5. Операторы ввода и вывода.

Мы рассмотрели на простых примерах два способа определения значения переменной: с помощью оператора «Read» и оператора присвоения. В этой главе мы приведем более детальное описание оператора ввода и его использования, а также рассмотрим различные форматы для вывода результатов.

Для ввода и вывода значений величин используются операторы Read (Readln) и Write (Writeln) соответственно. Синтаксически это стандартные процедуры, которые в программе вызываются оператором *вызова процедуры*. Без дополнительных уточнений подразумевается, что ввод осуществляется с клавиатуры, а вывод – на экран дисплея.

5.1. Управление вводом данных.

Рассмотрим оператор

```
Read (number);
```

Он переводит программу в состояние ожидания и считывает значение в переменной *number*, т.е. после того, как пользователь наберет на клавиатуре ее предполагаемое значение и нажмет на клавишу «Enter», переменной присвоится соответствующее набранное значение.

Общий вид оператора ввода:

```
Read(Список-вводимых-переменных);
```

где «Список-вводимых-переменных» — это список идентификаторов, значения которых пользователь определяет по ходу выполнения программы с помощью ввода с клавиатуры. Например,

```
Read(n1,n2);
```

берет два последующих числа, набранных пользователем, и присваивает эти значения переменным n1 и n2, причем первое из введенных значений присваивается переменной n1, а второе — переменной n2.

Программа должна считать значения всех переменных из вводимого списка прежде, чем начнет их дальнейшую обработку или перейдет на выполнение последующих инструкций. Порядок ввода чисел должен строго соответствовать порядку переменных в списке параметров оператора ввода, а числа должны быть точно того же типа, что и соответствующая ожидаемая переменная. Вводимые пользователем числа должны быть разграничены хотя бы одним пробелом. Обычно компилятор присваивает значения переменным только после ввода полного списка, т.е. набора соответствующих значений и нажатия клавиши «Enter». Если Вы набрали больше значений, чем это предусматривалось программой, то эти значения будут присваиваться переменным, ввод которых ожидается в теле программы в последующих операторах «Read».

Пусть даны следующее объявление и оператор ввода:

```
count, n: Integer;
value: Real;
...
Read(count, value, n);
```

и пользователь вводит последовательно числа:

```
23 -65.1 3
```

чтобы присвоить переменной *count* значение 23, переменной *value* — значение —65.1 и переменной *n* — значение 3. Так как при вводе данных никак не указывается, какой именно переменной присваивается какое значение, а соответствие контролируется самим программистом, принято перед вводом величин выводить на экран текстовое сообщение, подсказывающее пользователю, о вводе каких данных идет речь, как это было реализовано в программном примере в главе 4.

Данные при вводе должны быть разделены либо пробелами, либо новой линией — они будут игнорироваться компилятором. Таким образом, предыдущий ввод мог быть организован и так:

23
-65.1 3

Что произойдет, если при вводе численных данных случайно набрать нелегальный символ. Естественно, компилятор не сможет распознать вводимую информацию, будет иметь место логическая ошибка и программа прервет работу. Но можно установить такой режим для компилятора, при котором программного прерывания выполнения не произойдёт. Это можно осуществить с помощью т.н. *директив компилятора.* Это – специальные команды в программном коде, которые заключаются в фигурные скобки, при этом за открывающей скобкой должен следовать символ соответствующей директивы. Режим контролирования вводимой информации включается с помощью директивы {I+}, а отключается — с помощью {I-}. Это невыполняемые инструкции и поэтому за ней точку с запятой ставить не надо.

Если контроль за вводом данных отключен, а вводимая величина не соответствует ожидаемому типу, системная функция IoResult принимает отличное от нуля значение, и с помощью ее тестировки можно более гибко реализовать программы. Например,

```
...
Write(' Введите целое число:');
{I-} Read(n) {I+}
If IoResult <> 0 Then
Begin
Writeln ('Ошибка при вводе данных!');
Halt
End;
```

Такой ввод можно реализовать с помощью структуры повтора (см. ниже) так, чтобы гарантировать правильный ввод входных величин. Формат оператора ввода

Read(Список-переменных);

позволяет вводить данные с клавиатуры. Часто возникает необходимость считывать данные с файла. В этом случае перед списком переменных присутствует т.н. файловая переменная, которая идентифицирует, с какого именно файла производится чтение, а формат оператора будет иметь вид (подробно см. в главе о файлах):

Read (f, Список-вводимых-переменных);

где *f* — файловая переменная. Кроме того, для текстовых файлов, которые представляют набор строк, определен оператор «Read», который гарантирует переход на следующую строку файла после того, как перечисленные данные прочитаны.

5.2. Форматирование вывода результатов.

Операторы вывода используются для обеспечения и оформления интерфейса диалога: компьютер— пользователь. Современные программы должны быть организованы таким образом, чтобы тот, кто пользуется программой, не был вынужден знать ее текст наизусть. Поэтому при запросе о вводе данных удобно иметь на экране соответствующие комментарии, чтобы знать, какие именно данные нужно вводить в тот или иной момент. При выводе результатов также удобно иметь подходящие подсказки о том, что делает программа и какие данные вычисляются и выводятся в качестве результатов.

Вывод результатов вычислений осуществляется с помощью операторов «Write» и «Writeln». Следующий оператор

Write(count);

выводит текущее значение переменной *count* на экран дисплея. Значение будет напечатано на текущей линии сразу после последнего выведенного результата.

Общий формат оператора вывода результатов на экран имеет вид:

```
Write(Список-параметров);
```

где «Список-параметров» — это список переменных, констант или строковых выражений (текст в одинарных апострофах), разделенных друг от друга запятой. Оператор вывода позволяет просмотреть значения параметров, заключенных в скобки, в том порядке, в каком они расположены в списке вывода. Если данные не помещаются в одну линию, они разбиваются в правом конце строки вывода и переносятся на следующую.

Программный переход на новую линию происходит с помощью оператора «Writeln;» или «Writeln(Список-параметров)». Первый оператор выводит пустую линию, а второй — значения параметров из списка на одну линию. Так, например,

```
Writeln('Hello');
```

напечатает слово Hello на текущей линии экрана и переведет курсор на новую линию для вывода новых результатов. Рассмотрим следующий фрагмент:

```
Var length, breadth: Integer;
...
Write(' Введите длину и ширину:');
Read(length, breadth);
Writeln(' Длина равняется:', length);
Writeln(' Ширина равняется:', breadth);
...
```

Если пользователь на запрос введет числа 24 и 35, то на экране появится сообщение:

```
Длина равняется: 24
Ширина равняется: 35
```

Отметим, что значения параметров списка будут выводиться без разделительных пробелов. Поэтому в приведенном примере в конце

комментирующих вывод строковых выражений стоит пробел. Оператор

```
Write(length, breadth);
```

напечатает результаты в следующей форме:

```
2435
```

что явно не дает возможности дать правильную интерпретацию выводимым данным. Если хотим вывести значения нескольких величин на одной линии, надо их разделить несколькими пробелами, заключенными в апострофы, например,

```
Write(length,' ', breadth);
```

Таким образом, при выводе строковых выражений, а также символьных, целых и логических значений, выводимой величине отводится столько позиций на экране, сколько она занимает фактически. Для вывода символа апострофа «'» нужно набрать два апострофа подряд, например

```
Write(' There"re all rules for the "Write" operator! ');
```

Результат:

```
There're all rules for the 'Write'-operator!
```

Вывод переменных и значений действительного типа осуществляется тремя различными способами, которые называются *форматами* вывода.

✓ По-умолчанию действительные величины выводятся в т.н. *экспоненциальной форме* с *плавающей точкой.* Общий вид экспоненциальной формы можно записать так:

± x.xxxxx E±yyyy

что равнозначно записи:

$$\pm\ x.xxxxx\ E\pm yyyy = \pm\ x.xxxxx \cdot 10\pm yyyy.$$

Так, например,

$$-\ 5.678\ E{+}0012 = -\ 5.678 \cdot 10^{12} = -\ 5\ 678\ 000\ 000\ 000,$$

или

$$3.585\ E{-}0005 = 3.585 \cdot 10^{-5} = 0.00003585.$$

± x.xxxxx называется мантиссой и определяет значимые цифры. Экспоненциальная часть указывает, на сколько позиций нужно сместить десятичную точку в мантиссе, чтобы получить значение числа, при этом, если знак в экспоненциальной части положительный, точку нужно сместить вправо, а если отрицательный —влево.

✓Второй вид формата выводит действительные величины также в экспоненциальной форме, только в этом случае можно контролировать количество отведенных для числа позиций на экране. Для этого в операторе «Write» или «Writeln» после идентификатора выводимой переменной ставится двоеточие и полное количество отведенных для нее позиций,например:

```
Var r:Real;
...
Writeln(r:10);
...
```

Отметим, что при выводе надо зарезервировать по крайней мере 10 позиций с тем, чтобы обеспечить место для всех атрибутов экспоненциального формата, включая хотя бы один десятичный знак в мантиссе.

✓Третий вид формата позволяет выводить действительные величины в т.н. в формате с *фиксированной точкой,* т.е. это именно

та форма, которую использует человек для записи действительных чисел. В этом случае задаются две величины — полное количество отведенных позиций и количество позиций после десятичной точки, при этом, полное количество включает позицию для самой точки, а также позицию для знака, если число отрицательно. Например,

```
Writeln(r:10:3);
```

Таким образом, для значения переменной r на экране зарезервируется всего 10 позиций, а для дробной части отведется всего 3 позиции.

При организации вывода в формате с фиксированной точкой нужно вручную учитывать, сколько же позиций может занять каждая величина, а также требуемую точность.

Отметим, что второй формат вывода можно применить к переменным любого типа. В этом случае у нас возникает возможность более свободно контролировать распределение выводимых величин на экране. Например, если надо вывести символ «*» в десятой, двадцатой и тридцатой позициях на экране. Вывод можно реализовать двумя способами, сравните сами их эффективность:

```
(a)  Write('*         *         *');

(b)  Write('*':10, '*':10, '*'10);
```

В варианте (b) после вывода первой звездочки отсчитывается 9 позиций и в десятой, т.е. от начала строки в двадцатой позиции ставится вторая звездочка, и т.д.

На первых порах студенты часто путают комментарии к программному коду с комментариями к выводимым на экран результатам. В первом случае текстовое сообщение заключается в фигурные скобки { } (на экран не выводится!) и служит подсказкой для программиста, который составляет код программы. Во втором случае сообщение стоит на месте параметра оператора вывода, заключается в одинарные апострофы и служит для оформления интерфейса, т.е. диалога пользователь—программа (рис. 5.1)

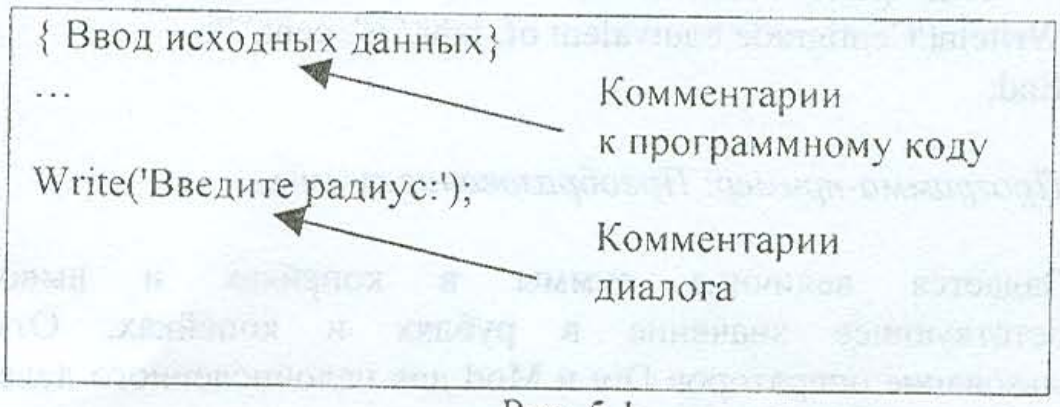

Рис. 5.1

5.3. Примеры использования операторов ввода и вывода.

Рассмотренные здесь примеры настолько просты, что их алгоритмическое описание отдельно не приводится. Нужно только внимательно проследить построчно, какие инструкции и в какой последовательности будут выполняться в процессе решения задачи.

Программа-пример: Преобразование температуры.

Задается величина температуры в Фаренгейтах (Fahrenheit) и выводится соответствующее значение в Цельсиях (Centigrade). Заметим, что константа *mult* определяется как выражение. В общем случае константа может быть определена с помощью выражения, которое, в свою очередь, может содержать как числа, так и идентификаторы предварительно определенных констант.

```
Program Temperature_Convertion;
{ Преобразует Фаренгейты в Цельсии }
{ Фаренгейт задается как входная величина,}
{ Вычислится Цельсий и выводится как результат }
Const mult = 5/9;
    sub = 32;
Var fahr, cent: Real;
Begin
    Write('Enter Fahrenheit temperature:');
```

```
    Read(fahr);
    cent= (fahr - sub) * mult;
Writeln('Centigrade equivalent of, fahr,' is', cent,'.');
End;
```

Программа-пример: Преобразование суммы.

Задается величина суммы в копейках и выводится соответствующее значение в рублях и копейках. Отметим использование операторов Div и Mod для целочисленного деления и определения остатка суммы в копейках.

```
Program Money_Convertion;
{ Преобразует сумму в копейках (входная величина) - }
                { в суммы в рублях и копейках }
Var rubl.kop: Integer;
Begin
    { Ввод суммы денег}
                Write('Enter the amount in kopiks:');
                Read(kop);
                { Начало вывода }
                Write(kop,' kopiks are');
                {Вычисление рублей }
                rubl:=kop Div 100;
                {Вычисление копеек}
                kop:=kop Mod 100;
                Writeln(rubl,' rubls and', kop,' kopiks.');
                End;
```