

## Глава 1

# Введение в Turbo Pascal

### ***В этой главе...***

- Примеры простых программ
- Кое-что о типах данных
- Символы и зарезервированные слова
- Комментарии
- Переменные
- Константы

**М**ногие книги, посвященные Turbo Pascal, начинают знакомить читателя с этим языком программирования с таких элементарных понятий, как алфавит, зарезервированные слова, типы, переменные, операторы и т.п. В принципе подход “от простого к сложному” себя оправдывает, однако в данном случае представляется, что читатель лучше поймет смысл перечисленных выше элементарных понятий, если начать с примера простой программы, включающей эти элементы. Так и поступим. Попробуем создать простую программу, а затем запустим ее на выполнение, чтобы убедиться, что она работает должным образом.

## Примеры простых программ

### Ваша первая программа

Попробуем создать предельно простую программу, которая бы выводила на экран единственное слово (приветствие Hello!). Как выглядит подобная программа, демонстрирует пример 1.1.

#### **Пример 1.1**

```
program hello;  
begin  
  Write ('Hello!');  
end.
```

Первая строка приведенной выше программы, начинающаяся с зарезервированного слова PROGRAM, представляет собой заголовок. (Здесь, вероятно, уместно сказать, что длина заголовка, а также любой другой строки программы на Turbo Pascal не должна превышать 127 символов.) Вообще-то, наличие в программе заголовка — это требование стандартного Pascal. В Turbo Pascal заголовок не является обязательным элементом текста программы. Тем не менее, чтобы каждая рассматриваемая програм-

ма имела собственное имя, все примеры программ, с которыми нам придется иметь дело в этой книге, будем дополнять заголовками. (Подробнее о структуре программы и о месте в этой структуре заголовка речь пойдет в Приложении Г.)

Слово PROGRAM (а также BEGIN и END, которые вы также найдете в приведенной программе) относится к так называемым *зарезервированным* (в Turbo Pascal) *словам*, имеющим специальное назначение, о которых будет идти речь дальше в этой главе. После слова PROGRAM, через пробел, следует имя программы, присваиваемое автором при ее создании.

В оставшейся части программы — между еще двумя зарезервированными словами BEGIN и END — содержится ее тело (или раздел операторов). Здесь представлены все описанные средствами Turbo Pascal действия, позволяющие получить нужные результаты. В нашем случае программа выполняет единственное действие — выводит на экран приветствие Hello!. (О структуре программы и о том, какое место в этой структуре занимает тело программы, речь пойдет в Приложении В.)

Теперь только осталось ввести текст данной программы в окне интегрированной среды Turbo Pascal, а затем инициировать ее выполнение. При этом предполагается, что интегрированная среда Turbo Pascal уже запущена и ее окно присутствует на экране.



#### Запуск интегрированной среды

Как запустить интегрированную среду? Если ваш компьютер работает под управлением Windows, для этого можно дважды щелкнуть на соответствующем значке на рабочем столе (если нужный значок там присутствует) либо выбрать подходящий пункт в меню Пуск (если такой пункт там имеется). В противном случае следует открыть папку, в которой содержатся файлы интегрированной среды, затем папку BIN и дважды щелкнуть на значке файла TURBO.EXE. Окно интегрированной среды Turbo Pascal с введенным в нем текстом нашей программы показано на рис. 1.1.

Для выполнения программы, текст ...следует в меню Run...  
которой присутствует в окне ...выбрать одноименный пункт  
интегрированной среды Turbo Pascal,...

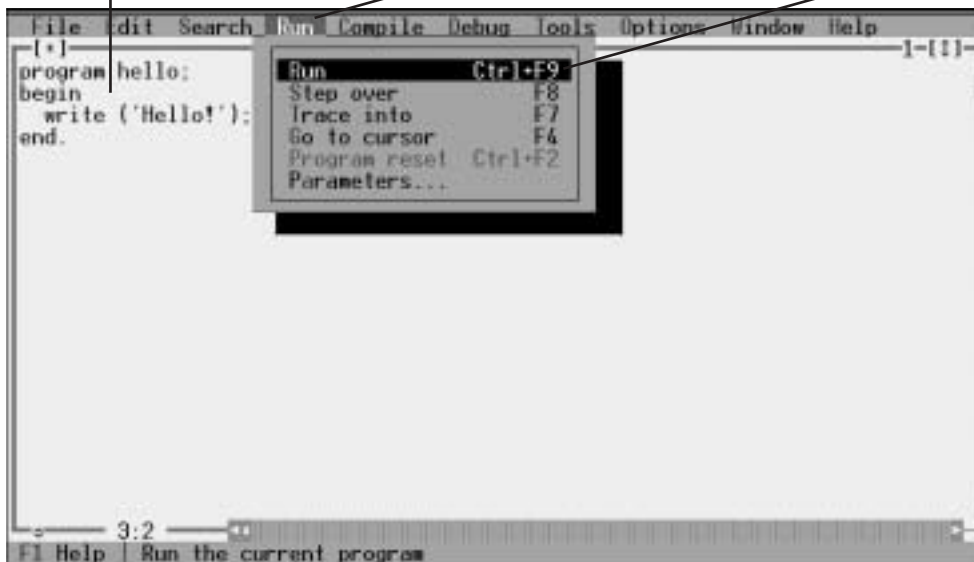


Рис. 1.1. Ввод программы в окне интегрированной среды ничем не отличается от ввода текста в окне редактора

После выбора пункта **Run** в одноименном меню, программа запускается на выполнение (рис. 1.1). В нашем случае выполняется одно только действие — на экран выводится единственное слово `Hello!`.



Как этот экран увидеть? Для этого следует воспользоваться комбинацией клавиш `<Alt+F5>`. В результате вместо интегрированной среды Turbo Pascal отображается экран вывода, в левом нижнем углу которого присутствует текст, выведенный нашей программой (рис. 1.2).



Рис. 1.2. Экран с текстом, выведенным программой



А если потребуется вернуть на экран окно интегрированной среды, воспользуйтесь еще раз указанной комбинацией клавиш.



Кстати, можно сделать так, чтобы на экране одновременно были видны исходный текст программы и результаты ее работы. Для этого в меню **Debug** (Отладка) среды разработчика выберем пункт **Output** (Вывод). В результате окно интегрированной среды примет вид, как на рис. 1.3.

О рассмотренной выше программе осталось только сказать, что оператор `Write` (вообще-то, это стандартная процедура) выполняет вывод представленного рядом в скобках текста. При этом предназначенный для вывода текст должен быть выделен апострофами.



В данном случае процедура `Write` осуществляет вывод на экран представленного рядом и выделенного апострофами текста.



С программой, о которой шла речь выше, полезно поэкспериментировать. Читателям предлагается вместо слова `Hello!` ввести свой текст. При этом следует помнить, что длина строки в программе на Turbo Pascal не должна превышать 127 символов.



Что если требуется вывести сообщение, длина которого превышает 127 символов? В этом случае ничто не мешает разделить это сообщение на несколько частей, используя для каждой части отдельный оператор `Write`.

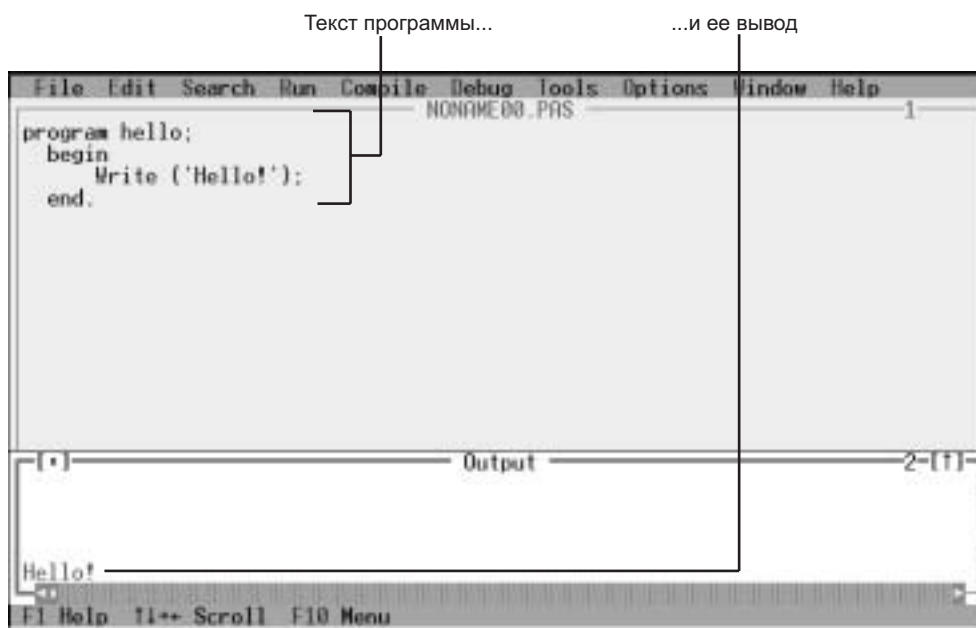


Рис. 1.3. Можно одновременно видеть текст программы и результаты ее работы

## Ваша вторая программа

То, с чем мы имели дело выше, лишь условно можно назвать программой. Попробуем создать что-то более сложное и полезное. Предположим, необходима программа для вычисления корней квадратного уравнения  $ax^2+bx+c=0$ . Конечно, для полного решения этого уравнения было бы необходимо предусмотреть различные случаи, когда его коэффициенты  $a$ ,  $b$  и  $c$  равны или не равны нулю. (Например, если  $a=0$ ,  $b=0$  и  $c=0$ , то уравнение имеет бесконечное множество решений. Если  $a=0$ ,  $b=0$ , а  $c\neq 0$ , то уравнение не имеет решений. Для этих и прочих частных случаев в будущей программе можно было бы предусмотреть вывод соответствующих текстовых сообщений.) Однако для упрощения задачи (и будущей программы) будем считать, что коэффициент  $a$  здесь не равен нулю, а дискриминант уравнения ( $b^2-4ac$ ) неотрицателен. Данная программа с учетом всех оговоренных выше условий содержится в примере 1.2 (рис. 1.4).

Наша программа именуется *Roots*<sup>1</sup>. Вторая строка в программе представляет собой раздел описания переменных. Все используемые переменные должны быть описаны в этом разделе. В нашем случае после слова **VAR** следует перечень из пяти переменных. Переменные  $a$ ,  $b$  и  $c$  представляют коэффициенты нашего квадратного уравнения, а  $x_1$  и  $x_2$  — его корни. Слово **Real** после этого перечня (через двоеточие) указывает, что данные переменные могут принимать только вещественные значения (рис. 1.4). Если какая-либо из этих переменных не фигурирует в разделе описания переменных, компилятор выведет соответствующее сообщение об ошибке. С наиболее используемыми типами данных мы познакомимся позже в этой главе.

Первая строка тела программы в приведенном выше примере представляет собой оператор чтения **Read** (так же, как **Write**, это стандартная процедура).

<sup>1</sup> *Roots* — корни (англ.).

### Пример 1.2

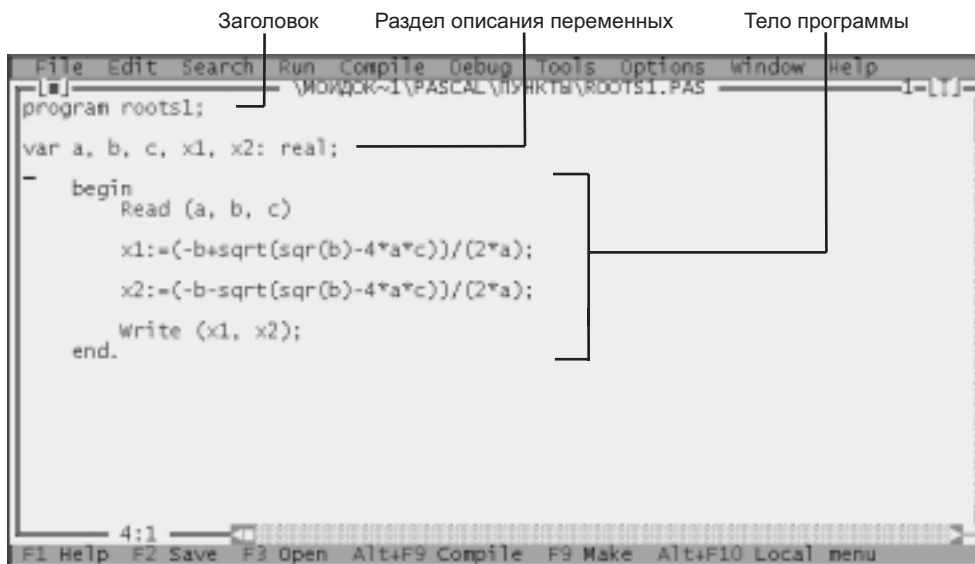


Рис. 1.4. Программа Roots1 в окне интегрированной среды разработчика



**Процедура Read** инициирует считывание соответствующего значения (значений) с клавиатуры (подробнее — в главе 5).

В данном случае процедура Read служит для ввода значений переменных  $a$ ,  $b$  и  $c$ . На практике это выглядит так. Дойдя до оператора Read, программа приостанавливает работу, и от пользователя программы (который заинтересован в получении результатов вычислений) требуется ввести значения соответствующих переменных. Пользователь вводит значение переменной  $a$  и нажимает клавишу <Enter>, а затем аналогично вводит значения переменных  $b$  и  $c$ . Если бы оператор чтения выглядел как Read ( $c$ ,  $a$ ,  $b$ ), то сначала пришлось бы ввести значение для переменной  $c$ , затем для  $a$  и, наконец, для  $b$ .

После операторов, позволяющих ввести значения переменных, следует оператор присваивания.

```
x1 := (-b+Sqrt (Sqr (b) -4*a*c)) / (2*a);
```

Знак присваивания ( $:=$ ) делит его на две части. В правой части представлено выражение, которое необходимо вычислить. Затем полученное значение присваивается переменной, содержащейся в левой части оператора. Данный оператор вычисляет значение одного из корней квадратного уравнения и присваивает его переменной  $x1$ . Если в традиционной форме записи, принятой в математике, это выражение выглядит как

$$\frac{-b+\sqrt{b^2-4ac}}{2a},$$

то представленное на языке программирования Turbo Pascal оно имеет несколько иной вид (см. выше). Во-первых, для возведения в квадрат предусмотрена специальная функция Sqr.



**Функция Sqr** возвращает значение, соответствующее квадрату переданного ей параметра (подробнее — в главе 4).

(Выражение  $\text{Sqr}(b)$  из Turbo Pascal соответствует  $b^2$ .) Во-вторых, выражение  $4ac$  (представленное в традиционной форме) в Turbo Pascal следует записывать как  $4*a*c$ , где символ  $*$  (звездочка) — знак умножения. Иными словами, в тексте программы на Turbo Pascal знак умножения нельзя опускать или заменять точкой.

Если же в традиционном математическом представлении выражение, вычисляющее один из корней квадратного уравнения, записано в две строки и операция деления представлена горизонтальной линией, разделяющей делимое и делитель, то в Turbo Pascal соответствующее выражение записывается в одну строку, а в качестве знака деления используется наклонная черта ( $/$ ). При этом вместо знака радикала в Turbo Pascal используется специальная функция  $\text{Sqr}$ .



**Функция  $\text{Sqr}$**  возвращает квадратный корень значения, переданного ей в качестве параметра (подробнее — в главе 4).

Оператор присваивания  $x2 := (-b - \text{Sqr}(\text{Sqr}(b) - 4*a*c)) / (2*a);$ , вычисляющий второй корень нашего квадратного уравнения, отличается от предыдущего оператора единственным знаком математической операции. Вместо знака плюс ( $+$ ) перед функцией вычисления квадратного корня здесь используется минус ( $-$ ).



К сказанному можно добавить, что операторы в тексте программы следует располагать определенным образом и при этом в меру использовать отступы и пустые строки. Отступы предназначены для того, чтобы показать, где одна структура включается в другую, а пустые строки — для отделения основных разделов программы.

В принципе текст программы можно записать и в одну строку — для компилятора это безразлично. Наша программа  $\text{Roots1}$ , представленная подобным образом, выглядит так:

```
program roots1; var a, b, c, x1, x2: real; begin Read (a, b, c)
x1 := (-b + Sqr(Sqr(b) - 4*a*c)) / (2*a); x2 := (-b - Sqr(Sqr(b) -
4*a*c)) / (2*a); Write (x1, x2); end.
```

И эта программа будет работать! Однако если данную программу потребуется когда-нибудь изменить или усовершенствовать, разобраться в таком исходном тексте будет непросто. Чтобы повысить их читабельность, при написании текстов программ следует придерживаться определенного стиля.

Кроме того, отладка (т.е. удаление ошибок) программы проводится построчно. Если в программе, записанной в одну строку, компилятор обнаружит ошибку, будет зафиксирован сам факт, но не выявлен оператор, содержащий ошибку. А если в строках программы содержатся по два-три оператора, при обнаружении ошибки программисту придется предпринимать дополнительные действия, чтобы выявить ее местонахождение.



Иными словами, исходный текст программы целесообразно максимально вытягивать по вертикали — стремиться в каждой строке иметь не более одного оператора.

Теперь попробуем выполнить нашу программу  $\text{Roots1}$ . Для этого в меню  $\text{Run}$  интегрированной среды разработчика выберем одноименный пункт. В результате программа запускается на выполнение.

Собственно, при этом программа выполняется, только если в ней нет ошибок. Если же без ошибок не обошлось, курсор позиционируется в месте предполагаемой ошибки (собственно, на следующей позиции), а в первой строке в окне редактора отображается информация об обнаруженной ошибке (рис. 1.5).

#### Сообщение об ошибке

Мы здесь просто забыли ввести точку с запятой после оператора  $\text{Read}$ . Более подробную информацию о той или иной ошибке вы найдете в системе справки (пункт меню  $\text{Help}$  интегрированной среды разработчика).

Курсор располагается в позиции, следующей за ошибкой

Сообщение об ошибке

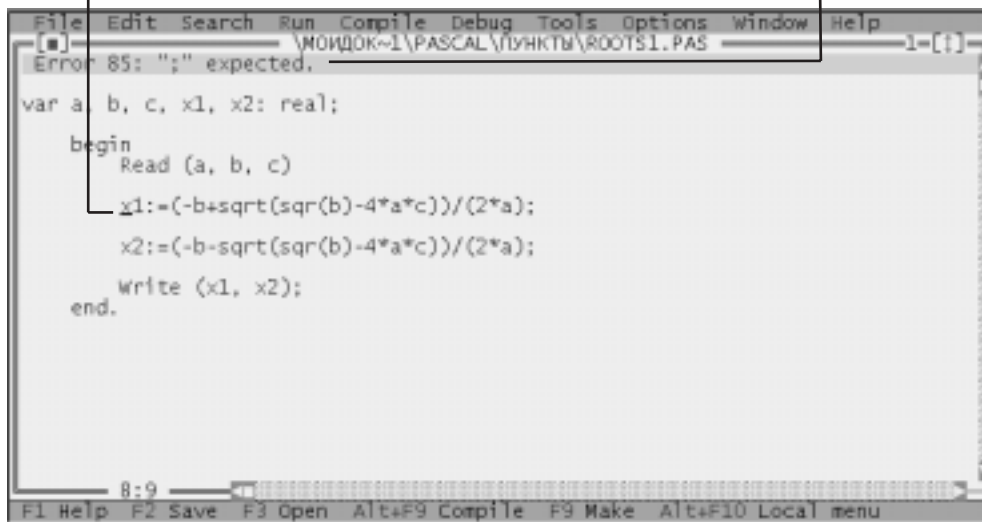


Рис. 1.5. Такое может случиться с каждым

Текст нашей программы, свободной от ошибок, а также результаты ее работы можно видеть на рис. 1.6. Здесь в окне Output в первой строке представлены введенные исходные данные, а во второй — вычисленные программой корни уравнения.

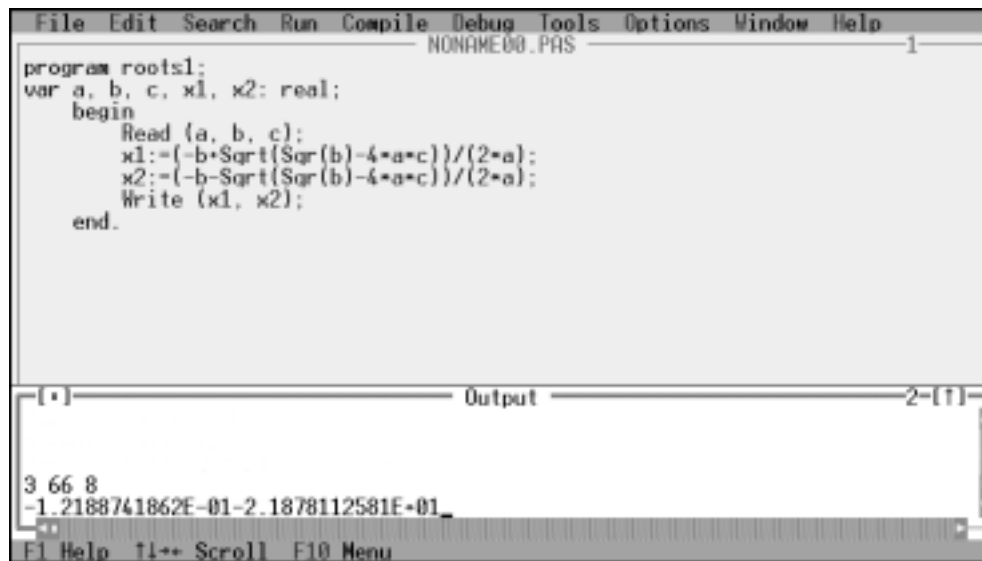


Рис. 1.6. Все правильно, хотя и не слишком вразумительно

Конечно, это никуда не годится! Во-первых, не помешало бы сделать так, чтобы программа отображала текстовое сообщение, приглашающее ввести значение каждой из переменных. Для этого вместо единого оператора Read, позволяющего задать по очереди все три переменные, добавим в программу три оператора Read, — для ввода значения каждой

переменной в отдельности, — предварив каждый из них оператором Write, обеспечивающим вывод на экран соответствующего сообщения. В результате фрагмент текста программы, позволяющий ввести значение одной из переменных, примет следующий вид.

```
Write ('Введите значение A');  
Read (a);
```

Во-вторых, значения корней уравнения здесь представлены в одну строку и не разделены даже пробелом. Было бы лучше вывести их в отдельных строках и как-нибудь обозначить. Для этого вместо одного оператора Write, обеспечивающего вывод на экран значений обоих корней уравнения, введем в программу два оператора WriteLn, чтобы значения корней отображались в разных строках.



**Процедура WriteLn** осуществляет вывод заданного значения и переводит курсор<sup>2</sup> в начало следующей строки. (Процедура Write осуществляет вывод заданного значения и переводит курсор на следующую позицию в той же строке.) Подробнее об этом в главе 5.

Кроме того, сделаем так, чтобы значения корней, выводимые на экран, предварялись соответствующими обозначениями. В результате операторы WriteLn будут выглядеть следующим образом.

```
WriteLn ('x1=', x1);  
WriteLn ('x2=', x2);
```

(Читателю уже известно, что если аргумент функции WriteLn заключен в апострофы, символы между апострофами будут выведены на экран в том виде, в каком они представлены в тексте программы.)

Наконец, неплохо бы весь вывод на экран как-нибудь озаглавить. Для этого можно ввести в начало программы следующий оператор.

```
WriteLn ('Вычисление корней квадратного уравнения');
```

Как теперь выглядит программа (пусть теперь она называется Roots2) в окне интегрированной среды Turbo Pascal, а также результаты ее работы в окне Output, демонстрирует пример 1.3 (рис. 1.7).

Ну вот, здесь уже что-то можно понять. Видны значения коэффициентов  $a$ ,  $b$  и  $c$ , а также вычисленные значения корней уравнения. Пусть вас не смущают, возможно, непонятные величины  $x1$  и  $x2$ . Здесь мы имеем дело с так называемым представлением чисел с плавающей точкой (известным также как экспоненциальный формат чисел). Число, расположенное перед символом  $E$ , в этом представлении заменяет основание степени 10, а непосредственно за ним следует порядок. В нашем случае  $1.2188741862E-01$  соответствует  $1.2188741862 \cdot 10^{-1}$  или  $0.12188741862$ , а  $2.1878112581E+01$  —  $2.1878112581 \cdot 10^1$  или  $21.878112581$ . Для чего это нужно? Для краткого представления чисел с большим количеством разрядов (например, двадцатью и больше). Дело в том, что компьютеры имеют ограниченную разрядность, и чтобы на них можно было обрабатывать огромные (или бесконечно малые) числа, приходится эти числа представлять в форме, подходящей для компьютера.



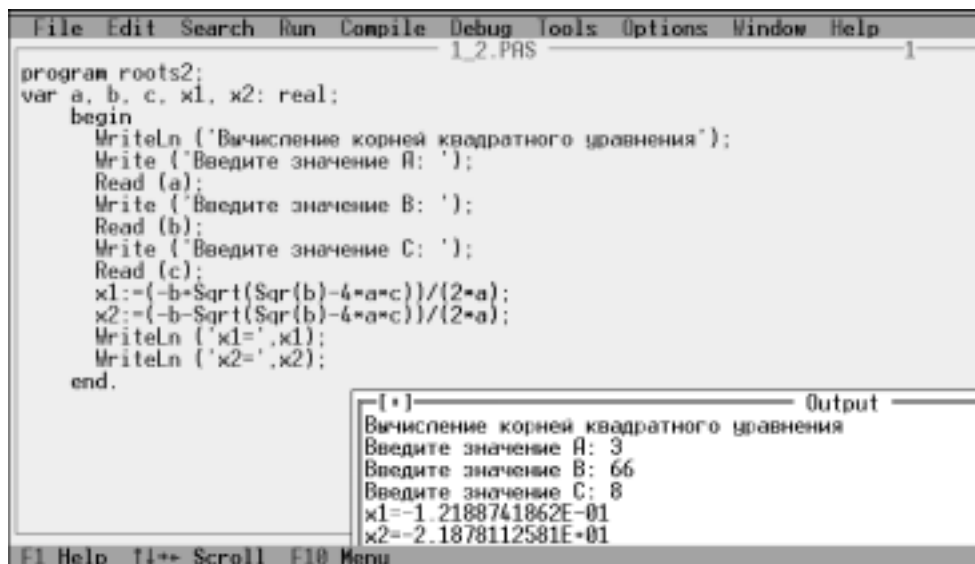
Почему окно Output на рис. 1.7 расположено не так, как на предыдущих рисунках? Дело в том, что данное окно можно разместить в любом месте экрана, перетаскивая его за заголовок. Кроме того, если дважды щелкнуть на том же заголовке, оно займет весь экран интегрированной среды.

---

<sup>2</sup> Курсор — это горизонтальная мигающая черточка на экране, указывающая, где будет отображен следующий символ, который вы введете с клавиатуры.



### Пример 1.3



```
File Edit Search Run Compile Debug Tools Options Window Help
1_2.PAS 1
program roots2;
var a, b, c, x1, x2: real;
begin
  Writeln ('Вычисление корней квадратного уравнения');
  Write ('Введите значение A: ');
  Read (a);
  Write ('Введите значение B: ');
  Read (b);
  Write ('Введите значение C: ');
  Read (c);
  x1:=(-b+Sqrt(Sqr(b)-4*a*c))/(2*a);
  x2:=(-b-Sqrt(Sqr(b)-4*a*c))/(2*a);
  Writeln ('x1=',x1);
  Writeln ('x2=',x2);
end.
```

```
[*] Output
Вычисление корней квадратного уравнения
Введите значение A: 3
Введите значение B: 66
Введите значение C: 8
x1=-1.2188741862E-01
x2=-2.1878112581E+01
```

Рис. 1.7. Так гораздо лучше

Как еще можно усовершенствовать программу вычисления корней квадратного уравнения? Например, вовсе не обязательно вводить в программу переменные  $x1$  и  $x2$  и использовать отдельные операторы для вычисления и вывода результатов. Эти действия можно совместить. В результате корни квадратного уравнения будут вычисляться и выводиться на экран следующим образом.

```
Writeln ('x1=', (-b+Sqrt (Sqr (b)-4*a*c)) / (2*a));
Writeln ('x2=', (-b-Sqrt (Sqr (b)-4*a*c)) / (2*a));
```

Нетрудно заметить, что выражения  $\text{Sqrt}(\text{Sqr}(b)-4*a*c)$  и  $2*a$  вычисляются здесь по два раза, и это увеличивает время работы программы. Конечно, в случае такой простой программы, как наша, это не ощутимо, однако в обширных программах, если допустить, чтобы одно и то же выражение вычислялось повторно несколько десятков (или сотен) раз, мелкие задержки суммируются в нечто существенное. Кроме того, хороший стиль программирования диктует исключать случаи повторного вычисления выражений. В силу этого введем в нашу программу две новые переменные —  $d$  и  $e$ , присвоим им вычисленные значения указанных выражений, а затем используем эти переменные для дальнейших вычислений. Программа с указанными изменениями (которая теперь называется *Roots3*) содержится в примере 1.4.

### Пример 1.4

```
program roots3;
var a, b, c, d, e: real;
begin
  Writeln ('Вычисление квадратного уравнения');
  Write ('Введите значение A: ');
  Read (a);
  Write ('Введите значение B: ');
  Read (b);
  Write ('Введите значение C: ');
  Read (c);
```

```

d:=Sqrt (Sqr (b)-4*a*c);
e:=2*a;
WriteLn ('x1=', (-b+d)/e);
WriteLn ('x2=', (-b-d)/e);
end.

```

Теперь испытаем нашу программу. При этом необходимо помнить, что изменения в последнем варианте нашей программы Roots3, по сравнению с Roots2, связаны исключительно с внутренним порядком вычислений и никак не влияют на вид экрана с результатами работы программы.

Как будто все очевидно, и читателю, вероятно, уже не терпится написать собственную программу. К сожалению, не все так просто. Для того чтобы программы на Turbo Pascal писать правильно, необходимо прежде познакомиться с некоторыми элементарными вещами. Этому и посвящаются оставшиеся разделы данной главы.



Целью предыдущих разделов, в которых мы создали две простейшие программы на Turbo Pascal, является демонстрация возможностей этого языка программирования (чтобы вызвать интерес читателей), а не изучение его особенностей. Поэтому, если какие-то нюансы остались для вас неясны, сюда вполне можно вернуться после изучения прочих глав данной книги.

## О типах данных

Этому специально посвящена глава 4 этой книги, однако до того нам часто придется иметь дело с различными типами данных при изучении тех или иных тем. Поэтому полезно усвоить о типах данных некоторые начальные сведения, отложив более подробный разговор до указанной главы.

Тип позволяет точно определить, как следует интерпретировать те или иные данные. В результате исключаются попытки производить над этими данными неприемлемые операции. Например, если в программе фигурирует переменная, имеющая смысл “количество штук”, понятно, что ее значение не должно представлять собой дробное число. Чтобы этого избежать, такой переменной при объявлении должен быть присвоен один из целочисленных типов (например, Integer). Также недопустимы арифметические операции над символами; чтобы этого не случилось, соответствующие переменные должны принадлежать символному типу (Char). А если в программе имеется переменная, способная принимать только значения Да или Нет (либо Правда или Ложь), чтобы обеспечить правильную трактовку ее значения, данная переменная должна принадлежать логическому типу (тип Boolean). Иными словами, принятая в Turbo Pascal типизация переменных позволяет исключить ошибочную интерпретацию данных и повышает надежность программ.

### Тип Real



Мы уже встречались с типом данных Real в программах Roots1–Roots3 и знаем, что это вещественный тип данных. Переменные типа Real могут принимать дробные значения и изменяться в пределах от 2.9E–39 до 1.7E38 (или от  $2.9 \cdot 10^{-39}$  до  $1.7 \cdot 10^{38}$ ).

Объявить переменные типа Real в разделе описания переменных можно следующим образом.

```

Var
a, b, c: real;

```



К вещественным значениям применимы четыре арифметических действия; полученный при этом результат — также вещественное число. К вещественным значениям применимы также операции сравнения =, <>, >, >=, <, <=, дающие логический результат.

С особенностями использования значений типа Real мы уже познакомились при создании программы вычисления корней квадратного уравнения.

## Тип Integer



Это целочисленный тип данных. Переменные типа Integer могут принимать значения, представляющие собой целые числа в диапазоне от  $-32768$  до  $32767$ .

Объявить переменные типа Integer в разделе описания переменных можно следующим образом.

```
Var  
a, b, c: integer;
```

После объявления переменных в разделе описаний программы, эти переменные могут участвовать в выражениях в теле программы.

```
a:=b+c;  
b:=c-2;  
c:=a*b;
```



К целочисленным значениям применимы четыре арифметических действия. Причем если операции сложения (+), вычитания (−) и умножения (\*) с двумя целыми значениями дадут целочисленный результат, то операция деления (/), примененная к двум целым значениям, даст вещественный результат. Например, значение, полученное в результате вычисления  $a/b$  (где  $a$  и  $b$  — целые числа), можно присвоить переменной только вещественного типа.



Кроме обычных арифметических операций, к целочисленным значениям в Turbo Pascal применимы две специальные операции деления, обозначаемые зарезервированными словами DIV и MOD.

Так, результатом действия  $a \text{ div } b$  будет целая часть частного от деления  $a$  на  $b$ .

```
25 div 2=12;  
5 div 7=0;
```

Результатом выполнения действия  $a \text{ mod } b$  будет остаток от деления  $a$  на  $b$  (не путать остаток с дробной частью).

```
25 mod 2=1;  
5 mod 7=5;
```



К целочисленным значениям применимы также операции сравнения =, <>, >, >=, <, <=, дающие логический результат (TRUE или FALSE).

Вот пример использования операции сравнения.

```
if a>b then a:=a+c  
      else a:=a-c;
```

Здесь в зависимости от того, соблюдается или не соблюдается условие  $a>b$ , выполняется либо действие  $a:=a+c$ , либо  $a:=a-c$ .

## Тип Char



Char представляет собой символьный тип данных. Область допустимых значений — все символы таблицы ASCII, принятой для персональных компьютеров (см. Приложение Д).

Объявить переменные типа Char в разделе описания переменных можно следующим образом.

```
Var  
  a,b,c:char;
```

После объявления переменных в разделе описаний программы, эти переменные могут участвовать в выражениях в теле программы.

```
a:='a';  
b:=Chr(97);  
if c>a then ...
```

В первой строке переменной *a* присваивается значение типа Char, соответствующее букве “a”. (Когда значения типа Char задаются в программе явно, они выделяются апострофами.)

Во второй строке переменной *b* присваивается значение типа Char, соответствующее коду 97 из таблицы ASCII (иными словами, той же букве “a”). Для преобразования здесь используется стандартная функция Chr.



Функция Chr преобразует код из таблицы ASCII в соответствующий символ, т.е. в значение типа Char.

Условный оператор в третьей строке, если значение переменной *c* больше значения переменной *a*, инициирует выполнение какого-то действия. Одно значение типа Char считается больше другого, если код первого значения больше кода второго. Например, выражение 'a'<'b' соответствует истине, точно так же как и 97<98.



Над значениями типа Char возможны все операции сравнения (=, <>, >, >=, <, <=). Причем они дают тот же результат, что и при применении к кодам соответствующих символов.

## Тип Boolean



Это логический тип данных. Переменные типа Boolean принимают значения TRUE и FALSE (правда и ложь), которые также могут быть представлены в виде двоичных цифр 1 и 0 соответственно.

Вот как можно объявить переменные типа Boolean в разделе описания переменных.

```
Var  
  a,b,c:boolean;
```

После объявления переменных в разделе описаний эти переменные могут участвовать в выражениях в теле программы.

```
a:=true;  
b:=a;  
c:=false;
```



К значениям типа Boolean применимы шесть операторов сравнения и четыре логические операции.

С операторами сравнения =, <>, >, >=, <, <= мы уже знакомы. Особенность их применения к логическим значениям состоит только в том, что значение TRUE (поскольку его также можно представить в виде двоичной цифры 1) считается больше значения FALSE (которое соответствует двоичной цифре 0).

Логические операции AND (И — логическое умножение), OR (ИЛИ — логическое сложение), XOR (исключающее ИЛИ), NOT (НЕ — логическое отрицание) подробнее рассматриваются в главе 4. Это же касается и принятых в Turbo Pascal типов данных.

## Алфавит и зарезервированные слова

Основными элементами “человеческого” языка (если он обладает письменностью) являются буква, слово, словосочетание, предложение. Что касается языка программирования, аналогичные элементы имеются и в нем, только здесь словосочетание называют *выражением*, а предложение — *оператором*. Слова в языке программирования образуются из определенного набора символов (букв, цифр и специальных знаков). Выражение здесь — группа слов, а оператор — некоторое сочетание выражений и слов. Иначе говоря, символы, слова, выражения и операторы составляют некоторую иерархическую структуру. Основе этой структуры — перечню символов, используемых в языке программирования Turbo Pascal, и посвящен следующий раздел.

### Алфавит языка Turbo Pascal

Для записи программ на языке Turbo Pascal, как самых простых, которые мы создали в предыдущем разделе, так и самых сложных, используется набор знаков, включающий буквы, цифры и специальные символы. Речь идет о следующих знаках.

1. Прописные и строчные буквы латинского алфавита.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

Здесь относятся и символ подчеркивания ( `_` ). (В языке Turbo Pascal символ подчеркивания считается *буквой*.)

2. Десятичные цифры.

0 1 2 3 4 5 6 7 8 9

3. Специальные символы.

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

К числу специальных символов относятся также пробел.

Из специальных символов образуются составные символы.

<code>:=</code>	присвоить
<code>&lt;&gt;</code>	не равно
<code>..</code>	диапазон значений
<code>(* *)</code>	можно использовать вместо фигурных скобок ( { } )
<code>&lt;=</code>	меньше или равно
<code>&gt;=</code>	больше или равно
<code>(. .)</code>	можно использовать вместо квадратных скобок ( [ ] )



Сказанное выше касается только зарезервированных слов и идентификаторов (т.е. имен переменных, меток, констант, процедур и функций) и не относится к сообщениям, предназначенным для вывода на экран. Примером могут служить программы `Roots1–Roots3` (см. выше), где в сообщениях широко используются буквы русского алфавита. Это же касается и текста комментариев (см. раздел “Комментарии” дальше в этой главе).

### Зарезервированные слова



Речь идет об ограниченном наборе слов, состоящих из букв. Смысл каждого из зарезервированных слов в Turbo Pascal строго фиксирован.



Зарезервированные слова нельзя использовать для обозначений переменных, констант и т.п.

Слова, о которых идет речь, представлены ниже в алфавитном порядке.

AND	GOTO	PROGRAM
ASM	IF	RECORD
ARRAY	IMPLEMENTATION	REPEAT
BEGIN	IN	SET
CASE	INHERITED	SHL
CONST	INLINE	SHR
CONSTRUCTOR	INTERFACE	STRING
DESTRUCTOR	LABEL	THEN
DIV	LIBRARY	TO
DO	MOD	TYPE
DOWNTO	NIL	UNIT
ELSE	NOT	UNTIL
END	OBJECT	USES
EXPORTS	OF	VAR
FILE	OR	WHILE
FOR	PACKED	WITH
FUNCTION	PROCEDURE	XOR

Если присмотреться, нетрудно заметить, что некоторые из зарезервированных слов мы уже использовали в программах Roots1, Roots2 и Roots3. Это следующие слова.

PROGRAM	Первое слово заголовка программы
VAR	Слово, открывающее раздел описания переменных
BEGIN	Слово, обозначающее начало раздела операторов (тела программы) <sup>1</sup>
END	Слово, завершающее тело программы

Смысл и назначение прочих зарезервированных слов мы будем выяснять по мере использования их в программах.

## Комментарии



Комментарии — это текстовые строки, вставляемые в текст программы для пояснения функций отдельных ее частей.

Подобные комментарии можно вставить и в текст созданной нами выше программы вычисления корней квадратного уравнения. Исходный текст указанной программы (которая теперь называется Roots4) с комментариями содержится в примере 1.5.

### Пример 1.5

```
program roots4;  
var a, b, c, d, e: real;  
begin  
  WriteLn ('Вычисление корней квадратного уравнения');
```

---

<sup>3</sup> В более широком смысле данное слово обозначает начало составного оператора. Раздел операторов программы представляет собой частный случай составного оператора. О составных операторах речь пойдет дальше.

```

Write ('Введите значение A: ');
Read (a);
Write ('Введите значение B: ');
Read (b);
Write ('Введите значение C: ');
Read (c);
d:=Sqrt(Sqr(b)-4*a*c);
e:=2*a;
{Вычисление и вывод первого корня}
WriteLn ('x1=', (-b+d)/e);
{Вычисление и вывод второго корня}
WriteLn ('x2=', (-b-d)/e);

```

end.



Ни в коем случае не следует путать комментарии с сообщениями, выводимыми на экран. Первые предназначены для пояснений в исходном тексте программы (на случай, если кому-нибудь когда-нибудь потребуется в них разобраться), а вторые — для информирования пользователя программы.

Разумеется, назначение операторов данной программы очевидно и без комментариев, однако если исходный текст программы состоит из сотен строк, без комментариев не обойтись. Следует подчеркнуть, что подобные комментарии предназначены исключительно для людей (чтобы им было легче разобраться в исходном тексте программы) и полностью игнорируются компилятором.

В заключение следует сказать, что, помимо фигурных скобок, для выделения комментариев в тексте программы можно использовать пары символов “круглая скобка — звездочка”. Иными словами, два комментария, которые мы только что вставили в текст программы Roots4, могут также выглядеть следующим образом.

```

(*Вычисление первого корня*)
(*Вычисление второго корня*)

```



Кстати, фигурными скобками можно воспользоваться, чтобы сделать для компилятора невидимым какой-либо фрагмент программы. Если некоторый оператор или группу операторов требуется “изъять из обращения”, не удаляя их полностью (а вдруг еще понадобятся), указанные операторы можно просто заключить в фигурные скобки, превратив их как бы в комментарий. Мы уже знаем, что комментарии полностью игнорируются компилятором.

## Переменные



В каждой программе для вычисления нужных результатов широко используются *переменные*. Что такое переменная? Это, по сути, некоторая область в памяти компьютера, для которой в данной программе предусмотрено уникальное имя и содержимое которой в ходе работы программы может изменяться (именно поэтому указанное понятие получило такое название). Когда переменной присваивается новое значение, ее старое значение теряется. Все используемые в программе на Turbo Pascal переменные должны быть объявлены в разделе описания переменных. (Подробнее о структуре программы и о месте в этой структуре раздела описания переменных речь пойдет в Приложении В.) При этом не только устанавливается сам факт существования переменной, но и задается ее тип, определяющий, какие значения может принимать данная переменная.

В последней версии программы вычисления корней квадратного уравнения (Roots4) раздел описания переменных выглядел следующим образом.

```
var a, b, c, d, e: real;
```

Здесь объявляются пять переменных типа Real. Что собой представляет этот тип и какие еще существуют типы, мы уже выяснили в разделе “О типах данных” в этой главе выше. Вот еще пример раздела описания переменных.

```
var
SqMeters,      {исходная величина – размер ткани в кв. метрах}
SqYards :real; {вычисляемая величина – размер ткани в кв. ярдах}
```

Здесь задаются две переменные, обе типа Real. Мы уже знаем, что текст в фигурных скобках — это комментарии, которые игнорируются компилятором и предназначены исключительно для легкости понимания исходного текста программы. Судя по идентификаторам переменных, это фрагмент программы, которая переводит квадратные метры в квадратные ярды, что подтверждают и комментарии.

Вероятно, читатель уже понял, что раздел описания переменных может содержать несколько строк, каждая из которых разделена двоеточием (:). В левой части такой строки содержится идентификатор переменной (или перечень идентификаторов, разделенных запятыми), а в правой части указан тип данных, которому принадлежит данная переменная (переменные). Каждая строка в разделе описания переменных завершается символом “точка с запятой” (;). Резервированное слово VAR присутствует в тексте программы в единственном экземпляре; число строк в разделе описания переменных, следующих за словом VAR, не ограничивается.

## Константы

Предположим, значение переменной *c* в программе Roots4 имеет некоторое постоянное значение, например 5.5. Разумеется, вместо переменной *c* в программе можно использовать указанное фиксированное значение (число). Но что если наше квадратное уравнение когда-нибудь потребует решить с другим значением коэффициента *c*? В этом случае данное значение пришлось бы изменять по всему тексту программы (в программе Roots4 всего дважды, но в других программах, возможно, десятки и сотни раз).

Можно оставить все как было и фиксированное значение для *c* (5.5) вводить каждый раз с клавиатуры вместе со значениями двух других переменных *a* и *b*. Можно также в раздел операторов (в тело программы) добавить оператор присваивания *c:=5.5*. Однако *c* в данном случае, по существу, переменной не является, и ее значение должно быть постоянным при каждом выполнении программы, независимо от входных данных.

Предлагая наилучшее решение, Turbo Pascal позволяет вводить в программы объекты, внешне похожие на переменные, но значения которых, в отличие от переменных, не изменяются в ходе работы программы. Подобные объекты называются *константами*. Наша программа (которую теперь назовем Roots5) с коэффициентом *c* квадратного уравнения, заданным в качестве константы, содержится в примере 1.6.

### Пример 1.6

```
program roots5;
const c=5.5;
var a, b, d, e: real;
begin
  WriteLn ('Вычисление корней квадратного уравнения');
  Write ('Введите значение A: ');
  Read (a);
  Write ('Введите значение B: ');
  Read (b);
  d:=Sqrt(Sqr(b)-4*a*c);
  e:=2*a;
```



```

    {Вычисление и вывод первого корня}
    WriteLn ('x1=', (-b+d)/e);
    {Вычисление и вывод второго корня}
    WriteLn ('x2=', (-b-d)/e);
end.

```

В программе появился новый структурный элемент — раздел описания констант, в котором задается значение константы *c*. Этот раздел начинается со слова **CONST**, входящего в число зарезервированных слов, имеющих специальное значение (перечень зарезервированных в Turbo Pascal слов см. выше). Подробнее о структуре программы и о месте в этой структуре раздела описания констант речь пойдет в Приложении В.



Преимущество задания какого-либо постоянного значения в виде константы, а не переменной, в том, что при этом блокируется возможность его случайного изменения (например, в результате ошибки программиста). В Turbo Pascal исключаются операторы присваивания, в левой части которых содержится идентификатор константы.

Например, если *c* — константа, то появление в теле программы строки наподобие `c := xxxx` (где *xxxx* — некоторое значение) побудит компилятор выдать соответствующее сообщение об ошибке.

Приведем пример раздела описания констант.

```

const
    number = 93;
    max = 10;
    min = -max;
    message = 'Ошибка';
    quantity = 2.33E11;

```

Первым двум константам в этом разделе присваиваются значения, представляющие собой целые числа, третьей константе (*min*) присваивается значение второй константы со знаком “минус”, четвертой — значение, представляющее собой текстовое сообщение. Наконец, пятая константа здесь будет содержать вещественное число, представленное с плавающей точкой.



Бросается в глаза, что при объявлении констант не указывается тип данных. Как же в этом случае трактуется значение той или иной переменной? Дело в том, что каждой константе с самого начала (в разделе объявления констант в начале программы) присваивается фиксированное значение, которое в программе (при использовании) изменяться не может. Так вот, тип каждой константы однозначно определяется по ее значению.

## Тесты

Эти тесты помогут вам закрепить материал данной главы. Ответы ищите в Приложении А.

### Истина или ложь?

Каждое утверждение либо верно, либо нет.

1. Раздел описания переменных начинается с зарезервированного слова **VAR**.
2. Знак присваивания (`:=`) делит на две части одноименный оператор.
3. В программе на Turbo Pascal знак умножения (звездочку — `*`) можно опустить или заменить точкой.

4. В качестве знака деления в Turbo Pascal используется наклонная черта (/).
5. Тип позволяет точно определить, как следует интерпретировать те или иные данные.
6. Для обозначения переменных, констант и т.п. можно использовать зарезервированные слова.
7. Когда переменной присваивается новое значение, ее старое значение теряется.

## Найти верный ответ

Каждый из предложенных вопросов может иметь несколько правильных ответов.

8. Какая строка из раздела описания переменных соответствует правилам Turbo Pascal?
  - а) `variable:real;`
  - б) `variable=real;`
  - в) `variable:=real;`
9. Какая строка из раздела описания констант корректна?
  - а) `a=10;`
  - б) `a:=10;`
  - в) `a:10;`
10. Какой из следующих операторов присваивания представлен без ошибок?
  - а) `a=b+c;`
  - б) `a:=b+c;`
  - в) `a:=b+c`
11. Какие из представленных ниже комментариев соответствуют правилам Turbo Pascal?
  - а) `{Это комментарий}`
  - б) `(*Это также комментарий*)`
  - в) `*(Еще один комментарий)*`
12. Какая из следующих групп символов принадлежит алфавиту Turbo Pascal?
  - а) А И Л Д Т А о в ч п № ! % &
  - б) F G H I J K u v w x [ \$ ( )
  - в) В Ж И М Л Г в и п о ^ < | »
13. Где здесь группа зарезервированных слов?
  - а) `Write, Read, WriteLn, Sqr, Sqrt`
  - б) `PROGRAM, VAR, CONST, BEGIN, END`
  - в) `Real, Integer, Boolean, Char`

## Найти соответствие

14. Укажите соответствие между вещественными значениями, представленными в экспоненциальном и традиционном формате.
 

а) 2.18E+01	1. 2.18
б) 2.18E+00	2. 0.218
в) 2.18E-01	3. 21.8

15. Укажите, какие из следующих выражений имеют значение TRUE и какие — FALSE.

а) 'a' > 'b';

б) 'a' < 'b';

в) 'a' = 'A';

г) 'c' >= 'c';

д) 'C' <= 'c';

е) 'B' <> 'b';