^n}$, где **`m`** – мантисса числа, а **`n`** – показатель числа. Мантисса – это целое число или вещественное число с фиксированной точкой, которое может принимать значение в диапазоне от 1 до 10. Показатель может быть только целым числом. Именно в таком виде по умолчанию выводятся на экран компьютера вещественные числа, причем запись в виде с плавающей точкой будет выглядеть следующим образом – **mEn**, где **E** заменяет собой основание степени – число 10. Перед показателем указывается его знак. Число 125.73 (далее в пособии мы будем использовать именно такую запись десятичных дробей) в виде с плавающей точкой будет записано как 1.2573000000E+02, то есть $1.2573 \cdot 10^2$, а число 0.00441 будет записано как 4.4100000000E-03, то есть $4.41 \cdot 10^{-3}$. Представление чисел в виде с плавающей точкой, как правило, используется для записи либо очень больших по модулю чисел, либо очень малых, так как и в том, и в другом случае при записи числа в обычном виде оно будет воспроизводиться с большим количеством нулей, что затрудняет восприятие данного числа. Но для представления результатов в большинстве задач все же более наглядным и удобным является вид числа с фиксированной точкой.

В качестве примера использования вещественных переменных рассмотрим задачу вычисления объема параллелепипеда, где его высота, длина и ширина – вещественные числа. В программе, написанной для решения данной задачи, используются 4 переменные. Все они описаны как вещественные. Это **`l`** – длина параллелепипеда, **`w`** – его ширина, **`h`** – высота и

v – искомый объем. Исходные данные вводятся с помощью операторов **readln.** Затем вычисляется произведение этих 3 величин и присваивается переменной **v**, значение которой выводится оператором **writeln.** Результат необходимо вывести в виде с фиксированной точкой. Ниже приводится текст программы.

```
program paral;
Uses Crt;
var l,w,h,v: real;
begin
ClrScr;
writeln('Введите длину параллелепипеда');
readln (l);
writeln('Введите ширину параллелепипеда');
readln (w);
writeln('Введите высоту параллелепипеда');
readln (h);
v:=l*w*h;
writeln ('Объем параллелепипеда равен ',v:6:2);
readln
end.
```

Для того чтобы представить вещественное число в виде с фиксированной точкой, в языке Паскаль предусмотрена возможность ***форматированного вывода,*** т. е. программист может сам определять внешний вид выводимого значения вещественной переменной.

Для этого в операторе вывода после имени переменной нужно поставить двоеточие. После двоеточия указывается общее количество символов, отводимое под значение данной переменной, далее ставится еще одно двоеточие, а затем указывается количество цифр в дробной части числа. Не нужно забывать при этом, что одну позицию нужно оставить для точки. Например, если мы хотим, чтобы в программе выводимое на экран значение некоторой переменной **t**, относящейся к вещественному типу, занимало 8 позиций, из которых 3 отводится под дробную часть, 1 – под десятичную точку, а 4 – под целую часть, то мы запишем наши пожелания в операторе вывода следующим образом:

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

Следует заметить, что если фактически в вещественном числе – значении форматируемой переменной содержится больше цифр после точки, чем указано в операторе вывода после второго двоеточия, то «лишние» цифры отбрасываются, а если число цифр после точки меньше, чем указанное, то недостающие цифры заменяются нулями.

Исходя из вышеизложенного, в нашей программе оператор вывода выглядит следующим образом:

```
writeln ('Объем параллелепипеда равен', v:6:2)
```

т. е. под значение переменной использовано 6 символов, в том числе 3 символа для целой части, 1 – для точки и 2 – для дробной части. Тогда при исходных данных h=4.2, w=2.5, h=1.8 результат будет выведен в следующем виде:

Объем параллелепипеда равен 18.90

т. е. результат будет представлен в виде, удобном для восприятия человеком.

В вышеприведенной программе мы использовали только переменные вещественного типа, но в дальнейшем нам придется составлять программы, в которых будут использоваться переменные как вещественного типа, так и целого, причем эти переменные будут взаимодействовать друг с другом. Взаимодействие это может проявляться и в том, что в программах будут встречаться математические выражения, в которых имеются и целые, и вещественные переменные, и константы. Естественно, что возникает вопрос: к какому типу будет относиться результат вычислений с участием величин разного типа. Это важно еще и потому, что если мы, например, присвоим целой переменной вещественное значение, то при компиляции программы будет выдано сообщение о несоответствии используемых типов. На поставленный вопрос отвечает специальный набор правил языка Паскаль.

Приведем список этих правил:

1. Если над двумя величинами целого типа производятся операции сложения, вычитания или умножения, то результатом данной операции будет величина целого типа.

2. Если операции сложения, вычитания или умножения произоводятся над двумя вещественными величинами, то результатом операции будет вещественная величина.

3. Если операции сложения, вычитания или умножения производятся над двумя величинами, одна из которых является целой, а другая вещественной, то результатом будет вещественная величина.

4. Если над любыми числовыми величинами (целочисленными или вещественными) производится операция деления (не целочисленного), то ее результатом будет вещественная величина.

На последнее правило следует обратить особое внимание, так как даже если в операции деления участвуют две целые величины, то результатом ее будет величина вещественная. Проиллюстрируем это следующим простым примером. Пусть нам нужно найти частное от деления двух целых чисел – 5 разделить на 4. Результатом будет число 1.25, то есть величина явно вещественная. Поэтому если, к примеру, в программе у нас имеются некие две переменные **k** и **l**, причем обе они относятся к целому типу, но в программе может встретиться операция деления одной переменной на другую, то для хранения результата этой операции следует использовать переменную **r**, которая должна быть описана как вещественная.

Задания для самостоятельной работы к главе 2

1. По заданной длине и ширине вычислить площадь прямоугольника. Исходные данные вводятся с клавиатуры, результат выводится на экран компьютера. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

2. По заданному радиусу вычислить длину окружности и площадь круга. Величину числа π принять равной 3.14. Исходные данные вводятся с клавиатуры, результат выводится на экран компьютера. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

3. По заданным катетам вычислить площадь прямоугольного треугольника. Исходные данные вводятся с клавиатуры, результат выводится на экран компьютера. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

4. Перевести объем оперативной памяти компьютера, выраженный в мегабайтах, в килобайты и байты. Для описания переменных использовать тип **`longint.`** Исходные данные вводятся с клавиатуры, результат выводится на экран компьютера. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

5. Составить программу, которая переводит вес, указанный в фунтах, в килограммы (1 фунт = 0,409 кг). Исходные данные вводятся с клавиатуры, результат выводится на экран компьютера. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

6. Составить программу, которая переводит длину, указанную в дюймах, в сантиметры (1 дюйм = 2,54 см). Исходные данные вводятся с клавиатуры, результат выводится на экран компьютера. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

7. Найти сумму цифр двузначного числа, введенного с клавиатуры. При составлении программы использовать операции **`div`** и **`mod`**. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

8. Найти сумму цифр трехзначного числа, введенного с клавиатуры. При составлении программы использовать операции **`div`** и **`mod`**. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

9. По заданным длине, ширине и высоте вычислить площадь поверхности параллелепипеда. Исходные данные вводятся с клавиатуры, результат выводится на экран компьютера. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

10. Найти среднее арифметическое трех чисел a, b и c, введенных с клавиатуры. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

11. Написать программу, которая величину угла, выраженную в градусах, переводит в радианы (для этого величину угла нужно умножить на π и разделить на 180). Величину числа π принять равной 3.14. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

12. Определить путь, пройденный пешеходом, если известны его средняя скорость (км/ч) и время его движения (в часах). Исходные данные вводятся с клавиатуры, результат выводится на экран компьютера. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

13. Определить среднюю скорость пешехода, если известны пройденный им путь (в километрах) и время его движения (в часах). Исходные данные вводятся с клавиатуры, результат выводится на экран компьютера. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

14. Подсчитать стоимость одной поездки в метро, исходя из стоимости проездной карточки и количества поездок, на которые эта карточка рассчитана. Исходные данные вводятся с клавиатуры, результат выводится на экран компьютера. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

15. Составить программу, которая бы определяла разницу между двумя моментами времени одних суток, выраженными в часах и минутах. Первый момент времени предшествует второму. Сама разница также должна быть выражена в часах и минутах. Значения как первого, так и второго моментов времени должны быть введены в компьютер с клавиатуры. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

Глава 3. Ветвления в программе

При решении различных задач нередко возникает ситуация, когда нужно выбирать один из двух или нескольких возможных вариантов дальнейшего хода программы. В этом случае в программе используется структура, называемая ***ветвлением***. В отличие от программ линейной структуры, в которых все операторы выполняются последовательно друг за другом, в программах с ветвлением в зависимости от значений вводимых исходных данных может работать одна из двух или нескольких ветвей, вариантов программы. 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.

Начнем мы рассмотрение условных операторов с более простого случая, когда 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`**.

3.1. Простой условный оператор

Общий вид оператора **`if`** следующий:

```
if  условие
then вариант 1
else вариант 2;
```

служебные слова **`if`**, **`then`** и **`else`** в переводе с английского означают соответственно **`если`**, **`то`**, **`иначе`**. Оператор действует следующим образом: сначала проверяется, 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.

Классическим примером использования оператора **if** является программа, определяющая большее из двух введенных чисел. В этой программе с помощью оператора **readln** первоначально вводятся два числа **a** и **b**, затем проверяется истинность условия **a** > **b**. Если данное условие истинно, то выполняется оператор **writeln**, записанный после служебного слова **then**. На экран компьютера выводится число **a**, которое является максимальным. Если же условие ложно, то выполняется оператор, находящийся после **else**. Тогда на экран выводится являющееся максимальным число **b**. Ниже приводится текст программы.

```
program maxim;
Uses Crt;
var a,b:integer;
begin
 ClrScr;
 writeln ('Введите первое число');
 readln(a);
 writeln('Введите второе число');
 readln(b);
 if a>b
 then writeln(a,' большее из двух чисел')
 else writeln(b,' большее из двух чисел');
readln
end.
```

Обратите внимание на то, что хотя конструкция **if...then...else** расположена на трех строчках, между этими строчками не ставится точка с запятой, так как вся эта конструкция представляет собой единый условный оператор, который в качестве своих элементов может включать в себя другие операторы (в данном случае операторы вывода).

Условный оператор **if** удобно использовать для составления различных программ-тестов, проверяющих знания пользователя. Например, программа, приведенная ниже, проверяет, знает ли пользователь дату первого полета человека в космос. В случае правильного ответа на экран выводится сообщение «Вы дали правильный ответ», в противном случае сообщение «Вы ошиблись».

В этой программе с помощью оператора **writeln** пользователю задается вопрос, ответ на который он должен дать, введя дату с клавиатуры. Введенное пользователем значение присваивается переменной **i**. Затем значение этой переменной сравнивается с правильным ответом – числом 1961. Если равенство выполняется, то оператором **writeln** выводится на

экран сообщение о правильном ответе, если значение **i** не совпадает с правильной датой, выводится сообщение об ошибке.

```
program flight;
  Uses Crt;
  var i:integer;
begin
  Clrscr;
  writeln ('В каком году был совершен первый полет человека в космос?');
  writeln ('Введите число и нажмите "Enter"');
  readln(i);
  if i=1961
  then writeln ('Вы дали правильный ответ')
  else writeln('Вы ошиблись');
  readln
end.
```

Вышеописанный вид условного оператора 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.

Как работают сокращенные условные операторы, разберем на примере следующей задачи: нам нужно отыскать максимальное число на сей раз среди трех введенных с клавиатуры чисел. Ниже приводится текст программы

```
program max3;
Uses Crt;
var a,b,c,max:integer;
begin
 ClrScr;
 writeln('Введите число');
 readln(a);
 writeln('Введите число');
 readln(b);
 writeln('Введите число');
```

```
 readln(c);
 max:=a;
   if b>max then max:=b;
   if c>max then max:=c;
writeln('Максимум равен ',max);
readln
end.
```

В данной программе операторами **readln** производится ввод исходных данных – 3 чисел, которые присваиваются переменным **a, b** и **c**. Кроме переменных, содержащих сравниваемые числа, в программе используется еще одна переменная **max**. Вначале переменной **max** присваивается значение, представляющее собой первое число, содержащееся в переменной **a**. Затем переменная **max** сравнивается с переменной **b** с помощью сокращенного условного оператора **if**. Если значение **b** больше, чем значение **max**, то переменной **max** присваивается новое значение, равное **b**. Если же значение **b** меньше или равно значению **max**, то никаких действий сокращенный условный оператор больше не производит, и программа переходит к выполнению следующего оператора. В любом случае в итоге действия условного оператора мы в переменной **max** получаем максимальное из двух чисел **a** и **b**.

Следующий оператор в программе также относится к числу сокращенных условных операторов. Этот оператор сравнивает имеющееся в переменной **max** максимальное из двух чисел с третьим, содержащимся в переменной **c**. Если значение **c**, больше, чем **max**, то значение **max** изменяется: ей присваивается значение **c**. Если **c** меньше или равно **max**, то **max** остается без изменений. В итоге последовательного действия двух сокращенных условных операторов мы получим в переменной **max** искомую величину – максимальное из трех чисел. Остается только вывести значение переменной **max** оператором **writeln** на экран компьютера – это и будет решение поставленной задачи.

3.2. Составные операторы в условном операторе

Во всех рассмотренных до сих пор примерах в каждой из ветвей условного оператора совершалось не более одного действия, производимого каким-либо одним оператором. Но нередко в ходе 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** (или после обоих этих слов).

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 (вложенным называется условный оператор, входящий в качестве составной части в другой условный оператор) и другие виды операторов. Количество простых операторов, входящих в составной оператор, не ограничено.

Операторы, входящие в составной оператор, друг от друга отделяются точками с запятой. После слова **begin** и перед словом **end** точка с запятой не ставится. Не ставится в данном случае точка с запятой и после слова **end** (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 мы рассмотрим в данном пособии ниже.

Рассмотрим использование составного оператора на примере следующей задачи. Администрация одного магазина для привлечения большего числа клиентов ввела правило, согласно которому каждый покупатель, который приобрел товар на сумму более 1000 руб., имеет право на скидку в размере 3% со стоимости покупок.

Требуется составить программу, облегчающую работу сотрудников магазина. Эта программа в случае, если стоимость покупки превышает указанную сумму, должна подсчитывать величину скидки и ту сумму, которую должен заплатить покупатель с учетом скидки. В случае же, если стоимость покупок меньше 1000 руб., программа должна выдавать сообщение о том, что покупка должна быть оплачена полностью. Ниже приводится программа, производящая необходимые расчеты.

```
program magazin;
Uses Crt;
  var sum,sumsk,skidka:real;
```

```
begin
 ClrScr;
 writeln('Введите сумму покупки');
 readln(sum);
 if sum>1000 then
   begin
   skidka:=sum*0.03;
   sumsk:=sum-skidka;
   writeln('Сумма скидки составляет ',skidka:7:2);
   writeln('Покупатель должен заплатить
   ',sumsk:7:2)
   end
            else
   writeln('Покупатель оплачивает покупку
   полностью');
  readln
end.
```

В этой программе используются три переменные вещественного типа: **sum** – сумма покупки без учета скидок, **skidka** – величина 3%-й скидки, **sumsk** – стоимость покупки с учетом скидки. В начале программы с клавиатуры вводится сумма покупки. Затем в действие вступает условный оператор. Если величина переменной **sum** больше чем 1000, то будет работать тот вариант программы, который находится после служебного слова **then**. Этот вариант представляет собой составной оператор, состоящий в свою очередь из 4 простых операторов.

В первом из этих операторов находится величина скидки путем умножения значения **sum** на 0.03. Найденная величина присваивается переменной **skidka.** Во втором операторе определяется сумма покупки со скидкой путем вычитания величины скидки из начальной суммы. Эта величина присваивается переменной **sumsk.** В третьем и четвертом операторах производится вывод подсчитанных в предыдущих двух операторах величин на экран компьютера.

Вывод этот производится в отформатированном виде. Для значения каждой выводимой переменной предусмотрено два разряда в дробной части, соответствующие копейкам. Следовательно, вывод переменных будет осуществляться таким образом, что цифры перед точкой в каждой из сумм будут соответствовать рублям, а цифры, расположенные после точки – копейкам.

В том случае, если сумма покупки составляет меньше 1000 руб., будет работать вариант программы, находящийся после служебного слова **else**. В этой ветви программы содержится только один простой оператор **writeln**, который выводит на экран компьютера сообщение о том, что покупатель должен оплатить покупку полностью.

3.3. 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
Значение 1:  Вариант1;
Значение 2: Вариант2;
.....................................
Значение n: Вариант N;
Else Вариант N+1
end;
```

где **Case, of , Else** – служебные слова языка Паскаль. Словосочетание **Case of** переводится на русский как «В случае если». Селектором называется переменная целого или символьного типа (о символьных переменных мы расскажем более подробно во II части Практикума).

Переменная-селектор может принимать различные значения. 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 одно 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,** 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 **case** служебным словом **end**, после которого ставится точка с запятой.

Разберем следующий фрагмент программы, в которой использован оператор **case**:

```
case k of
1: x:=x+5;
2: x:=x+10;
3: x:=x+20;
else x:=0
end;
```

В данном условном операторе в роли переменной-селектора выступает целочисленная переменная **k**. Если **k** принимает значение равное 1 (значения переменной-селектора могут либо вводиться с клавиатуры оператором **read** или **readln**, либо присваиваться в программе оператором присваивания), то текущее значение другой целочисленной переменной **x** увеличивается на 5; если **k** принимает значение равное 2, то текущее значение **x** увеличивается на 10; если **k** принимает значение 3, то значение **x** увеличивается на 20; если же переменная-селектор принимает любое другое значение, то переменная **x** обнуляется. Каждое из указанных после двоеточия действий производится простым оператором присваивания. Составные операторы внутри данного оператора **case** отсутствуют.

Возможности, предоставляемые программисту оператором **case**, мы используем в программе, которая имитирует работу простейшего электронного калькулятора. По запросу пользователя программа выполняет одно из четырех основных арифметических действий над любыми двумя введенными им с клавиатуры целыми числами.

В начале работы программы-калькулятора пользователю предлагается ввести с клавиатуры два целых числа, над которыми будет произведена одна из арифметических операций. Затем, после ввода чисел операторами **readln,** на экран компьютера операторами **writeln** выводится текст программного меню. Этот текст сообщает пользователю программы, нажатие какой клавиши соответствует выполнению какой арифметической операции. Например, нажатие клавиши с цифрой 1 производит сложение введенных чисел, 2 – вычитание и т. д.

Введенное пользователем значение считывается оператором **readln** и присваивается переменной **x**, которая в данной программе выступает в роли переменной-селектора. Далее в действие в программе вступает условный оператор **case**. В трех случаях (сложение, вычитание, умножение) для получения результата используется простой оператор **writeln**.

Каждый из операторов **writeln** вначале выводит значения операндов, над которыми производится действия, а между ними указывается знак производимой арифметической операции. Затем на экран выводится знак

равенства (так как этот знак и знак операции являются здесь фрагментами текста, то в списке вывода заключены в апострофы), а в конце указывается само вычисляемое выражение, которое подсчитывается компьютером и выводится на экран.

В последнем, четвертом случае необходимо использовать составной оператор. Причина заключается в том, что в этом случае результат операции будет числом вещественным. Поэтому вначале подсчитывается результат деления, он присваивается вещественной переменной **d**, а затем в списке вывода оператора **writeln** выводится значение переменной **d** в отформатированном виде, что обеспечивает точность выведенного вещественного числа-результата до третьего знака после запятой. Ниже приводится текст программы:

```
program kalkulat;
  Uses Crt;
  var a,b,x:integer; d:real;
begin
  ClrScr;
  writeln('Введите первое целое число');
  readln(a);
  writeln('Введите второе целое число');
  readln(b);
  writeln('Выберите действие, производимое над
числами');
  writeln('(введите соответствующую цифру)');
  writeln ('1 - сложение, 2 - вычитание, 3 -
умножение, 4 - деление');
  readln(x);
  case x of
    1: writeln(a,'+',b,'=',a+b);
    2: writeln(a,'-',b,'=',a-b);
    3: writeln(a,'*',b,'=',a*b);
    4: begin
       d:=a/b;
       writeln(a,'/',b,'=',d:7:3)
       end
  end;
  readln
end.
```

Рассмотренный выше вариант оператора **case** является несколько упрощенным. В нем каждому варианту действий соответствует только одно значение переменной-селектора. Возможна, однако, и другая форма того же оператора, когда каждому варианту действий соответствует какая–либо группа значений селектора.

Эта группа значений, находящаяся в операторе перед двоеточием, может представлять собой как несколько отдельных значений, перечисленных через запятую, так и целый диапазон значений. В последнем случае в соответствующем разделе оператора указывается только начальное и конечное значение диапазона, а между ними ставятся 2 символа точки.

Поясним сказанное на следующем примере. Перед нами фрагмент программы с условным оператором **case**, в котором переменной **y** присваиваются различные значения, представляющие собой степени **x**. Какое именно значение будет присвоено переменной **y**, зависит от значения переменной-селектора **n**:

```
case n of
3,5,7: y:=x*x;
10..20: y:=x*x*x;
30..40: y:=x*x*x*x;
else y:=x
end;
```

В случае, если переменная **n** принимает значение, равное одному из чисел 3, 5 или 7, то значение **y** будет равно квадрату **x**. Если **n** примет любое целочисленное значение из диапазона от 10 до 20 (10,11, 12 и т. д. до 20 включительно), то **y** будет присвоено значение, равное кубу **x**. Если же **n** примет любое значение из диапазона от 30 до 40, то **y** получит значение, равное четвертой степени **x**. Наконец, если **n** не примет ни одно из вышеуказанных значений, то **y** будет равно первой степени **x**.

Приведенную выше форму записи оператора **case** мы используем для решения следующей задачи: требуется составить программу, которая запрашивала бы пользователя о том, какой сейчас месяц, и по введенному пользователем номеру месяца определяла бы текущее время года. При этом сообщение о времени года должно выводиться на экран компьютера соответствующим сезону цветом. Если сейчас зима, то сообщение об этом должно выводиться белым цветом, для весны сообщение выводится зеленым цветом, для лета – красным цветом и для осени - желтым. Текст программы приведен ниже

```
program month;
Uses Crt;
var m:integer;
begin
ClrScr;
 writeln('Введите номер месяца по григорианскому
 календарю');
 readln(m);
 case m of
 1,2,12:
```

```
    begin
    textcolor(White);
    writeln('Зима')
    end;
  3,4,5:
    begin
    textcolor(Green);
    writeln('Весна')
    end;
  6,7,8:
    begin
    textcolor(Red);
    writeln('Лето')
       end;
  9,10,11:
    begin
    textcolor(Yellow);
    writeln('Осень')
    end;
   else writeln('Месяца с таким номером не
   существует')
  end;
readln
end.
```

В данной программе используется всего одна переменная – номер текущего месяца **m**, выступающий в роли переменной-селектора. В зависимости от вводимого пользователем значения переменной выполняется один из пяти возможных вариантов работы программы.

В первом случае для зимних месяцев с номерами 1, 2 или 12 можно использовать один простой оператор **writeln**, который выводит сообщение о том, что сейчас зима. Для вывода информации в Паскале по умолчанию используется белый цвет, но в данной программе лучше все же указать цвет выводимого сообщения, т.к. при предыдущих запусках программы цвет текста мог изменяться.

Следующие три варианта работы программы аналогичны друг другу. Каждый из них (для месяцев с третьего по пятый, с шестого по восьмой или с девятого по одиннадцатый) представляет собой составной оператор, который включает в себя два простых. Первый из этих операторов устанавливает цвет сообщения (соответственно, зеленый, красный или желтый). Вторым оператором является оператор вывода, который и выводит на экран компьютера нужным цветом соответствующий текст.

Наконец, последний вариант работы программы соответствует тому случаю, когда пользователь случайно ошибся (например, указал несуществующий в григорианском календаре нулевой или тринадцатый

месяц). В этом случае программа сообщает пользователю о совершенной им ошибке и на этом прекращает свою работу.

Задания для самостоятельной работы к главе 3

1. Составить программу, которая определяет наименьшее из двух чисел, введенных с клавиатуры. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

2. Составить программу, которая определяет наименьшее из трех чисел, введенных с клавиатуры. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

3. Составить программу, которая проверяет, является ли введенное с клавиатуры целое число четным или нечетным. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

4. Составить программу, которая проверяет, делится ли целое число, введенное с клавиатуры, без остатка на 5. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

5. Составить программу, которая проверяет, является ли число, введенное с клавиатуры, положительным числом, нулем или отрицательным числом. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

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 число 1, 2 или 3. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

7. Составить программу, которая предлагает пользователю ввести три 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 число 1, 2 или 3. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

8. Составить программу, которая предлагает пользователю ввести два числа и находит их сумму. Исходные данные вводятся белым цветом на черном фоне. Затем пользователю предлагается выбрать цвет для вывода результата с помощью меню: 1 – красный, 2 – зеленый, 3 – желтый, 4 – бирюзовый.

9. Составить программу, которая предлагает пользователю ввести три числа и находит их среднее арифметическое. Исходные данные вводятся белым цветом на черном фоне. Затем пользователю предлагается выбрать цвет для вывода результата с помощью меню: 1 – красный, 2 – зеленый, 3 – желтый, 4 – бирюзовый.

10. Составить программу, которая предлагает пользователю ввести длину и ширину прямоугольника и находит его площадь. Исходные данные вводятся белым цветом на черном фоне. Затем пользователю предлагается выбрать цвет для вывода результата с помощью меню: 1 – красный, 2 – зеленый, 3 – желтый, 4 – бирюзовый.

11. Составить программу, которая спрашивает у пользователя, в каком году был запущен первый искусственный спутник Земли. Исходные данные вводятся белым цветом на черном фоне. Если пользователь дал правильный ответ (1957), то сообщение об этом выводится зеленым цветом; если ответ был неверным, то сообщение об ошибке выводится красным.

12. Составить программу, которая предлагает пользователю ввести три числа, а затем проверяет, являются ли они пифагоровыми числами (пифагоровыми называются такие три числа, у которых сумма квадратов двух чисел равна квадрату третьего). Исходные данные вводятся белым цветом на черном фоне. Если числа являются пифагоровыми, то сообщение выводится зеленым цветом, если нет – то красным.

13. Составить программу, которая возводит введенное пользователем число в квадрат, если оно четное, и возводит его в куб, если оно нечетное. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

14. Составить программу, которая предлагает пользователю ввести три любых числа и находит сумму только тех из них, которые являются положительными. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

15. Составить программу, которая предлагает пользователю ввести три любых целых числа и находит сумму только тех из них, которые являются четными. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

Глава 4. Применение циклов

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 незначительно отличаются лишь значениями некоторых переменных или вводимых исходных данных. Рассмотрим эту ситуацию на следующем примере. Пусть нам необходимо разработать программу, создающую таблицу для перевода величины, выраженной в одних единицах измерения, в другие.

Конкретизируем задачу следующим образом: нам требуется составить таблицу для перевода веса, выраженного в тоннах, в килограммы. Созданная программой таблица должна охватывать диапазон от 1 до 50 т.

Для каждого целого значения, выраженного в тоннах (1,2,3 и т. д. вплоть до 50), процесс перевода этого веса в килограммы и создания соответствующей строки в таблице будет осуществляться с помощью двух операторов. Первый из этих операторов производит вычисление суммы в килограммах, а второй оператор выводит исходную величину (в тоннах) и результат вычислений (в килограммах) на экран компьютера. Данная группа операторов будет выглядеть следующим образом:

```
kg:=tn*1000;
writeln(tn,kg);
```

где **tn** – вес, выраженный в тоннах, а **kg** – тот же вес, выраженный в килограммах, причем, как нетрудно понять, значение переменной **kg** напрямую зависит от значения **tn**.

Для того чтобы выполнить поставленную задачу, можно, конечно, поступить следующим образом: повторить данную группу операторов в программе 50 раз, а перед каждой группой операторов присваивать переменной **tn** новое значение либо с помощью оператора присваивания, либо, используя оператор ввода **readln**.

При этом начальным значением переменной **tn** должна быть единица, после выполнения вышеприведенных двух операторов переменная **kg** получит значение 1000. Затем, перед следующим выполнением группы из двух операторов **tn** должно быть присвоено значение 2, а после их выполнения **kg** будет иметь значение 2000. Последнее значение **tn** будет равно 50, а **kg** соответственно 50000. Понятно, однако, что при решении данной задачи указанным способом получившаяся программа будет чересчур громоздкой за счет многократного повторения одних и тех же действий.

Поэтому естественно, что возникает вопрос: нельзя ли использовать при решении этой и других подобных задач какую-либо структуру, которая бы облегчала и делала менее трудоемким решение этой задачи путем автоматического повторения данных операторов или групп операторов с соответствующим изменением значений переменной или переменных? Такая

структура в языке Паскаль существует, и называется она циклом. В общем виде цикл состоит из двух основных блоков:

1.Управляющая конструкция. Эта конструкция содержит некоторое условие, которое определяет число повторений цикла.

2. Тело цикла. Телом цикла называется простой или составной оператор (как известно из материала, изложенного в разд. 3.2, составным оператором называется группа операторов, заключенная в операторные скобки **begin** и **end**), 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.

Перед началом первого выполнения тела цикла или по окончании его выполнения (в зависимости от вида конкретного цикла) производится проверка содержащегося в управляющей конструкции условия. Если условие выполняется, то происходит выполнение тела цикла, после чего программа вновь возвращается к управляющей конструкции цикла, и осуществляется новая проверка условия и т. д.

Если же условие в управляющей конструкции оказывается невыполненным, то работа цикла завершается, и управление программой переходит к оператору, расположенному в программе вслед за циклом. При этом в правильно работающем цикле не должно происходить бесконечного повторения тела цикла, приводящего к зацикливанию программы.

Поэтому условие, находящееся в управляющей конструкции, должно быть составлено таким образом, чтобы в какой-то момент в ходе выполнения программы оно переставало выполняться, что должно привести к завершению работы цикла. В нашем примере с программой перевода тонн в килограммы таким изменением могло бы стать получение переменной **tn** значения большего, чем 50. В случае, когда **tn,** например, получает значение 51, цикл должен прекращать свою работу.

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 do**).

2. Цикл с предусловием (цикл **while**).

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

Каждый из этих видов цикла имеет свои особенности и область применения, которые мы и рассмотрим ниже. Общим для всех этих операторов цикла является то, что они включают в себя управляющие конструкции (в операторах **for..to** и **while** – это строка заголовка, а в операторе **repeat..until** – строка заголовка и завершающая строка) и тело цикла. Если количество повторений тела цикла заранее известно, то рекомендуется использовать оператор цикла **for..to**. В других случаях

нужно использовать оператор **repeat..until** (если для решения поставленной задачи требуется, чтобы тело цикла выполнялось хотя бы один раз) либо оператор **while** (в этом случае перед выполнением тела цикла проверяется, есть ли вообще необходимость в его выполнении). Далее перейдем к более подробному рассмотрению каждого из этих видов цикла.

4.1. Цикл с заранее известным 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. При этом начальное значение должно быть меньше 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, допускаются и некоторые другие типы, которые мы рассмотрим позднее. Категорически нельзя использовать в качестве счетчика переменные вещественного типа. 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;

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 общем случае в качестве тела цикла выступает составной оператор, в который входит несколько других. Однако это может быть и один оператор. В последнем случае телом цикла считается первый оператор, расположенный после служебного слова **do**. Следует обратить внимание на то, что между заголовком цикла и телом цикла не ставится точка с запятой.

Решение задачи по переводу центнеров в килограммы, рассмотренной нами в начале данной главы, будет тогда выглядеть следующим образом:

```
for tn:=1 to 50 do
  begin
  kg:=tn*1000;
  writeln(tn,kg)
  end;
```

В приведенном фрагменте программы роль счетчика цикла выполняет переменная **tn** (вес в тоннах). Начальное значение переменной цикла равно 1. Для этого начального значения выполняются действия, указанные в составном операторе, то есть будет вычислено значение **kg**, равное 1000, а затем значения 1 и 1000 будут выведены на экран компьютера.

Далее будет подсчитано значение **kg** для **tn,** равного 2, и числа 2 и 2000 будут выведены на экран в следующей строке. Аналогичные действия будут производиться до тех пор, пока не будет выведена последняя строка со значениями **tn** и **kg** – 50 и 50000. Цикл прекратит свою работу, когда переменная **tn** превысит конечное значение – 50. Исходя из приведенных выше пояснений понятно, что всего действия, содержащиеся в теле цикла, будут повторены 50 раз.

Таким образом, мы решили задачу, поставленную в начале данной главы, написав фрагмент программы из 5 строк. Нетрудно подсчитать, что если решать эту задачу без использования цикла, как мы делали раньше, то такой фрагмент программы включал бы более, чем 100 операторов.

Приведенный вариант цикла **for** не является единственным. Цикл типа **for** 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.

В качестве примера работы цикла типа **for** приведем программу, которая составляет таблицу для перевода расстояния, выраженного в милях, в километры. На экран компьютера должна быть выведена таблица для расстояний от 1 до **n** миль, где **n** – целое положительное число, вводимое с клавиатуры пользователем. 1 миля составляет 1,609 км. Таблица, выводимая на экран компьютера, должна иметь заголовок и состоять из двух столбцов. В левом столбце должны быть указаны расстояния в милях, а в правом - соответствующие им расстояния в километрах. Ниже приводится текст данной программы:

```
program milkm;
Uses Crt;
 var mile,n:integer; km: real;
begin
 ClrScr;
 writeln('Введите расстояние в милях');
 readln(n);
```

```
 writeln('Таблица перевода расстояний из миль в
километры');
 writeln('мили    километры');
 for mile:=1 to n do
   begin
   km:=mile*1.609;
   writeln(mile:3,'     ',km:7:3)
   end;
 readln
end.
```

В данной программе для решения задачи используются три переменные. Это, во-первых, переменная целого типа **mile**, содержащая расстояния в милях, которая одновременно является счетчиком цикла. Во-вторых, – это переменная **n**, содержащая максимальное расстояние в милях, которое нужно перевести в километры. Наконец – это переменная вещественного типа **km**, представляющая собой расстояние в километрах.

В программе с помощью оператора **readln** пользователь вводит величину **n**. Это требуется для определения параметров цикла. Начальное значение счетчика цикла **mile** равно 1, а конечное – **n**.

Тело цикла представляет собой составной оператор, содержащий 2 оператора. Первый оператор является оператором присваивания, в котором для каждого значения **mile** (расстояние в милях) вычисляется соответствующее ему значение **km** – расстояние в километрах.

Во втором операторе**,** входящем в состав тела цикла, - операторе **writeln** для обеих выводимых переменных используется форматный вывод. Разница между форматами вывода для первой и второй переменной заключается в следующем.

В первом случае (для целочисленной переменной **mile**) указывается лишь общее количество выводимых на экран символов (в данном случае – 3). Во втором случае (для вещественной переменной **km**) указывается и общее количество символов, и их количество в дробной части (в данном случае соответственно 7 и 3). Это делается для того, чтобы выровнять столбцы выводимых чисел.

4.2. Цикл с постусловием

Общий вид оператора цикла с постусловием следующий:

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

где **repeat** и **until** – служебные слова (**repeat** означает повторять, **until** – до тех пор).

Цикл работает следующим образом: вначале выполняется оператор или группа операторов, входящая в тело цикла. Затем проверяется, 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**, ложно. Этот цикл называют циклом с постусловием (латинская приставка «пост» означает «после»). Указанное 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.

Цикл с постусловием можно использовать для решения следующей задачи: требуется определить средний рост учеников в классе. Данные о росте каждого ученика вводятся с клавиатуры. Сколько учащихся в классе, заранее неизвестно, но известно, что признаком окончания ввода исходных данных является ввод нуля.

Для решения поставленной задачи требуется произвести следующие действия: вначале обеспечить ввод требуемых исходных данных, представляющих собой последовательность целых чисел (условимся, что рост мы измеряем с точностью до сантиметра). Далее нужно найти среднее арифметическое этой последовательности чисел. Для этого сначала нужно найти сумму членов последовательности.

Большую часть этих действий можно выполнить в ходе работы цикла. В разделе описаний программы опишем следующие переменные: **rost** – используется для хранения роста каждого отдельного ученика; **n** – общее число членов последовательности, которая будет на единицу больше, чем общее число учеников в классе, так как последний член последовательности – это, как мы помним не чей-либо рост, а признак конца; **sum** – сумма членов последовательности; **sred** – искомый средний рост учеников в классе (эта величина должна быть вещественной, так как она является результатом деления).

Перед началом работы цикла обнуляем переменные **sum** и **n**, так как число членов пока что должно быть равно нулю, а соответственно нулю должна быть равна и их сумма.

Далее после служебного слова **repeat** идут операторы тела цикла. Первыми двумя операторами тела цикла являются оператор вывода, содержащий приглашение на ввод роста очередного ученика, и оператор, напоминающий о том, что для окончания ввода требуется ввести нуль. Следующий оператор – оператор ввода обеспечивает присваивание

введенной с клавиатуры величины (роста данного ученика) переменной **rost**.

Далее в цикле находится оператор присваивания, который увеличивает текущее значение переменной **sum** на величину, равную введенному росту. Таким образом, к концу работы цикла в переменной **sum** мы получим сумму ростов всех учеников класса (при последнем выполнении цикла будет введен еще и дополнительный нуль, но он, понятно, на общую сумму не повлияет).

Но знания значения только переменной **sum** нам недостаточно для решения задачи, так как нам необходимо знать еще и число учеников. Для определения этого числа мы подсчитаем сначала число членов последовательности, которое будет лишь на единицу больше числа учеников. Для такого подсчета в цикле мы будем использовать счетчик **n**. Этот счетчик будет увеличиваться на единицу после ввода очередного члена последовательности.

В итоге после окончания работы цикла нам останется только определить средний рост, представляющий собой частное от деления итоговой суммы на число учеников класса. Первая величина нам уже известна, а вторую величину мы найдем, вычтя из итоговой величины **n** (общего числа членов последовательности) единицу. Частное от деления этих двух величин **sred** и будет искомой величиной, которую останется только вывести на экран компьютера оператором вывода. Ниже приводится текст данной программы.

```
program srrost;
Uses Crt;
 var rost,n,sum:integer;
     sred:real;
begin
 ClrScr;
 sum:=0;
 n:=0;
 repeat
   writeln('Введите рост очередного ученика в см');
   writeln('Для окончания ввода введите ноль');
   readln(rost);
   sum:=sum+rost;
   n:=n+1
 until rost=0;
 sred:=sum/(n-1);
 writeln('Средний рост учеников равен ',sred:7:2,' см');
```

```
 readln
end.
```

4.3. Использование генератора случайных чисел совместно с оператором цикла

Как уже ранее отмечалось, оператор цикла с постусловием целесообразно использовать в тех случаях, когда заранее нельзя предсказать, сколько раз цикл будет повторяться. В частности, с такой ситуацией часто можно столкнуться при разработке различных компьютерных игр, когда те или иные действия игрока повторяются заранее неизвестное количество раз.

Создадим собственными силами компьютерную игру со следующими правилами. Компьютер загадал некоторое целое число в диапазоне от 1 до 20. Игрок должен угадать загаданное число. Всего игроку дается 5 попыток угадать число. Игра продолжается до тех пор, пока игрок не угадал число или не исчерпал имеющиеся у него попытки. Игрок выигрывает в том случае, если он угадал число с любой из 5 попыток, и считается проигравшим, если попытки закончились, а он так и не угадал число. При этом требуется, чтобы число, которое загадал компьютер, не повторялось от игры к игре и каждый раз было другим.

Для того чтобы составить такую программу, нам нужно будет познакомиться с двумя новыми командами языка Паскаль, которые используются для работы с генератором случайных чисел.

Генератором случайных чисел называется специальная подпрограмма, входящая в состав языка Паскаль, которая выдает случайным образом выбранные случайные числа. Эти числа, в зависимости от указанных параметров, могут быть как целыми, так и дробными.

Мы для нашей программы рассмотрим тот вариант работы генератора, когда он выдает целые числа. Для того чтобы можно было использовать генератор случайных чисел в программе, его следует предварительно запустить в работу, или, как говорят, инициализировать. Инициализация генератора случайных чисел производится командой

```
Randomize;
```

Следующее, что требуется сделать, – это взять некоторое число из созданного ряда случайных чисел. Это действие производится посредством команды

```
random(n),
```

где **n** – указываемое пользователем целое число или имя целочисленной переменной. Команда **random** будет выбирать числа, которые находятся в диапазоне от 0 до **n-1**. Полученное командой **random** число можно в дальнейшем вывести на экран компьютера оператором вывода или

использовать в качестве составной части арифметического выражения. Если в программе будет, например, указана команда

```
random(10),
```

то данная команда выдаст число из ряда 0,1,2,…..9, т. е. первое число ряда равно 0, а последнее будет на единицу меньше, чем указанное в скобках.

Если же мы хотим, чтобы выводимое на экран компьютера посредством **random** число было натуральным однозначным числом (т. е. принимало значения 1,2,…9), то используем прием, показанный в нижеприведенном фрагменте программы:

```
Randomize;
writeln(random(9)+1);
```

в этом случае к выводимому значению будет добавлена единица. Поэтому минимальное из возможных выводимых чисел будет равно 1(0+1), а максимальное будет равно 9 (8+1).

Текст программы, использующей генератор случайных чисел, приведен ниже. Разберем эту программу.

```
program igra;
Uses  Crt;
 var
    x,y,n:integer;
begin
 ClrScr;
 Randomize;
 n:=1; x:=random(20)+1;
 Textcolor(cyan);
 writeln('Компьютер загадал целое число (от 1 до 20)');
 writeln('попробуйте угадать это число (всего у Вас будет
5 попыток)');
 repeat
    Textcolor(green);
    writeln(n,' попытка ');
    writeln('Введите число и нажмите клавишу Enter');
    readln(y);
    n:=n+1
until (x=y) or (n>5);
if x=y then
    begin
```

```
    Textcolor(lightred);
    writeln('Поздравляю Вас с выигрышем')
    end
       else
    begin
    Textcolor(lightblue);
    writeln ('Вы проиграли. Загаданное компьютером число
равно ',x)
    end;
 readln
end.
```

В данной программе после очистки экрана производится инициализация генератора случайных чисел. Затем командой **random** берется случайное число, которое присваивается целочисленной переменной **x**. Это и будет число, загаданное компьютером, но неизвестное заранее пользователю, которое он должен угадать.

В процессе угадывания числа компьютер должен фиксировать согласно условиям игры сделанное пользователем количество попыток. Это количество попыток будет содержать другая целочисленная переменная **n**. Так как попытки мы будем считать, начиная с первой, а не с нулевой, то мы присвоим переменной **n** начальное значение, равное единице.

Сам процесс угадывания числа будет производиться в цикле с постусловием. Тело цикла включает в себя следующие операции. Вначале пользователю оператором вывода предлагается ввести предполагаемое значение **x**. Для справки указывается номер попытки, который будет равен текущему значению переменной **n**. Предполагаемое значение **x** вводится в компьютер оператором **readln** и присваивается переменной **y**.

Последним оператором тела цикла является оператор присваивания, который увеличивает текущее значение переменной **n** на единицу, для того, чтобы при следующем повторении тела цикла (если оно будет иметь место) правильно фиксировался номер попытки.

Постусловие, которое определяет, нужно ли следующее повторение тела цикла, является сложным и состоит из двух простых условий.

Необходимость повторения тела цикла отпадает, во-первых, в том случае, если пользователь угадал число, т. е. введенное им предполагаемое значение **y** совпало с действительным значением **x**. Равенство значений этих двух переменных и будет одним из возможных условий окончания цикла.

Второй вариант окончания работы цикла заключается в том, что пользователь исчерпал имевшееся у него число попыток. В этом случае после завершения последней неудачной попытки (пятой по счету) переменной **n** будет присвоено новое значение, равное 6 (5+1). Таким

образом, вторым условием окончание цикла будет превышение перменной **n** значения 5, т. е. **n** > 5. Оба этих условия заключаются, согласно правилам языка Паскаль, в скобки и объединяются операцией **or** (логическое «или»), т. е. работа цикла завершится в том случае, если будет выполняться или одно, или другое условие.

По завершении работы цикла остается выяснить, было ли все-таки угадано загаданное компьютером число, и вывести для пользователя соответствующее сообщение. Для такой проверки используется условный оператор **if**, в заголовке которого указывается проверяемое условие – это опять-таки равенство значений переменных **x** и **y**. Если эти значения на данный момент равны, остается поздравить пользователя с победой и вывести сообщение об этом оператором вывода.

Сообщение о победе выводится в ветви программы, находящейся после служебного слова **then**. Если же значения переменных не совпадают, то нужно сообщить пользователю, что он проиграл, и проинформировать его о том, какое число было загадано на самом деле. Эту задачу выполняет составной оператор, находящийся в условном операторе после служебного слова **else**.

Для того чтобы процесс игры для пользователя был менее монотонным, выводимая программой информация раскрашена в разные цвета с помощью операторов **Textcolor**. Так, начальное сообщение об условиях игры выводится на экран бирюзовым цветом, информация о номере очередной попытки и приглашение на ввод пользователем очередного числа выводятся зеленым цветом. В том случае, если пользователь выиграл, то информация о победе выводится светло-красным цветом, а в случае проигрыша соответствующая информация и загаданное число – светло-синим цветом.

4.4. Использование оператора цикла для защиты от неправильного ввода данных

Еще одним применением цикла с постусловием является защита программы от неправильного ввода данных. В качестве примера рассмотрим уже составленную ранее программу, моделирующую работу калькулятора, рассмотренную в разделе 3.3. В первоначальном варианте этой программы отсутствует защита от неправильного ввода данных при выборе в меню одного из возможных вариантов действий. Напомним, что для выбора одного из действий, которое можно произвести над двумя введенными числами, в этой программе нужно ввести целое число от 1 до 4.

Для защиты программы от неверного ввода данных мы используем цикл с постусловием. Телом этого цикла будут операторы вывода, сообщающие пользователю о том, какие числа можно вводить для выбора вариантов дальнейших действий, и оператор ввода, который присваивает значение переменной **x**, являющейся переменной-селектором, используемой для выбора варианта. Текст тела цикла приводится ниже:

```
    writeln('Выберите действие, производимое над
числами');
    writeln('(введите соответствующую цифру)');
    writeln ('1 - сложение, 2 - вычитание, 3 -
умножение, 4 - деление');
    readln(x)
```

Это тело цикла будет повторяться до тех пор, пока не будет выполнено одно из условий, находящихся в завершающей строке цикла:

```
    (x=1) or (x=2) or (x=3) or (x=4)
```

Тогда весь оператор цикла будет выглядеть следующим образом:

```
    repeat
     writeln('Выберите действие, производимое над
числами');
     writeln('(введите соответствующую цифру)');
     writeln ('1 - сложение, 2 - вычитание, 3 -
умножение, 4 - деление');
     readln(x)
    until (x=1) or (x=2) or (x=3) or (x=4);
```

Теперь, если пользователь попробует ввести при выборе варианта в меню число 5 или 7, программа вновь будет возвращать его к вводу данных. Целиком же усовершенствованная программа будет выглядеть следующим образом:

```
    program kalk1;
    Uses Crt;
    var a,b,x:integer; d:real;
    begin
    ClrScr;
    writeln('Введите первое число');
    readln(a);
    writeln('Введите второе число');
    readln(b);
    repeat
     writeln('Выберите действие, производимое над
числами');
     writeln('(введите соответствующую цифру)');
     writeln ('1 - сложение, 2 - вычитание, 3 -
умножение, 4 - деление');
     readln(x)
    until (x=1) or (x=2) or (x=3) or (x=4);
    case x of
    1: writeln(a,'+',b,'=',a+b);
```

```
2: writeln(a,'-',b,'=',a-b);
3: writeln(a,'*',b,'=',a*b);
4: begin d:=a/b; writeln(a,'/',b,'=',d:7:3) end
end;
readln
end.
```

4.5. Цикл с предусловием

Общий вид оператора цикла с предусловием следующий:

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

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

Цикл работает таким образом: первоначально проверяется некоторое 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.

В качестве примера использования цикла типа **while** приведем программу разложения целого положительного числа на простые множители. Напомним, что простым называется целое число (кроме единицы), которое делится только на единицу и само на себя. Составным, или сложным называется число, которое делится еще и на другие числа. Любое составное число можно представить единственным образом в виде произведения простых чисел, которые и называются простыми множителями.

Наиболее простой (хотя и достаточно трудоемкий для человека, но не для компьютера) алгоритм разложения числа на множители заключается в следующем. Исходное число делится на все числа, начиная с 2. Если число нацело разделилось на делитель, то этот делитель является одним из искомых простых множителей. Тогда мы определяем частное от деления этих двух чисел и делим уже его на все числа, начиная с 2.

Если же исходное число или последнее получившееся частное не делится нацело на делитель, то значение делителя увеличиваем на единицу и уже это новое значение используем при дальнейшей работе. Так мы будем действовать до тех пор, пока значение делителя не станет равным частному

или самому числу (в том случае, если простых множителей для исходного числа не было найдено). На этом поиск простых множителей завершается.

В начале программы оператором вывода пользователю предлагается ввести с клавиатуры исходное число **c**, которое вводится оператором **readln**. Затем следующий оператор вывода сообщает о том, что далее будет производиться вывод простых множителей.

Для этого мы будем делить введенное число на некоторое число **d**. Если число **c** разделится на число **d** без остатка, то **d** будет одним из искомых множителей, на которые нужно разложить **c**. Деление без остатка подразумевает, что остаток от деления будет равен нулю, то есть должно выполняться соотношение **c mod d = 0**. Начинать поиск делителей мы начнем с наименьшего возможного – числа 2. Поэтому переменной **d** мы присвоим начальное значение 2.

Поиск простых множителей мы будем производить в цикле. Этот цикл с предусловием будет работать до тех пор, пока значение делителя **d** будет меньше или равно делимому **c**. Как только **d** станет больше **c,** работа цикла прекратится, так как дальнейший поиск не имеет смысла.

В теле цикла первым оператором будет условный оператор, проверяющий соотношение **c mod d = 0.** Если данное соотношение истинно, то текущее значение **d** является одним из искомых простых множителей, и тогда нужно будет выполнить две операции.

Во-первых, нужно вывести на экран компьютера текущее значение переменной **d**. Во вторых, для того, чтобы продолжить дальнейший поиск простых множителей, нужно найти частное от деления **c** на **d**. Это частное мы найдем посредством операции целочисленного деления **c div d**. Результат этой операции мы присвоим переменной **c,** и в дальнейшем будем работать с этим новым ее значением, ища для него возможные делители.

Вышеуказанные операции находятся в той ветви условного оператора, которая расположена после слова **then**. Если же **c** не разделилось на **d** без остатка, то работает ветвь условного оператора, которая расположена после **else**. В этом случае текущее значение делителя **d** увеличивается на единицу, и при новом выполнении тела цикла будет использоваться это новое значение делителя. Ниже приводится текст данной программы:

```
program razl;
Uses Crt;
 var c,d:integer;
begin
 Clrscr;
 writeln('введите  целое положительное число');
 readln(c);
 writeln('Число ',c,' можно разложить на следующие
простые множители:');
 d:=2;
```

```
 while d<=c do
  if c mod d = 0
  then begin write (' ',d); c:=c div d  end
  else d:=d+1;
 readln
end.
```

Таким образом, мы переберем все возможные значения простых множителей и выведем на экран те из них, которые действительно являются таковыми. Если исходное число является простым, то в результате работы программы на экран будет выведен единственный результат – само исходное число. Если же число является составным, то простых множителей будет как минимум два.

Задания для самостоятельной работы к главе 4

1. Составить программу, которая находит сумму всех натуральных чисел от 1 до **n**, где **n** – натуральное число, введенное пользователем с клавиатуры (натуральным называется целое и положительное число). Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

2. Составить программу, которая вычисляет факториал натурального числа **n**, введенного с клавиатуры. Факториалом натурального числа **n** называется произведение всех натуральных чисел от 1 до **n** (**1*2*3*...*n**).

3. Составить программу, которая выводит на экран компьютера таблицу перевода градусов в радианы для величин от 1 до **n** градусов, где **n** – натуральное число, введенное пользователем (**n** должно быть не более 360). Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

4. Составить программу, которая для введенного пользователем однозначного числа **n** выводит таблицу умножения этого числа на все числа от 1 до 9. Таблица должна выглядеть следующим образом (пример приведен для **n**, равного 6):

```
1*6=6
2*6=6
3*6=18
……….
9*6=54
```

Таблица должна быть выведена на экран компьютера на светло-сером фоне символами синего цвета.

5. Составить программу, которая выводит на экран компьютера таблицу. Данная таблица для величин от 0 до 50 миль должна переводить

расстояния, выраженные в милях, в километры с шагом, равным 5 милям. Внешний вид таблицы должен быть следующим:

```
мили    км
 0       0.000
 5       8.045
10      16.090
……….……….
45      72.405
50      80.450
```

Таблица должна выводиться на светло-сером фоне символами синего цвета.

6. Составить программу, которая выводит на экран компьютера таблицу. Данная таблица для величин от 0 до 100 фунтов должна переводить вес, выраженный в фунтах, в килограммы с шагом, равным 10 фунтам. Внешний вид таблицы должен быть следующим:

```
фунты  кг
 0      0.000
10      4.090
20      8.180
……….……….
 90     36.810
100     40.900
```

Таблица должна выводиться на светло-сером фоне символами синего цвета.

7. Составить программу, которая осуществляет ввод с клавиатуры последовательности из 10 целых чисел. Программа должна определять, сколько чисел, входящих в данную последовательность, являются четными, а сколько – нечетными. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

8. Составить программу, которая осуществляет ввод с клавиатуры последовательности из 10 целых чисел. Программа должна отдельно определять суммы всех четных элементов последовательности и всех нечетных элементов. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

9. Составить программу, которая осуществляет ввод с клавиатуры последовательности из 10 чисел. Программа должна определять, сколько чисел, входящих в данную последовательность, являются положительными, а сколько отрицательными. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

10. Составить программу, которая осуществляет ввод с клавиатуры последовательности из 10 чисел. Программа должна отдельно подсчитать суммы всех положительных элементов последовательности и всех

отрицательных элементов. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

11. В университете формируется баскетбольная команда, членами которой могут стать студенты, рост которых составляет не менее 175 см. Составить программу, которая позволила бы тренеру команды определить, сколько потенциальных кандидатов в команду имеется в студенческой группе, насчитывающей **n** человек. Количество студентов в группе (**n**) и данные о росте каждого из студентов группы вводятся в компьютер с клавиатуры. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

12. Клиент положил в банк сроком на 5 лет некоторую денежную сумму под определенный процент. Составить программу, которая бы определяла, какую величину составит сумма вклада по истечении срока хранения данного вклада в банке, если по условиям договора о вкладе, заключенного между клиентом и банком, в течение всего срока хранения вклада банковский процент не должен изменяться. Сумма вклада и процент вводятся в компьютер с клавиатуры. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

13. Составить программу, которая осуществляет ввод последовательности целых чисел с клавиатуры. Количество элементов последовательности заранее неизвестно, но известно, что признаком окончания ввода является ноль (который не является элементом последовательности). Требуется определить, сколько в данной последовательности чисел, которые делятся без остатка на 3. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

14. Составить программу, которая осуществляет ввод последовательности целых чисел с клавиатуры. Количество элементов последовательности заранее неизвестно, но известно, что признаком окончания ввода является ноль (который не является элементом последовательности). Требуется определить сумму всех элементов последовательности, которые делятся без остатка на 5. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

15. Составить программу, которая осуществляет ввод последовательности целых чисел с клавиатуры. Количество элементов последовательности заранее неизвестно, но известно, что признаком окончания ввода является ноль (который не является элементом последовательности). Требуется определить среднее арифметическое всех нечетных элементов последовательности. Фон экрана должен быть черным, исходные данные вводятся светло-красным цветом, вывод результата – светло-зеленым цветом.

Библиографический список

Бежанова М.М., Москвина Л.А. Практическое программирование. Приемы создания программ на языке Паскаль. — М.: Научный мир, 2000.

Емелина Е.И. Основы программирования на языке Паскаль. — М.: Финансы и статистика, 1997.

Культин Н.Б. Программирование в Turbo Pascal 7.0 и Delphi. СПб: БХВ, 1999.

Немнюгин С.А. Turbo Pascal: практикум. – СПб.: Питер, 2002.

Немнюгин С.А. Turbo Pascal: учебник. – СПб.: Питер, 2002.

Пестриков В.М., Маслобоев А.Н., Федоров О.К. Программирование в системе Turbo Pascal 7.0: учебное пособие /СПбГТУ РП. — СПб., 2002.

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

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

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

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

Ставровский А.Б. Turbo Pascal 7.0: учебник. — Киев: Издательская группа BHV, 2000.

Учебное издание

Артур Николаевич Маслобоев

Практикум по алгоритмическому программированию в системе Turbo Pascal

Часть I

Учебно-методическое пособие

Редактор и корректор Н.П. Новикова

Техн. редактор Л.Я. Титова Темплан 2010 г., поз. 110

Подп. к печати Формат 60x84/16. Бумага тип. № 1.

Печать офсетная. 3,75 уч.-изд. л. ; 3,75 усл. печ. л. Тираж 50 экз.

Изд. № 110. Цена «С». Заказ

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