

**\* Процедуры и функции.  
Заголовок и тело  
процедур и функций,  
классификация  
параметров. Вызов  
процедур и функций,  
особенности их  
использования.**

**Лекция 9**

В программах на Pascal используются подпрограммы двух видов: *процедуры и функции*.

Имея один и тот же смысл и аналогичную структуру, они несколько различаются назначением и способом их использования.

Все процедуры и функции, в свою очередь, подразделяются на две группы:

- стандартные (встроенные);
- определенные пользователем.

*Встроенные (стандартные) процедуры и функции* входят в стандартные библиотеки и могут вызываться по имени без *предварительного описания* (например, **read** и **write**).

*Процедуры и функции пользователя* пишутся самим программистом и помещаются в *раздел описаний процедур и функций*. Их вызов для выполнения записывается в *разделе операторов* основной программы или другой подпрограммы.

# \* Процедуры

*Процедура* — это независимая именованная часть программы, которую после *однократного* описания можно *множественно* вызывать по имени из последующих частей программы. Вызов процедуры оформляется как отдельный оператор, процедура не может выступать как операнд в выражении.

*Структура процедуры* повторяет структуру программы, это "программа в миниатюре", состоящая из заголовка и тела. В отличие от программы, для процедур и функций наличие заголовка обязательно.

*Заголовок* состоит из зарезервированного слова **procedure**, идентификатора (*имени*) процедуры и *необязательного* списка *формальных параметров*. *Тело процедуры* по своей структуре аналогично обычной программе:

**procedure** **ИмяПроцедуры** (**ФормальныеПараметры**);

**{** **Описательная часть процедуры** **}**

**begin**

**{** **Инструкции исполнительской части процедуры** **}**

**end;**

*Обратите внимание* — в конце тела процедуры, как и в конце программы, стоит **end**, однако после **end** ставится *точка с запятой*, а не *точка*.

Для обращения к процедуре используется *оператор вызова процедуры*. Он состоит из *имени* процедуры и *необязательного списка фактических параметров*, отделенных друг от друга запятыми:

### **ИмяПроцедуры(ФактическиеПараметры);**

При вызове процедуры работа главной программы *приостанавливается* и начинает выполняться *вызванная процедура*. Когда процедура выполнит свою задачу, программа *продолжится* с оператора, следующего за оператором вызова процедуры.

Для *принудительного выхода* из процедуры в ее теле записывается **оператор завершения exit**, который обеспечивает выход во внешний блок (обычно — в основную программу).

Пример. Написать программу вычисляющую сумму  $1+x+x^2+x^3+\dots+x^n$ .

```
Var  
s,x,n,i,p:integer;  
procedure power (n,m:integer; var p:integer);  
var  
    i:integer;  
begin  
    p:=1;  
    for i :=1 to m do  
        p:=p*n;  
    end;
```

```
begin
```

```
  write('Введите x');
```

```
  readln(x);
```

```
  write('Введите n');
```

```
  readln(n);
```

```
  s:=1;
```

```
for i:=1 to n do
```

```
  begin
```

```
    power(x,i,p);
```

```
    s:=s+p;
```

```
  end;
```

```
  readln;
```

```
end.
```

# \* Механизм передачи параметров

Параметры, которые указываются при описании процедур и функций, называются **формальными**. Название "формальные" параметры получили в связи с тем, что они нужны только для записи алгоритма, а при вызове подпрограммы на их место будут *подставлены конкретные фактические* параметры.

Соответствие между формальными и фактическими параметрами обеспечивается выполнением следующих требований:

- формальных и фактических параметров должно быть *одинаковое количество*;
- *порядок следования* фактических и формальных параметров должен быть *один и тот же*;
- *тип* фактического параметра *должен быть совместим* с типом соответствующего ему формального параметра

Имеются два основных вида параметров, которые отличаются способом их передачи в подпрограмму:

- параметры-значения;
- параметры-переменные;
- параметры-константы.

**Параметры-значения:** Подпрограмме передается лишь значение параметра, которое помещается в переменную, специально созданную для этой цели. Таким образом, в подпрограмме используется копия фактического параметра и его изменение в подпрограмме никак не скажется на значении его в самой программе, т. к. меняется копия.

Поэтому параметры-значения нельзя использовать для передачи результатов из подпрограммы в основную программу.

**Параметры-переменные:** Каждому формальному параметру-переменной должен соответствовать фактический параметр в виде *переменной* (выражение не допускается). В этом случае при вызове подпрограммы ей передается адрес фактического параметра в памяти и в дальнейшем подпрограмма работает именно с этой ячейкой памяти, а не с копией, как при использовании параметра-значения, т.е. если переменная меняется в подпрограмме, то она изменится и в программе

В списке формальных параметров перед ними ставится ключевое слово **var**.

**Вывод:** исходные данные в подпрограмму могут передаваться как через параметры-значения, так и через параметры-переменные, а результаты работы подпрограммы возвращаются в программу только через параметры-переменные.

Пример:

```
var a,b: integer;  
procedure kub (x: integer; var y: integer);  
begin  
x:= x*x*x;  
y:= y*y*y;  
writeln(x, '    ',y);  
end;  
begin  
a:=3; b:=2;  
kub (a, b); writeln(a, '    ',b);  
end.
```

При выполнении программа выведет на экран:

27 8

3 8

**Параметры-константы** передаются по адресу, как параметры-переменные, но значения их запрещено изменять в подпрограмме. За этим строго следит компилятор, при нарушении запрета он выдает сообщение об ошибке.

При описании подпрограммы перед параметрами-константами добавляется служебное слово **const**, например:

```
procedure proc (const p: integer);
```

Параметры-константы — это очень хороший способ передачи исходных данных в подпрограмму, т. к. он не требует дополнительных затрат памяти для хранения копий. При этом сами исходные данные в своих ячейках памяти остаются в неприкосновенности.

# \* Функции

Если результатом подпрограммы является только одно значение, то имеет смысл оформить такую подпрограмму не в виде процедуры, а в виде функции.

*Функция пользователя* во всем аналогична процедуре, за исключением двух отличий:

- функция передает в программу результат своей работы — единственное значение, носителем которого является имя самой функции;
- имя функции может входить в выражение как операнд.

Функция, определенная пользователем, состоит из заголовка и тела функции. *Заголовок* содержит зарезервированное слово **function**, *имя* функции, *необязательный* список *формальных* параметров и *тип* возвращаемого функцией значения. *Тело функции* по своей структуре аналогично обычной программе:

```
function ИмяФункции (ФормальныеПараметры):
```

```
ТипРезультата;
```

```
{ Описательная часть функции }
```

```
begin
```

```
    { Инструкции исполнительной части функции }
```

```
    ИмяФункции := Результат;
```

```
end;
```

В разделе операторов функции должен находиться по крайней мере один оператор, который присваивает ее имени значение *результата работы* функции. Если таких присваиваний *несколько*, то результатом функции будет значение *последнего* выполненного оператора присваивания. Если же такой оператор отсутствует, то значение, возвращаемое функцией, *не определено*.

Пример. Написать программу, которая находит для любых двух точек А и В плоскости находит расстояние между ними и вектор АВ:

```
function rasst(x1,y1,x2,y2:real; var vect1,vect2:real):real;  
begin  
    vec1:=x2-x1;  
    vec2:=y2-y1;  
    rasst:=sqrt(sqr(vec1)+sqr(vec2));  
end;  
  
var  
    x1,x2,y1,y2,r,vec1,vec2:real;  
  
begin  
    writeln('Введите координаты точки А');  
    readln(x1,y1);  
    writeln('Введите координаты точки В');  
    readln(x2,y2);
```

```
r:=rasst(x1,y1,x2,y2,vec1,vec2);  
  writeln('Вектор АВ=(',vec1:5:2,',',vec2:5:2,')', 'Расстояние  
АВ=', r:5:2);  
  readln;  
end.
```

*Обратите внимание* – функция `rasst` возвращает не один результат, а три, причем один из них (расстояние) передается как результат функции, а переменные `vec1` и `vec2` передаются как *параметры-переменные*.

Анализ приведенной программы дает основания считать функции *более универсальным видом подпрограмм*, чем процедуры, тем более что в современных версиях Pascal разрешен прямой вызов функций (по типу вызова процедуры). В некоторых языках (C/C++ и им подобных) вообще отсутствуют процедуры, и единственным видом подпрограмм являются функции.

# \* Область видимости и время жизни переменной

Все объекты (метки, константы, типы, переменные, процедуры и функции), которые описываются в теле подпрограммы, являются **локальными объектами**, т. е. доступны *только* в пределах этой подпрограммы и недоступны вызывающей программе. Иначе говорят, что они "видимы" в той подпрограмме, в которой описаны.

При запуске любой программы для нее формируется специальная область памяти, которая называется *программным стеком*. Эта область используется для хранения локальных переменных подпрограмм, которые помещаются в стек непосредственно при вызове подпрограммы.

Параметры подпрограммы также помещаются в стек, причем для параметров-значений в стек помещается *само значение*, а для параметров-переменных — *адрес фактического параметра*, в котором хранится значение.

Кроме того, в стек помещается *точка возврата*, т. е. адрес инструкции вызывающей программы, к которой нужно вернуться после окончания работы подпрограммы.

Как только подпрограмма закончит работу, вся память, отведенная для нее в стеке, считается свободной и в дальнейшем может использоваться для данных других подпрограмм. Так обеспечивается *независимость* подпрограмм друг от друга и от основной программы.

Все объекты, описанные в вызывающей программе, называются *глобальными*. Память под глобальные переменные выделяется при компиляции программы. Эта часть памяти называется *сегментом данных*. Иначе ее называют *глобальной памятью*.

Глобальные переменные находятся в сегменте данных от начала до конца выполнения программы, поэтому *ими можно пользоваться и в программе, и во всех подпрограммах*, к которым обращается программа.

- *обмен данными между программой и подпрограммой может производиться не только через параметры, но и через глобальные переменные,*
- *если одно и то же имя определено и в программе, и в подпрограмме, то внутри подпрограммы глобальный объект недоступен, т. е. он как бы экранируется (маскируется) локальным объектом с таким же именем;*
- *если одно и то же имя определено в нескольких подпрограммах одной программы, то в каждой подпрограмме это будут разные локальные объекты, не зависящие друг от друга (если подпрограммы не вложены одна в другую).*

Для пояснения рассмотрим следующую программу:

```
var a,b,c,d: integer;
```

```
procedure prim(x: integer; var y: integer);
```

```
var c: integer;
```

```
begin
```

```
c:=1;d:=1;x:=1; y:=1;
```

```
writeln(x,' ',y,' ',c,' ',d);
```

```
end;
```

```
begin
```

```
a:=0;b:=0;c:=0;d:=0;
```

```
prim(a,b);
```

```
writeln(a,' ',b,' ',c,' ',d);
```

```
end.
```

При выполнении программа напечатает строки:

1111

0101

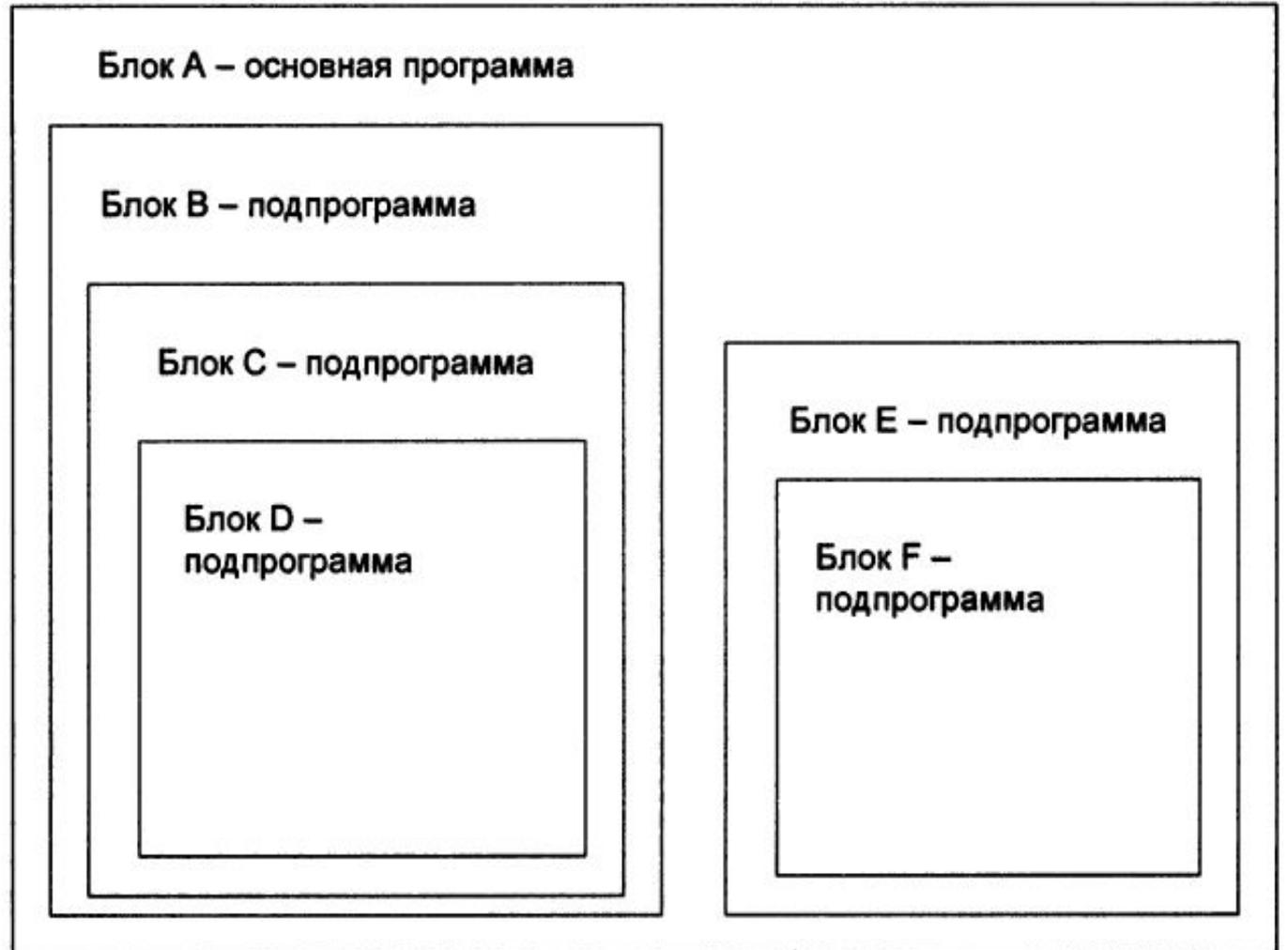
$x$  — формальный параметр-значение, которому соответствует фактический параметр  $a=0$ . В процедуре его значение заменится на 1, после чего результат будет напечатан. На переменной  $a$  это никак не отразится, и после выполнения процедуры  $a$  по-прежнему будет равно нулю,  $y$  — параметр-переменная, поэтому при выполнении процедуры вместо  $y$  действия будут проводиться с переменной  $b$ , которая получит значение 1.  $c$  — локальная переменная, которая маскирует глобальную переменную  $c$  основной программы,  $d$  — глобальная переменная.

# \* Вложенные процедуры и функции

В Pascal допускается *любой уровень вложенности процедур и функций*. Процедура, описанная в основной программе, в свою очередь, может содержать описания внутренних процедур и функций. При этом объекты, описанные в вызывающей процедуре, являются **глобальными** по отношению к вызываемой процедуре.

- имена объектов, описанных в некотором блоке, считаются известными в пределах данного блока, включая и все вложенные блоки;
- имена объектов, описанных в блоке, должны быть уникальны в пределах данного блока и могут совпадать с именами объектов из других блоков;
- если в некотором блоке описан объект, имя которого совпадает с именем объекта, описанного в вышестоящем блоке, то это последнее имя становится недоступным в

Если применить эти правила к схеме, можно сказать, что объекты, описанные в блоке *B*, известны (видимы), кроме самого блока *B*, еще и в блоках *C* и *D*, но невидимы в блоке *A*. Объекты, описанные в блоке *F*, видимы только в пределах этого блока.



# \* Рекомендации по разработке программ

- При разработке программ с большим количеством подпрограмм всегда стремитесь сократить число глобальных переменных до минимума, давайте им осмысленные имена и держите каждую из них под постоянным контролем;
- все рабочие переменные, которые используются в подпрограмме для реализации ее алгоритма, описывайте как локальные; при этом желательно, чтобы имена переменных не совпадали с именами глобальных переменных;
- если планируется использование подпрограммы в нескольких разных программах, в ней вообще не должны использоваться глобальные переменные, т. е. она должна быть полностью *автономной* и должна обмениваться данными с основной программой только через параметры;
- избегайте вложенных процедур, т. к. при их использовании вероятность нечаянно изменить значение переменной резко возрастает.

# \* Домашнее задание

1. Составить опорный конспект лекции по теме «Процедуры и функции» на основе презентации.

2. Программирование на языке Pascal. Рапаков Г. Г., Ржеуцкая С. Ю. СПб.: БХВ-Петербург, 2004, стр. стр. 157-174.

Составить программы с использованием процедур и функций:

- Пользователь вводит натуральное число. Вывести на экран все его простые делители (простое число делится нацело только на 1 и на само себя).
- Пользователь вводит размерность массива и его натуральные элементы. Найти элемент массива в записи которого наибольшее количество нулей.