Функции и процедуры

Наумов Д.А., доц. каф. КТ

Программирование и алгоритмические языки, 2020

Содержание лекции

Подпрограмма

идентифицированная часть компьютерной программы, содержащая описание определённого набора действий, которая может быть многократно вызвана из разных частей программы.

Назначение подпрограмм:

- выделить целостную подзадачу, имеющую типовое решение;
- сделать программу более понятной и обозримой.

Преимущества подпрограмм:

1. декомпозиция сложной задачи;
2. уменьшение дублирования кода;
3. возможность повторного использования кода;
4. разделение задач между исполнителями или стадиями проекта;
5. сокрытие деталей реализации;
6. упрощение отладки.

Виды подпрограмм в языке *Pascal*: **функции** и **процедуры**.
Синтаксическая форма описания функции:

```
1 function <ИмяФункции>([<СписокФормПарам>]):<ТипВозврЗнач>;
2 [<РазделыОписаний>]
3 begin
4   <Оператор 1>;
5   ...;
6   <Оператор N>;
7 end;
```

- **ИмяФункции** - имя функции, идентификатор;
- **СписокФормПарам** - список формальных параметров с указанием их типа.
- **ТипВозврЗнач** - тип значения, возвращаемого функцией.
- **РазделОписаний** - раздел описаний локальных меток, констант, переменных, типов данных, процедур и функций.

Пример описания функции для вычисления степени с натуральным показателем:

```
1 function CalcPower(x: real; n: integer):real;
2 var
3   i: integer;
4   p: real;
5 begin
6   p := 1;
7   for i := 1 to n do
8       p := p * x;
9   CalcPower := p; (*возвращаем значение*)
10 end;
```

- **x**, **n** - формальные параметры;
- **i**, **p** - локальные переменные;

При обращении к функции происходит:

1. вычисление значений фактических параметров (слева направо);
2. подстановка значений фактических параметров на место формальных параметров;
3. выполнение операторов тела функции;
4. возврат значения функции в основную программу;
5. возврат управления в точку вызова;

Вычисление выражения $z = x^5 + (x+1)^3$:

```
11 var x, z: real;
12 begin
13   x := 1.001;
14   z := CalcPower(x, 5) + CalcPower(x+1, 3);
15   writeln('x=', x:6:4, 'z=', z:6:4);
16 end.
```

Вызов функции должен осуществляться в некотором выражении (иначе "потеряется"возвращаемое значение).

Синтаксическая форма вызова функции:

`<ИмяФункции>(<ФактПар1>,<ФактПар2>...<ФактПарN>)`

- **ИмяФункции** - имя вызываемой функции;
- **ФактПар1..N** - фактические параметры: выражения, совместимые по типу с соответствующими формальными параметрами;

Пример:

```
1 function Tan(x: real):real;
2 begin
3   Tan := Sin(x)/Cos(x);
4 end;
5 begin
6   Tan(Pi/4); //возвращаемое значение "теряется"
7 end.
```

В теле функции должен выполниться хотя бы раз оператор вида *ИмяФункции := Значение* (иначе возвращаемое значение будет неопределено).
Пример: ошибка в функции проверки года на то, что он является високосным.

```
1 function IsLeapYear(aYear: integer):boolean;
2 begin
3   if aYear div 400 = 0 then
4      IsLeapYear := true
5   else if aYear div 100 = 0 then
6      IsLeapYear := false
7   else if aYear div 4 = 0 then
8      IsLeapYear := true;
9 end;
```

Синтаксическая форма описания процедуры

```
1 procedure <ИмяПроцедуры>([<СписокФормПарам>]);
2 [<РазделОписаний>]
3 begin
4   <Оператор 1>;
5   <Оператор 2>;
6   ...;
7   <Оператор N>;
8 end;
```

- **ИмяПроцедуры** - имя функции, идентификатор;
- **СписокФормПарам** - список формальных параметров с указанием их типа.
- **РазделОписаний** - раздел описаний локальных меток, констант, переменных, типов данных, процедур и функций.

Процедура табулирования функции y=x*sin(x)

```
1 procedure Tabulate(Left, Right, Step: real);
2 var x, y: real;
3 begin
4   writeln('x':10,'y:10');
5   if (Right-Left) * Step <= 0 then exit;
6   x := Left;
7   while x <= Right do
8   begin
9     y := x * sin(x);
10    writeln(x:10:2,y:10:4);
11    x := x + Step;
12  end; {while x <= Right}
13 end; {procedure Tabulate}
```

Обращение к процедуре

При обращении к процедуре происходит:

- вычисление значений фактических параметров (слева направо);
- подстановка значений фактических параметров на место формальных параметров;
- выполнение операторов тела процедуры;
- возврат управления в точку вызова;

Обращение к процедуре табулирования:

```
1 var XMin, XMax, XStep: TIndex;
2 begin
3  write('Tabulate from = '); readln(XMin);
4  write('Tabulate to   = '); readln(XMax);
5  write('Tabulate step = '); readln(XStep);
6  Tabulate(XMin, XMax, XStep);
7 end.
```

Вызов процедуры должен осуществляться в отдельном операторе.

Блочный принцип организации программы

Программа, процедура или ункция представляют собой блоки со своими разделами описаний и определений, которые могут быть вложены друг в друга.

Внешний блок

блок, содержащий в своем описании другие блоки (по отношению к этим блокам).

Объекты, описанные во внешнем блоке, и не имеющие других описаний, являются глобальными для внутренних блоков.

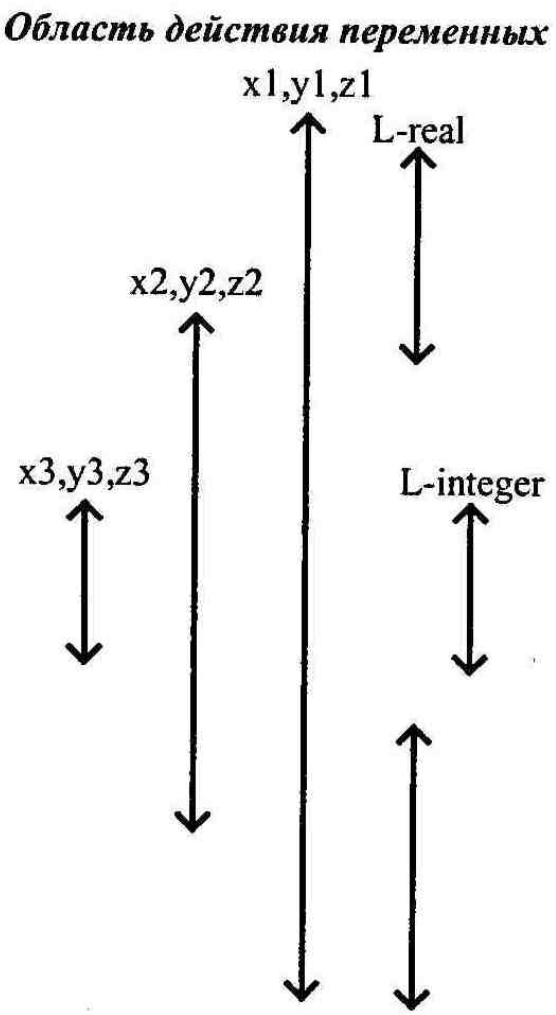
Внутренний блок

блок, содержащийся в другом блоке.

Объекты, описанные внутри блока, являются локальными по отношению к нему, не доступны во внешних блоках, но доступны во внутренних блоках начиная с момента описания.

Области видимости

```
Program Pr1;          Блок1
  Var
    x1,y1,z1:real;
    L:real;
  Procedure Pr2;      Блок2
    Var
      x2,y2,z2:real;
    Procedure Pr3;    Блок3
      Var
        x3,y3,z3:real;
        L:integer;
      Begin{Pr3}
        . . .
      End;{Pr3}
    Begin{Pr2}
      . . .
    End;{Pr2}
  Begin{Pr1
    . . .
  End.{Pr1}
```



Обмен данным с подпрограммой

- обмен данными между подпрограммами осуществляется при помощи формальных и фактических параметров.
- использование глобальных переменных для обмена данных не рекомендуется, так как это усложняет разработку и отладку, и может привести к неконтролируемому изменению данных и контекста выполнения;
- для передачи нестандратных типов данных необходимо их описание в разделе **type**.

```
1 type
2   TComplex = record
3     Re, //вещественная часть
4     Im: TCElem; //мнимая часть
5   end;
6 procedure Add(const A, B: TComplex; var C: TComplex);
```

Процедура сортировки массива

```
1 type
2   TIndex = 1..10;
3   TElem = real;
4   TArray = array[TIndex] of TElem;
5 procedure Sort(var V:TArray; const n:TIndex);
6 var
7   i, j: TIndex;
8   tmp:  TElem;
9 begin
10  for i := 1 to n-1 do
11    for j := i + 1 to n do
12      if V[i] > V[j] then
13      begin
14        tmp := V[i]; V[i] := V[j]; V[j] := tmp;
15      end;
16 end;
```

Обращение к процедуре сортировки

```
11 var vector: TVector;
12 var i: TIndex;
13 begin
14   for i := 1 to 10 do
15     vector[i] := random(100);
16   Sort(vector, 10);
17 end.
```

Вызов процедуры должен осуществляться в отдельном операторе.

Виды формальных параметров подпрограмм

- параметры-значения;
- параметры-переменные;
- параметры-константы;
- параметры процедурного (функционального) типа.

```
1 type
2   TArray = array[1..100] of real;
3   TFunction = function(x: real):real;
4  //x - параметр-значение
5 function Sqr(x: real): real;
6  //v - параметр-переменная
7 procedure Sort(var v: TArray; n: integer);
8  //m - параметр-константа
9 function Find(x: real; const m: TArray; n: integer): real;
10 //f - параметр функционального типа
11 function Integral(a, b: real; f: TFunction): real;
```

Параметры-значения

Параметры-значения

формальные параметры, описанные в загловоке подпрограммы без использования ключевых слов var или const, и не являющиеся параметрами процедурного типа.

- для чего нужны: передача входных данных в подпрограмму;
- как передается: по значению (в стеке создается копия значения актического параметра), которая после завершения подпрограммы уничтожается;
- может ли изменяться внутри подпрограммы: да;
- влияет ли изменение формального параметра на фактический: нет;
- что может являться актическим параметром: выражение, совместимое по присваиванию с типом формального параметра.

Параметры-значения

```
1 function IsIdentifier(s: string): boolean;
2 const
3   LETTERS = ['A'..'Z', 'a'..'z'];
4   DIGITS = ['0'..'9'];
5 begin
6   if not (s[1] in LETTERS + ['_']) then begin
7     IsIdentifier := false;
8     exit; //выход из подпрограммы
9   end else
10   for i: integer := 2 To length(s) do
11     if not (s[i] In LETTERS + DIGITS + ['_']) then begin
12       IsIdentifier := false;
13       exit; //выход из подпрограммы
14     end;
15   IsIdentifier := true;
16 end.
```

Параметры-переменные

Параметры-переменные

это формальные параметры, описанные в загловоке подпрограммы с использованием ключевого слова **var**.

- для чего нужны: передача входных данных в подпрограмму и выходных данных в программу;
- как передается: по адресу фактического параметра;
- может ли изменяться внутри подпрограммы: да;
- влияет ли изменение формального параметра на фактический: да;
- что может являться фактическим параметром: L-value (выражение, указывающее на переменную (область памяти для хранения изменяемого значения) соответствующего типа).

Вопрос: использовать ли параметры-переменные для функции?

Параметры-переменные

```
//do: получение частного комплексных чисел
//in: A - делимое, B - делитель
//out: C - частное A / B
1 procedure Div(const A, B: TComplex; var C: TComplex;
    var ErrorCode: integer);
2  var d: TCElem;
3  begin
4    d := B.Re * B.Re + B.Im * B.Im;
5    if abs(d) < 1e-10 then begin
7      ErrorCode := -1;
8      exit;
9    end;
10   ErrorCode := 0;
11   C.Re := (A.Re * B.Re + A.Im * B.Im) / d;
12   C.Im := (A.Im * B.Re - A.Re * B.Im) / d;
13 end;
```

Параметры-константы

Параметры-константы

формальные параметры, описанные в загловоке подпрограммы с использованием ключевого слова **const**.

- для чего нужны: передача входных данных в подпрограмму с контролем из неизменности;
- как передается: по адресу фактического параметра;
- может ли изменяться внутри подпрограммы: нет (попытка приведет к ошибке компиляции);
- влияет ли изменение формального параметра на актический: нет;
- что может являться фактическим параметром: выражение, совместимое по присваиванию формальным параметром.

Параметры-константы

```
 1 type
 2  TIndex = 1..Nmax;
 3  TElem = real;
 4  TVector = array[TIndex] Of TElem;
 5 function SearchElement(const V:TVector;
     const Element:TElem; StartIndex, EndIndex: TIndex):TIndex;
 6 var
 7   i: TIndex;
 8 begin
 9 for i := StartIndex to EndIndex do
10   if abs(V[i] - Element) <= 1e-8 then begin
11     SearchElement := i;
12     exit
13   end;
14   SearchElement := -1;
15 end;
```

Процедурный тип

описывается в разделе **type** подобно описанию процедур и функций без указания их имени.

```
1 type
2  {$F+}
3  TFunction = function(x:real):real;
4  TProcedure = procedure(var x:real);
```

Описание процедурного типа позволяет использовать в качестве фактических параметров процедуры и функции.

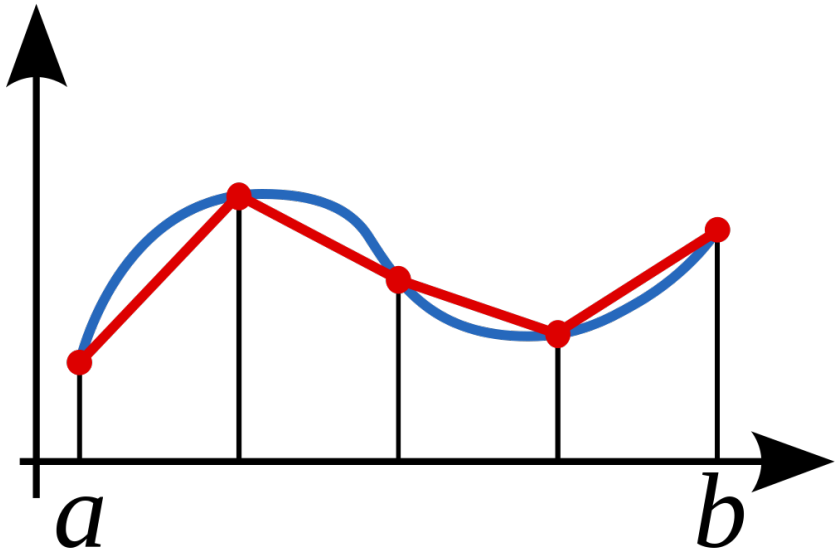
- подпрограмма-фактический параметр должна быть в области видимости (не локальная, не стандартная подпрограмма);
- должны совпадать сигнатуры (тип и количество формальных параметров и возвращаемое значение) у фактического параметра-подпрограммы и формального параметра;
- в старых компиляторах должна использоваться директива комплиятора far.

Пример использования параметра процедурного типа

Метод трапеций

Метод трапеций — метод численного интегрирования функции одной переменной, заключающийся в замене на каждом элементарном отрезке подынтегральной функции на многочлен первой степени, то есть линейную функцию.

Схема для численного интегрирования



Метод трапеций

Формула для интегрирования

$$\int_a^b f(x)\,dx = h\left(\frac{f_0 + f_n}{2} + \sum_{i=1}^{n-1} f_i\right) + E_n(f)$$

```
1 type TFunction = function(x:real):real;
2 function FSin(x: real):real;
3 begin
4   FSin := sin(x);
5 end;
6 function FCos(x: real):real;
7 begin
8   FCos := cos(x);
9 end;
```

```
//функция численного интегрирования по формуле трапеций для
//  заданного числа шагов
1 function CalcIntegralStep(F: TFunction; LimitA, LimitB,
     Step: real): real;
2 var
3   s, x: real;
4   i, n: integer;
4 begin
//получаем количество разбиений
5   n := round((LimitB - LimitA) / Step);
6   s := 0;
7   for i := 1 to n - 1 do begin
8     x := LimitA + Step * i;
//обращаемся к параметру F функционального типа
9     s := s + F(x);
10  end;
11  CalcIntegralStep := Step*((F(LimitA)+F(LimitB))/2+s);
12 end;
```

```
//функция численного интегрирования по формуле трапеций
//  с заданной точностью
1 function CalcIntegral(F:TFunction;LimitA,LimitB:real):real;
2 var
3  i: integer;
4  prev, curr, step: real;
5 begin
6  step := (LimitB - LimitA) / FIRST_STEP;
7  i := 0;
8  curr := CalcIntegralStep(F, LimitA, LimitB, step);
9  repeat
10   i := i + 1;
11   step := step / 2; //уменьшаем шаг вдвое
12   prev := curr;
13    curr := CalcIntegralStep(F, LimitA, LimitB, step);
14  until (abs(prev - curr) < EPS) or (i > LIMIT);
15  CalcIntegral := curr;
16 end;
```

Пример сортировки массива

```
1 //функция сравнения элементов массива
2  TCompareFunction = function(FirstValue,
      SecondValue: TElem):boolean;
3 //функция сравнения элементов для сортировки по возрастанию
4 function SortAscending(FirstValue,
    SecondValue: TElem):boolean;
5 begin
6   SortAscending := FirstValue <= SecondValue;
7 end;

8 //функция сравнения элементов для сортировки по убыванию
9 function SortDescending(FirstValue,
    SecondValue: TElem):boolean;
10 begin
11  SortDescending := FirstValue >= SecondValue;
12 end;
```

Пример сортировки массива

```
//процедура сортировки
1 procedure Sort(var aVector: TVector; aCount: TIndex;
     aSortFunction: TCompareFunction);
2 var
3    i, k : TIndex; tmp: TElem;
4 begin
5   for i:=1 to NMax-1 do
6     for k:=i downto 1 do
7       //если значения расположены не в требуемом порядке
8        if not aSortFunction(aVector[k], aVector[k+1]) then
9       begin
10          tmp := aVector[k];
11          aVector[k] := aVector[k + 1];
12          aVector[k + 1] := tmp;
13        end;
14 end;
```

Пример сортировки массива

```
//сортируем по возрастанию, передавая в качестве параметра
//функцию SortAscending

Sort(MyVector, NMax, SortAscending);

//сортируем по убыванию, передавая в качестве параметра
//функцию SortDescending

Sort(MyVector, NMax, SortDescending);
```