# Subprogramas

Subprogramas são partes de um programa que contém um cabeçalho, uma seção de definição e declaração de dados e uma seção de comandos.

Os subprogramas são definidos na seção de definição e declaração de dados, e podem ser de dois tipos:

- Procedimentos
- Funções

A diferença essencial entre *funções e procedimentos* é o fato de que as *funções* retornam valores, enquanto os procedimentos não. O valor retornado por uma função é qualquer um dos tipos primitivos **char**, **integer**, **boolean**, **real** ou **string**.

A ativação de um subprograma é feita através de uma *chamada* ao subprograma. Quando um subprograma é *chamado* uma sequência de comandos definida na *seção de comandos* do subprograma é executada, após o qual a execução do programa retorna para instrução seguinte à chamada do subprograma. Um subprograma é chamado através do nome que o define.

Os subprogramas podem ser embutidos; isto é, um subprograma pode ser definido dentro do bloco de declarações de um outro subprograma. Um subprogama embutido pode ser chamado somente pelo subprograma que o contém, sendo visível somente para o subprograma que o contém.

A chamada a um procedimento é reconhecida pelo compilador como um comando, enquanto que uma chamada a uma função é reconhecida como uma expressão.

# Procedimentos e Funções

A declaração de procedimentos e funções difere apenas no cabeçalho.

O cabeçalho de um procedimento segue a seguinte sintaxe:

Sintaxe

```
Procedure NomeProcedimento ;
```

Onde NomeProcedimento identifica o procedimento

O cabeçalho de uma função segue a seguinte sintaxe:

Sintaxe

```
Function NomeFunção : tipo ;
```

Onde:

- NomeFunção identifica a função
- tipo define o tipo do dado retornado pela função, que pode ser um dos tipos primitivos **char**, **integer, boolean**, **real** ou **string**.

A seção de definição e declaração de dados segue o cabeçalho do subprograma, e é o local onde são definidas as constantes e tipos passíveis de uso. Também nessa seção são declaradas as variáveis locais ao subprograma, e definidas as funções e procedimentos que podem vir a serem utilizados pelo subprograma.

A seção de comandos segue a seção de definição e declaração de dados. É iniciada com a palavra reservada **Begin** e terminada com a palavra reservada **End**, seguida de um ponto e vírgula. Entre as palavras **Begin** e **End** são colocados os comandos da função.

As funções retornam dados através de uma atribuição ao identificador da função de um valor a ser retornado pela função, em alguma parte da seção de comandos da função.

Funções e procedimentos podem receber parâmetros, e podem ou não serem recursivos.

## Parâmetros

Um subprograma pode receber parâmetros. A definição dos parâmetros passados a um subprograma deve ser especificada no cabeçalho do subprograma, dentro de parênteses.

Os parâmetros podem ter qualquer um dos tipos predefinidos da linguagem Pascal ( dentre os tipos primitivos implementados no compilador ) ou ainda um tipo que pode ser um dentre os definidos pelo usuário.

A sintaxe do cabeçalho de uma função contendo *n* parâmetros é dada, genericamente, por::

```
  Function identificador( parâmetro1: tipo ; parâmetro2: tipo ; ... ;
parâmetron : tipo ) : tipo;
```

A passagem de parâmetros para a função pode ser de dois tipos, a saber:

- Passagem por valor
- Passagem por referência

No primeiro caso o parâmetro assume o valor passado como argumento pela rotina de chamada, e no segundo caso o parâmetro assume o endereço da variável passada como argumento pela rotina de chamada.

A passagem por referência é diferenciada da passagem por valor pela presença da palavra reservada **var** antes do nome identificador do parâmetro.

Exemplo. Dado o seguinte procedimento:

```
  Procedure exemplo( var parametroPassadoPorReferencia : integer );
```

Esse procedimento poderia ser chamado através de um comando tal como:

```
  exemplo ( x ) ;
```

onde x é uma variável ( ou constante ) do tipo inteiro.

## Funções Pré-Definidas

O compilador implementa as seguintes funções:

- Abs
- ArcTan
- Chr
- Concat
- Copy
- Cos
- Eof
- Eoln
- Exp
- Frac
- Int
- Keypressed
- Length
- Ln
- Odd
- Ord
- Pos
- Pred
- Random
- Readkey
- Round
- Sin
- Sqr
- Sqrt
- Succ
- Trunc
- Upcase

## Regras de Escopo

As regras de escopo, definidas para um programa escrito na linguagem Pascal**,** são as seguintes**:**

- Um identificador definido na *seção de definição e declaração de dados* do programa principal é acessível por qualquer subprograma;

- Um identificador definido na *seção de definição e declaração de dados* de um subprograma é acessível na *seção de comandos* do subprograma na qual foi definido e também na *seção de comandos* de todos subprogramas declarados na sua *seção de definição e declaração de dados*, a menos que esse identificador seja redefinido no subprograma de mais baixo nível na escala hierárquica..

- Os identificadores definidos em um subprograma não existem nem antes nem depois da chamada àquele subprograma e, por isso, não podem ser referenciados em tais momentos.

## Funções Recursivas

Uma função pode chamar a si mesma de dentro de sua própria *seção de comandos*. Quando isto é feito, a função é denominada *função recursiva*.

O uso de *funções recursivas* consegue fornecer soluções elegantes para certos tipos de programas, como mostrado no exemplo abaixo, que calcula, para um número inteiro n, seu fatorial:

```
function fatorial (n :integer ) : integer ;
 begin
  if n > 1 then
    fatorial :=  n * fatorial (n-1)
 else
    fatorial := 1;
 end;
```

## abs

Retorna o valor absoluto de um argumento numérico.

Sintaxe

```
function abs ( x : < integer, real > ): < integer, real > ;
```

Exemplo

```
Program PascalZIM;
 var
    r: Real;
    i: Integer;
 begin
     r := abs( -2.3 );     // r recebe 2.3
     i := abs( -157 );     // i recebe 157
 end.
```

## chr

Recebe como parâmetro um inteiro e retorna o caracter ASC II correspondente ao código identificado com esse inteiro.

Sintaxe

```
function chr( x: integer ): char;
```

Exemplo

```
Program PascalZIM;
 var
  i: integer;
 begin
   for i := 32 to 126 do
    write( chr(I) );
 end.
```

# copy

Retorna parte de uma cadeia de caracteres.

Sintaxe

```
copy( cadeia, posInicio, quantidade ) : string ;
```

Onde:

- cadeia é uma expressão do tipo **string**.
- posInicio é uma expressão do tipo **integer.**
- quantidade é uma expressão do tipo **integer**.

Funcionamento

- Retorna uma subcadeia de cadeia, que começa na posição dada por posInicio. Quantidade denota a quantidade de caracteres que serão retornados a partir da posição informada.
- O primeiro caractere da cadeia está armazenado na posição 1
- Se quantidade for menor ou igual a zero, será retornada uma cadeia vazia.
- Se posInicio for maior que o tamanho da cadeia, será retornada uma cadeia vazia.
- Se posInicio for menor ou igual a zero, será assumido que posInicio correponde ao inicio da cadeia.
- Se a soma de posInicio e quantidade for maior que o tamanho da cadeia, retorna a subcadeia de cadeia que começa em posInicio.

Exemplo

- copy( 'abcdef', 3, 4 ) produz como resultado a cadeia 'cdef'
- copy( 'abcdef', 3, -4 ) produz como resultado a cadeia vazia
- copy( 'abcdef', 30, 4 ) produz como resultado a cadeia vazia
- copy( 'abcdef', -3, 4 ) produz como resultado a cadeia 'abcd'
- copy( 'abcdef', 3, 20 ) produz como resultado a cadeia 'cdef'

Exemplo

```
Program PascalZIM;
 var
    cadeia: string ;
 begin
     cadeia := 'abcdef' ;
     writeln( copy(cadeia, 3, 4) ) ; // Exibe cdef
     writeln( copy(cadeia, -3, 4) ) ; // Exibe abcd
     writeln( copy(cadeia, 30, 4) ) ; // Exibe cadeia vazia
     writeln( copy(cadeia, 4, -2) ) ; // Exibe cadeia vazia
 end.
```

## int

Retorna a parte inteira de um valor numérico.

Sintaxe

```
int( valor ): real ;
```

Onde:

- valor é uma expressão do tipo integer ou real

Exemplo

```
Program PascalZIM;
  begin
    writeln( int(12.34) ) ;   // Mostra 12.00
    writeln( frac(12.34) ) ;  // Mostra 0.34

    writeln( int(12) ) ;   // Mostra 12.00
    writeln( frac(12) ) ;  // Mostra 0.00
  end.
```

## length

Retorna o comprimento de uma cadeia de caracteres.

Sintaxe

```
length( expressãoLiteral ): integer ;
```

Onde expressãoLiteral é uma cadeia de caracteres ou uma expressão envolvendo a concatenação de várias cadeias.

Exemplo

```
Program PascalZIM;
  var
    s: string;
  begin
   write( 'Digite uma cadeia: ' );
   readln( s );
   writeln( ' O comprimento da cadeia lida = ', length( s ));
  end.
```

## ord

Recebe como parâmetro um caractere e retorna o inteiro correspondente ao código ASC II referente ao caracter.

Sintaxe

```
function Ord ( X : char ): integer ;
```

Exemplo

```
Program PascalZIM;
begin
   writeln( 'O codigo ASCII para "c" = ', ord('c'), ' decimal' );
end.
```

## pos

Retorna a posição de uma subcadeia dentro de uma cadeia de caracteres.

Sintaxe

```
pos( subcadeia , cadeia ): integer ;
```

Onde:

- subcadeia é a cadeia que será utilizada na busca
- cadeia é a cadeia onde será procurada a subcadeia

Funcionamento

- Se a subcadeia não for encontrada na cadeia, a função pos retorna zero.
- A posição do primeiro caractere da cadeia é um.

Exemplo

pos( 'zim', 'Pascalzim' ) produz como resultado 7

pos( 'zIm', 'Pascalzim' ) produz como resultado 0

Exemplo

```
Program PascalZIM;
 begin
    writeln( pos('zim', 'Pascalzim') ); // Mostra 7
    writeln( pos('zIm', 'Pascalzim') ); // Mostra 0
 end.
```

## random

Recebe como parâmetro um inteiro *x* e retorna um número *n* no intervalo 0 <= *n* < *x*.

Sintaxe

```
random( x ): integer;
```

Exemplo

```
Program PascalZIM ;
```

```
var
   i: integer ;
begin
    randomize;
    repeat

      i:= i + 1;
      writeln ( random(1000) );

    until i>10 ;
 end.
```

## readkey

Solicita a leitura de um caracter do teclado. Pode ser utilizado como um comando ou como uma função.

Sintaxe

```
readkey ;
```

Exemplo

```
Program PascalZIM ;
begin
   writeln( 'O programa vai terminar...' );
   readkey;
 end.
```

Como função, sua sintaxe é:

```
readkey: integer;
```

Exemplo

```
Program PascalZIM ;
 var
   umCaractere: char ;
 begin
    writeln( 'Digite um caracter:' );
    umCaractere:= readkey;
    writeln( 'Você digitou: ', umCaractere );
 end.
```

## round

Arredonda um valor real em um valor inteiro.

Sintaxe

```
function Round ( X: Real ): integer;
```

Exemplo

```
Program PascalZIM ;
   begin
    writeln( 1.4, ' é arredondado para ', round( 1.4 ) ) ;
    writeln( 1.5, ' é arredondado para ', round( 1.5 ) ) ;
    writeln( -1.4, ' é arredondado para ', round( -1.4 ) ) ;
    writeln( -1.5, ' é arredondado para ', round( -1.5 ) ) ;
  end.
```

## sqr

Retorna o quadrado do argumento.

Sintaxe

```
function sqr ( x : < integer, real > ): < integer, real > ;
```

Exemplo

```
Program PascalZIM ;
  begin
    writeln( 'O quadrado de 5 = ', sqr(5) ) ;
    writeln( 'A raiz quadrada de 2 = ', sqrt(2.0) ) ;
  end.
```

## sqrt

Retorna a raiz quadrada do argumento.

Sintaxe

```
function sqrt ( x: real ): real;
```

Exemplo

```
Program PascalZIM ;
  begin
    writeln( 'O quadrado de 5 = ', sqr(5) ) ;
    writeln( 'A raiz quadrada de 2 = ', sqrt(2.0) ) ;
  end.
```

## trunc

Trunca um valor real em um valor inteiro.
Sintaxe

```
function trunc ( x: real ): integer;
```

Exemplo

```
Program PascalZIM ;
  begin
    writeln( 1.4, ' se torna ', trunc( 1.4 ) ) ;      { 1.0 }
    writeln( 1.5, ' se torna ', trunc( 1.5 ) ) ;      { 1.0 }
    writeln( -1.4, ' se torna ', trunc( -1.4 ) ) ;   { -1.0 }
    writeln( -1.5, ' se torna ', trunc( -1.5 ) ) ;   { -1.0 }
  end.
```

## upcase

Recebe como parâmetro um caractere e retorna sua representação em caixa alta.

Sintaxe

```
function upcase( c: char ): char;
```

Exemplo

```
   Program PascalZIM;
   var
      umCaractere: char;
      cadeia: string;
      i: integer;
    begin
       cadeia:= 'Uma frase' ;
       for i:=0 to length( cadeia ) do
         write( upcase( cadeia[ i ] ) ) ;
     end.
```