

Функции и процедуры

Наумов Д.А., доц. каф. КТ, ИТГД

Алгоритмические языки и программирование, 2019

Содержание лекции

- 1 Подпрограммы: процедуры и функции
 - Подпрограммы
 - Описание функций
 - Описание процедуры
- 2 Варианты заданий

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     [<РазделОписаний>]  
3 begin  
4     <Оператор1>;  
5     <Оператор2>;  
6     ...;  
7     <Оператор3>;  
8 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);

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

- в теле функции должен выполняться хотя бы раз оператор вида *ИмяФункции := Значение* (иначе возвращаемое значение будет неопределено).

```
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     <Оператор1>;  
5     <Оператор2>;  
6     ...;  
7     <Оператор3>;  
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.

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).

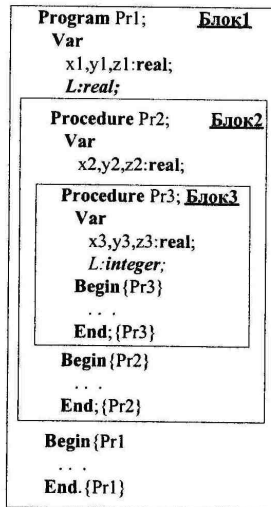
Объекты, описанные во внешнем блоке, и не имеющие других описаний, являются глобальными для внутренних блоков.

Внутренний блок

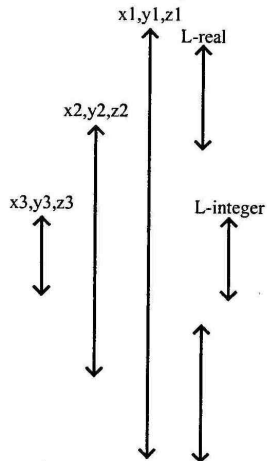
блок, содержащийся в другом блоке.

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



Область действия переменных



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 нестандартных 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;
```

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;

Сортировка массива:

```
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 заголовке 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 фактического параметра), которая после завершения подпрограммы уничтожается;
- может ли изменяться внутри подпрограммы: да;
- влияет ли изменение формального параметра на фактический: нет;
- что может являться фактическим 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 заголовке 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
6       ErrorCode := -1;
7       exit;
8     end;
9     ErrorCode := 0;
10    C.Re := (A.Re * B.Re + A.Im * B.Im) / d;
11    C.Im := (A.Im * B.Re - A.Re * B.Im) / d;
12  end;
```

Параметры-константы

Параметры-константы

формальные параметры, описанные в заголовке 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 фактический: 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;
6   const Element:TElem; StartIndex, EndIndex: TIndex):TIndex;
7 var
8   i: TIndex;
9 begin
10  for i := StartIndex to EndIndex do
11    if abs(V[i] - Element) <= 1e-8 then begin
12      SearchElement := i;
13      exit
14    end;
15  SearchElement := -1;
16 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.

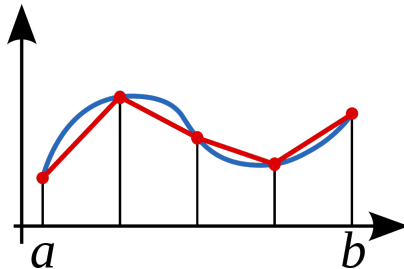
- подпрограмма-фактический параметр должна быть в области видимости (не локальная, не стандартная подпрограмма);
- должны совпадать сигнатуры (тип и количество формальных параметров и возвращаемое значение) у фактического параметра-подпрограммы и формального параметра;
- в старых компиляторах должна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

Варианты заданий

Вариант 1. Дана целочисленная прямоугольная матрица.

Определить:

- 1 количество строк, не содержащих ни одного нулевого элемента (оформить в виде функции);
- 2 максимальное из чисел, встречающихся в заданной матрице более одного раза (оформить в виде процедуры).

Вариант 2. Дана целочисленная прямоугольная матрица.

Определить:

- 1 количество столбцов, не содержащих ни одного нулевого элемента (оформить в виде функции);
- 2 Характеристикой строки целочисленной матрицы назовем сумму ее положительных четных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с ростом характеристик (оформить в виде процедуры).

Варианты заданий

Вариант 3. Дана целочисленная прямоугольная матрица.

Определить:

- 1 количество столбцов, содержащих хотя бы один нулевой элемент (оформить в виде функции);
- 2 номер строки, в которой находится самая длинная серия одинаковых элементов (оформить в виде процедуры).

Вариант 4. Дана целочисленная квадратная матрица. Определить:

- 1 произведение элементов в тех строках, которые не содержат отрицательных элементов (оформить в виде функции);
- 2 максимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы (оформить в виде процедуры).

Варианты заданий

Вариант 5. Дана целочисленная квадратная матрица. Определить:

- 1 сумму элементов в тех столбцах, которые не содержат отрицательных элементов (оформить в виде функции);
- 2 минимум среди сумм модулей элементов диагоналей, параллельных побочной диагонали матрицы (оформить в виде процедуры).

Вариант 6. Дана целочисленная квадратная матрица. Определить:

- 1 такие k , что k -я строка матрицы совпадает с k -м столбцом (оформить в виде процедуры);
- 2 найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент (оформить в виде функции).

Варианты заданий

Вариант 7. Дана целочисленная квадратная матрица. Определить:

- 1 сумму элементов в тех столбцах, которые не содержат отрицательных элементов (оформить в виде функции);
- 2 минимум среди сумм модулей элементов диагоналей, параллельных побочной диагонали матрицы (оформить в виде процедуры).

Вариант 8. Дана целочисленная прямоугольная матрица.

Характеристикой столбца целочисленной матрицы назовем сумму модулей его отрицательных нечетных элементов.

- 1 переставляя столбцы заданной матрицы, расположить их в соответствии с ростом характеристик (оформить в виде процедуры);
- 2 Найти сумму элементов в тех столбцах, которые содержат хотя бы один отрицательный элемент (оформить в виде функции).

Варианты заданий

Вариант 9. Дана целочисленная квадратная матрица. Определить:

- 1 сумму элементов в тех столбцах, которые не содержат отрицательных элементов (оформить в виде функции);
- 2 минимум среди сумм модулей элементов диагоналей, параллельных побочной диагонали матрицы (оформить в виде процедуры).

Вариант 10.

- 1 Коэффициенты системы линейных уравнений заданы в виде прямоугольной матрицы. С помощью допустимых преобразований привести систему к треугольному виду (оформить в виде процедуры);
- 2 Найти количество строк, среднее арифметическое элементов которых меньше заданной величины (оформить в виде функции).