Основы программирования

Учитель информатики
Букина Н.С.

1. Основы
2. Основные типы данных
3. Основные математические функции
4. Оператор присваивания
5. Раздел описания переменных VAR
6. Раздел описания констант const
7. Раздел описания меток Label
8. Оператор вывода данных на экран write(ln)
9. Оператор ввода данных с клавиатуры read(ln)
10. Оператор безусловного перехода goto
11. Оператор условия if…then….else
12. Цикл с параметром for…do
13. Цикл с предусловием while…do
14. Цикл с постусловием repeat….until
15. Случайные числа randomize….random
16. Оператор выбора case…else
17. Операции над текстовыми данными
18. Массивы array
19. Модульное программирование
    1. Функции function
    2. Процедуры procedure
20. Блок описания типов type
21. Работа с файлами
    1. Текстовые файлы
    2. Типизированные файлы
22. Указатели. Общие сведения
    1. Типизированные указатели
23. Перенос программ, написанных на Turbo Pascal в Delphi

ВЫХОД

Основы

Исходная программа должна принимать следующий вид:

program <имя>; заголовок программы

[uses] раздел описания модулей

[var] раздел описания переменных
[const] раздел констант
[label] раздел меток
[type] раздел типов
[function] раздел функций
[procedure] раздел процедур

begin начало программы

[тело программы, операторы]

end. конец программы (*точка обязательна*!)

Те команды, которые заключены в скобки «<» и «>», можно изменять на свое усмотрение. Команды, которые заключены в «[» и «]», можно пропускать, то есть их может и не быть.

В отличие от Basic каждую команду в теле программы нужно разделять точкой с запятой «;», а как именно рассмотрим на частных примерах команд.

Рассмотрим простейший пример:

```
program first;
var a, b, c: word;
const word1 = ‘ответ= ';
 begin
  a:=2;
   b:=3;
  c:=a*b;
   writeln (word1, c);
end.
```

Результат:

ответ= 6

назад

Основные типы данных

Тип	Обозначение	Диапазон	Размер, байт
Короткое целое	Shortint	-128…127	*1*
Целое	Integer	-32768…32767	*2*
Длинное целое	Longint	-2 E9…2 E9	*4*
Вещественное	Real	$-10^{-38}…10^{38}$	*6*
Короткое целое	Byte	0…255	*1*
целое	Word	0…65535	*2*
Целое Длинное	Longword	$0…2^{32}$	*4*
Строковый	String [...]	1…255	*1 — 256*
Символьный	Char	1 символ	*1*
Логический	Boolean	True, False	1

Основные математические функции

В Паскале существуют следующие арифметические выражения: +, -, *, \

Но кроме них есть еще и следующие: DIV и MOD.

DIV - целая часть от деления, например: 5 div 2 = 2 или 20 div 3 = 6.

MOD - остаток от деления, например: 5 mod 2 = 1 или 20 mod 3 = 2. «mod» довольно часто применяется в различных программах, поэтому важно запомнить его значение, в принципе, как и все остальные значения других операций и функций.

Встроенные арифметические функции:

SQR (x); — x^2

SQRT (x); — $x^{1/2}$ (корень из x)

ABS (x); — |x| (модуль x)

ROUND (x); — округление до ближайшего целого

TRUNC (x); — отбрасывание дробной части числа

[**ARC**]**SIN** (x);, [**ARC**]**COS** (x);, [**ARC**]**TAN**(x); — (arc)sinx, (arc)cosx, (arc)tanx

EXP(x); — e^x (экспонента числа x)

LN (x);, **LOG2**(x);, **LOG10**(x);, **LOGN**(x); — натуральный логарифм, логарифм по основанию 2, десятичный логарифм и логарифм по основанию n (любое целое число). *В Turbo Pascal есть только ln(x).*

Не редко возникает вопрос: как возвести x в более высокую степень? ответ есть, но не для всех x:

EXP (n***LN**(x)); — $x^n = e^{n\ln x}$. Здесь на x накладывается следующее условие: x>0.

В PascalABC. NET существует еще одна некоторая элементарная операция, условие ее выполнения: переменная должна быть целого типа.

a += n; //увеличиваем переменную a на n едениц

b *= m; //увеличиваем b в m раз.

назад

Оператор присваивания

Оператор присваивания в общем виде выглядит следующим способом:

<имя 1-ой переменной>:=<имя 2-ой переменной>;

В данном случае говорят, что первой переменной присвоилось значение второй. При этом вместо второй переменной может быть любое выражение, либо строка, которые Вы хотите присвоить первой переменной.

Существуют некоторые основные правила присваивания:

1. *Переменная* и присваиваемое ей *значение* должны быть *одного типа*.
2. *Каждой переменной* должно быть присвоено какое-либо значение, *перед* ее *использованием*.
3. Ничто не *исчезает* так бесследно, как *старое значение* переменной.

Примеры:

a:=5; //a: integer;
b:=2; //b: integer;
c:=a+b; //переменной c присвоено значение 7, при этом если c типа integer или real эта операция пройдет, а если c:char или string появится ошибка, о том, что типы не совпадают.
b:=’дом’; //ошибка! Нельзя присвоить переменной с числовым типом значение типа строки.
word1:=’дом’; // word1: string;

назад

Раздел описания переменных VAR

Раздел «Var» (variable — переменная) предназначен для описания переменных. Переменные могут быть разных типов: целыми, строковыми, массивами (так называемый структурированный тип) и другими. Обозначать переменные следует только английскими буквами, а также цифрами, при этом на первом месте обязательно должна быть буква. Общий вид данного раздела выглядит следующим образом:

var <имя переменной>: <тип переменной>;

Пример:

```
var
  a,b: integer; //a и b целочисленные
  c: string [30]; //c - строка длиной в 30 символов
  line: array [1..40] of  real; //line - одномерный массив на
    40 элементов
```

наза

Раздел описания констант const

Раздел «Const» (constant — константа) предназначен для описания констант. Константы, как и переменные, могут быть разных типов: целыми, строковыми, массивами (так называемый структурированный тип) и другими. Обозначение констант одинаковое, что и у переменных. Общий вид данного раздела выглядит следующим образом:

const <имя константы> = <тип константы>;

Пример:

```
const
pi=3,14159;
answer='answer is';
```

наза

Раздел описания меток Label

Раздел «label» используется для описания, как видно из названия, меток. В общем виде этот раздел выглядит так:

label <имя метки>;

Имя метки может состоять из английских букв и положительных целых чисел. Метка используется для обеспечения, так называемого *безусловного перехода*, осуществляемого с помощью оператора «goto».

Пример:

label 1, a1, new;

наза

Оператор вывода данных на экран write(ln)

Оператор вывода данных на экран в общем виде выглядит следующим способом:

Write[**ln**][(данные):n:m];

Расшифрую, что значит данная запись:

Во-первых, можно использовать оператор «write» без приставки «ln», только в этом случае перехода на следующую строку на экране монитора не будет, т.е. все, что будет выводиться на экран — это одна длинная строка.

Во-вторых, данные — это арифметические выражения, которые в данном операторе можно не упрощать, он их сам посчитает и выведет на экран без сохранения в памяти. Можно указать *формат* вывода данных с помощью двух чисел через двоеточие (:n:m). Также вместо арифметических выражений можно написать какую-либо строку, которую Вы хотите увидеть на экране.

Число **n** — это общее количество цифр, включая знак и десятичную точку. Число **m** — это количество цифр после десятичной точки. Если формат введен больше, чем необходимо, то перед целой частью будут пробелы, а после дробной — нули.

Можно вывести на экран сразу несколько данных, только их надо будет *разделить запятой*.

Пример:

```
program PR1;
var  a,b: integer;
       c,d: real;
        m: string;
begin
   c:=6.8;
   b:=5;
   c:=c*2;
   b:=b-3;
   d:=c*b;
   m:='Ответ:';
   write (m, d:6:2);
   writeln ('  Ответ: при b=',b,' выражение будет равно ',b+c);
  writeln ('Конец программы');
end.
```

Результат:

Ответ:*27.20 Ответ: при b=2 выражение будет равно 15.6
Конец программы

*-пробел. Как видно, когда я не добавила «ln» два ответа получились на одной строке.

наза

Оператор ввода данных с клавиатуры

read(ln)

Оператор ввода данных с клавиатуры в общем виде выглядит следующим способом:

Read[**ln**][(переменные)];

«ln», как и в операторе вывода, здесь переход на следующую строку, но чтобы избежать ошибок чаще всего используют «readln». Переменных может быть любое количество, их следует разделять запятыми. При вводе с клавиатуры, при запущенной программе, несколько переменных, либо разделяют пробелом, либо нажатием клавиши ввод («Enter»).

Немного из теории:

Специализированная программа применяется для решения одной уникальной задачи.

Универсальная программа решает целый класс одинаковых (похожих) задач с различными входными данными.

Хорошим тоном в программировании считается разработка универсальной задачи, поэтому нам понадобится знание работы с входными данными.

Пример:

Допустим, что дана задача найти периметр и площадь прямоугольника, причем не указаны размеры его сторон.

Решение:

```
program square;
var a, b, S, P: real; //размеры сторон как и площадь вместе с периметром могут быть не целыми
Begin
    write ('Введите длины сторон прямоугольника:');
    readln (a,b);
    P:=2*(a+b);
    S:=a*b;
    writeln ('Периметр прямоугольника равен ',P);
    writeln ('Площадь прямоугольника равна ',S);
end.
```

Результат:

```
Введите длины сторон прямоугольника:3 4
Периметр прямоугольника равен 14
Площадь прямоугольника равна 12
```

наза

Оператор безусловного перехода goto

Начнем с того, что данный оператор используется крайне редко. Чаще всего вместо его используют операторы условия, циклы и процедуры, как раз использование данных операторов является хорошим тоном в программировании, когда «goto» не приветствуется. Но как бы там ни было — он все-таки есть и надо иметь о нем представление, хотя бы в цели того, что его очень удобно использовать при поиске ошибки в уже написанной программе (при отладке).

В общем виде он выглядит так:

Goto <метка>;

Метка определяется (описывается) в разделе «Label».

Пример:

```
program zaciklivanie;
    label 1;
begin
    1:write('*');
    readln;
    goto 1;
end.
```

Данный пример выводит бесконечное число звездочек на экран. Для того, чтобы прекратить работу программы нажмите «Ctrl+Break».

наза

Оператор условия if…then….else

Один из самых важных операторов в паскале. Данный оператор необходим тогда, когда нужно выполнить команды при определенном условии. Общий вид данного оператора:

if <логическое выражение> **then** <оператор>[;]

[else <оператор>];

if - если, then — тогда, else — иначе

Данная схема является простой реализацией данного оператора. [;] — данный символ обозначает то, что перед «else» *точка с запятой никогда не ставится*. Как упоминалось раньше оператор условия имеет несколько видов:

1. if — then;
2. if — then — else;
3. begin — end (блочная форма)

Блочная форма (составной оператор) применяется, когда по условию требуется выполнить не одну команду, а несколько. В общем виде:

if <логическое выражение> **then**

begin

<команды>;

end [;]

else

begin

<команды>;

end; //конец оператора условия (if — end;)

Перед «end» *можно* не ставить точку запятой.

Следует запомнить следующее правило программистов: «При двух каких-либо условиях пишется один оператор «if»".

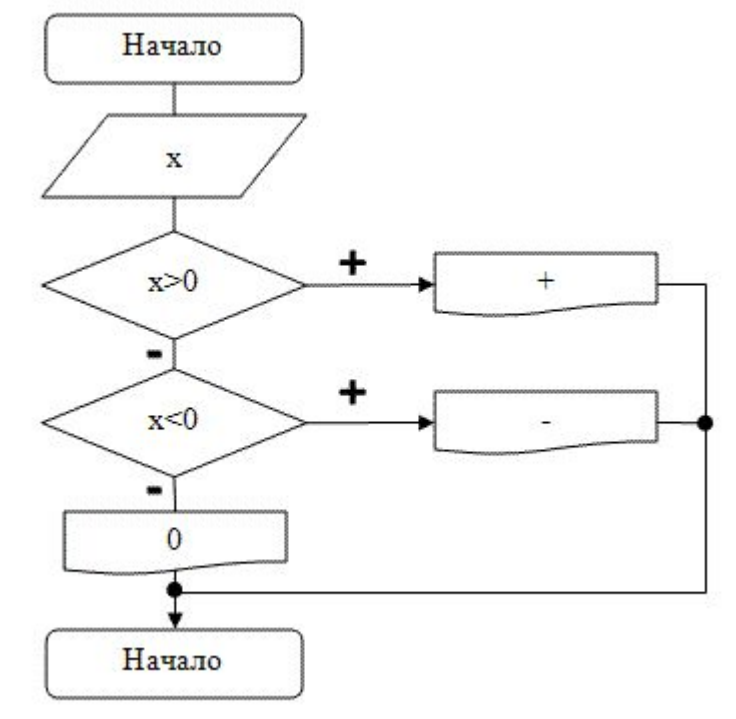
Пример: возьмем задачу на определение знака числа. Требуется ввести число и определить его знак.

Рекомендуется составлять небольшие таблички, типа «входные, исходные и выходные данные», составим ее:

Допустим, что входным будет число «x», исходных данных у нас нет, а выходными данными будут три варианта, либо плюс, либо минус, либо равен нулю. Таким образом получаем:

Входные данные: x;
Исходные данные: нет;
Выходные данные: +, -, 0.

Затем следует построить блок-схему задачи (алгоритм):



Блок схема к примеру

В разговорной речи мы бы могли просто сказать так:

1. Если x>0, тогда знак у него плюс,
2. Если x<0, тогда знак у него минус,
3. Если же x=0, тогда говорим, что икс равен нулю.

Но можно перефразировать немного иначе данный ответ:

1. Если x>0, тогда знак у него плюс, иначе
2. Если x<0, тогда знак у него минус, иначе икс равен нулю.

Как раз вот это слово «иначе» будет «вшито» в один оператор условия (смотрите вид номер 2), поэтому вместо трех «if» у нас будет два — это называется оптимизацией

Перейдем непосредственно к написанию **программы**:

```
program sgn;
var x: real;
begin
 write (‘Введите любое число:’);
 readln (x);
  if x>0 then writeln (‘Число положительное’)
  else if x<0 then writeln (‘Число отрицательное’)
    else writeln (‘Число равно 0’);
end.
```

назад

Цикл с параметром for…do

Циклы применяются, когда нужно выполнить одни и те же команды несколько раз.

Рассмотрим общий вид цикла с параметром:

For x:=a <**to, downto**> b **do** [оператор][;]

[**Begin**]

[тело цикла];

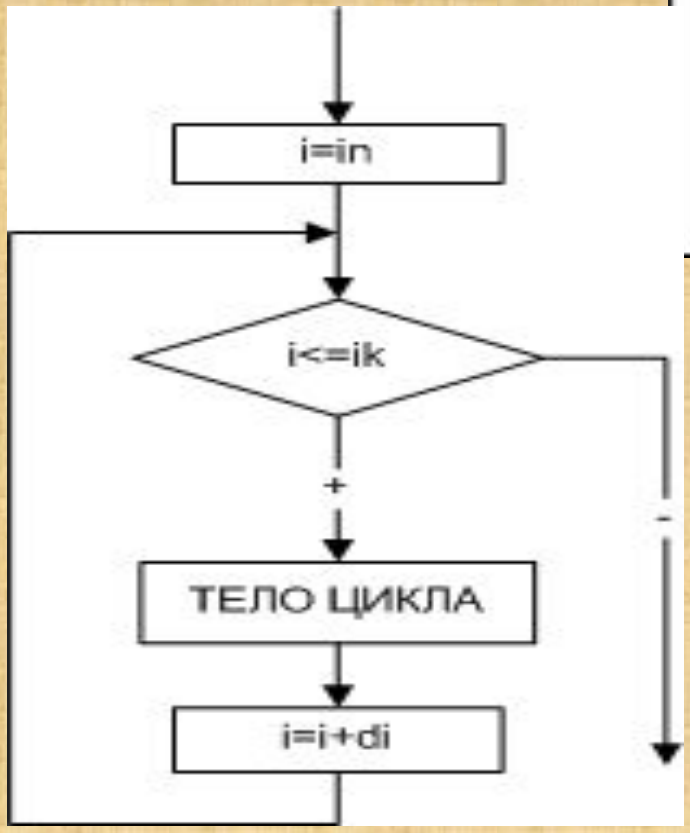
[**End**];

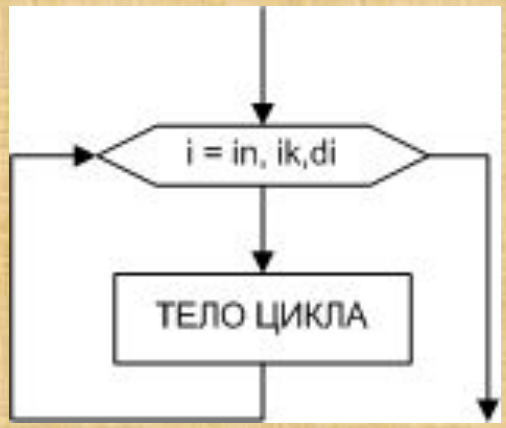
«to» применяется при увеличении аргумента a на единицу, а «downto» при уменьшении на единицу. Данный оператор читается следующим образом: управляющая переменная (x) принимает значения от (a) и изменяется с шагом (1 или -1) до конечного значения (b), при этом пока (x) не больше, чем (b), то выполняются команды, находящиеся в теле цикла.

Аналогично как и в операторе условия, так и в циклах (с параметром, с предусловием, с постусловием) бывают простые и блочные формы. Напомню, что *простой* формой называется запись оператора (цикла, условия и тд.), в теле которого присутствует только одна команда или один оператор, в блочной форме их может быть несколько.


ТЕЛО ЦИКЛА
УСЛОВИЕ
-
+


УСЛОВИЕ
+
-
ТЕЛО ЦИКЛА


i=in
i<=ik
+
-
ТЕЛО ЦИКЛА
i=i+di


i = in, ik,di
ТЕЛО ЦИКЛА

Существуют следующие правила:

1. Управляющая переменная (x) только целого типа.
2. Если a>b при шаге +1, то оператор не выполняется (аналогично для шага -1 (a<b)).
3. В теле цикла нельзя изменять основные параметры (x, a, b)
4. Допускается блочная форма оператора (тогда, когда нужно выполнить несколько команд в теле цикла).

Рассмотрим следующую задачу: вывести на экран все числа от 1 и до 10 включительно.

Решим ее без применения цикла, то есть с помощью операторов условия и безусловного перехода.

```
program num1;
var x:integer;
label 1;
Begin
    x:=1;
    1:  if x<=10 then begin
                        write (x:2);
                        x:=x+1;
                        goto 1
                       end
end.
```

Теперь решим задачу при помощи цикла:

```
program num2;
var x:integer;
Begin
  for x:=1 to 10 do write(x:2)
end.
```

Решение получилось в два раза короче

назад

Цикл с предусловием while…do

Циклы применяются, когда нужно выполнить одни и те же команды несколько раз.

Рассмотрим общий вид цикла с предусловием:

While <логическое выражение> **do** [оператор][;]

[**Begin**]

[тело цикла];

[**End**];

Главными плюсами данного цикла являются:

1. Индекс можно изменять не только с шагом плюс или минус один, а *на любую величину*, даже *не целого типа*(конечно в пределах разумной меры), в отличие от цикла с параметром.
2. В данном цикле можно *реализовать повторение* команд до *определенного условия* — до *логического выражения*.

Стоит правильно понимать суть этого цикла: «Пока данное условие выполняется (или логическое выражение не ложь) повторяем команды, написанные в теле цикла, иначе завершаем цикл».

Через цикл с предусловием можно реализовать цикл с параметром, за одним «но» — на практике лучше использовать «for».

Например:

У нас есть задача: вывести на экран все числа от 1 и до 10 включительно, которую уже мы делали, применяя цикл с параметром, пойдем другим путем:

```
program num3;
var x:integer;
Begin
    x:=1;
    while x<=10 do begin
                    write (x:2);
                    x:=x+1
                   end
end.
```

Как видно, программа получилась несколько длиннее, но сказано выше у данного цикла свои плюсы.

назад

Цикл с постусловием repeat….until

Циклы применяются, когда нужно выполнить одни и те же команды несколько раз.

Рассмотрим общий вид цикла с постусловием:

Repeat

[тело цикла];

Until <логическое выражение>;

Плюсы у данного цикла те же самые, что и у цикла с предусловием.

Стоит правильно понимать суть этого цикла: «Повторять команды, приведенные в теле цикла до тех пор, пока данное условие не будет выполнено (логическое выражение не будет истинным)».

Например:

Снова возьмем следующую задачу: вывести на экран все числа от 1 и до 10 включительно, которую уже мы делали, но пойдем другим путем:

```
program num4;
var x:integer;
Begin
  x:=1;
  repeat
  write (x:2);
   x:=x+1
  until x>10
end.
```

назад

Случайные числа randomize….random

Прежде, чем использовать случайные числа вначале программы пишут команду «**randomize**», таким образом, происходит активация *генератора случайных чисел*.

Получение случайных чисел:

1. <переменная>:=**random** — случайное вещественное число от 0 до 1 (не включая единицу)
2. <переменная>:=**random(<N>)** — случайное число от 0 до N-1
3. <переменная>:=trunc ((B-A)***random**+A) — то, что стоит в скобках вещественное число, но при помощи оператора «trunc» число получается целым от A до B.

Примечание к пункту 3: В «Pascal ABC. NET» все выражение можно записать при помощи только одной единственной команды **random (A, B)**. A и B — *целые числа*.

Пример:

Вывести на экран N случайных чисел от -20 до 20, определить количество отрицательных чисел и произведение всех положительных.

```
program rnd;
var z,n,s,p,i: integer;
Begin
      randomize; //активируем генератор случайных чисел
      write('Введите количество случайных чисел:');
      readln (n);
      s:=0;  //это сумматор, где будем считать количество отрицательных чисел
      p:=1;  //здесь будем считать произведение всех положительных чисел
     for i:=1 to n do begin
      z:=trunc(20-(-20))*random+(-20)); //В Pascal ABC. NET это выражение запишется так: z:=random (-20, 20);
      write (z:3);
      if z<0 then s:=s+1
     else if z>0 then p:=p*z //мы не учитываем ноль, так как сказано в условии задачи только про положительные и отрицательные числа, про ноль не оговаривалось
    end;
    writeln;
    writeln ('Количество отрицательных чисел:', s);
   writeln ('Произведение положительных чисел:', p)
end.
```

назад

Оператор выбора case…else

Данный оператор применяется вместо нескольких операторов условия. Общий вид:

Case <выражение> **of**

<список констант 1>:<оператор 1>;

…

< список констант n>:<оператор n>;

[**Else** <оператор>]

End;

Если значение выражения равно одной из констант (из списков от 1 до n), то выполняется соответствующий ей (константе) оператор, затем управление передается за пределы оператора выбора. Если же значение не равно ни одной константе, то управление передается по ветке «else», если же и этой ветки нет, то «case» не выполняет никаких действий.

Список констант может состоять из одной или несколько констант. Между константами должна стоять запятая. В Pascal ABC. NET и в Turbo Pascal можно указать промежуток констант например: 2..5.

Важно! Списки констант *не должны пересекаться*! То есть элементы списков не должны совпадать, и по значению, и по содержанию.

Например: Ввести число от 1 до 12, исходя из его значения, вывести на экран соответственно: месяц зимы, месяц весны, лета или осени…

```
program month;
var m: integer;
Begin
  write ('Введите число:');
  readln (m);
  case m of
    12,1..2: writeln ('Это месяц зимы');
    3..5: writeln ('Это месяц весны');
    6..8: writeln ('Это месяц лета')
    else writeln ('Это месяц осени')
  end
end.
```

назад

Операции над текстовыми данными

Напомню, как назначать текстовую переменную с помощью раздела «var»:

Var <имя переменной>:**string** [N];

N — это количество символов в строке (от 1 до 255). По умолчанию, если не указывать N, оно равно 255. Но если известно, что данная строка будет не длиннее, чем N символов, то рекомендуется его указывать. Например: вряд ли найдется имя, состоящее более, чем из 20 букв, поэтому «var name: string [20];».

Существует специальный тип данных, который вмещает в себя только один символ и не более. Данный тип обозначается: «char»:

Var <имя переменной>:**char**;

Над строками допустимы операции склеивания (+) и сравнения (<, >, = и т.д.).

Операция сравнения

Для операции сравнения верно следующее:

Цифры (от 0 до 9) < Большие буквы русского алфавита (от А до Я) < Маленькие буквы русского алфавита (от а до я) < Большие буквы английского алфавита (от A до Z) < Маленькие буквы английского алфавита (от a до z).

Причем для цифр верно следующее: 0<1<2<…<8<9,

Для букв русского алфавита:
А<Б<…Ю<Я<а<б<…<ю<я,

Для букв английского алфавита:
A<B<…<Y<Z<a<b<…<y<z.

Например: ‘ИВАНОВ’ < ‘Иванов’, ’121′<’125′, ’125′<’13′.

Операция склеивания

Допустим, что были созданы в разделе «var» две строковые переменные: A и B. Для них верны следующие операции:

```
A:=’хол’;
  B:=’од’;
  C:=A+B;
  writeln (C);
```

На экране после запуска программы появится следующее слово: «холод».

К отдельным символам строки можно обратиться с помощью номеров в квадратных скобках:

```
B:=C[5]+C[4]+C[3];
  writeln (B);
```

На экране появится слово: «дол».

Основные функции и процедуры для обработки строк

Copy (<текст>, [N], [M]) — возвращает M символов строки <текст>, начиная с символа под номером N этой строки.

Пример:

writeln (C, 5, 3);

Выведет слово: ‘дол’

Length (<текст>) — определяет длину строки (или количество символов)

Пример:

writeln (length (A));

Выведет цифру: 3.

Chr (<код>) — преобразует код в символ (по таблице ASCII).

Пример:

writeln(chr(70));

Выведет букву: «F»

UpCase (<символ>) и **LowCase** (<символ>) — функции, которые преобразуют символы в строке из нижнего в верхний регистр и из верхнего в нижний регистр соответственно (т.е. из прописных в заглавные и из заглавных в прописные). Минус данных функций, что они преобразуют только один символ, поэтому для того, чтобы преобразовать всю строку, например, из нижнего регистра в верхний организовывают символ.

Пример

```
program upc;
var sent: string [6];
  i:byte;
  begin
  sent:='pascal';
  for i:=1 to length(sent) do sent[i]:=upcase(sent[i]);
  writeln(sent);
end.
```

Выведется: «PASCAL».

назад

Массивы array

Массивы используются для хранения большого количества данных. Существую несколько видов массивов:

1. Одномерный (Вектор)
2. Двумерный (Матрица)
3. Трехмерный (Кубический)
4. Многомерный

В программах чаще всего используют массивы первых двух видов.

0	1	2	3	4	5	…	…	…	n-1
3	4	7	21	4.5	…				[э]

[э] — какой-либо элемент

На данном рисунке изображен одномерный числовой массив (назовем его A), состоящий из n элементов. Следует обратить внимание, что индекс элементов начи

	0	1	…	i-1
0	4	11	…	[э]
1	24.5	78	…	[э]
…	…	…	…	…
j-1	[э]	[э]	[э]	[э]

На этом рисунке изображен двумерный числовой массив из *i*x*j* элементов (назовем его B). Чаще используется квадратная матрица, где i=j=n, иначе обозначаемая *n*x*n*.

Примечание: На данных примерах массивов A и B, я указал наиболее часто используемые и более понятные виды массивов, с которыми намного легче работать. На самом деле индексы начала и конца массива могут задаваться пользователем на усмотрение, о чем будет говориться далее.

Можно провести аналогию массивов и переменных. Имя массива обозначается так же, как и имя переменной, за исключением индексов, стоящих в квадратных скобках, например: для первого одномерного массива: A[1] он будет равным 4, — для второго массива: B[0,1] он будет равным 24.5, — и так далее. Массивы, как и переменные, могут быть разного типа: числового (целого, вещественного…), строкового, символьного и другого.

Описание массивов

Описывать массивы можно, как и через блок описания переменных «Var», через блок описания констант «Const», так и с помощью блока описания типов «Type».

Рассмотрим два первых метода описания (последний способ можно найти, перейдя по ссылке).

Общий вид описания в блоке «Var»:

Var <имя массива>: **array** [тип индекса в кв. скобках] **of** <тип элементов>=[(элементы)];

Часто не указывают тип индекса, а задают тип промежутками, например: [1..10,-5..5]. Пример:

```
Var massiv: array [1..10,-5..5] of real;
```

Таким образом, мы создали двумерный массив «massiv» со строками с индексами от i=1,2,…, 10 и со столбцами с индексами j=-5,-4,…,5.

В PascalABC. NET вводится следующее наименование таких массивов — *статические*. Как видно из общего вида данные массивы можно изначально заполнить какими-либо элементами, которые затем можно будет изменить в течение программы. При этом индексами могут быть не только цифры, но и буквы, например: ['a'..'z'] или [d1..d5].

Общий вид описания в блоке «Const»:

Const <имя массива>=**array** [тип индекса в кв. скобках] **of** <тип элементов> = (элементы);

Пример:

```
Const massivc = array [0..2,0..2] of integer = ((2,4,7),(10,23,-54));
```

Можно копировать содержимое массивов из одного в другой, но только при условиях, что:

1. Массивы одного типа
2. Массивы одинакового размера,

воспользовавшись оператором присваивания: m1:=m2;

При работе с массивами, а именно: заполнении, счете, чтении и т.д., — используют циклы. Для работы, например, с двумерным массивом, используют два вложенных цикла. Обработка матрицы возможна по строкам или по колонкам, аналогично и для других случаев.

Пример:

Задача: создать массив и заполнить данный массив с клавиатуры.

```
program arr;
var table: array [0..4, -2..2] of integer;
   i,j: byte;
Begin
   for i:=0 to 4 do
   for j:=-2 to 2 do begin
   write ('Введите элемент с индексами (',i,',',j,'):');
   readln(table[i,j])
   end
end.
```

назад

Модульное программирование

В широком смысле модульное программирование — это последовательное выделение из исходной задачи (программы) более простых подзадач (подпрограмм).

Подпрограммы применяются как отдельные, логически законченные части программ. Как правило, их выполняют несколько раз.

В паскале используются два вида подпрограмм:

1. Функции (*Function*) — для получения только одного выходного значения;
2. Процедуры (*Procedure*) — для получения нескольких выходных значений.

Основные термины

Формальные параметры — это параметры, которые указываются при описании подпрограммы. Если он описан после ключевого слова *«var«*, то такой параметр называют параметром-переменной.

Фактические параметры — это параметры, которые указываются при вызове подпрограммы. Такой параметр, говорят, передается *по ссылке,* если в описании подпрограммы используется параметр-переменная соответствующая фактическому. В данном случае фактический параметр и соответствующая ему параметр-переменная объявляются *эквивалентными*. То есть фактический параметр на выходе будет равен параметру-переменной, после завершения работы подпрограммы; либо самому себе, если параметр-переменная не изменялась в ходе выполнения функции или процедуры.

Типы фактических переменных и формальных параметров должны совпадать или, как говорят, быть совместимыми по присваиванию.

Пример: Вызываем процедуру kub с фактическим параметром n: kub(n);

```
procedure kub(var a:integer);
Begin
  a:=a*a*a
end;
```

После завершения процедуры n изменится и будет равно a. То есть мы нашли, таким образом, n в кубе.

Локальными переменными называются переменные, которые описываются в блоке описаний функций или процедур. Их значения никогда не покидают границы подпрограммы, другими словами, их кроме как в подпрограмме больше нигде не видно.

Очевидно, что существуют и ***глобальные переменные***, которые видны везде (например такие объекты, как классы).

назад

Функции function

Описание функции:

Function <имя> [(входные формальные параметры: тип);...)]:тип значения функции;

<Блок описания переменных функции>;

Begin

<Блок операторов>;

End;

Описываются функции *до начала* основной программы.

Несколько входных формальных параметров разных типов следует разделять точкой запятой, одинаковых типов — запятой. Данных параметров может и не быть. Выходное значение у функции *только одно*.

К функции можно обращаться из тела самой этой функции (вызвать саму себя). Данный вид планирования решения задачи или алгоритма программы называется ***рекурсией***.

При решении задач с использованием функций необходимо определить ее назначение (тип значения), количество и тип входных формальных параметров.

Пример: Определить значение следующего выражения:

$$x = \frac{\max(5,a) + \max(3+a,b)}{\max(a,b)}$$

где max — это значение, максимальное из двух аргументов.

```
program reshenie;
var x, a, b: real;
function max(fa, fb: real): real; //описание функции
begin                  //начало блока операторов
  if fa>fb then max:=fa
  else max:=fb
end;
begin            //начало основной программы
  write ('Введите a и b:');
  readln (a,b);
  x:=(max(5,a)+max(3+a,b))/max(a,b);
  writeln ('Ответ:', x:8:2);
end.
```

назад

Процедуры procedure

Описание процедуры:

Procedure <имя> [(формальные параметры: тип);...)];

<Блок описания переменных процедуры>;

Begin

<Блок операторов>;

End;

Описываются процедуры *до начала* основной программы.

Несколько входных формальных параметров разных типов следует разделять точкой запятой, одинаковых типов — запятой. Данных параметров может и не быть. Количество выходных значений у процедуры *равно количеству формальных параметров*.

В блоке описания процедур можно описывать переменные, константы, типы, вложенные процедуры и функции, при условии, что они будут являться *локальными*.

Пример: Посчитать факториал числа.

```
program factorial;
var n:integer;
procedure fct (a:integer);
var i:byte;
    P:integer;
Begin
  P:=1;
  for i:=1 to a do P:=P*i;
  write (n,'!=',P)
end;
Begin
  write ('Введите n:');
  readln (n);
  fct(n);
end.
```

Данный пример демонстрирует работу процедуры.

На самом деле для того, чтобы посчитать факториал (или какую-нибудь сумму) один раз, не надо создавать процедуру. Посчитать это можно в основной программе.

назад

Блок описания типов type

Блок описания типов выглядит следующим образом:

type

[описание типов];

[описание записей];

[описание файловых переменных];

Описание типов в общем случае выглядит так:

<имя переменной> = <тип>;

Между именем переменной и типом стоит «равно», а не «двоеточие», как в блоке «Var».

Например:

```
Type
   myarray = array [0 .. 100] of char;
   stek = myarray; //В разделе описания переменных теперь можно не вводить каждый раз данный символьный массив, а просто написать a: myarray; или a: stek;
```

Теперь разберем следующее составляющее блока «Type» — это описание записей:

<имя записи> = **record**

<1 поле>:<тип>;

<2 поле>:<тип>;

…

<N поле>:<тип>;

end;

Данную конструкцию, помимо *записи*, называют также *составной переменной* или *объектом*. Типы полей могут, и различаться, и совпадать. Доступ к полям осуществляется через **точку:**

<имя записи>**.**<имя поля>

Существует специальный оператор, с помощью которого можно упростить инициализацию записей. Его вид:

with <имя записи> **do begin**

<1 поле>:=<выражение>;

…

<N поле>:=<выражение>;

end;

Пример:

```
type Tdate = record
   day: 1..31;
   month: 1..12;
   year:1500..2010
end;
type Tbook = record
   autor: string[50];
   name: string[50];
   published: Tdate;
end;
var book:Tbook;
Begin
    //Инициализация может проходить двумя способами
   //Первый способ:
   book.autor:='Иванов Иван Иванович';
   book.name:='Приключения';
   book.published.day:=12;
   book.published.month:=2;
   book.published.year:=2010;
   //Второй способ:
   with book do begin
      autor:= 'Иванов Иван Иванович';
      name:= 'Приключения';
      published.day:=12;
      published.month:=2;
      published.year:=2010
   end
 end.
```

Для работы с файлами в паскале существуют специальные *файловые переменные*, их можно описать следующим образом:

<имя файловой переменной>=**file of** <тип>

Типом файловой переменной может быть, и основной, и созданный тип пользователем в блоке «Type». Данный метод описания файлов называется *типизированным*, но об этом позже.

назад

Работа с файлами

Для того, чтобы работать с файлами в паскале, объявляют переменную, которую называют файловой. Файловые переменные бывают следующих видов:

Текстовые

Типизированные

Не типизированные

Общий вид объявления такой, соответственно для каждого вида:

var

<имя файловой переменной>: **text**;
<имя файловой переменной >: **file of** <тип>;
<имя файловой переменной >: **file**;

Файловые переменные (далее ф.п.) второго вида могут объявляться любым основным типом. Следующим шагом надо связать файловую переменную и файл (физический файл), находящийся на жестком диске или на съемном носителе:

assign (<имя ф.п.>, <директория>);

Примечание: в Pascal ABC .NET можно использовать оператор **AssignFile**(<имя ф.п.>, <директория>);

Запомните! После того, как Вы связали ф.п. с физическим файлом, работать с последним Вы еще не можете, так как он еще не открыт.

Для того чтобы открыть файл на чтение и на запись используют оператор **reset** (<имя ф. п.>);. Примечание: файл, связанный с переменной *текстового* типа открывается просто на *чтение*.

Для того чтобы открыть файл на запись, обнулив все его содержимое, даже если он уже существовал, используют **rewrite** (<имя ф.п.>);. Примечание: Рекомендуется данную команду применять сразу после того, как создан новый файл.

Для закрытия файла используют **close** (<имя ф.п.>);. В Pascal ABC .NET можно использовать **CloseFile** (<имя ф. п.>);. Примечание: Рекомендуется при завершении работы приложения закрывать все файлы, открытые данной программой.

Erase (<имя ф.п.>); — удаляет файл, связанный с файловой переменной, с диска.

Rename (<имя ф.п.>); — переименовывает файл на диске, связанный с файловой переменной.

Функция **EOF** (<имя ф.п.>); — возвращает значение *True*, если достигнут конец файла, иначе *False*, если конец не достигнут.

Для считывания информации из файлов применяют следующие команды:

Read (<имя ф.п.>, <список переменных>); — считывает информацию из файла в переменные и оставляет указатель на этой же строке в файле.

Write (<имя ф.п.>, <список переменных>); — записывает в файл информацию, содержащуюся в переменных, и не переводит указатель на следующую строку.

Разборы команд, которые применяются только для определенного вида файловых переменных и разборы примеров находятся на следующих страницах:

Текстовые файлы и Типизированые файлы

назад

Текстовые файлы

Теперь разберемся с командами, которые могут быть использованы только в случае одного из видов файловых переменных.

Append (<имя ф.п.>); — открывает текстовый файл, ставя указатель на его конец. Таким образом, можно дописать какую-нибудь информацию.

Readln (<имя ф.п.>, <список переменных>); — считывает информацию из файла в переменные и переводит указатель в файле на следующую строку.

Writeln (<имя ф.п.>, <список переменных>); — записывает в файл информацию, содержащуюся в переменных, и переводит указатель на следующую строку.

Eoln (<имя ф.п.>); — возвращает *True*, если достигнут конец строки в файле, иначе *False*.

- **Пример:**

```
Program PutGetBirthdays;
type TBirthday = record
    day:string;
    month:string;
    year:string;
end;
type TFIO = record
    Family: string ;
   Name: string ;
    Otch: string ;
end;
type TLichn = record
    FIO: TFIO;
   Birt: TBirthday;
end;
var p:TLichn;
  f:text;
    ans:string[3];
    i,n:byte;
//Функция посимвольного разбора данных в файле
function OsymB (OsmB:string):string;
Var SymB:char;
```

```
Begin
 //Цикл пока не достигнем конца строки, файла или данного
  repeat
    read(f,SymB); //читаем символ
    //Если символ не пуст, то формируем строку
    if (Symb<>' ') and (Symb<>'') then OsmB:=OsmB+SymB;
  until (Symb=' ') or (EOF(f)=True) or (Eoln(f)=True);
  Result:=OsmB; //Возвращаем полученную строку
End;
Begin
  Assign (f,'C:\\temp\birthdays.txt');
  Rewrite (f);
  n:=1;
  //Вводим нужную информацию
  repeat
    writeln ('Ввод ',n,'-го человека:');
    write ('Введите фамилию:');
     readln (p.FIO.family);
    write ('Введите имя:');
    readln (p.FIO.name);
    write ('Введите отчество:');
    readln (p.FIO.Otch);
   write ('Введите день рождения:');
     readln (p.Birt.day);
     write ('Введите месяц рождения:');
    readln (p.Birt.month);
    write ('Введите год рождения:');
    readln (p.Birt.year);
     write ('Продолжить ввод?(да/нет):');
    readln (ans);
```

```
//Записываем данные в файл, разделяя пробелами
      write (f,p.FIO.family,' ',p.FIO.name,' ',p.FIO.Otch,' ');
      write (f,p.Birt.day,' ',p.Birt.month,' ',p.Birt.year);
      writeln (f); //Переходим на следующую строчку в файле
      n:=n+1;
   until ans='нет';
   close (f);    //Закрываем файл
   reset (f);    //Открываем
   i:=0;
   //Выводим
   while (i<n-1) do begin
      writeln ('');
      writeln ('Вывод ',i+1,'-го человека:');
      p.FIO.family:=OsymB ('');
      p.FIO.name:=OsymB ('');
      p.FIO.Otch:=OsymB ('');
      p.Birt.day:=OsymB ('');
      p.Birt.month:=OsymB ('');
      p.Birt.year:=OsymB ('');
      writeln ('ФИО:',p.FIO.family,' ',p.FIO.name,' ',p.FIO.Otch);
      writeln ('ДР:',p.Birt.day,'.',p.Birt.month,'.', p.Birt.year);
      readln(f);
      i:=i+1;
   end;
   close (f);
end.
```

назад

Типизированные файлы

Как упоминалось ранее, для того, чтобы создать типизированный файл, нужно объявить файловую переменную с любым типом, даже с составным, который Вы создали сами. Общий синтаксис объявления такой переменной:

var <имя ф.п.>: **file of** <тип>;

Для работы с типизированными файлами, кроме общих команд для всех файлов, используют следующие:

FilePos (<имя ф.п.>); — возвращает положение указателя в файле.

FileSize (<имя ф.п.>); — возвращает количество элементов в файле.

Seek (<имя ф.п.>; <номер элемента>); — перемещает указатель на конкретный элемент в файле.

Truncate (<имя ф.п.>); — удаляет все элементы в файле с позиции указателя.

```
program PutGetBirthday;
type TBirthday = record
    day: 1..31;
    month: 1..12;
   year: 1..2010;
end;
type TFIO = record
    Family: string [50];
    Name: string [50];
    Otch: string [50];
end;
type TLichn = record
    FIO: TFIO;
    Birt: TBirthday;
end;
var p:TLichn;
    f:file of TLichn;
    ans:string[3];
    i,n:byte;
```

```
Begin
  Assign (f,'C:\\temp\Birthdays.txt');
   Rewrite (f);
   n:=1;
   //Вводим данные
   repeat
      writeln ('Ввод ',n,'-го человека:');
      write ('Введите фамилию:');
      readln (p.FIO.family);
      write ('Введите имя:');
      readln (p.FIO.name);
      write ('Введите отчество:');
      readln (p.FIO.Otch);
      write ('Введите день рождения ДД ММ ГГ:');
      readln (p.Birt.day, p.Birt.month, p.Birt.year);
      write (f,p);
      write ('Продолжить ввод?(да/нет):');
      readln (ans);
      n:=n+1;
   until ans='нет';
```

```
//Для того, чтобы вывести все на экран, передем к началу
    файла
   seek (f,0);
   i:=0;
   //Выведем на экран записанную в файле информацию
   while i<n-1 do begin
      writeln ('');
      writeln ('Вывод ',i+1,'-го человека:');
      read (f,p);
      writeln ('ФИО:',p.FIO.family,' ',p.FIO.name,' ',p.FIO.Otch);
      writeln ('ДР:',p.Birt.day,'.',p.Birt.month,'.', p.Birt.year);
      i:=i+1;
   end;
    close (f);
end.
```

Данная программа работает визуально абсолютно одинаково, как и программа, приведенная в примере с текстовыми файлами. На самом деле, как видно, это не так. В данном случае процесс более оптимизирован и выполняется гораздо быстрее, требует меньше выделяемой памяти под все созданные типы. Но на выходе файл получается гораздо больше размером, чем текстовый, и его нельзя прочитать, если не использовать специальную программу, либо программу вроде этой.

Примечание:

Я заметил небольшое отличие в работе компилятора PascalABC и PascalABC. NET. Данная программа будет выполняться и там и там. Но, если обозначить тип TFIO следующим образом:

```
type TFIO = record
   Family: string;
   Name: string;
   Otch: string;
end;
```

То есть убрать длину строки у типа string, то компилятор в PascalABC. NET выдаст ошибку о том, что переменная f не может быть типизированной файловой переменной с типом TLichn. В PascalABC компилятор не обращает на это внимание.

назад

Указатели. Общие сведения

Указатели – это ячейки памяти, в которых хранится адрес.

Каждая переменная, константа, массив (а также процедуры, функции, структуры и т.д.) хранятся в памяти, поэтому имеют какой-либо определенный адрес. При компиляции и компоновке компилятор к каждой переменной (массиву или константе), которая используется в выражении, подставляет адрес, для того, чтобы процессор смог определить в каком месте, по какому адресу располагается то или иное значение переменной.

Таким образом, компилятор выполняет несколько действий, как минимум это: поиск по названию переменной ее объявление, определение типа, для того, чтобы определить смещение и подстановка адреса. Если бы мы знали адрес переменной изначально, то легче было бы его просто изначально его и подставить, тогда компилятор не будет выполнять некоторые «лишние» действия. В результате мы выиграем очень дорогой ресурс в программировании – процессорное время.

Но это еще не все достоинства указателей. Приведу еще один пример: сортировка массива строк. Допустим, у нас есть массив, состоящий из имен студентов. Нам надо его отсортировать по алфавиту.

Самым «легким» способом будет взять этот массив и просто напросто переставлять строки местами так, как делают это с числами. Только с числами все совсем по-другому. Число – это одна ячейка памяти, когда строка – это целый массив, элементами которого являются символы, входящие в данную строку. Поэтому числовой массив отсортировать гораздо легче, чем строковый. Возможно, программируя на паскале, Вы это не заметите, но например, в Си или Си++ эта разница очень заметна.

Есть другой способ сортировки массива строк, как раз таки – *сортировка указателей* в массиве. Каждая ячейка массива имеет свой адрес, каждый начальный элемент строк имеет адрес, при этом указывает на следующие элементы строки. Поэтому намного проще будет переставить указатели на соответствующую строку в массиве, другими словами поменять адреса в массиве, нежели переставить строку.

Следует научиться использовать указатели – это очень мощное средство в программировании. Они применяются везде, в какой бы среде программирования Вы не работали.

Существуют два вида указателей:

Типизированные, содержащие адрес на ячейку памяти определенного типа.

Не типизированные, используются в тех случаях, когда программисту неизвестно какой тип будет переменной (константы, массива …).

назад

Типизированные указатели

Краткие теоретические сведения находятся в пункте «Указатели: Общие сведения»

Указатель – это тип. Переменную типа указатель также называют указатель.

Типизированные указатели указывают на ячейку памяти определенного, либо основного типа данных, либо пользовательского.

Объявление типизированного указателя:

var

<имя указателя>:=^<тип>;

Пример:

```
Var
   a: integer;
   pI: ^integer;
```

Данная запись читается так: pI – указатель на integer (целочисленную переменную).

Для того чтобы указать pI на a следует применить операцию взятия адреса **@**, а для того, чтобы получить значение переменной через указатель применяют операцию разыменования **^**:

```
Begin
   pI:=@a; //Присвоили указателю адрес переменной a
   pI^:=5; //Записали по указателю значение 5, при этом a стало равно 5
   writeln ('Значение переменной через указатель:', pI^);
end.
```

Если при выводе значения после pI не поставить операцию разыменования, то на экран выведется адрес переменной a.

Следует помнить, что если Вы по указателю на integer запишете, например, не целое, а вещественное число, то произойдет ошибка компиляции, по аналогии с типами переменных.

С самими указателями (без операции разыменования) можно делать практически все операции, кроме умножения и деления. Что происходит, например, когда к указателю прибавить единицу?

```
begin
   pI:=@a; //Присвоили указателю адрес переменной a
   pI^:=5; //Записали по указателю значение 5, при этом a стало
     равно 5
   writeln ('Значение переменной через указатель:', pI^);
   writeln ('Адрес, который записан в указателе:', pI);
   pI:=pI+1;
   writeln ('Новый адрес, который записан в указателе:', pI);
end.
```

Если значение адреса в pI было, к примеру: $0A652F04, то после увеличения на единицу, казалось бы, должен стать $0A652F05. Но не тут то было, вывелось $0A652F08.

Все просто: тип нашего указателя: integer — можно посмотреть в таблице типов, сколько занимает данный тип места в памяти: ровно 4 байта — поэтому адрес увеличился на 4. Другими словами операции сложения и вычитания делают со значениями указателей следующее: здесь целое число (у нас оно равно 1) означает количество ячеек указуемого типа, а знак операции — куда сдвигаться (в нашем случае мы сдвигаемся в памяти “вправо”).

Примечание: все адреса хранятся в шестнадцатеричной системе счисления

Примечание: арифметические действия с указателем нельзя выполнять для указателей на пользовательские типы. Видимо, компилятор паскаля запрещает это делать.

Указатели часто используют, когда в качестве фактического параметра в процедурах и в функциях передают какие-либо крупные массивы, структуры, либо с ориентацией на изменение локальной переменной за пределами одной функции или процедуры. (Об этом поговорим подробнее позже).

Приведу пример:

```
program func;
Var
   a: integer;
   p:^integer;
 function square (p:^integer):integer; //Здесь говорят так, что значение формального
      параметра передается по указателю
Begin
   p^:=p^*p^; //Локальная переменная a в вызывающей функции будет изменяться
end;
Begin
   a:=10;
   p:=@a;
   square(p); //Поэтому здесь возвращаемое значение функции можно ни к чему не
      присваивать, так как оно уже записано в a
   writeln (a); //Выведется 100
end.
```

назад

Перенос программ, написанных на Turbo Pascal в Delphi