



ПОДПРОГРАММЫ ЯЗЫКА ВЫСОКОГО УРОВНЯ PASCAL

*Цивилизация развивается за счет
расширения числа важных операций,
которые можно выполнять, не думая о них.
А. Н. Уайтхед*

Содержание

2

- Понятие подпрограммы
- Передача параметров в подпрограмму
- Разработка подпрограмм
- Блочная организация программы

Подпрограммы

3

- *Подпрограмма* – синтаксически обособленная часть программы, решающая отдельную подзадачу.
 - Имеет *параметры*, соответствующие входным и выходным данным подзадачи.
 - Не выполняется сама по себе: всегда *вызывается* из программы или другой подпрограммы.
 - *Процедура* – подпрограмма, не возвращающая значения.
 - *Функция* – подпрограмма, возвращающая одно значение, называемое *результатом* (подобно математической функции).

Виды подпрограмм

4

Спецификация подпрограммы

Заголовок подпрограммы

Объявление констант

Объявление типов данных

Объявление подпрограмм

Объявление переменных

Объявления подпрограммы

Головной блок подпрограммы

Тело подпрограммы

```
{ Нахождение корней КВУР  
a* (x^2)+b*x+c=0.  
x1,x2 - корни КВУР  
N - количество корней (0..2) }  
procedure Eqtn(a,b,c: Real;  
var x1,x2: Real; var N: Integer);  
  
{ Нахождение корней КВУР  
a* (x^2)+b*x+c=0.  
x1,x2 - корни КВУР  
Возвращает количество корней (0..2) }  
function Eqtn(a,b,c: Real;  
var x1,x2: Real): Integer;
```

Спецификация подпрограммы

5

1. Назначение подпрограммы.
 2. Описание формальных параметров и их семантики.
 3. Описание возвращаемого результата и его семантики (для функций).
- Допускается не специфицировать самоочевидные подпрограммы и/или параметры.

Параметры подпрограммы

6

- *Формальные параметры* указываются в заголовке объявления подпрограммы.
 - Символические обозначения, заменяемые при вызове подпрограммы фактическими параметрами.
- *Фактические параметры* указываются в операторе вызова подпрограммы.
 - Конкретные значения или имена конкретных переменных, подставляемые при вызове подпрограммы вместо формальных параметров.

Формальные и фактические параметры

7

```
Program Example;
```

```
{ Нахождение корней КВУР  $a \cdot (x^2) + b \cdot x + c = 0$ .
```

```
x1, x2 - корни КВУР
```

```
N - количество корней (0..2) }
```

```
procedure Eqtn(a, b, c: Real; var x1, x2: Real; var N: Integer);
```

```
...
```

```
var
```

```
a, b, c, x1, x2: Real;
```

```
N: Integer;
```

```
...
```

```
begin
```

```
...
```

```
Eqtn(a, b, c, x1, x2, N);
```

```
...
```

```
end.
```

Формальные параметры

Фактические параметры

Формальные и фактические параметры

8

```
Program Example;
```

```
{ Нахождение корней КВУР  $a \cdot (x^2) + b \cdot x + c = 0$ .
```

```
x1, x2 - корни КВУР
```

```
N - количество корней (0..2) }
```

```
procedure Eqtn(a, b, c: Real; var x1, x2: Real; var N: Integer);
```

```
...
```

```
var
```

```
q, r, s, w1, w2: Real;
```

```
Z: Integer;
```

```
...
```

```
begin
```

```
...
```

```
Eqtn(q, r, s, w1, w2, Z);
```

```
...
```

```
end.
```

Формальные параметры

Фактические параметры

Соответствие параметров

9

Program Example;

{ Нахождение корней КВУР $a \cdot (x^2) + b \cdot x + c = 0$.

x_1, x_2 - корни КВУР

N - количество корней (0..2) }

procedure Eqtn(a, b, c : Real; var x_1, x_2 : Real; var N : Integer);

...

var

q, r, s, w_1, w_2 : Real;

Z : Integer;

...

begin

...

Eqtn(q, r, s, w_1, w_2, Z);

...

end.

- ❑ Позиционное соответствие.
- ❑ Совпадение количества.
- ❑ Совместимость типов.

Объявление подпрограмм

10

- **procedure** <ИД> [(<Список параметров>)] ;
- **function** <ИД> [(<Список параметров>)] :
<Тип результата>;
 - ▣ Тип результата ::= <Простой тип> | <Указатель> | String
- <Список параметров> ::= [var] <ИД> : <Тип> { ; [var] <ИД> : <Тип> }

Пример: объявления и вызовы процедур

11

```
procedure PrintLine;
```

```
begin
```

```
...
```

```
end;
```

```
procedure
```

```
PrintLineChar (Ch:Char) ;
```

```
begin
```

```
...
```

```
end;
```

```
procedure Calculate (A,B:  
Integer; C: Real; var R:  
Real) ;
```

```
begin
```

```
...
```

```
end;
```

```
var
```

```
A, B, a1, a2: Integer;
```

```
C, a3, R, Result: Real;
```

```
Ch: Char;
```

```
...
```

```
PrintLine;
```

```
PrintLineChar ('*');
```

```
Ch:='$'; PrintLineChar (Ch);
```

```
Calculate (A,B,C,R) ;
```

```
Calculate (a1,a2,a3,Result) ;
```

```
Calculate (a2,A,Result,a3) ;
```

Подпрограмма-функция

12

- Тело функции должно содержать как минимум один оператор присваивания вида **Result := <Значение>;**
 - Тип значения должен быть совместим по присваиванию с типом результата функции.
 - Результатом функции будет последнее присвоенное значение.
 - Вместо **Result** можно использовать имя функции.
- Если тело функции не содержит такого оператора присваивания или ни один такой оператор не выполняется, то результат функции *не определен*.

Пример: объявление и вызов функции

13

```
function IsDigit (Ch:Char) :  
Boolean;  
begin  
    Result := Ch in ['0'..'9'];  
end;  
function IsLetter (Ch:Char):  
Boolean;  
begin  
    Result := Ch in ['a'..'z',  
    'A'..'Z'];  
end;  
function DigitsInNum (N: Integer):  
Integer;  
var D: Integer;  
begin  
    D:=0;  
    repeat  
        D := D+1;  
        N := N div 10;  
    until N=0;  
    Result := D;  
end;
```

```
function AlphaNum (Ch: Char) :  
Boolean;  
begin  
    Result := IsDigit (Ch) or  
    IsLetter (Ch);  
end;  
...  
if not AlphaNum (Ch) then  
    WriteLn (Ch, '- спецсимвол.');
```



```
repeat  
    Write ('Введите целое число: ');  
    ReadLn (Num);  
    WriteLn ('В этом числе ',  
    DigitsInNum (Num), ' цифр');  
until Num=0;
```

Предописание подпрограмм

14

- *Предописание* позволяет определять *взаимно рекурсивные* подпрограммы (вызывающие друг друга прямо или косвенно).
- `procedure <Имя>[(<Формальные параметры>)];
forward;`
- `function <Имя>[(<Формальные параметры>)] :
<Тип результата>; forward;`

Пример: взаимно рекурсивные подпрограммы

15

```
Program FlipFlop;

procedure Flip(N: Integer);
forward;

procedure Flop(N: Integer);
begin
  WriteLn('Flop');
  if N > 0 then Flip(N - 1);
end;

procedure Flip;
{ Здесь возможна сокращенное
объявление процедуры. }

begin
  WriteLn('Flip');
  if N > 0 then Flop(N - 1);
end;
```

```
begin
  Flip(3);
end.
```

```
begin
  Flop(3);
end.
```

```
Flip
Flop
Flip
Flop
```

```
Flop
Flip
Flop
Flip
```

Способы передачи параметров

16

- *Передача по значению (call-by-value)* – создание *копии* фактического параметра по тому адресу памяти, который назначен для формального параметра.
 - Все изменения в подпрограмме производятся над копией и не отражаются на фактическом параметре.
 - По значению можно передавать выражения и переменные.
- *Передача по ссылке (call-by-reference)* – назначение формальному параметру адреса памяти фактического параметра.
 - Все изменения в подпрограмме производятся *непосредственно* над фактическим параметром.
 - По ссылке можно передавать только переменные.

Пример: передача по значению

17

```
procedure DoZero (A: Integer);  
begin  
  A:=0;  
end;  
var  
  B: Integer;  
begin  
  B:=1;  
  DoZero (B);  
  WriteLn (' B=' , B); { B=1 }  
end.
```

Пример: передача по ссылке

18

```
procedure DoZero (var A: Integer);  
begin  
  A:=0;  
end;  
var  
  B: Integer;  
begin  
  B:=1;  
  DoZero (B);  
  WriteLn ('B=' , B); { B=0 }  
end.
```

Передача по значению vs по ссылке

19

- Передача по значению
 - технологически надежна (не "портит" фактический параметр)
 - используется, как правило, для передачи параметров "входных данных" в подпрограмму
 - требует дополнительных затрат памяти; не всегда возможна (например, файловые переменные)
- Передача по ссылке
 - не требует дополнительных затрат памяти
 - используется, как правило, для передачи параметров "результатов" в подпрограмму
 - технологически ненадежна ("портит" фактический параметр)

Параметры-константы

20

```
function MakeStr(const Ch: Char; const Len:
Byte): String;
var
  i: Byte;
  S: String;
begin
  S:='';
  for i:=1 to Len do
    S:=S+Ch;
  Result:= S;
end;
```

- Изменение параметра-константы в теле подпрограммы трактуется как синтаксическая ошибка.

Выбор вида формальных параметров

21

- *Параметрами-переменными* следует объявлять только те, посредством которых подпрограмма передает результаты вызывающей программе.
- *Параметрами-значениями* следует объявлять параметры, посредством которых подпрограмма получает данные от вызывающей программы.
- *Параметрами-константами* рекомендуется объявлять параметры, значения которых не изменяются внутри подпрограммы.

Открытые параметры

- *Открытый параметр* позволяет передавать одной и той же подпрограмме массивы различной длины.
- Открытые параметры-массивы доступны по директиве компилятора $\{ \$P+ \}$.
- Соответствующий фактический параметр может быть массивом *произвольной* длины с элементами аналогичного типа.
- Стандартные функции **Low** и **High** выдают минимальное и максимальное значения индекса открытого параметра-массива.

Пример: открытый параметр подпрограммы

23

```
function MaxInVector(V: array of Real): Integer;
var
  i, N: Integer;
  M: Real;
begin
  N := Low(V);
  M := V[N];
  for i:=Low(V) to High(V) do
    if V[i]>M then
      begin
        N:=i;
        M:=V[i];
      end;
  MaxInVector := N;
end;
```

Процедурный тип

24

```
type
  TFunc = function(X: Real): Real;

procedure PrintFunc(a, b, S: Real; f:
  TFunc); far;
{ Печатает таблицу значений функции
  f на отрезке [a;b] с шагом S }
var x: Real;
begin
  x:=a;
  while x<=b do begin
    WriteLn('f(',x:5:5,')=', f(x):5:5);
    x:=x+S;
  end;
end;
```

```
function tg(X: Real): Real;
begin
  tg := sin(X)/cos(X);
end;

function MyF(X: Real): Real;
begin
  MyF := sin(X*X)+cos(X*X);
end;

begin
  PrintFunc(1, 2, 0.01, tg);
  PrintFunc(0, 1, 0.001, MyF);
end.
```

Разработка подпрограмм

25

□ Автономность

- Подпрограмма должна решать *одну* подзадачу и быть как можно более *независимой* от работы других подпрограмм.

□ Детерминированность

- Подпрограмма должна выполнять одни и те же определенные программистом действия независимо от места ее вызова.

□ Самообъяснимость

- Подпрограмма должна быть объяснима, исходя исключительно из ее текста
 - назначение – с помощью спецификации
 - алгоритм – с помощью тела подпрограммы.

Разработка подпрограмм

26

- Приемлемая сложность
 - Подпрограмма должна иметь обозримый размер и сложность.
- Повторное использование
 - Подпрограммы должны быть "кирпичиками", из которых можно построить много "зданий" программ.
- Иерархия
 - Модульная структура программы должна представлять собой иерархию.
 - Связи (вызовы подпрограмм) вида "через уровень", "снизу вверх", "горизонтальные" недопустимы.

Определяющее и использующее вхождение идентификаторов

27

- *Определяющее вхождение идентификатора* — вхождение идентификатора в объявление переменной, метки и др.
- *Использующее вхождение идентификатора* — вхождение идентификатора в оператор.

Определяющее и использующее вхождение идентификаторов

28

Program Example;

```
{ Нахождение корней КВУР  $a \cdot (x^2) + b \cdot x + c = 0$ .  
x1, x2 - корни КВУР, N - количество корней (0..2) }  
procedure Eqtn(a, b, c: Real; var x1, x2: Real; var N: Integer);  
var  
  D: Real;  
begin  
  D := b*b - 4*a*c;  
  if D < 0 then  
    N := 0  
  else  
    ...  
end;  
...  
var  
  a, b, c, x1, x2: Real;  
  N: Integer;  
begin  
  ...  
  Eqtn(a, b, c, x1, x2, N);  
  ...  
end.
```

Определяющее вхождение N

Использующее вхождение N

Определяющее вхождение N

Использующее вхождение N

Область действия декларации

- *Область действия декларации* – это часть текста программы, начинающаяся с данной декларации и завершающаяся концом текущего блока.
- Понятие области действия декларации необходимо для связывания использующих вхождений идентификаторов с правильными определяющими вхождениями данных идентификаторов.

Пример: область действия декларации

30

```
Program Scope;
```

```
var
```

```
  i, A, K, N: Integer;
```

```
procedure P(R: Real; var Z: Integer);
```

```
var
```

```
  i: Integer;
```

```
begin
```

```
  i:=2*Z*K;
```

```
  R:=6.28*Z+i;
```

```
end ;
```

```
begin
```

```
  i:=1;
```

```
  N:=i;
```

```
  A:= 628*N;
```

```
  P(A, N);
```

```
end.
```

Связывание использующих вхождений с определяющим

31

- Каждое использующее вхождение идентификатора связывается с тем определяющим вхождением данного идентификатора, которое имеет *наименьшую* область действия.

Пример: область действия декларации

32

```
Program Scopel;  
var  
  i, A, K, N: Integer;  
  
procedure P(A: Real; N: Integer);  
var  
  i: Integer;  
begin  
  N := i + K;  
  A := 6.28 * N;  
end ;  
  
begin  
  i := 1;  
  N := i;  
  A := 628 * N;  
  P(A, N);  
end.
```

Правила видимости

33

- Идентификатор, определенный в блоке, является *видимым* в данном блоке (может быть использован в операторах и выражениях данного блока).
- Идентификатор, определенный во внешнем блоке, может быть переопределен во внутреннем блоке. В этом случае объект, связанный с внешним определением, становится *невидимым* во внутреннем блоке.

Пример: видимость программных объектов

34

```
Program Scope2;  
var  
  K, i, A, N: Integer;  
  
procedure P(A: Real; N: Integer );  
const  
  K=413;  
var  
  i: Integer;  
begin  
  N:=i*K;  
  A:=6.28*N;  
end ;  
  
begin  
  K:=314;  
  i:=1;  
  N:=i;  
  A:= 628*N;  
  P(A, N);  
end.
```

Локальные и глобальные переменные

35

- Переменная называется *локальной по отношению к данному блоку*, если блок содержит связанное с ней определяющее вхождение, и *глобальной* – в противном случае.

Пример: локальные и глобальные переменные

36

```
Program LocalAndGlobal;  
const  
  N=10;  
type  
  TArr=array[1..N] of Real;  
var  
  A, B, C, Res: Real;  
  
procedure InpData(var A, B, C:  
Real);  
begin  
  ...  
end;  
  
procedure Calculate(A, B, C:  
Real; var Res: Real);  
var  
  i, j, k: Integer;
```

```
function Search(A: TArr; E:Real):  
Integer;  
begin  
  ...  
end;  
begin  
  ...  
end;  
procedure OutData(Res: Real);  
begin  
  ...  
end;  
  
begin  
  InpData(A, B, C);  
  Calculate(A, B, C, Res);  
  OutData(Res);  
end.
```

Глобальные переменные вредны

37

- Использование глобальных переменных снижает читаемость программы и может привести к *побочным эффектам*.
- *Побочный эффект подпрограммы* – выполнение подпрограммой действий, результат которых обнаруживается вне данной подпрограммы, например, изменение значений глобальных по отношению к ней переменных.
- По возможности **следует избегать использования глобальных переменных** (использовать локальные переменные и/или параметры подпрограмм).

Пример: побочный эффект

38

Глобальная
переменная

Побочный
эффект

```
Program SideEffectIsBad;
var
  C: Real;
  i: Integer;
function Deg(A: Real;
N: Integer): Real;
{Вычисляет A^N}
begin
  C:=1;
  for i:=1 to N do
    C:=C*A;
  Result:=C;
end;
```

```
function Sum(N: Integer): Real;
{Вычисляет 1/2+1/4+1/8+...+1/(2^N)}
begin
  C:=0;
  for i:=1 to N do
    C:=C+1/Deg(2,i);
  Result:=C;
end;

begin
  WriteLn('0.875=', Sum(3):5:3);
end.
```

0.875=8.125

Пример: побочный эффект

39

Локальная
переменная

```
Program NoSideEffects;
```

```
function Deg(A: Real;  
N: Integer): Real;  
{Вычисляет A^N}
```

```
var  
(C: Real;  
(i: Integer;
```

```
begin  
  C:=1;  
  for i:=1 to N do  
    C:=C*A;  
  Result:=C;  
end;
```

```
function Sum(N: Integer): Real;  
{Вычисляет 1/2+1/4+1/8+...+1/(2^N)}  
var
```

```
(C: Real;  
(i: Integer;  
begin  
  C:=0;  
  for i:=1 to N do  
    C:=C+1/Deg(2,i);  
  Result:=C;  
end;
```

Побочного
эффекта
НЕТ

```
begin  
  WriteLn('0.875=', Sum(3):5:3);  
end.
```

0.875=0.875

Разработка подпрограмм

40

- Минимальное использование глобальных переменных в подпрограмме (вместо них – локальные переменные, в т.ч. параметры подпрограммы).

```
var
  a,b,c, { коэффициенты КВУР }
  x1,x2: Real; { корни КВУР}
  N: Integer; { количество корней (0..2) }

{ Нахождение корней КВУР a*(x^2)+b*x+c=0 }
procedure Eqtn;
...
```

Нет параметров подпрограммы

```
{ Нахождение корней КВУР  $a*(x^2)+b*x+c=0$ .
a,b,c - коэффициенты КВУР
D - дискриминант КВУР
x1,x2 - корни КВУР
N - количество корней (0..2) }
procedure Eqtn(a,b,c,D:Real, var x1,x2: Real; var N: Integer);
...
```

Ненужный параметр подпрограммы

Заключение

- Подпрограмма – синтаксически обособленная часть программы, решающая отдельную подзадачу. Виды подпрограмм – процедуры и функции.
- Формальные параметры подпрограммы – в ее заголовке. Фактические параметры – в операторе вызова подпрограммы. Установление соответствия параметров – позиционное. Типы фактических параметров должны быть совместимы с типами формальных параметров.
- Способы передачи фактических параметров: по ссылке, по значению.
- Подпрограммы должны быть "кирпичиками", из которых строится "здание" программ.

Заключение

- Текст программы может содержать более одного определяющего вхождения одного и того же идентификатора. Каждое использующее вхождение идентификатора связывается с некоторым определяющим вхождением этого идентификатора в соответствии с областью действия декларации и правилами видимости.
- По отношению к данному блоку переменная может быть локальной или глобальной. Использование глобальных переменных снижает читаемость программы и может привести к побочным эффектам.