


# Подпрограммы



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

А.Н. Уайтхед

Компьютерные науки © М.Л. Цымблер

---

---

---

---

---

---

---

---

# Содержание

- Понятие подпрограммы
- Способы передачи фактических параметров
- Виды формальных параметров
- Процедурный тип
- Разработка подпрограмм

Подпрограммы © М.Л. Цымблер 2

---

---

---

---

---

---

---

---

# Подпрограммы

- *Подпрограмма* – синтаксически обособленная часть программы, решающая отдельную подзадачу.
- Может иметь *параметры*, соответствующие входным и выходным данным подзадачи.
- Может выдавать *результат*, соответствующий значению, выдаваемому подзадачей как математической функцией.
- Никогда не выполняется сама по себе: всегда *вызывается* прямо или косвенно из основной программы или из другой подпрограммы.

Подпрограммы © М.Л. Цымблер 3

---

---

---

---

---

---

---

---

## Виды подпрограмм

- *Процедура* – подпрограмма, не возвращающая значения.
- *Функция* – подпрограмма, возвращающая одно значение, называемое *результатом*.
- *Рекурсивная подпрограмма* – подпрограмма, вызывающая себя (прямо или косвенно).

Подпрограммы

© М.Л. Цымблер

4

---

---

---

---

---

---

---

---

## Объявление подпрограмм

- **procedure** <Имя>[(<Формальные параметры>)];  
{ <Спецификация> }
- **function** <Имя>[(<Формальные параметры>)] :  
<Тип результата>;  
{ <Спецификация> }
- <Формальные параметры> ::= <Простой тип> | <Тип "указатель"> | String  
{; [var]<Имя>:<Тип>  
; [var]<Имя>:<Тип> }
- *Формальные параметры* – символическое обозначение тех параметров подпрограммы, которые будут заменены конкретными значениями или ссылками на конкретные переменные (*фактические параметры*) в момент вызова данной подпрограммы.

Подпрограммы

© М.Л. Цымблер

5

---

---

---

---

---

---

---

---

## Спецификация подпрограммы

- Подпрограмма должна иметь *спецификацию* – комментарий до/после заголовка, содержащий:
  1. Назначение подпрограммы.
  2. Описание формальных параметров и их семантики.
  3. Описание возвращаемого результата и его семантики (для функций).
  4. Описание возможных побочных эффектов.

Подпрограммы

© М.Л. Цымблер

6

---

---

---

---

---

---

---

---

## Вызов подпрограммы

- <Имя подпрограммы>[(*Фактические* параметры)];
- *Фактические параметры* – конкретные значения или имена конкретных переменных, соответствующие формальным параметрам.
- Вызов процедуры – самостоятельный оператор.
- Вызов функции – только как операнд некоторого выражения. В ходе вычисления данного выражения происходит вызов и выполнение функции, и значением операнда становится значение, возвращаемое функцией.

Подпрограммы

© М.Л. Цымблер

7

---

---

---

---

---

---

---

---

## Установление соответствия параметров

- Соответствие формальных и фактических параметров *позиционное* (первому слева формальному параметру ставится в соответствие первый слева фактический параметр, второму слева формальному – второй слева фактический и т.д.).
- Количество формальных и фактических параметров подпрограммы должны быть равны.
- Типы фактических параметров должны быть совместимы с типами формальных параметров.

Подпрограммы

© М.Л. Цымблер

8

---

---

---

---

---

---

---

---

## Пример: объявления и вызовы процедур

**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);

Подпрограммы

© М.Л. Цымблер

9

---

---

---

---

---

---

---

---

## Тело подпрограммы-функции

- Тело функции должно содержать как минимум один оператор присваивания, в котором функции присваивается некоторое значение.  
 <Имя функции> := <Значение>;  
 { тип значения совместим по присваиванию с типом результата функции }  
 Результатом функции будет последнее присвоенное значение.
- 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

```

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

- *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**

```

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 сокращенная декларация }
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 фактических параметров**

- **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.

---

---

---

---

---

---

---

---

---

---

**Передача по значению vs по ссылке**

- **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)

---

---

---

---

---

---

---

---

---

---

### Виды формальных параметров

Вид	Ключевое слово	Передача фактического параметра	Примечание
Параметр-значение	[нет]	По значению	Локальная переменная подпрограммы. Получает начальное значение от соответствующего фактического параметра в момент вызова подпрограммы. Изменения формального параметра-значения НЕ отражаются на фактическом параметре.
Параметр-переменная	var	По ссылке	Локальная переменная подпрограммы. Синоним (другое имя) фактического параметра. Изменения формального параметра-переменной отражаются на фактическом параметре.
Параметр-константа	const	По ссылке	Локальная типизированная константа подпрограммы. Получает начальное значение от соответствующего фактического параметра в момент вызова подпрограммы, которое впоследствии НЕ может быть изменено.

---

---

---

---

---

---

---

---

---

---

### Пример: процедура с параметром-значением

```

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

---

---

---

---

---

---

---

---

---

---

### Пример: процедура с параметром-переменной

```

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

---

---

---

---

---

---

---

---

---

---

Пример: функция с параметром-константой

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

---

---

---

---

---

---

---

---

Открытые параметры (Турбо Паскаль)

- *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: функция с открытым параметром-массивом

```
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;
```

---

---

---

---

---

---

---

---

### Выбор вида формальных параметров

- *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.

---

---

---

---

---

---

---

---

### Процедурный тип

- *Процедурные типы* позволяют передавать процедуры и функции в качестве фактических параметров подпрограммы.
- В подпрограмме, передаваемой в качестве фактического параметра, после заголовка должна быть указана директива **far**.

---

---

---

---

---

---

---

---

### Пример: использование процедурного типа

```

type
  TFunc = function(X: Real): Real;
function tg(X: Real): Real; far;
begin
  tg := sin(X)/cos(X);
end;
function ctg(X: Real): Real; far;
begin
  ctg := cos(X)/sin(X);
end;

procedure PrintFunc(Start, Stop,
  Step: Real; f: TFunc);
var
  x: Real;
begin
  x := Start;
  repeat
    WriteLn(x:5:5, ' ', f(x));
    x := x + Step;
  until x>=Stop;
end;
...
PrintFunc(1, 2, 0.01, tg(x));
PrintFunc(1, 2, 0.01, ctg(x));
    
```

---

---

---

---

---

---

---

---



### Разработка подпрограмм (1)

- Подпрограмма должна решать *одну* подзадачу и быть как можно более *независимой* от (работы) других подпрограмм.
- Подпрограмма должна выполнять одни и те же определенные программистом действия независимо от места ее вызова (быть *детерминированной*).
- Подпрограмма должна быть объяснима, исходя исключительно из ее текста (назначение – с помощью спецификации, алгоритм – с помощью тела подпрограммы).

---

---

---

---

---

---

---

---

### Разработка подпрограмм

- Использование глобальных переменных в подпрограмме должно быть сведено к минимуму ("побочный эффект"). Вместо них следует использовать локальные переменные и параметры подпрограммы.
- Модульная структура программы должна представлять собой *иерархию*. Связи (вызовы подпрограмм) вида "через уровень", "снизу вверх", "горизонтальные" недопустимы.
- Подпрограммы должны быть "кирпичиками", из которых строится "здание" программ.

---

---

---

---

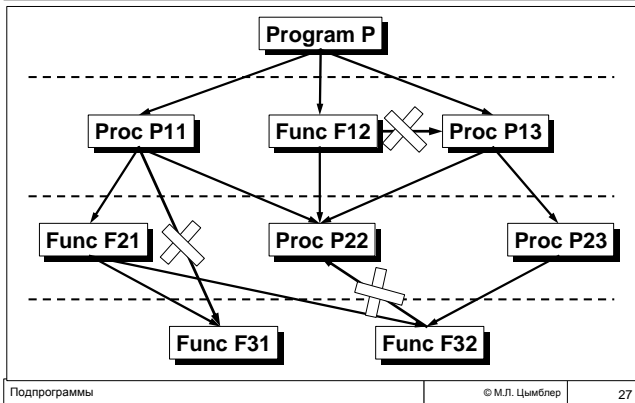
---

---

---

---

### Иерархия подпрограмм




---

---

---

---

---

---

---

---