

## 9. Простейший ввод-вывод на Паскале

### 9.1 Стандартные файлы *Input* и *Output*

Все программы должны обмениваться с внешней средой информацией, т. е. принимать из нее исходные данные и передавать в нее полученные результаты.

Устройство, через которое происходит такой обмен, называется консолью (*console*).

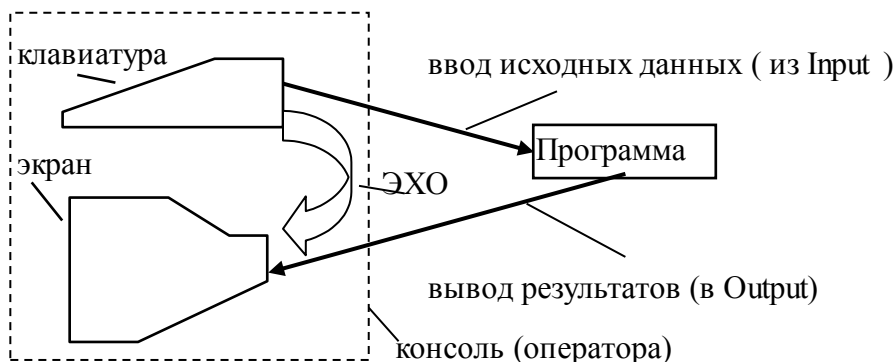
*Консоль* - это комбинированное устройство, в котором для ввода данных используется клавиатура, а для вывода - экран монитора (в текстовом режиме 25 строк по 80 символов).

Поток символов, вводимый с клавиатуры и поток символов, выводимый на экран, принято называть **файлом** (так принято в ОС Unix, а обычно *файл* - именованная область данных, размещенная на носителе информации).

В Паскале за двумя этими потоками символов закреплены имена: *Input* (ввод с клавиатуры) и *Output* (вывод на экран). Это стандартные файлы, которые открываются и закрываются автоматически (соответственно, в начале выполнения программы и по ее завершении). Именно с ними работают процедуры ввода-вывода, если явно в операторах ввода-вывода не указывать имена файлов.

Файлы *Input* и *output* являются **текстовыми файлами**, т.е. они состоят из символьных **строк переменной длины**. Каждая такая строка представляет из себя последовательность символов, в конце которой стоит специальный признак End of line = #13+#10 (из них символ #10 - символ перехода на новую строку или просто символ «новая строка»). В конце текстового файла размещается символ конца файла (#26). С каждым файлом связан **указатель на текущую компоненту файла**, начиная с которой будет производиться текущая операция ввода-вывода. При запуске программы для каждого стандартного файла эти указатели **устанавливаются на самое его начало**, а в процессе работы с файлом после каждой очередной операции ввода-вывода указатель перемещается так, чтобы указывать на компоненту файла, с которой начнется следующая операция.

Особенностью файлов *Input* и *Output* является то, что ввод происходит в режиме **эхо-отображения** вводимых данных. Т.е. на экране происходит отображение того, что вводят с клавиатуры. Эхо-отображение заключается в том, что информация в файле *Output* изменяется не только при выводе в него, но и при вводе информации из файла *Input*. Это хорошо тем, что мы видим на экране то, что вводится с клавиатуры, т.е. ввод идет не вслепую.



### 9.2 Процедуры ввода информации (с клавиатуры).

Есть две процедуры ввода информации на Паскале: *ReadLn* и *Read*.

Для *ReadLn* существует 2 варианта: со списком параметров и без списка

переменная<sub>1</sub>, переменная<sub>2</sub>, переменная<sub>3</sub>, ...

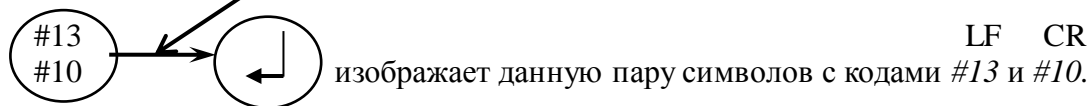
```
ReadLn;  
ReadLn (список ввода);  
Read (список ввода);
```

Список параметров для процедуры ввода будем называть **списком ввода**.

Список ввода (как и левая часть оператора присваивания) может состоять только из имен переменных (имя переменной в программе является синонимом **адреса переменной**). Эти переменные могут быть только одного из **следующих типов**: целый, вещественный, строковый и символьный (массивы, записи и множества м.б. введены лишь **поэлементно**).

Выполнение процедур ввода *Read* и *ReadLn* начинается с того, что программа переходит в **состояние ожидания ввода**. Из этого состояния программа **выйдет** только тогда, когда самая правая в списке ввода переменная получит значение (из буфера клавиатуры). Значения переменные из списка ввода получают из т.н. входного потока.

**На вход процедур *Read* и *ReadLn* данные** (из входного потока) поступают только после нажатия клавиши “Ввод” (Enter) на клавиатуре (если не все переменные из списка ввода успели получить при этом свои значения, то программа останется в состоянии ожидания). После нажатия клавиши “Ввод” (Enter) на клавиатуре к потоку символов, который уже введен до нажатия клавиши “Ввод”, добавляются два символа с кодами #13 и #10. Символ с кодом #13 часто называют “Возвратом каретки” (ВК или CR). При попытке вывода символа на экран курсор на экране установился в начало текущей строки. Символ с кодом #10 («Новая строка» = LF) относится (также как и символ с кодом #13) к множеству пустых символов. При попытке вывода его на экран курсор устанавливается на следующей строке в текущей позиции. После нажатия клавиши "Ввод" процедура ввода **начинает интерпретацию** введенного. То, что введено, представляет собой входной поток для процедуры ввода.



Пример:

```
var
  a,b,c:integer;
begin
  read(a,b,c); //введем 1 2 3
  write (a, #10, b, #10, c, #10);
  readln;
end.
```

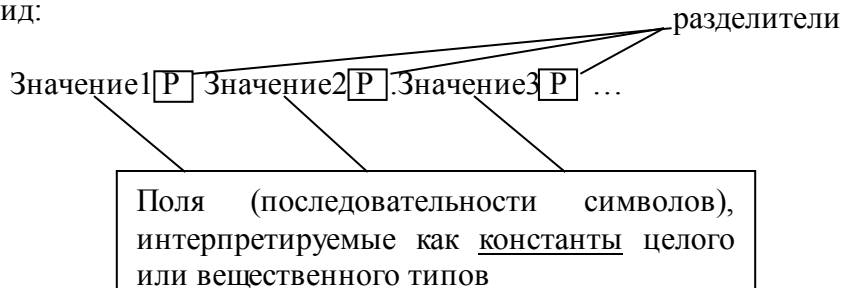
на экране будет так:

1	2	3			
		1			
			2		
				3	
					здесь встанет курсор

**Интерпретация входного потока** зависит от того, какие переменные находятся в списке ввода. Поэтому рассмотрим интерпретацию входного потока для разных сочетаний типов переменных в списке ввода.

**1)-----**

Если в списке ввода находятся переменные вещественного и целого типов, то входной поток рассматривается как последовательность значений (вещественного или целого типов), между которыми стоят разделители. В качестве таких разделителей во входном потоке используются 3 символа: “пробел” (#32), символ табуляции (#9), и признак конца строки (#13+#10). Входной поток имеет вид:



Соответственно, если фрагмент программы имеет вид.

```
var
  x, y, z : integer;
begin
  read(x,y,z);
```

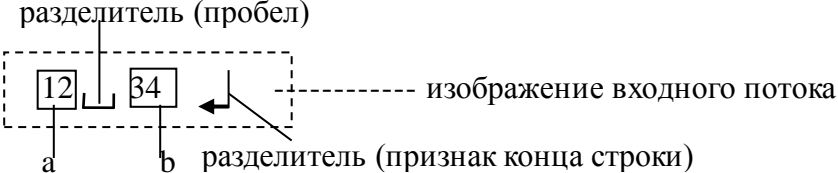
то выполнение процедуры ввода  $\text{read}(x,y,z)$  для указанного выше входного потока будет эквивалентно выполнению трех операторов присваивания следующего вида:

```
x := Значение1;  
y := Значение2;  
z := Значение3;
```

Процесс интерпретации входного потока производится **следующим образом**. Вначале рассматривается самая первая (левая) переменная из списка ввода. Для нее начиная с самого первого символа во входном потоке просматриваются все символы до первого разделителя. Особенностью процедур ввода в Паскале является то, что преобразования типа значения (во входном потоке) к типу переменной (из списка ввода, соответствующей данному значению) при вводе не производится (бесформатный ввод). Поэтому программой делается попытка интерпретировать последовательность символов из входного потока (до ближайшего разделителя) как значения (константы) того типа, который имеет первая (слева) переменная из списка ввода. Такая попытка будет удачной, если символы входного потока (до разделителя) образуют константу, записанную по правилам для констант нужного нам типа (типа данной переменной из списка ввода).

Например, для рассматриваемого случая (в списке ввода - вещественные и/или целые переменные) входной поток будет интерпретирован следующим образом:

```
Var  
  a, b: byte;  
begin  
  read (a, b);  
end.
```



, что эквивалентно следующим операторам присваивания:

```
a := 12;  
b := 34;
```

Если во входном потоке для переменной  $b$  (целого типа) вместо цепочки символов '34' (целая константа) будет цепочка '3.4' (вещественная константа), при выполнении процедуры ввода будет получено сообщение об ошибке ввода. Программа аварийно завершится, поскольку 3.4 не является правильной константой целого типа.

Если попытка интерпретации текущей последовательности символов из входного потока как значения текущей переменной из списка ввода закончилась удачно (успешно), то текущая переменная из списка ввода получает соответствующее значение, происходит перемещение в списке ввода на следующую переменную. Во входном потоке также происходит перемещение текущего указателя входного потока на начало последовательности символов, расположенной справа от последнего найденного разделителя до следующего разделителя. После этого производится попытка интерпретации последовательности символов между двумя разделителями как значения текущей (следующей) переменной из списка ввода. Если такая попытка будет удачной, то текущая переменная получает значение и осуществляется переход к следующей переменной из списка ввода. Так продолжается до тех пор, пока не закончится список ввода.

Примечание. Если входной поток исчерпан раньше, чем количество переменных в списке ввода, то программа перейдет (или останется) в состояние ожидания ввода (без упоминания вас об этом).

Исходя из сказанного, оператор ввода (процедуру ввода) можно рассматривать как форму оператора присваивания. При этом левую часть оператора (операторов) составляют переменные из списка ввода, а в правую часть входят константы из соответствующих полей из входного потока.

## 2)-----

Если в списке ввода находятся переменные символьного типа, то в отличие от случая для переменных целого и вещественного типов, во входном потоке для переменных символьного типа нет разделителей. То есть входной поток для переменных символьного типа рассматривается как поток символов, в котором (из которого) для каждой переменной из списка ввода выбирается один символ.

### ПРИМЕР:

Var

a,

b,

c,

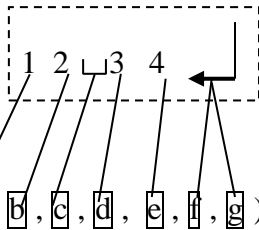
d,

e,

f, g: char;

begin

read ( a, b, c, d, e, f, g );



входной поток (тот же, как в предыдущем случае)

Для заданного входного потока переменные с *a* по *f* получают следующие значения, при этом пробел рассматривается не как разделитель во входном потоке, а как его информационная часть:

a = '1'

b = '2';

c = ' '

d = '3'

e = '4'

f = #13

g = #10

Следует обратить внимание на то, что во входном потоке для переменной символьного типа апострофы не задаются. Каждый апостроф во входном потоке будет рассматриваться как отдельный символ.

### 3)-----

Для переменной строкового типа интерпретация производится следующим образом. Рассмотрим три случая.

а) если Список ввода состоит только из единственной переменной – строкового типа - то входной поток рассматривается как **одна** строка (не включая признак конца строки);

б) если переменная строкового типа в списке ввода следует за переменными других типов, то из входного потока для строковой переменной берется последовательность символов, начиная с позиции, следующей за последним обработанным разделителем до конца строки (но не включая признак конца строки).

#### ПРИМЕР 1.

Var

a:integer;

b:real;

s: string;

begin

read(a,b,s);

или

read (a,b); read(s);

end.

входной поток

12 34 56

что получили в итоге переменные

a = 12;  
b = 34;  
s = '56';

Если для той же программы входной поток изменится на

12 34

то переменные *a*, *b* и *s* получат следующие значения:

a = 12;  
b = 34;  
c = '';

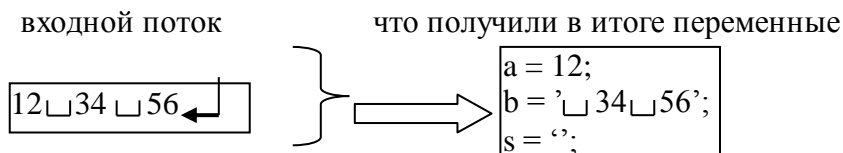
То есть, значения переменных *a* и *b* получатся таким, какие и ожидалось. А переменная *s* получит указанное значение (пустая строка), поскольку при вводе значений строкового типа с использованием процедуры *Read* никогда не происходит во входном потоке переход через признак конца строки (т. е. через символы с кодами #13 и #10). В данном случае после завершения ввода

значений переменной *b* текущий указатель во входном потоке установится не на начало следующего входного потока (даже если мы его ввели), а на символ с кодом #13 в данном входном потоке. Поэтому значение следующей переменной (*c*) будет начинаться с этого символа (с кодом #13) и на нем же и заканчиваться. Но, так как символ с кодом #13 в строку-результат не включается, то в строку *s* попадет 0 значащих символов.

в) если в конце списка ввода не одна строковая переменная, а две или более

### ПРИМЕР 2.

```
Var
a:integer,
b:string;
s:string;
begin
read(a,b,s);
или
read(a,b); read(s);
end.
```



**Вывод:** никогда не следует пользоваться для считывания последовательности строк процедурой *Read*. Надо использовать процедуру ***ReadLn***.

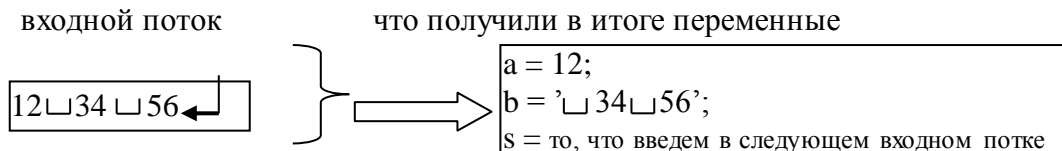
Действие *ReadLn* отличается от действия *Read* следующим:

- при достижении конца входного потока раньше исчерпания списка ввода происходит переход к началу следующего входного потока (если он есть),
- после достижения конца списка ввода оставшаяся часть входного потока пропускается и осуществляется переход через символ конца строки к началу следующего входного потока.

5

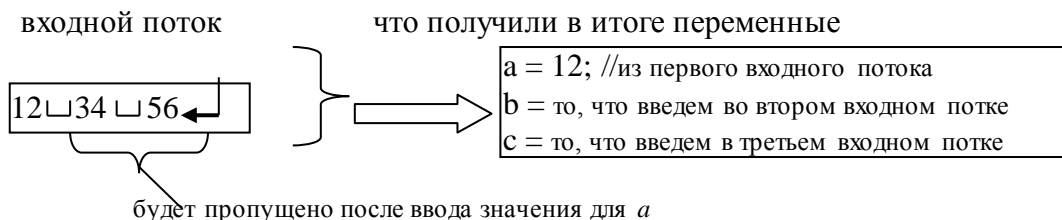
### ПРИМЕР 3.

```
Var
a:integer,
b:string;
s:string;
begin
readln(a,b);
read(s);
end.
```



### ПРИМЕР 4.

```
Var
a:integer,
b:string;
s:string;
begin
readln(a);
readln(b);
readln(s);
end.
```



4)-----  
 Один и тот же входной поток для переменных разных типов интерпретируется по разному:

```
Var
  c1,
  c2,
  c3,
  c4: char;
  i: word;
  s: string;
```

Пусть входной поток имеет вид:

1234 ↵

Тогда переменные получают следующие значения:

```
begin
  read(c1,c2,c3,c4); ----- c1 = '1' c2 = '2' c3 = '3' c4 = '4'
  или ----- i = 1234
  read(i); ----- s = '1234'
  или -----
  read(s); -----
end.
```

5)-----  
 И еще раз об упомянутой выше особенности использования процедуры ReadLn, которую не следует забывать. Эта особенность проявляется при вводе значений символьного, целого и вещественного типов.

Рассмотрим пример:

Пусть входной поток имеет вид:

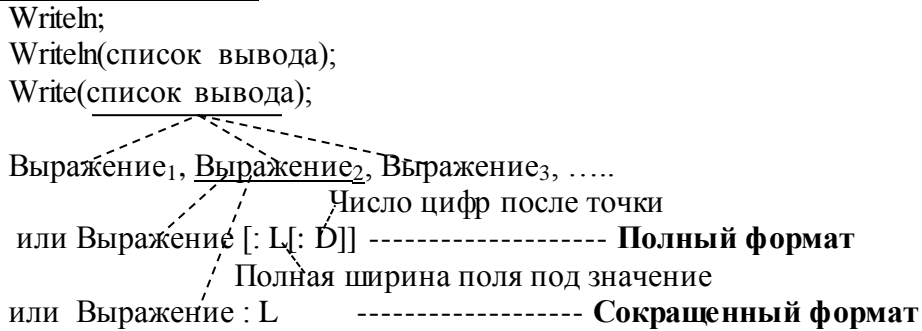
```
var
  a,
  b,
  c,
  d:byte;
begin
  readln(a,b);
  readln(c,d);
```

1 2 3 4 5 6  
 a b c d

это будет пропущено (исчерпан список ввода)

Для той программы переменные *a* и *b* получают значения, которые ожидаются (*a* = 1, *b* = 2). Возникает вопрос, получают ли переменные *c* и *d* значения 3 и 4? **Особенностью** использования процедуры **readln**, при работе с нестроковыми переменными, является то, что после исчерпания в **readln** списка ввода, оставшаяся часть входного потока для нее пропускается. В данном случае, после считывания значения переменной *b* будет достигнут конец списка ввода для этой процедуры, и остаток текущего входного потока будет пропущен. После чего переменные *c* и *d* получают значения, которые будут введены в следующий входной поток.

### 9.3 Процедуры вывода в ТР.



Две процедуры имеются в ТР для вывода данных writeLn и write.

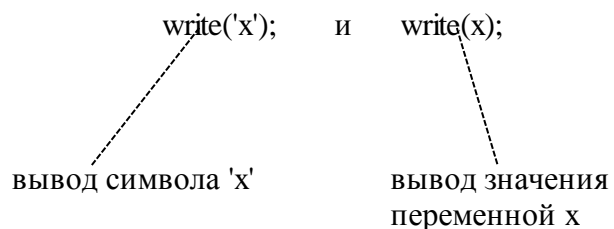
Поговорим вначале о write. В этой процедуре параметры образуют список, называемый списком вывода. Рассмотрим различия в использовании процедур ввода и вывода.

**Первое отличие** процедур вывода от процедур ввода состоит в том, что в списке вывода могут быть использованы не 4 типа значений, а 5: **символьные, строковые, целые, вещественные, и булевские (логические)**. Причем так же, как и при вводе, здесь **не производится преобразование** типа выводимого выражения (**бесформатный вывод**).

**Второе отличие** в том, что в списке вывода размещаются **не только переменные**, а в общем случае - выражения (т.е. допускаются и **константы и вызовы функций**). Т.о. список вывода - это список выражений.

```
Var
a,b,c:byte;
d: char;
begin
read(a, b, c);
read('c'); - нельзя!
read(1,2,3,a+b); - нельзя!
Write(1,2,3,a+b); - верно!
End.
```

При этом **надо различать:**



7

**Третье отличие** состоит в том, что каждый элемент списка вывода может снабжаться т.н. **форматом вывода**. Формат вывода образуется в общем случае из 2-х полей - поле *l* и поле *d*, разделенных двоеточием:

переменная:*l*:*d*

Поле *l* задает минимальную ширину поля (в символах), в котором должно быть размещено значение того выражения, для которого этот формат приписан. Поле *d* задает количество знаков после точки для вещественного значения выражения. Отсюда следует, что **полный формат можно задавать только для вывода вещественного значения**. Для всех прочих вы можете задавать значение с указанием в формате только поля *l*.

**Замечание:** если задается из формата вывода из формата вывода только одно поле, то подразумевается, что задано поле *l*.

Если **ширина поля больше** количества символов, реально необходимых для вывода значения, то значение выводится в поле шириной *l* со **смещением его вправо** так, что слева это значение дополняется необходимым числом пробелов.

```
Var
c: char;
begin
c := 'a';
write ('a':6);
end.
```

На экране получится следующее:

□ □ □ □ □ a

↑  
текущая позиция в строке (до начала вывода)

Если **ширина поля *l* меньше**, чем количество символов, реально необходимых для вывода значений, то **значение *l* игнорируется** и выводится на экране столько символов, сколько необходимо для вывода значения. На экране получится:

```
Var
s: string;
begin
s:='1234'
write (s:1);
end.
```

1234

↑  
текущая позиция в строке (до начала вывода)

**Четвертое отличие** процедуры вывода от процедуры ввода в том, что процедура вывода выводит информацию на экран **без разделителей** (между выводимыми полями данных).

```

a,
b,
c: char;
begin
  a:='1';
  b:='2';
  c:='3';
  write(a,

```

123

текущая позиция в строке (до начала вывода)

end.

При выводе целых чисел знак "-" отображается, а знак "+" не печатается (вместо него даже пробел не отображается). Так, что для данного примера получится следующая картина:

```

Var
a,
b,
c: shortint;
begin
a:=1;
b:=2;      текy
c:=3;
write(a, b, c,);
end.

```

текущая позиция в строке (до начала вывода)

end.

Знак числа                      точка

$\{+/-\}$  Ц Ц Ц Ц Е  $\{+/-\}$  Ц Ц Ц Ц  
 знак пор.  
 дробная часть  
 обязательная цифра  $\triangleleft 0$   
 (нормализация)

знак порядка    порядок

дробная часть

обязательная цифра  $\diamond 0$   
(нормализация)

--- всего 22 позиции на экране

Видно, что при выводе на экран **знак "+" заменяется на пробел, знак "-" печатается.**

Формат представления значения при выводе его в формате с фиксированной точкой:

До нужной ширины поля для отрицат. чисел

[пробелы][ \_ /- ]Ц.[цифры]

для положит. чисел

дробная часть

не менее одной цифры



В **общее количество выводимых символов** надо включать: число символов дробной части, точка, одна цифра в целой части и знак.

Теперь **правила вывода вещественных значений**:

```
if (l задана) and (d задано) and (d >= 0)
then
```

```
begin
```

```
  if (d > 15)
```

```
  then
```

```
    d := 15;
```

```
  if (d = 0)
```

```
  then
```

```
    дробная часть не выводится;
```

```
  if (l < чем нужно для представления значения)
```

```
  then
```

```
    l := столько, сколько минимально нужно для представления значения ;
```

```
  вывод значения в формате с ФТ
```

```
end
```

```
else { либо d не задано, либо d задано, но d < 0 }
```

```
  if (l задана) and (d пропущено) or ((l задана) and (d задано) and (d < 0))
```

```
  then
```

```
    if (l < чем нужно для представления значения)
```

```
    then
```

```
      begin
```

```
        l := столько, сколько минимально нужно для представления значения ;
```

```
        d := 1;
```

```
      end
```

```
    else { l пропущено (и d тоже, естественно) }
```

```
      begin
```

```
        l := 22 ;
```

```
        d := 15;
```

```
      end
```

```
      вывод значения в формате с ПП (в соответствии с указанными выше l и d)
```

```
write(111.111:1:1);
```

выведет на экран

```
└ 111.1
```

```
write(111.111:1);
```

выведет на экран

```
└ 1.1E+002
```

```
write(111.111);
```

выведет на экран

```
└ 1.1111110000000000E+002
```

При выводе значений **булевого типа** на экране выводятся слова: true или false.

Вызов процедуры **writeln** без списка вывода **Writeln;**

производит следующие действия: в файл Output записывается пустая строка, включающая только признак конца строки (#13#10), что при выводе на экран приводит к тому, что курсор на экране перескакивает на начало следующей строки. Исходя из этого, выполнения процедуры **writeln** со списком вывода выполняется как бы в два этапа:

1-й этап: **write**(список вывода);

2-й этап: **writeln;** (без списка вывода).

**Замечание1:** При указании формата вывода отрицательным может быть не только значение *d*, но и значение *l*. Когда указывается *l* < 0, то значение выводится в поле шириной *l* с выравниванием влево, а *l* принимается столько, сколько нужно для представления значения на экране.

**Замечание2:** В процедурах вывода **write** и **writeln** для полей формата можно использовать не только константы, но и переменные. (*l* и *d* могут быть константами или переменными). Это можно использовать при выводе на текстовый экран графиков функций.

**Замечание3:** если в процессе вывода информация как бы переползает за 80-ю позицию (правой границы экрана), то в этом случае не производится скроллинг вправо т.е. выводимая информация начинает выводиться с начала следующей строки.

Когда выводимая информация перелезает за нижнюю границу экрана, т.е. за 25-ю строку экрана, то производится скроллинг вверх, т.е. вся ранее выделенная информация как бы смещается вверх на одну строку, освобождая строку снизу.

Перед вызовом процедур вывода можно использовать вызов процедуры **delay**, такой как **delay(100)**. В данном случае в программе будет сделана задержка на 100 миллисекунд.

Дальше - 10.6 Уточнение многоместных операций