Pascalzim

Table of contents

Introdução

A linguagem Pascal foi desenvolvida pelo professor Niklaus Wirth no ano de 1972, na cidade de Genebra, Suíça. O nome da linguagem foi uma homenagem ao filósofo e matemático Blaise Pascal (1623-1662), inventor da primeira calculadora mecânica. O desejo de Wirth era dispor, para o ensino de programação, de nova linguagem que fosse simples, coerente e capaz de incentivar a confecção de programas claros e facilmente legíveis, favorecendo a utilização de boas técnicas de programação.

A linguagem Pascal se tornou amplamente conhecida e utilizada com o lançamento da mundialmente famosa série de compiladores Turbo Pascal pela Borland, em 1985, devido a uma combinação de simplicidade e poder.

O compilador Pascalzim, desenvolvido no Departamento de Ciências da Computação da Universidade de Brasília, é fruto de muitos anos de pesquisa e trabalho na área de tradutores e linguagens de programação. Adotado como ferramenta de apoio ao ensino e aprendizagem da linguagem Pascal pelos alunos matriculados no curso de Introdução à Ciência da Computação nesta instituição, o compilador foi utilizado no primeiro semestre do ano 2000.

No segundo semestre de 2001 a ferramenta foi utilizada pelos alunos do Instituto de Ensino Superior de Brasília - IESB para o aprendizado da disciplina Algoritmos e Programação Estruturada.

A ferramenta foi exaustivamente testada em laboratório, mas ainda assim não se encontra livre de erros de implementação. A correção destes será efetuada tão breve quanto sua descoberta, levando à disponibilidade de versões mais atualizadas do compilador.

O compilador implementa um subconjunto da linguagem Pascal e contém as estruturas de dados, funções e comandos mais utilizados por iniciantes no estudo dessa linguagem. O arquivo de ajuda que acompanha o produto especifica as instruções suportadas.

Críticas e sugestões são bem-vindas!

O Pascalzim foi concebido com finalidade meramente educacional e sua distribuição é livre.

Formato básico de um programa Pascal

Um programa escrito na linguagem Pascal pode ser, basicamente, estruturado em três regiões significativas:

1. Um cabeçalho, que dá nome ao programa;
2. Uma seção de definição e declaração de dados;
3. Uma seção de comandos, que contém as instruções do programa.

O cabeçalho de um programa é iniciado com a palavra reservada **Program**, seguido de um nome que identifica o programa e um ponto e vírgula.

Exemplo

```
Program MeuPrograma ;
```

A seção de definição e declaração de dados segue o cabeçalho do programa, e é o local onde são definidas as constantes e tipos que serão usados dentro do programa. Nesta seção também são declaradas as variáveis globais do programa, assim como as funções e procedimentos que podem ser utilizados pelo programa principal.

Essa seção consiste das seguintes partes:

1. A parte para declaração de constantes;
2. A parte para definição de tipos;
3. A parte para declaração de variáveis;
4. A parte para definição de funções e procedimentos;
5. A parte para declaração de rótulos para o comando goto.

A definição de cada uma dessas partes é opcional, e não precisa seguir necessariamente a ordem estabelecida. Uma função, por exemplo, pode ser definida antes da declaração de uma variável.

Seguindo a seção de definição e declaração de dados vem a seção de comandos. Esta seção é iniciada com a palavra reservada **Begin** e terminada com a palavra reservada **End**, seguida de ponto. Entre as palavras **Begin** e **End** devem ser colocados os comandos do programa.

De maneira geral, o formato de um programa Pascal possui a seguinte estrutura:

```
Program NomePrograma ;

Seção de definições e declarações

Begin
  Comandos
End.
```

Onde **Program**, **Begin** e **End** são palavras reservadas da linguagem Pascal.

Identificadores

Um identificador válido na linguagem Pascal é qualquer sequência de caracteres que obedeça às seguintes regras:

1. Seja iniciada por uma letra (a, b, ..., z);
2. Possui, depois da primeira letra, uma sequência de caracteres que podem ser letras, dígitos (1, 2, ..., 9, 0) ou ainda o caractere _ ;
3. Não é uma das palavras reservadas da linguagem Pascal.

Exemplo

Identificadores válidos na linguagem Pascal:

A
Nota
P1
Meu_Identificador

Identificadores inválidos na linguagem Pascal:

1A
E(13)
A:B

A linguagem Pascal não diferencia palavras maiúsculas de palavras minúsculas. Assim, para o compilador as seguintes palavras denotam um mesmo identificador:

PASCAL = pascal = Pascal

Palavras reservadas

O conjunto das palavras reservadas da linguagem Pascal, identificadas pelo compilador é dado pela tabela abaixo:

abs	cos	for	lightmagenta	readkey	then
and	crt	frac	lightred	readln	to
append	cursoroff	function	ln	real	true
arctan	cursoron	getdir	longint	record	trunc
array	cyan	goto	lowvideo	red	type
assign	darkgray	gotoxy	magenta	rename	unit
begin	dec	green	maxint	repeat	until
black	delay	highvideo	mkdir	reset	upcase
blink	delete	if	mod	rewrite	uses
blue	delline	implementation	new	rmdir	val
boolean	dispose	in	nil	round	var
break	div	inc	normvideo	seek	wherex
brown	do	insert	not	set	wherey
byte	downto	insline	odd	shl	while
case	else	int	of	shortint	white
char	end	integer	or	shr	window
chdir	eof	interface	ord	sin	with
chr	eoln	ioresult	pi	sqr	word
close	erase	keypressed	pos	sqrt	write
clreol	exit	label	pred	str	writeln
clrscr	exp	length	procedure	string	xor
concat	false	lightblue	program	succ	yellow
const	file	lightcyan	random	text	
continue	filepos	lightgray	randomize	textbackground	
copy	filesize	lightgreen	read	textcolor	

Constantes predefinidas

O compilador reconhece as seguintes constantes pré-definidas:

- **maxint**

Guarda o valor máximo de um inteiro, 32.767.

- **pi**

Guarda o valor da constante pi, 3.14159265358979.

Declaração de constantes

As constantes são declaradas na seção de declaração de constantes, que está contida na seção de definição e declaração de dados.

O início da seção de declaração de constantes é indicada por meio da palavra reservada **const**. A palavra reservada **const** marca o início da seção de definições de constantes, e deve aparecer somente uma única vez dentro da seção de declarações e definições.

Sintaxe

```
const identificador1, identificador2, ... , identificadorn = expressão ;
```

Onde expressão é uma expressão que, em tempo de compilação, pode ser avaliada em uma constante inteira, real, uma cadeia de caracteres ou um único caractere.

Nesse tipo de declaração, o tipo de uma constante é implicitamente definido pelo valor que ela recebe.

Exemplo. A declaração abaixo define uma constante inteira cujo valor é 10:

```
const dez = 10 ;
```

Constantes tipadas

Uma constante tipada é uma constante cujo tipo é informado no momento da declaração da constante.

Sintaxe

```
const identificador1, identificador2, ... , identificadorn : tipo = expressão
;
```

A expressão, nesse caso, deve ser compatível com o tipo informado.

```
const a: integer = 26 ;
      b: char = 'L' ;
      c: boolean = true ;
      d: real = 3.1418 ;
      e: string = 'Pascalzim' ;
```

A expressão de inicialização pode ser utilizada para inicializar um vetor. Nesse caso, os elementos do vetor devem aparecer dentro de um par de parênteses, e devem estar separados por vírgula (,):

```
const m: array[1..3] of integer = ( 50, 60, 70 );
      n: array[1..3] of char = ( 'a', 'b', 'c' );
      o: array[1..3] of boolean = ( true, false, true );
      p: array[1..3] of real = ( 50.0, 60.0, 70.0 );
      q: array[1..3] of string = ( 'abc', 'def', 'ghi' );
```

A inicialização de arrays de mais de uma dimensão segue a ideia acima, com a ressalva de que cada dimensão adicional do array deve aparecer entre um par de parênteses, e os elementos correspondentes a cada dimensão são também separados por vírgula (,).

```
const c: array[1..3, 1..2] of integer = ((1,2),(3,4),(5,6));
      d: array[1..3, 1..2, 1..2] of integer = ( ((1,2),(3,4)) , ((5,6),(7,8)),
((10,11),(12,13)) );
```

A expressão de inicialização pode também ser utilizada para inicializar um registro. Nesse caso, os valores de inicialização do registro são informados entre um par de parênteses, e a inicialização de cada campo é separado da inicialização seguinte por um sinal de ponto e vírgula (;). A inicialização de um campo é dada pelo nome do campo seguido por um sinal de dois pontos (:) e o valor de inicialização do campo.

```
type r1 = record x, y: integer ; End ;
     r4 = array[1..2] of record x, y: integer ; End ;

const s1: r1 = (x:10 ; y:20);
      s2: r2 = ((x:15 ; y:25),(x:35 ; y:45));
```


Tipos de dados

Todas as variáveis declaradas dentro de um programa devem ser especificadas através de um tipo.

Um tipo é uma especificação que:

- Indica o espaço em memória necessário para o armazenamento de um dado (ou conjunto de dados);
- Define o conjunto de operações que pode ser aplicada a um dado (ou conjunto de dados).

Os tipos implementados no compilador podem, basicamente, ser classificados em três categorias:

1. Tipos predefinidos
2. Tipos estruturados

3. Tipos definidos

Tipos predefinidos

Os tipos de dados predefinidos na linguagem Pascal, e implementados no compilador, são:

- **Boolean**
  - Define dois valores lógicos: FALSE e TRUE.
  - Um dado do tipo booleano ocupa um byte de espaço na memória.
  - No Pascalzim, um dado do tipo booleano ocupa quatro bytes de espaço na memória (mesmo tamanho que um integer).

- **Char**
  - Define os elementos do conjunto de caracteres que compõem o alfabeto ASCII, adicionados dos caracteres representados pelos códigos de 128 a 255.
  - Um dado desse tipo ocupa um byte de espaço na memória.
  - No Pascalzim, um dado do tipo booleano ocupa quatro bytes de espaço na memória (mesmo tamanho que um integer).

- **Integer**
  - Define os valores inteiros compreendidos no intervalo de -2.147.483.648 até 2.147.483.647.
  - Um dado desse tipo ocupa quatro bytes de espaço na memória.

- **Real**
  - Define os valores reais definidos no intervalo de 3.4*(10^-38) até 3.4 * (10^+38).
  - Um dado desse tipo ocupa quatro bytes de espaço na memória.
  - No Pascalzim, um dado do tipo real ocupa oito bytes de espaço na memória (mesmo tamanho que um integer).

- **String**
  - Define uma cadeia de caracteres. Se nenhuma restrição de tamanho for especificada, um dado do tipo string é capaz de armazenar uma sequência contendo até 255 caracteres, onde cada caractere ocupa um byte de espaço na memória.
  - No Pascalzim, cada caractere ocupa quatro bytes de espaço na memória (mesmo tamanho que um integer).

Uma cadeia de caracteres pode ter seu tamanho definido (contendo menos de 255 caracteres), onde o tamanho especifica o número máximo de caracteres contidos na cadeia. Essa especificação deve ser indicada entre colchetes, logo após a palavra reservada string.

Exemplo

```
Nome: string [6];
```

define uma cadeia capaz de armazenar até 6 caracteres.

Uma cadeia de caracteres definida com *n* caracteres ocupa *n* bytes de espaço na memória

Como uma extensão da linguagem Pascal, o compilador implementa também os seguinte tipos

de dados inteiros:

- **Byte**
  - Define os valores inteiros compreendidos no intervalo de 0 a 255.
  - Um dado desse tipo ocupa um byte de espaço na memória.
  - No Pascalzim, um dado do tipo byte ocupa quatro bytes de espaço na memória (mesmo tamanho que um integer).

- **ShortInt**
  - Define os valores inteiros compreendidos no intervalo de -128 a 127.
  - Um dado desse tipo ocupa um byte de espaço na memória.
  - No Pascalzim, um dado do tipo shortint ocupa quatro bytes de espaço na memória (mesmo tamanho que um integer).

- **Word**
  - Define os valores inteiros compreendidos no intervalo de 0 a 65.535.
  - Um dado desse tipo ocupa dois bytes de espaço na memória.
  - No Pascalzim, um dado do tipo word ocupa quatro bytes de espaço na memória (mesmo tamanho que um integer).

- **LongInt**
  - Define os valores inteiros compreendidos no intervalo de -2.147.483.648 até 2.147.483.647.
  - Um dado desse tipo ocupa quatro bytes de espaço na memória.

Exemplo

```
Program PascalZIM ;
Begin
   writeln('Tamanho tipo boolean:', sizeof(boolean));
   writeln('Tamanho tipo char:', sizeof(char));
   writeln('Tamanho tipo integer:', sizeof(integer));
   writeln('Tamanho tipo real:', sizeof(real));
   writeln('Tamanho tipo string:', sizeof(string));
   writeln('Tamanho tipo byte:', sizeof(byte));
   writeln('Tamanho tipo shortint:', sizeof(shortint));
   writeln('Tamanho tipo word:', sizeof(word));
   writeln('Tamanho tipo longint:', sizeof(longint));
End.
```

O compilador também reconhece constantes inteiras nas bases hexadecimal, octal e binária. Nesse caso:

- Constantes hexadecimais iniciam com %
- Constantes octais iniciam com &
- Constantes binárias iniciam com $

Exemplo

- %101 (constante binária com o valor 5)
- &144 (constante octal com o valor 100)
- $64 (constante hexadecimal com o valor 100)

Tipos estruturados

Os tipos de dados predefinidos podem ser organizados em tipos de dados complexos, denominados tipos estruturados.

A linguagem Pascal oferece, basicamente, cinco destes tipos:

1. Conjuntos
2. Enumerações
3. Ponteiros
4. Registros
5. Vetores
6. Tipos Procedurais

Conjuntos

Tipos de dados utilizados para denotar um conjunto de valores.

Declaração de conjuntos

```
var nomeConjunto: set of tipo ;
```

Onde nomeConjunto denota um identificador válido na linguagem Pascal, e tipo denota um tipo enumerado, integer, char ou um intervalo de valores dentro desses tipos.

Exemplo de declaração de conjuntos

```
type diassemana = ( dom, seg, ter, qua, qui, sex, sab );

var s1: set of real ;      // tipo inválido
    s2: set of string ;    // tipo inválido
    s3: set of char ;
    s4: set of integer ;
    s5: set of diassemana ;
    s6: set of 1..10 ;
    s7: set of 'a'..'g' ;
    s8: set of seg..sex ;
```

Os elementos de um conjunto são informados entre colchetes:

Elementos de um conjunto

```
[ elemento1, elemento2, ... ,elemento3 ];
```

Onde cada elemento denota uma expressão, e todos os elementos tem o mesmo tipo de dados.
elemento, também pode denotar um intervalo.

Exemplo de atribuições

```
s3:= [ 'a' .. 'z' ];
s4:= [ 1, 2, 3, 4, 5, 6..10 ];
s5:= [ seg..sex ];
s6:= [ 3, 4 ];
s7:= [ 'b', 'e'];
s8:= [ ter, qua ];
```

As seguintes operações podem ser realizadas sobre conjuntos:

- união(+);
- intersecção(*);
- diferença(-);
- igualdade(=);
- desigualdade(<>);
- contém (>=);
- está contido (<=);
- pertinência (in).

Exemplo

```
Program PascalZIM ;
Begin
   writeln( [1,2,3] + [4,5,6,7] );  // Exibe [1..7]
   writeln( [1..10] * [3,4,5] );    // Exibe [3..5]
   writeln( [1..10] - [3,4,5] );    // Exibe [1..2, 6..10]
   writeln( [1..3] = [1,2,3] );     // Exibe true
   writeln( [1..3] <> [1,2,3] );    // Exibe false
   writeln( [1..10] >= [3,4,5] );   // Exibe true
   writeln( [3,4,5] <= [1..10] );   // Exibe true
   writeln( 2+3 in [1..10] );       // Exibe true
End.
```


Enumerações

Tipos de dados enumerados são utilizados para denotar um elemento dentro de um conjunto de constantes.

Declaração de enumerações

```
var nomeEnumeracao: ( identificador, ... , identificador );
```

Onde identificador denota um identificador válido na linguagem Pascal.

Exemplo

```
Program PascalZIM ;
var diasSemana: ( domingo, segunda, terca, quarta, quinta, sexta, sabado );
Begin
   writeln( 'Depois de segunda vem quinta? ' , succ(segunda) = quinta );
```

```
   writeln( 'Depois de segunda vem terca? '  , succ(segunda) = terca );
   writeln( 'Antes de quinta vem quarta? '   , pred(quinta) = quarta );
   writeln( 'Antes de quinta vem segunda? '  , pred(quinta) = segunda );
End.
```

Exemplo

```
Program PascalZIM ;
type diaSemana = ( domingo, segunda, terca, quarta, quinta, sexta, sabado );
var dia: diaSemana ;
Begin
   for dia:= domingo to sabado do
   Begin
      case ( dia ) of
         domingo: writeln( 'O dia é domingo' );
         segunda: writeln( 'O dia é segunda' );
         terca  : writeln( 'O dia é terca' );
         quarta : writeln( 'O dia é quarta' );
         quinta : writeln( 'O dia é quinta' );
         sexta  : writeln( 'O dia é sexta' );
         sabado : writeln( 'O dia é sabado' );
      End ;
   End ;
   readkey ;
End.
```


Ponteiros

Ponteiros são variáveis que podem armazenar o endereço de uma outra variável.

Declaração de ponteiros

```
var nomePonteiro: ^tipoDados ;
```

O símbolo **^** deve ser lido como o ponteiro para...

Na declaração acima temos que nomePonteiro é um ponteiro para variáveis do tipo tipoDados.

Exemplo

```
var ponteiro: ^integer ;
```

Exemplo

```
type TAluno = Record
                 nome: string ;
                 matricula: string ;
              End ;

var ponteiroAluno: ^TAluno ;
```

Operações sobre ponteiros

- Guardar no ponteiro endereço de uma variável:

ponteiro:= @variável ;

- Guardar no ponteiro o endereço armazenado em um outro ponteiro:

ponteiro:= outroponteiro ;

- Dizer que o ponteiro não guarda nenhum endereço:

ponteiro:= **nil** ;

- Referenciar o dado apontado pelo ponteiro (o elemento que tem o tipo de dados definido pelo ponteiro, e que está no endereço de memória que o ponteiro armazena):

ponteiro^

- Fazer o ponteiro apontar para o objeto anterior/seguinte:

inc(ponteiro) ;

dec(ponteiro);

Exemplo

```
Program Ponteiros ;
Var
   a: integer ;
   p: ^integer ;
Begin
   a:= 8 ;      // Guardamos o valor 8 em a
   p:= nil ;    // O ponteiro não guarda nenhum endereço
   writeln( 'Valor armazenado em a: ' , a );
   // Guardamos no ponteiro o endereço da variável a
   p:= @a ;
   writeln( 'Valor apontado por p: ' , p^ );
   // Esse comando é equivalente a "a:= 2 * a ;" , pois p
   // aponta para o endereço de a
   a:= 2 * p^ ;
   writeln( 'O valor de a agora: ' , a );       // Imprime 16
   writeln( 'Valor apontado por p: ' , p^ );    // Imprime 16
   readln ;
End.
```

Alocação dinâmica de memória

É possível alocar, dinamicamente, espaço na memória para um ponteiro. A quantidade de memória é determinada pelo tipo do ponteiro.

Sintaxe

```
new( ponteiro );
```

Deve-se tomar cuidado para que a memória alocada com um new seja liberada antes do programa terminar.

Sintaxe

```
dispose( ponteiro );
```

Como alternativa, pode-se utilizar também os métodos GetMem e FreeMem para alocar e liberar memória para um ponteiro.

Exemplo

```
Program AlocacaoDinamica ;
var p: ^integer ;
    v: integer ;
Begin
   new( p );   // Aloca espaço para armazenar um inteiro
   p^:= 10 ;   // Guarda um inteiro na posição apontada por p
   writeln( 'Valor armazenado na posicao de memoria: ', p^ );
   v:= p^ ;   //Guardamos em v o valor apontado por p
   writeln( 'Valor armazenado em v: ', v );
   dispose( p );   // Liberamos a memoria associada a p
   readln ;
End.
```

Exemplo

```
// ----------------------------------------------------------------
// Este programa mostra ilustra a utilização de listas lineares
// usando ponteiros.
//
// Problema. Construir uma lista linear e imprimir seus dados.
// ----------------------------------------------------------------

Program PercorrendoLista ;

// Definição de um tipo para representar um nó da lista
type TNo = record
              dado: integer ;   // Dado armazenado pelo nó
              prox: ^TNo ;      // Ponteiro p/ próximo nó
           End ;

var pinicio: ^TNo ;    // Guarda endereço 1° nó da lista
    p1: ^TNo ;         // Auxiliar. Guarda endereço de um nó
    resposta: char ;   // Auxiliar. Controla repetição.

Begin
   pinicio:= nil ;

   // Repetição que define os nós da lista
   repeat
      new( p1 );
      write( 'Entre com novo dado: ' );
      readln( p1^.dado );
      p1^.prox:= pinicio ;
      pinicio:= p1 ;
      write( 'Novo dado(S/N)?' );
      readln( resposta );
```

```
      resposta:= upcase( resposta );
   until resposta = 'N' ;

   // Percorrer a lista, imprimindo seus elementos
   p1:= pinicio ;
   while( p1 <> nil ) do
      Begin
         writeln( 'Achei: ' , p1^.dado );
         p1:= p1^.prox ;
      End ;

   // Percorrer a lista, desalocando memória para os elementos
   while( pinicio <> nil ) do
      Begin
         p1:= pinicio ;
         pinicio:= pinicio^.prox ;
         dispose( p1 );
      End ;

   readln ;
End.
```


Registros

Um registro é um tipo composto por um conjunto formado por dados de tipos diferentes, onde cada um dos dados é definido como sendo um campo.

Um tipo registro é definido através da palavra reservada **record**, seguida por uma série de declaração de campos. A palavra reservada **End** seguida de um ponto e vírgula encerra a definição de um registro.

A sintaxe genérica para definição de registros segue o seguinte formato:

Sintaxe

```
Record
   identificador de campo : tipo ;
   identificador de campo : tipo ;
   ...
   identificador de campo : tipo ;
End ;
```

Exemplo. Declaração de um registro simples:

```
var dados: Record
               numero: integer ;
               caractere: char ;
               preenchido: boolean ;
            End ;
```

Exemplo. Declaração de um registro contendo registros aninhados:

```
var umRegistro : Record
```

```
    numero: integer ;
    dado: Record
    caractere: char ;
    End ;
preenchido: boolean ;
End ;
```

A referência a um campo de um registro é feita através do nome da variável do tipo registro seguida por um ponto e pelo nome do campo, como por exemplo:

```
umRegistro.numero
```


Vetores

Um vetor é uma estrutura de dados que contém um número fixo de elementos que possuem um mesmo tipo de dados, tipo esse que pode ser qualquer um dos tipos predefinidos na linguagem Pascal (integer, char, boolean ou string), um tipo vetor, um tipo registro ou ainda um tipo definido pelo usuário.

O número de elementos de um vetor é determinado pelo intervalo de indexação de elementos do vetor, que é especificado por duas constantes ordinais separadas por dois pontos, entre colchetes.

A sintaxe para definição de vetores segue o seguinte formato:

```
array[limiteInferior..limiteSuperior] of tipo ;
```

Onde:

- **array** e **of** são palavras reservadas da linguagem Pascal, usadas na declaração de vetores;
- limiteInferior e limiteSuperior são constantes ordinais;
- tipo define o tipo de dados de cada elemento do vetor.

Exemplo. A declaração abaixo define um vetor do tipo inteiro, identificado por dias:

```
var dias: array[1..24] of integer ;
```

Nesse vetor, os elementos estão armazenados nas posições de 1 a 24.

A referência ao elemento que está armazenado na posição x de um vetor é dado da seguinte forma:

```
nomeVariavel[x]
```

Os vetores podem ter, ainda, mais de uma dimensão.

Vetores com várias dimensões

Vetores podem ter mais de uma dimensão. Nesse caso, cada nova dimensão é declarada de acordo com as regras do item anterior, e as *n* dimensões do vetor são separadas por vírgulas.

A sintaxe para definição vetores *n*-dimensionais segue o seguinte formato:

```
array[limite1..limite2, limite3..limite4, ... ,limiten-1..limiten] of tipo ;
```

Exemplo. A declaração abaixo define um vetor de duas dimensões, do tipo inteiro:

```
var matriz: array[1..10, 1..20] of integer ;
```


Tipos Procedurais

O tipo de dados procedural permite armazenar em uma variável o endereço de uma função ou procedimento definidos pelo usuário.

Declaração do tipo procedural

É semelhante à declaração do cabeçalho de uma função ou um procedimento, sem o identificador da função ou procecimento.

Para declarar uma variável que guarda o endereço de uma função ou procedimento sem parâmetros, usa-se:

```
var minhaFuncao: function : tipo ;
    meuProcedure: procedure ;
```

Para declarar uma variável que guarda o endereço de uma função ou procedimento com parâmetros, usa-se:

```
var minhaFuncao: function( parâmetro1: tipo ; parâmetro2: tipo ; ... ;
parâmetron : tipo ) : tipo ;
    meuProcedure: procedure( parâmetro1: tipo ; parâmetro2: tipo ; ... ;
parâmetron : tipo ) ;
```

Exemplo

```
var minhaFuncao : function(a,b: integer): integer ;

    meuProcedure : procedure(texto: string);
```

Nesse exemplo, minhaFuncao pode armazenar o endereço de uma função que recebe dois parâmetros do tipo integer, e que retorna um valor do mesmo tipo.

Também, meuProcedure pode armazenar o endereço de um procedimento que recebe um parâmetro do tipo string.

Operações sobre variáveis do tipo procedural

- Guardar na variável o endereço de uma função definida pelo usuário:

minhaFuncao := funcaoDefinidaUsuario ;

- Guardar na variável o endereço armazenado em uma outra variável do tipo procedural:

minhaFuncao := minhaOutraFuncao ;

- Dizer que a variável do tipo procedural não guarda nenhum endereço:

minhaFuncao := **nil** ;

- Referenciar o endereço armazenado na variável do tipo procedural:

@minhaVariavel

Exemplo

```
Program Pascalzim;

// Função que calcula o minimo de a e b
function min(a,b: integer): integer;
Begin
  if (a < b) then
    min:= a
  else
    min:= b ;
End;

// Função que calcula o máximo de a e b
function max(a,b: integer): integer;
Begin
  if (a > b) then
    max := a
  else
    max := b ;
End;

// Variáveis que podem guardar o endereço das funções min e max
var F, G: function(a,b: integer): integer;

Begin
  F := min ;
  writeln( F(50,60) );  // Mostra 50, valor retornado pela funcao min

  F := max ;
  writeln( F(50,60) );  // Mostra 60, valor retornado pela funcao max

  G:= F ;
  writeln( G(50,60) );  // Mostra 50, valor retornado pela funcao max
End.
```

Exemplo

```
Program Pascalzim;

// Função que calcula o minimo de a e b
```

```
function min(a,b: integer): integer;
Begin
  if (a < b) then
    min:= a
  else
    min:= b ;
End;

// Função que calcula o máximo de a e b
function max(a,b: integer): integer;
Begin
  if (a > b) then
    max := a
  else
    max := b ;
End;

// Variáveis que podem guardar o endereço das funções min e max
var F, G: function(a,b: integer): integer;

Begin
  F:= nil ;
  writeln( @F = nil ); // Mostra true, pois F não guarda o endereço de uma
função

  F := min ;
  G := max ;
  writeln( @F = @G );  // Mostra false, pois F guarda o endereço de min, G
guarda o endereço de max

  G := min ;
  writeln( @F = @G );  // Mostra true, pois F e G guardam ambas o endereço de
min

  F := G ;
  writeln( @F = @G );  // Mostra true, pois F e G guardam ambas o endereço de
min
End.
```


Definição de tipos

A definição de um novo tipo é feita na seção de definição de tipos, contida na seção de definição e declaração de dados.

O início da seção de definição de tipos é indicada através da palavra reservada **type**. A palavra reservada **type** deve aparecer uma única vez dentro da seção de definição e declaração de dados.

Sintaxe

```
type nomeTipo = tipoDefinido ;
```

Onde tipoDefinido é um dos tipos estruturados vetor, registro, ponteiro, procedural ou ainda o nome de um tipo definido pelo usuário ou um tipo de dados simples.

Exemplo

```
type vetorInteiros = array[1..100] of integer ;
     matrizReais = array[0..9, 0..9] of real ;
     ponteiroInteiro = ^integer ;
     ponteiroFuncao = function(a, b: integer) : integer ;
```


Declaração de variáveis

A declaração de uma variável faz com que o compilador reserve uma quantidade de espaço na memória suficientemente grande para armazenar um tipo de dados, além de associar também um "nome" a esta posição na memória. As variáveis são declaradas na seção de declaração de variáveis, contida na seção de definição e declaração de dados.

O início da seção de declaração de variáveis é indicada por meio da palavra reservada **var**. A palavra reservada **var** deve aparecer somente uma única vez dentro da seção de definição e declaração de dados.

Sintaxe

```
var identificador1, identificador2, ... , identificadorn : tipo ;
```

Exemplo. A declaração abaixo define três variáveis dos tipos inteiro, caractere e booleano, respectivamente.

```
var inteiro: integer ;
    caractere: char ;
    booleano: boolean ;
```

Variáveis inicializadas

Uma variável pode ser inicializada no momento da sua declaração.

Sintaxe

```
var identificador1, identificador2, ... ,identificadorn : tipo = expressão ;
```

A expressão, nesse caso, deve ser compatível com o tipo informado.

```
var a: integer = 26 ;
    b: char = 'L' ;
    c: boolean = true ;
    d: real = 3.1418 ;
    e: string = 'Pascalzim' ;
```

A expressão de inicialização pode ser utilizada para inicializar um vetor. Nesse caso, os elementos do vetor devem aparecer dentro de um par de parênteses, e devem estar separados por vírgula (,):

```
var m: array[1..3] of integer = ( 50, 60, 70 );
    n: array[1..3] of char = ( 'a', 'b', 'c' );
```

```
    o: array[1..3] of boolean = ( true, false, true );
    p: array[1..3] of real = ( 50.0, 60.0, 70.0 );
    q: array[1..3] of string = ( 'abc', 'def', 'ghi' );
```

A inicialização de arrays de mais de uma dimensão segue a ideia acima, com a ressalva de que cada dimensão adicional do array deve aparecer entre um par de parênteses, e os elementos correspondentes a cada dimensão são também separados por vírgula (,).

```
var c: array[1..3, 1..2] of integer = ((1,2),(3,4),(5,6));
    d: array[1..3, 1..2, 1..2] of integer = ( ((1,2),(3,4)) , ((5,6),(7,8)),
((10,11),(12,13)) );
```

A expressão de inicialização pode ser também utilizada para inicializar um registro. Nesse caso, os valores de inicialização do registro são informados entre um par de parênteses, e a inicialização de cada campo é separado da inicialização seguinte por um sinal de ponto e vírgula (;). A inicialização de um campo é dada pelo nome do campo seguido por um sinal de dois pontos (:) e o valor de inicialização do campo.

```
type r1 = record x, y: integer ; End ;
     r2 = array[1..2] of record x, y: integer ; End ;

var s1: r1 = (x:10 ; y:20);
    s2: r2 = ((x:15 ; y:25),(x:35 ; y:45));
```


Expressões

O termo expressão se refere a qualquer combinação de uma ou mais constantes ou identificadores de variáveis, com um ou mais operadores. As constantes e variáveis que aparecem numa expressão são chamadas de operandos.

Quando mais de um operador aparece numa expressão, a sequência de cálculo efetuada pelo compilador depende da precedência definida para cada operador da linguagem, onde o operador com mais alta precedência é o primeiro a capturar seus operandos. No caso de dois ou mais operadores terem o mesmo nível de precedência, o cálculo é feito da esquerda para a direita.

São definidos quatro níveis de precedência para os operadores da linguagem, definidos abaixo em ordem decrescente:

1. - (menos unário), not
2. *, div, mod, and
3. +, -, or
4. =, <>, <, >, <=, >=

Parênteses alteram a ordem de precedência de um conjunto de operadores, forçando o programa a calcular a expressão dentro dos parênteses antes das outras.

Por exemplo, a adição é calculada antes da multiplicação em 5* (3+4).

Operadores

Grande parte da manipulação de dados que ocorre na seção de comandos é feita através do uso de um operador.

Um operador, na linguagem Pascal, pertence basicamente a uma dentre as quatro categorias básicas abaixo:

1. Operadores aritméticos
2. Operadores lógicos
3. Operadores relacionais
4. Operadores binários

Operadores aritméticos

Os operadores aritméticos são utilizados nas expressões aritméticas.

Os operadores aritméticos definidos pelo compilador são:

- **-** (menos unário)
  Tipo de operandos permitidos: inteiros e reais.
  Operação executada: inverte o valor numérico do operando.

- **DIV**
  Tipo de operandos permitidos: ambos do tipo inteiro.
  Operação executada: o operando à esquerda do DIV é dividido pelo operando à sua direita, sendo o resultado desta operação um valor inteiro resultante da divisão.

- **MOD**
  Tipo de operandos permitidos: ambos do tipo inteiro.
  Operação executada: o operando à esquerda do MOD é dividido pelo operando à sua direita, sendo o resultado desta operação o resto inteiro da divisão.

- **+**
  Tipo de operandos permitidos: inteiros, reais e cadeias de caracteres.
  Operação executada: no caso de inteiros e reais o operando à esquerda do + é somado ao operando a sua direita, sendo o tipo do resultado dessa operação dependente de seus operandos:

  - Se os dois operandos são inteiros, o resultado da soma é um valor inteiro.
  - Se os dois operandos são reais, o resultado da soma é um valor real.
  - Se um dos operandos é real, e o outro é inteiro, o resultado da soma é um valor real.

No caso dos operandos serem ambos cadeias de caracteres o resultado da operação é dada pela cadeia obtida pela concatenação da cadeia dada pelo primeiro operando com a cadeia dada pelo segundo operando.

- **-**
  Tipo de operandos permitidos: inteiros e reais.
  Operação executada: o operando à esquerda do - é subtraído do operando a sua direita, sendo o tipo do resultado dessa operação dependente de seus operandos:

  - Se os dois operandos são inteiros, o resultado da operação é inteiro.

  - Se os dois operandos são reais, o resultado da operação é real.
  - Se um dos operandos é real, e o outro é inteiro, o resultado da operação é real.

- *****
  Tipo de operandos permitidos: inteiros e reais.
  Operação executada: o operando à esquerda do * é multiplicado pelo operando a sua direita, sendo o tipo do resultado dessa operação dependente de seus operandos:

  - Se os dois operandos são inteiros, o resultado da operação é um valor inteiro.
  - Se os dois operandos são reais, o resultado da operação é um valor real.
  - Se um dos operandos é real, e o outro é inteiro, o resultado da operação é um valor real.

- **/**
  Tipo de operandos permitidos: inteiro ou real.
  Operação executada: o operando à esquerda do / é dividido pelo operando a sua direita, sendo o resultado dessa operação real.

Operadores lógicos

Os operadores lógicos são usados nas expressões lógicas, com operandos do tipo booleano.

Os operadores lógicos definidos pelo compilador são:

- **not**
  Operação executada: o operador inverte o valor verdade de um operando booleano.

- **and**
  Operação executada: é efetuado um and lógico entre os dois operandos do operador, sendo o resultado da operação verdadeiro quando ambos operandos são verdadeiros.

- **or**
  Operação executada: é feito um or lógico entre os dois operandos do operador, sendo o resultado da operação verdadeiro se um dos operandos for verdadeiro.

- **xor**
  Operação executada: é feito um xor lógico entre os dois operandos do operador, sendo o resultado da operação verdadeiro se os dois operandos contiverem valores lógicos diferentes.

A tabela verdade para os operadores lógicos é:

Primeiro Operando	Operador	Segundo Operando	Resultado
True	Not	----	False
False	Not	----	True
True	And	True	True
True	And	False	False
False	And	True	False

False	And	False	False
True	Or	True	True
True	Or	False	True
False	Or	True	True
False	Or	False	False
True	Xor	True	False
True	Xor	False	True
False	Xor	True	True
False	Xor	False	False

Expressões contendo os operadores AND e OR são avaliadas em curto-circuito, a exemplo do compilador Turbo Pascal, da Borland. Dessa forma, se o primeiro operando do AND for avaliado em false, o segundo operando não é analisado. Também, se o primeiro operando do OR for avaliado em true, o segundo operando não é analisado.

Operadores relacionais

Os operadores relacionais são usados nas expressões relacionais.

Os tipos de operandos permitidos para esse tipo de operadores são:

- Ambos operandos do mesmo tipo primitivo (integer, char, boolean, char ou string)
- Operandos de tipos diferentes, onde:
  - Um operando é do tipo **integer** e outro do tipo **real**.
  - Um operando é do tipo **string** e outro do tipo **char**.

O resultado de expressões contendo operadores relacionais é um valor booleano, definido de acordo com a tabela a seguir.

Operador	Resultado
=	Verdadeiro se os dois operandos para o operador forem iguais. Falso em caso contrário.
<>	Verdadeiro se os dois operandos para o operador forem diferentes. Falso em caso contrário.
<	Verdadeiro se o operando à esquerda do operador for menor que o operando à direita. Falso em caso contrário.
<=	Verdadeiro se o operando à esquerda do operador for menor ou igual o operando à direita. Falso em caso contrário
>	Verdadeiro se o operando à esquerda do operador for maior do que o operando à direita. Falso em caso contrário.
>=	Verdadeiro se o operando à esquerda do operador for maior ou igual que o operando à direita. Falso em caso contrário.

Operadores binários

Os operadores binários são usados para manipular os bits de operandos do tipo inteiro.

Os operadores binários definidos pelo compilador são:

- **not**
  Inverte os bits do operando inteiro.
- **and**
  Realiza um and lógico entre os bits correspondentes dos dois operandos.
- **or**
  Realiza um or lógico entre os bits correspondentes dos dois operandos.
- **xor**
  Realiza um xor lógico entre os bits correspondentes dos dois operandos.
- **shl**
  Desloca para os bits do primeiro operando para a esquerda, de acordo com a quantidade de posições informada no segundo operando.
- **shr**
  Desloca para os bits do primeiro operando para a direita, de acordo com a quantidade de posições informada no segundo operando.

Comandos

Os comandos são inseridos na seção de comandos e podem ser, basicamente, classificados em oito categorias:

1. Comandos de atribuição
2. Comandos compostos
3. Comandos de entrada e saída
4. Comandos condicionais
5. Comandos de repetição
6. Comandos para tratamento de arquivos
7. Comandos de desvio
8. Comandos auxiliares

O ponto e vírgula é usado na linguagem Pascal como separador de comandos, servindo para separar um comando dos comandos subsequentes.

Comandos de atribuição

Um comando de atribuição é definido através da seguinte sintaxe:

```
variável:= expressão ;
```

O tipo da expressão deve ser igual ao tipo da variável, com exceção de dois casos especiais onde:

- A variável é do tipo **real** e a expressão é do tipo **integer**.
- A variável é do tipo **string** e a expressão é do tipo **char**.

Exemplo. Sendo dados:

```
var item: integer ;
    saída: boolean ;
    soma, Valor: real ;
    caractere: char ;
    cadeia: string ;
```

São válidos os seguintes comandos de atribuição:

```
item:= 0 ;
saida:= FALSE ;
soma:= valor1 + valor2 ;
caractere:= 'a' ;
cadeia:= 'Isso é uma cadeia de caracteres' ;
soma:= 9 ;
cadeia:= 'a' ;
```


Comandos compostos

Além de marcar o início e o fim da seção de comandos, o par **Begin** e **End** define um par de instruções usado para combinar um conjunto de comandos em um comando composto, também chamado de bloco de comandos.

Exemplo

```
if first < last then
   Begin
      Temp:= First ;
      First:= Last ;
      Last:= Temp ;
   End ;
```


Comandos de entrada e saída

Os comandos usados para entrada e saída de dados são definidos, pelo compilador, por meio de quatro comandos:

1. read
2. readln
3. write
4. writeln

read/readln

Os comandos **read** e **readln** são usados para ler o valor de uma variável de um dispositivo de entrada de dados. A diferença entre os dois comandos é que o comando **readln** processa uma quebra de linha após a leitura do valor de uma variável, enquanto o **read** não o faz.

A leitura de dados pode ser direcionada para um arquivo, identificado por uma variável do tipo **text**.

Sintaxe:

```
read( listaVariáveis );
```

Onde listaVariáveis é uma sequência de uma ou mais variáveis separadas por vírgula.

A sintaxe de um comando **read** para leitura a partir de um arquivo é:

```
read( variavelArquivo, listaVariáveis );
```

Onde variavelArquivo é uma variável do tipo **text**.

Exemplo

```
Program PascalZIM ;
var arq: text ;
    caractere: char ;
Begin
   assign( arq, 'Teste.txt' );
   reset( arq );
   while not eof( arq ) do
      Begin
         read( arq, caractere );
         write( caractere );
      End ;
End.
```


write/writeln

Os comandos **write** e **writeln** são usados para imprimir o valor de uma sequência de expressões em um dispositivo de saída de dados. A diferença entre os dois comandos é que o comando **writeln** processa uma quebra de linha após imprimir o valor de uma sequência de expressões.

A escrita de dados pode ser direcionada para um arquivo, identificado através de uma variável do tipo **text**.

A sintaxe de um comando **write** / **writeln** para impressão na tela de uma sequência de expressões é:

```
write( expressão1 , expressão2 , ... , expressãon );
```

A sintaxe de um comando **write** / **writeln** para impressão em arquivo de uma sequência de expressões é:

```
write( variavelArquivo, expressão1 , expressão2 , ... , expressãon );
```

Onde variavelArquivo é uma variável do tipo **text**.

Exemplo

```
Program PascalZIM ;
var c: char ;
Begin
   writeln( 'Por favor, pressione uma tecla' );
   c:= readkey ;
   writeln( 'Você pressionou ', c, ', cujo valor ASCII é ', ord(c), '.' );
End.
```

Os parâmetros do comando **write** podem conter a especificação do seu comprimento. Tal especificação é definida através da seguinte regra sintática:

```
expressão : tamanho
```

Onde expressão define um parâmetro e tamanho é uma expressão do tipo inteiro.

A impressão de constantes em ponto flutuante pode conter, além da especificação de comprimento, a especificação do número de casas decimais a serem impressas. Essa especificação é dada através da seguinte regra sintática:

```
expressão : tamanho : casas decimais
```

Onde expressão é um parâmetro do tipo real, tamanho e casas decimais são expressões do tipo inteiro.

A impressão de uma linha em branco é dada através de um comando **writeln** como abaixo:

```
writeln ;
```

Exemplo

```
Program PascalZIM ;
var arq: text ;
Begin
   assign( arq, 'Teste.txt' );
   rewrite( arq );
   writeln ;   //Impressão na tela
   writeln( 1:10, 2:20, 3:30 );
   writeln( 'a':10, 'b':20, 'c':30 );
   writeln( 'asd':10, 'bnm':20, 'cvb':30 );
   writeln( 2.1:10, 3.2:20, 4.3:30 );
   writeln( 2.1:10:2, 3.2:20:3, 4.3:30:4 );
   writeln ;   //Impressão em arquivo
   writeln( arq, 1:10, 2:20, 3:30 );
   writeln( arq, 'a':10, 'b':20, 'c':30 );
   writeln( arq, 'asd':10, 'bnm':20, 'cvb':30 );
```

```
   writeln( arq, 2.1:10, 3.2:20, 4.3:30 );
   writeln( arq, 2.1:10:2, 3.2:20:3, 4.3:30:4 );
   close( arq );
End.
```


Comandos condicionais

Os comandos condicionais permitem restringir a execução de comandos.

Os seguintes comandos condicionais são reconhecidos pelo compilador:

- if...then
- if...then...else
- case
- case...else

if

Possibilita restringir a execução de um conjunto de comandos.

A sintaxe de um comando **if**...**then** é:

```
if expressão then comando
```

Onde expressão é uma expressão condicional e comando é um comando simples ou um bloco de comandos.

O comando funciona da seguinte forma: se expressão for true, então comando é executado; caso contrário comando não é executado.

Exemplo

```
if j <> 0 then result:= I/J ;
```

A sintaxe de um comando **if**...**then**...**else** é:

```
if expressão then comando1 else comando2
```

Onde expressão é uma expressão condicional, comando1 e comando2 um comando simples ou um bloco de comandos.

O comando funciona da seguinte forma: se expressão for true, então comando1 é executado; caso contrário, comando2 é executado.

Exemplo

```
if j = 0 then
   write( j )
   else
   write( m );
```

Em uma série de comandos **if** aninhados a cláusula **else** está ligada ao **if** mais próximo no aninhamento.
Uma sequência de comandos como:

```
if expressão1 then if expressão2 then comando1 else comando2 ;
```

É reconhecido pelo compilador da seguinte forma:

```
if expressão1 then [ if expressão2 then comando1 else comando2 ];
```

Pode-se utilizar também, no lugar de vários ifs aninhados, um comando case.

case

Possibilita a escolha de um conjunto de comandos que serão executados, dentre várias alternativas de escolha.

Sintaxe

```
case seletor of
   lista de constantes : comandos ;
   lista de constantes : comandos ;
   ...
   lista de constantes : comandos ;
   else comandos ;
End ;
```

Onde:

- seletor é uma expressão do tipo **integer** ou **char ;**
- lista de constantes é uma sequência de constantes do tipo **integer** ou **char**, separadas por vírgula (ao invés de uma constante é possível usar um intervalo de constantes, que consiste em duas constantes separadas por um par de pontos).

A cláusula **else** não é obrigatória, e os comandos associados a essa cláusula serão executados somente se nenhuma outra opção do case foi selecionada.

Exemplo

```
Program PascalZIM ;
var opcao: integer ;
Begin
   write( 'Entre com uma opcao: ' );
   readln( opcao );
   // escolha da opcao
   case opcao of
      1: writeln( 'Você escolheu a opção 1...' );
```

```
      2: writeln( 'Você escolheu a opção 2...' );
      3: writeln( 'Você escolheu a opção 3...' );
   else
      writeln( 'Você escolheu uma opção diferente de 1, 2, 3...' );
   End ;
End.
```

Exemplo

```
Program PascalZIM ;
const opSoma = '+' ;
      opSubtracao = '-' ;
      opProduto = '*' ;
      opDivisao = '/' ;
var opcao: char ;
Begin
   write( 'Entre com um operador: ' );
   readln( opcao );
   // escolha da opcao
   case opcao of
      opSoma: writeln( 'Você escolheu soma...' );
      opSubtracao: writeln( 'Você escolheu subtracao...' );
      opProduto: writeln( 'Você escolheu produto...' );
      opDivisao: writeln( 'Você escolheu divisao...' );
   End ;
End.
```

Exemplo

```
Program PascalZIM ;
var opcao: integer ;
Begin
   write( 'Entre com uma opcao: ' );
   readln( opcao );
   // escolha da opcao
   case opcao of
      1, 2: writeln( 'Você escolheu a opção 1 ou 2...' );
      3: writeln( 'Você escolheu a opção 3...' );
   else
      writeln( 'Você escolheu uma opção diferente de 1, 2, 3...' );
   End ;
End.
```

Exemplo

```
Program PascalZIM ;
var c: char ;
Begin
   write( 'Digite um caractere: ' );
   readln( c );
   case c of
      'A'..'Z', 'a'..'z': writeln( '=> Você digitou uma letra!' );
      '0'..'9': writeln( '=> Você digitou um digito!' );
      '+', '-', '*', '/': writeln( '=> Você digitou um operador!' );
      else
      writeln( '=> Você digitou um caractere!' );
```

```
   End ;
End.
```


Comandos de repetição

Os comandos de repetição permitem repetir a execução de um conjunto de comandos.

Os seguintes comandos de repetição são reconhecidos pelo compilador:

- repeat
- while
- for

Os comandos de desvio que podem ser utilizados em comandos de repetição são os seguintes:

- break
- continue

repeat

O comando **repeat** executa repetidamente uma sequência de comandos até que uma dada condição, resultantes da avaliação de uma expressão booleana, seja verdadeira.

Sintaxe

```
repeat
   comando1 ;
   ...
   comandon ;
until expressão ;
```

Onde expressão é uma expressão condicional.

Os comandos internos ao **repeat** são executados no mínimo uma vez, pois a condição de parada da repetição é avaliada somente após a primeira repetição.

Exemplo

```
repeat
   k:= i mod j ;
   i:= j ;
   j:= k ;
until j = 0 ;
```

while

O comando **while** é semelhante ao comando **repeat**, com a diferença de que a condição para a execução repetida de comandos é avaliada antes da execução de qualquer comando interno da repetição.

Dessa forma, se a condição inicial para o **while** for falsa, a sequência de comandos definidos para o **while** não será executada nenhuma vez.

Sintaxe

```
while expressão do
   comando
```

Onde expressão é uma expressão lógica e comando pode ser um comando composto.

Exemplo

```
while dado[ i ] <> x do
   i:= i + 1 ;
```


for

O comando **for**, diferente dos comandos **repeat** e **while**, permite que uma sequência de comandos seja executada um número definido de vezes.

Sintaxe

```
for contador:= valorInicial to valorFinal do
   comando

for contador:= valorInicial downto valorFinal do
   comando
```

Onde:

- contador é uma variável do tipo **integer** ou **char.**
- valorInicial e valorFinal são expressões do tipo **integer** ou do tipo **char**.
- comando pode ser um comando simples ou um comando composto.

Funcionamento

1. O comando **for** armazena na variável contador o valor da expressão correspondente à valorInicial.
2. Se contador é maior (**for**...**to**) ou menor (**for**...**downto**) que valorFinal o comando **for** para de executar. Caso contrário, comando é executado.
3. Após executar comando o valor armazenado em contador é incrementando ou decrementando (o **for**...**to** incrementa, e **for**...**downto** decrementa).

4. Volta para o passo 2.

Exemplo

```
for i:= 2 to 63 do
   if data[ i ] > max then
      max:= data[ i ] ;
```

Exemplo

```
for c:= 'a' to 'z' do
   write( c );
```


Comandos para tratamento de arquivos

Dentre os comandos usados para tratamento de arquivos estão inclusos comandos para identificação, abertura e fechamento de arquivos.

O uso de arquivos do tipo texto na linguagem Pascal é feito de duas maneiras distintas:

- Arquivos do tipo texto
- Arquivos binários

append

Abre um arquivo já existente para escrita no final.

Sintaxe

```
procedure append( var variavelArquivo : text );

procedure append( var variavelArquivo : file );
```

Exemplo

```
Program Pascalzim ;
var arq: text ;
Begin
   // Cria um novo arquivo, adicionando texto nele
   assign( arq, 'TEST.TXT ' );
   rewrite( arq );
   writeln( arq, 'texto inicial ' );

   // Fecha o arquivo, salvando as alteracoes efetuadas
   close(arq);
```

```
   // Abre o arquivo para adicionar mais texto no final
   append( arq );
   writeln( arq, 'mais texto! ' );

   // Fecha o arquivo, salvando as alteracoes efetuadas
   close( arq );
End.
```


assign

Associa o nome de um arquivo com uma variável do tipo arquivo.

Um arquivo, seja ele do tipo texto ou ainda um arquivo binário, é referenciado dentro de um programa Pascal por meio de uma variável do tipo arquivo.

Uma variável do tipo arquivo, por sua vez, deve ser definida com o tipo text (para manipulação de arquivos do tipo texto) ou o tipo FILE (para manipulação de arquivos binários).

As diversas operações realizadas no arquivo, como abertura, fechamento, leitura e escrita são realizadas sobre a variável do tipo arquivo.

A associação entre uma variável do tipo arquivo e o arquivo correspondente é realizada através do comando **assign**.

Sintaxe

```
procedure assign( var variavelArquivo : text ; nomeArquivo : string );

procedure assign( var variavelArquivo : file ; nomeArquivo : string );
```

Exemplo

```
assign( arq, 'c:\dados.txt' );
```

Exemplo

```
Program Pascalzim ;
var nomeArquivo: string [15];
    arq: text ;
Begin
   write( 'Entre com o nome do arquivo: ' );
   readln( nomeArquivo );
   assign( arq, nomeArquivo );
End.
```


close

Fecha um arquivo.

Sintaxe

```
procedure close( var variavelArquivo : text );

procedure close( var variavelArquivo : file );
```

Exemplo

```
Program Pascalzim ;
var arq: text ;
Begin
   // Cria um novo arquivo, adicionando dados nele
   assign( arq, 'TEST.TXT' );
   rewrite( arq );
   writeln( arq, 'texto inicial' );

   // Fecha o arquivo, salvando as alteracoes efetuadas
   close( arq );

   // Abre o arquivo para adicionar mais texto no final
   append( arq );
   writeln( arq, 'mais texto!' );

   // Fecha o arquivo, salvando as alteracoes efetuadas
   close( arq );
End.
```


filepos

Retorna a posição corrente (em número de registros) do cursor de leitura/escrita no arquivo binário.

Sintaxe

```
function filepos ( var variavelArquivo : file ) : integer ;
```

Exemplo

```
Program Pascalzim ;

// Define o tipo de registro que será armazenado no arquivo
type registro = record
                   nome: string[20];
                   idade: integer ;
                End ;

// Define variáveis para manipulação do arquivo
var arq: file of registro ;
    reg: registro ;

Begin
   // Abre o arquivo para escrita
   assign( arq, 'dados.bin' );
```

```
   rewrite( arq );

   // Grava dados no arquivo
   reg.nome:= 'Joao Batista' ;
   reg.idade:= 30 ;
   write(arq, reg);
   writeln( 'Gravou registro, está na posicao ', filepos(arq));

   // Grava dados no arquivo
   reg.nome:= 'Pedro Pereira' ;
   reg.idade:= 40 ;
   write(arq, reg);
   writeln( 'Gravou registro, está na posicao ', filepos(arq));

   // Grava dados no arquivo
   reg.nome:= 'Miguel da Silva' ;
   reg.idade:= 50 ;
   write(arq, reg);
   writeln( 'Gravou registro, está na posicao ', filepos(arq));

   // Mostra a quantidade registros armazenados
   write( 'Foram armazenados ' , filesize(arq) , ' registros ' );
   close( arq );
End.
```


filesize

Retorna a quantidade de registros armazenada em um arquivo binário.

Sintaxe

```
function filesize ( var variavelArquivo : file ) : integer ;
```

Exemplo

```
Program Pascalzim ;

// Define o tipo de registro que será armazenado no arquivo
type registro = record
                   nome: string[20];
                   idade: integer ;
                End ;

// Define variáveis para manipulação do arquivo
var arq: file of registro ;
    reg: registro ;

Begin
   // Abre o arquivo para escrita
   assign( arq, 'dados.bin' );
   rewrite( arq );

   // Grava dados no arquivo
   reg.nome:= 'Joao Batista' ;
   reg.idade:= 30 ;
   write(arq, reg);
```

```
   // Grava dados no arquivo
   reg.nome:= 'Pedro Pereira' ;
   reg.idade:= 40 ;
   write(arq, reg);

   // Grava dados no arquivo
   reg.nome:= 'Miguel da Silva' ;
   reg.idade:= 50 ;
   write(arq, reg);

   // Mostra a quantidade registros armazenados
   write( 'Foram armazenados ' , filesize(arq) , ' registros ' );
   close( arq );
End.
```


reset

Abre um arquivo já existente, posicionando o cursor de leitura no início do arquivo.

Sintaxe

```
procedure reset( var variavelArquivo : text );

procedure reset( var variavelArquivo : file );
```

Exemplo

```
Program Pascalzim ;
var arq: text ;
    texto: string ;
Begin
   // Abre o arquivo e lê uma linha de texto dele
   assign( arq, 'TEST.TXT' );
   reset( arq );
   readln( arq, texto );
   writeln( texto );

   // Fecha o arquivo, salvando as alteracoes efetuadas
   close( arq );

   // Abre o arquivo para adicionar mais texto no final
   append( arq );
   writeln( arq, 'mais texto!' );

   // Fecha o arquivo, salvando as alteracoes efetuadas
   close( arq );
End.
```


rewrite

Cria um arquivo do tipo texto ou, se ele já existe, apaga seu conteúdo e cria um novo arquivo,

posicionando o cursor de leitura no início do arquivo.

Sintaxe

```
procedure rewrite( var variavelArquivo : text );

procedure rewrite( var variavelArquivo : file );
```

Exemplo

```
Program Pascalzim ;
var arq: text ;
Begin
   // Cria um novo arquivo, adicionando dados nele
   assign( arq, 'TEST.TXT' );
   rewrite( arq );
   writeln( arq, 'texto inicial' );

   // Fecha o arquivo, salvando as alteracoes efetuadas
   close( arq );

   // Abre o arquivo para adicionar mais texto no final
   append( arq );
   writeln( arq, 'mais texto!' );

   // Fecha o arquivo, salvando as alteracoes efetuadas
   close( arq );
End.
```


seek

Posiciona o cursor de leitura/escrita antes de uma posição específica (em número de registros) do arquivo binário.

O comando considera que o primeiro registro armazenado no arquivo está na posição zero.

Sintaxe

```
procedure seek ( var variavelArquivo: file ; posicao: integer );
```

Onde posicao é uma expressão inteira que informa o registro antes do qual será posicionado o cursor de leitura/escrita.

Exemplo

```
Program Pascalzim ;

// Define o tipo de registro que será armazenado no arquivo
type registro = record
                   nome: string[20];
                   idade: integer ;
                End ;
```

```
// Define variáveis para manipulação do arquivo
var arq: file of registro ;
    reg: registro ;
    i: integer ;

Begin
   // Abre o arquivo para escrita
   assign( arq, 'dados.bin' );
   rewrite( arq );

   // Grava dados no arquivo
   reg.nome:= 'Joao Batista' ;
   reg.idade:= 30 ;
   write(arq, reg);

   // Grava dados no arquivo
   reg.nome:= 'Pedro Pereira' ;
   reg.idade:= 40 ;
   write(arq, reg);

   // Grava dados no arquivo
   reg.nome:= 'Miguel da Silva' ;
   reg.idade:= 50 ;
   write(arq, reg);

   // Mostra os registros que foram gravados
   for i:= 0 to filesize(arq)-1 do
   Begin
      seek( arq, i );
      read( arq, reg );
      writeln( 'Leu do arquivo => nome:', reg.nome, ' idade:', reg.idade );
   End ;
End.
```


Arquivos text

Os arquivos do tipo texto são manipulados por meio de variáveis do tipo **text**. Os comandos para tratamento de arquivos desse tipo, implementados no compilador, são os seguintes:

- append
- assign
- close
- ioresult
- reset
- rewrite

Arquivos binários

Os arquivos binários são manipulados por meio de variáveis do tipo **file of**. Os comandos para tratamento de arquivos desse tipo, implementados no compilador, são os seguintes:

- append
- assign
- close
- filesize
- filepos
- ioresult
- reset
- rewrite
- seek

ioresult

Retorna o resultado da última operação realizada sobre um arquivo.

Se a função retornar 0, nenhum erro foi encontrado durante a operação. Caso contrário, foi encontrado um erro.

Sintaxe

```
function ioresult : integer ;
```

Exemplo

```
Program PascalZIM ;
var arq: text ;
Begin
   assign(arq, 'c:\dados.txt' );
   reset(arq);
   if (ioresult = 0) then
      writeln( 'O arquivo foi aberto para leitura' )
   else
      writeln( 'O arquivo nao pode ser aberto para leitura' );
End.
```


Comandos de desvio

Os comandos de desvio permitem interromper a execução sequencial de comandos.

Para comandos de repetição, os seguintes comandos de desvio são reconhecidos pelo compilador:

- break
- continue

O comando goto permite desviar a execução do programa para outro ponto.

O comando exit termina a execução do procedimento, função ou programa.

break

Comando usado para forçar a saída de uma estrutura de repetição (while, for, repeat).

Sintaxe

```
break ;
```

Onde:

- O comando deve estar dentro do corpo de uma estrutura de repetição.
- O próximo comando a ser executado após o break é o comando que segue a estrutura de repetição.

Exemplo

```
Program PascalZIM ;
var contador: integer ;
Begin
   contador:= 1 ;

   // Repetição que é executada 5 vezes
   while ( true ) do
      Begin
         writeln( 'Contador vale:' , contador );
         contador:= contador + 1 ;
         if ( contador > 5 ) then
            break
         else
            continue ;
      End ;

   // Impressão de uma mensagem após sair da repetição
   writeln( 'Agora estou fora do while!' );
End.
```


continue

Usado para desviar a execução dos comandos de uma estrutura de repetição (while, for, repeat) para a avaliação da condição de loop.

Sintaxe

```
continue ;
```

Onde:

- O comando deve estar dentro do corpo de uma estrutura de repetição.
- Após a execução do comando a repetição pode parar (se a condição de loop assim indicar) ou prosseguir com a execução do primeiro comando da repetição.

Exemplo

```
Program PascalZIM ;
var contador: integer ;
Begin
   contador:= 1 ;

   // Repetição que é executada 5 vezes
   while ( true ) do
      Begin
         writeln( 'Contador vale:' , contador );
         contador:= contador + 1 ;
         if ( contador > 5 ) then
            break
         else
            continue ;
      End ;

   // Impressão de uma mensagem após sair da repetição
   writeln( 'Agora estou fora do while!' );
End.
```


exit

Termina a execução do procedimento, função ou programa.

Sintaxe

```
exit ;
```

Funcionamento

- Se o comando aparecer dentro de um procedure, o procedimento será encerrado.
- Se o comando aparecer dentro de uma função, a função será encerrada.
- Se o comando aparecer dentro do programa principal, o programa será encerrado.

Exemplo

```
Program PascalZIM ;

procedure proc ;
   Begin
      writeln('abc');    // Mostra abc
      exit ;             // Sai do procedure
      writeln('def');    // Nunca chega aqui
   End ;

function func : integer ;
   Begin
      writeln( 'ghi' );   // Mostra ghi
```

```
      func:= 5 ;        // 5 é o valor retornado
      exit ;             // Sai da funcao
      writeln( 'jkl' );   // Nunca chega aqui
   End ;

Begin
   proc ;              // Mostra abc
   writeln(func);      // Mostra ghi, depois 5
   exit ;              // Sai do programa
   writeln( 'nada' );   // Nunca chega aqui
End.
```


goto

Comando usado para forçar o desvio do programa para um ponto específico.

Sintaxe

```
goto rotuloInstrucao ;
```

Quando o comando é executado, o programa é desviado para a instrução que foi rotulada com o nome informado.

Uma instrução pode ser rotulada da seguinte maneira:

```
rotuloInstrucao : instrução
```

Onde:

- `rotuloInstrucao` denota uma constante inteira ou um identificador válido na linguagem Pascal
- `instrucao` denota um comando na linguagem Pascal

O rótulo de uma instrução deve ainda ser declarado na seção de declaração de dados do programa, da seguinte maneira:

```
label rotuloInstrucao1, ...., rotuloInstrucaon ;
```

Exemplo

```
Program PascalZIM ;
   // Declaracao de rotulos para o comando goto
   label 1, 2, tres, 4, 5, 6, sete ;
   var x: integer ;
   // Declaracao de outros rotulos
   label 8, 9 ;
begin
   // Desvia o programa para o rotulo 'tres'
   goto tres ;
   1: writeln('linha 1');
   2: writeln('linha 2');
   tres: writeln('linha 3');
```

```
   4: writeln('linha 4');
   5: writeln('linha 5');
   6: writeln('linha 6');
   sete: writeln('linha 7');
   8: writeln('linha 8');
end.
```


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 a 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 subprograma 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, ou ainda um tipo definido pelo usuário (usando type).

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 ser recursivos.

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 os 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.

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 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 ;
```

Funções auxiliares

O compilador implementa funções e procedimentos em unidades.

As unidades implementadas pelo compilador são as seguintes:

- Unidade padrão
- Unidade graph

Unidade Graph

O compilador implementa as seguintes funções e procedimentos da unidade graph:

- Arc
- Bar
- Bar3D
- Circle
- ClearDevice
- ClearViewPort
- CloseGraph
- DetectGraph
- DrawPoly
- Ellipse
- FillEllipse
- FillPoly
- FloodFill
- GetArcCoords
- GetAspectRatio
- GetBkColor
- GetColor
- GetDefaultPalette
- GetDriverName
- GetFillPattern
- GetFillSettings
- GetGraphMode
- GetImage
- GetLineSettings
- GetMaxColor
- GetMaxMode
- GetMaxX
- GetMaxY
- GetModeName
- GetModeRange
- GetPalette
- GetPaletteSize
- GetPixel
- GetTextSettings

- GetViewSettings
- GetX
- GetY
- GraphDefaults
- GraphErrorMsg
- GraphResult
- ImageSize
- InitGraph
- Line
- LineRel
- LineTo
- MoveRel
- MoveTo
- OutText
- OutTextXY
- PieSlice
- PutImage
- PutPixel
- Rectangle
- RestoreCrtMode
- Sector
- SetActivePage
- SetAllPalette
- SetAspectRatio
- SetBkColor
- SetColor
- SetFillPattern
- SetFillStyle
- SetGraphBufSize
- SetGraphMode
- SetLineStyle
- SetPalette
- SetRGBPalette
- SetTextJustify
- SetTextStyle
- SetUserCharSize
- SetViewPort
- SetVisualPage
- SetWriteMode
- TextHeight
- TextWidth

As seguintes funções, presentes no Turbo Pascal, não foram implementadas:

- InstallUserDriver
- InstallUserFont
- RegisterBGIdriver
- RegisterBGIFont

Tipos

O compilador implementa os seguintes tipos da unidade graph:

- ArcCoordsType
- FillPatternType
- FillSettingsType
- LineSettingsType
- PaletteType
- PointType
- TextSettingsType
- ViewPortType

ArcCoordsType

Especifica as coordenadas da última chamada ao procedure Arc ou Ellipse.

Esse tipo é utilizado por GetArcCoords.

Sintaxe

```
type ArcCoordsType = record
    X : integer ;
    Y : integer ;
    Xstart : integer ;
    Ystart : integer ;
    Xend : integer ;
    Yend : integer ;
  end;
```

Nesse registro, temos o seguinte:

- (x, y) é a coordenada (x, y) onde está centralizado o arco
- (Xstart, Ystart) é a coordenada correspondente ao ponto no ângulo anglnicio
- (Xend, Yend) é a coordenada correspondente ao ponto no ângulo angFim

FillPatternType

Define um padrão de preenchimento definido pelo usuário.

Esse tipo é utilizado por GetFillPattern e SetFillPattern.

Sintaxe

```
type FillPatternType = array [1..8] of byte;
```

FillSettingsType

Define o padrão e cor usado para preencher uma área.

Esse tipo é utilizado por GetFillSettings.

Sintaxe

```
type FillSettingsType = record
   Pattern : integer ;
   Color : integer ;
end ;
```


LineSettingsType

Define o estilo, o padrão e a espessura para desenho de linhas.

Esse tipo é utilizado por GetLineSettings.

Sintaxe

```
type LineSettingsType = record
    LineStyle : integer;
    Pattern   : integer;
    Thickness : integer;
  end;
```


PaletteType

Define o tamanho e as cores usadas pela paleta.

Esse tipo é utilizado por GetPallete, GetDefaultPalette e SetAllPalette.

Sintaxe

```
const
    MaxColors = 15;

type PaletteType = record
      Size    : Byte;
      Colors  : array[0..MaxColors] of Shortint;
    end;
```

Na paleta padrão, a cor na posição zero define a cor de fundo da tela gráfica.

PointType

Especifica as coordenadas (x, y) de um ponto.

Sintaxe

```
type PointType = record
   x : integer ;
   y : integer ;
end ;
```


TextSettingsType

Define os atributos utilizados para impressão de texto.

Esse tipo é utilizado por GetTextSettings.

Sintaxe

```
type TextSettingsType = record
    Font      : integer;
    Direction : integer;
    CharSize  : integer;
    Horiz     : integer;
    Vert      : integer;
  end;
```


ViewPortType

Define o estado de uma janela (ViewPort).

Esse tipo é utilizado por GetViewSettings.

Sintaxe

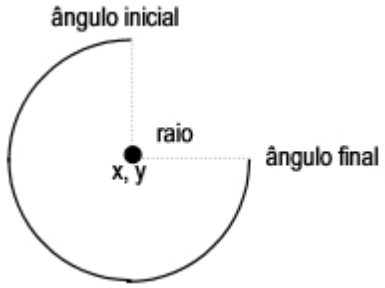
```
type ViewPortType = record
    x1, y1, x2, y2 : integer;
    clip           : boolean;
  end;
```

Nesse registro, temos o seguinte:

- (x1, y1) é a coordenada superior esquerda da janela
- (x2, y2) é a coordenada inferior direita da janela
- clip determina se os desenhos estão sendo realizados na janela.

Arc

Desenha parte de um círculo de raio r, centrado na coordenada (x, y), iniciando no ângulo angInicio e finalizando no ângulo angFim.



Sintaxe

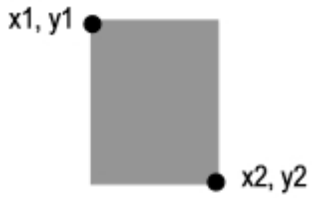
```
procedure Arc ( x, y, angInicio, angFim, r : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(BLUE);
  cleardevice;

  // Desenha arco amarelo de raio 40, com angulo entre 45 e 315, centrado em
(60, 70)
  setColor(yellow);
  arc(60, 70, 45, 315, 40);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```

Bar

Desenha um retângulo com borda nas coordenadas (x1, y1) e (x2, y2), preenchendo o mesmo com a cor e estilo de preenchimento atuais.



Sintaxe

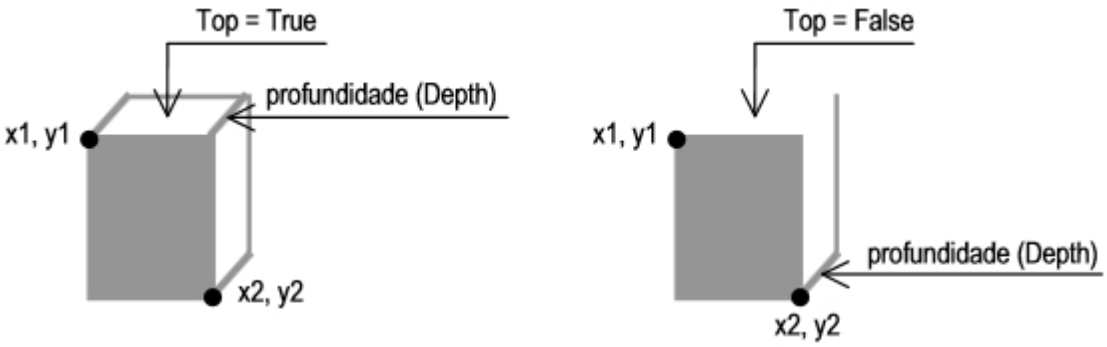
```
procedure Bar ( x1, y1, x2, y2 : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(BLUE);
  cleardevice;

  // Desenha retangulo com borda em (50, 50) e (300, 15)
   bar(20, 20, 100, 50);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


Bar3D

Desenha um bloco retangular tridimensional, com profundidade p, e borda nas coordenadas (x1, y1) e (x2, y2), preenchendo o mesmo com a cor e estilo de preenchimento atuais. Se top é true, então um topo tridimensional é desenhado.



Sintaxe

```
procedure Bar3D ( x1, y1, x2, y2, p: integer; top: boolean ) ;
```

Sintaxe

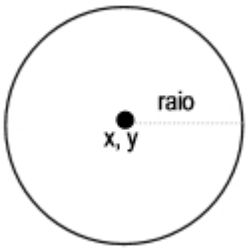
```
procedure Circle ( x, y : integer; r : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(BLUE);
  cleardevice;

  // Desenha bloco com borda em (50, 30) e (300, 15) e profundidade 20
  bar3d(50, 30, 300, 150, 20, true);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


Circle

Desenha um círculo de raio r, centrado na coordenada (x, y).



Sintaxe

```
procedure Circle ( x, y : integer; r : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(BLUE);
  cleardevice;

  // Define a cor de desenho em amarelo
  setColor(YELLOW);

  // Desenha circulo de raio 5 em (20, 10)
  circle(50, 50, 20);

  // Desenha ellipse de raio 20, com angulo entre 0 e 270, na coordenada (100,
50)
  ellipse(100, 50, 0, 270, 20, 20);

  // Desenha pizza verde, de raio 20, com angulo entre 45 e 315, na coordenada
(50, 100)
  setColor(green);
  pieSlice(50, 100, 45, 315, 20);

  // Desenha pizza amarela, de raio 20, com angulo entre 45 e 315, na
coordenada (100, 100)
  setColor(yellow);
  sector(100, 100, 45, 315, 20, 20);

  // Fecha o modo grafico
  readkey;
  closegraph;
```

```
End.
```


ClearDevice

Limpa a tela com cor de background atual. O cursor é coloca na coordenada (0,0).

Sintaxe

```
procedure ClearDevice ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(BLUE);
  cleardevice;

  // Define a cor de desenho em amarelo
  setColor(YELLOW);

  // Desenha circulo de raio 5 em (20, 10)
  circle(50, 50, 20);

  // Desenha ellipse de raio 20, com angulo entre 0 e 270, na coordenada (100,
50)
  ellipse(100, 50, 0, 270, 20, 20);

  // Desenha pizza verde, de raio 20, com angulo entre 45 e 315, na coordenada
(50, 100)
  setColor(green);
  pieSlice (50, 100, 45, 315, 20);

  // Desenha pizza amarela, de raio 20, com angulo entre 45 e 315, na
coordenada (100, 100)
  setColor(yellow);
  sector(100, 100, 45, 315, 20, 20);

  // Fecha o modo grafico
  readkey;
  closegraph;
```

```
End.
```


ClearViewport

Limpa a janela corrente com cor de background atual. O cursor é colocando na coordenada (0,0).

Sintaxe

```
procedure ClearViewPort ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela
  SetBkColor(white) ;
  cleardevice ;

  // Cria janela e pinta de verde
  SetViewPort(20, 20, 200, 100, false);
  SetBkColor(green) ;
  ClearViewPort;

  // Cria janela e pinta de azul
  SetViewPort(20, 120, 200, 200, false);
  SetBkColor(blue) ;
  ClearViewPort;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


CloseGraph

Finaliza o sistema gráfico.

Sintaxe

```
procedure CloseGraph ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(BLUE);
  cleardevice;

  // Define a cor de desenho em amarelo
  setColor(YELLOW);

  // Desenha circulo de raio 5 em (20, 10)
  circle(50, 50, 20);

  // Desenha ellipse de raio 20, com angulo entre 0 e 270, na coordenada (100,
50)
  ellipse(100, 50, 0, 270, 20, 20);

  // Desenha pizza verde, de raio 20, com angulo entre 45 e 315, na coordenada
(50, 100)
  setColor(green);
  pieSlice (50, 100, 45, 315, 20);

  // Desenha pizza amarela, de raio 20, com angulo entre 45 e 315, na
coordenada (100, 100)
  setColor(yellow);
  sector(100, 100, 45, 315, 20, 20);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


DetectGraph

Verifica o hardware e determina automaticamente qual driver e modo gráfico deve ser utilizado.

Sintaxe

```
procedure DetectGraph ( var driver, modo : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  detectGraph(driver, modo);
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Exibe o driver e modo selecionado
  writeln('Selecionado driver ', driver, ' e modo ', modo);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


DrawPoly

Desenha o contorno de um polígono usando o estilo de linha e cores atuais.

Sintaxe

```
procedure DrawPoly ( numeroPontos : integer; var pontosPoligono : PolyPoints )
;
```

O primeiro argumento especifica o número de pontos informados em *pontosPoligono*.

O segundo argumento especifica um vetor de elementos do tipo PointType.

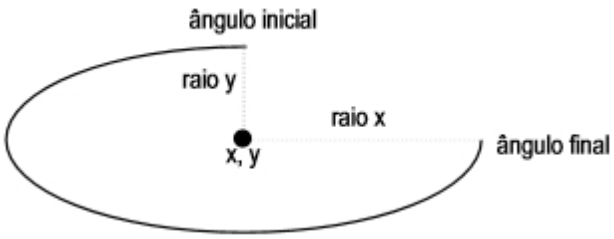
Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    vetPontos: array[1..6] of PointType ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');
```

```
  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Define os pontos do polígono
  vetPontos[1].x := 100 ;  vetPontos[1].y := 10;
  vetPontos[2].x := 150 ;  vetPontos[2].y := 160;
  vetPontos[3].x := 20 ;   vetPontos[3].y := 70;
  vetPontos[4].x := 180 ;  vetPontos[4].y := 70;
  vetPontos[5].x := 50 ;   vetPontos[5].y := 160;
  vetPontos[6].x := 100 ;  vetPontos[6].y := 10;

  // Desenha o contorno do polígono
  setColor(yellow);
  DrawPoly(6, vetPontos);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


Ellipse

Desenha uma elipse de raio horizontal rx, raio vertical ry, centrada na coordenada (x, y), iniciando no ângulo anglnicio e finalizando no ângulo angFim.



Sintaxe

```
procedure Ellipse ( x, y, angInicio, angFim, rx, ry : integer ) ;
```

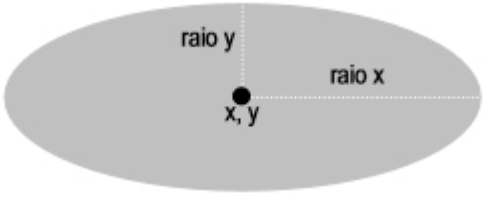
Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');
```

```
  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(BLUE);
  cleardevice;

  // Define a cor de desenho em amarelo
  setColor(YELLOW);

  // Desenha circulo de raio 5 em (20, 10)
  circle(50, 50, 20);

  // Desenha ellipse de raio 20, com angulo entre 0 e 270, na coordenada (100,
50)
  ellipse(100, 50, 0, 270, 20, 20);

  // Desenha pizza verde, de raio 20, com angulo entre 45 e 315, na coordenada
(50, 100)
  setColor(green);
  pieSlice (50, 100, 45, 315, 20);

  // Desenha pizza amarela, de raio 20, com angulo entre 45 e 315, na
coordenada (100, 100)
  setColor(yellow);
  sector(100, 100, 45, 315, 20, 20);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


FillEllipse

Desenha uma elipse de raio horizontal rx, raio vertical ry, centrada na coordenada (x, y), preenchendo a mesma com a cor e estilo de preenchimento atuais.



Sintaxe

```
procedure FillEllipse ( x, y, rx, ry : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(BLUE);
  cleardevice;

  // Desenha ellipse
  fillellipse(100, 70, 80, 50);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


FillPolly

Desenha os contornos de um polígono, depois preenche um polígono.

Sintaxe

```
procedure FillPoly ( numeroPontos : integer; var pontosPoligono : PolyPoints )
;
```

O primeiro argumento especifica o número de pontos informados em *pontosPoligono*.

O segundo argumento especifica um vetor de elementos do tipo PointType.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    vetPontos: array[1..6] of PointType ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
```

```
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Define os pontos do polígono
  vetPontos[1].x := 100 ;  vetPontos[1].y := 10;
  vetPontos[2].x := 150 ;  vetPontos[2].y := 160;
  vetPontos[3].x := 20 ;   vetPontos[3].y := 70;
  vetPontos[4].x := 180 ;  vetPontos[4].y := 70;
  vetPontos[5].x := 50 ;   vetPontos[5].y := 160;
  vetPontos[6].x := 100 ;  vetPontos[6].y := 10;

  // Desenha o polígono
  setFillStyle(solidFill, green);
  FillPoly(6, vetPontos);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


FloodFill

Preenche a área que contém o ponto (x, y). Essa área está limitada pela corBorda.

Sintaxe

```
procedure FloodFill ( x, y, corBorda : integer ) ;
```

Nesse caso, a área é preenchida com o padrão definido por SetFillStyle.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela
  SetBkColor(white) ;
  cleardevice ;
```

```
  // Desenha linha e dois retangulos vermelhos
  SetColor(red);
  Line(0, 0, 400, 400);
  Rectangle(300, 300, 400, 400);
  Rectangle(320, 320, 380, 380);

  // Define padrao de preenchimento amarelo
  SetFillStyle(SolidFill, yellow);

  // Preenche regiao do ponto (315, 310) delimitada pela cor vermelha
  FloodFill(315, 310, red);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetArcCoords

Recupera as coordenadas dos pontos correspondentes ao último comando Arc.

Sintaxe

```
procedure GetArcCoords ( var coordenadas : ArcCoordsType ) ;
```

O argumento especifica um registro do tipo ArcCoordsType, que vai armazenar os dados desejados.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    coordenadasArco: ArcCoordsType ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Desenha arco, pega suas coordenadas
  Arc(100, 100, 0, 270, 30);
  GetArcCoords(coordenadasArco);

  // Desenha linha partindo do centro do arco até suas extremidades
  with coordenadasArco do
```

```
  begin
    Line(x, y, Xstart, Ystart);
    Line(x, y, Xend, Yend);
  end ;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetAspectRatio

Recupera, nas variáveis xasp e yasp, a proporção efetiva da tela.

Sintaxe

```
procedure GetAspectRatio ( var xasp, yasp : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    xasp, yasp : integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Define a cor de desenho em amarelo
  setColor(YELLOW);

  // Desenha moldura na janela
  getAspectRatio(xasp, yasp);
  rectangle(10, 10, xasp-10, yasp-10);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetBkColor

Retorna a cor de fundo atual.

Sintaxe

```
function GetBkColor : integer ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

 // Limpa a tela com a cor de fundo vermelho
 setbkcolor(RED);
 cleardevice;
 readkey;

  // Limpa a tela com a cor de fundo verde
 setbkcolor(GREEN);
 cleardevice;
 readkey;

 // Limpa a tela com a cor de fundo azul
 setbkcolor(BLUE);
 cleardevice;
 readkey;

  // Fecha o modo grafico
  closegraph;
End.
```


GetColor

Retorna a cor de desenho atual.

Sintaxe

```
function GetColor : integer ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;

// Funcao utilizada para converter um inteiro em string
function intToStr(valor: integer) : string ;
var s : string ;
begin
  str(valor, s);
  intToStr := s ;
End;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Escreve mensagem
  setcolor(lightgreen);
  outTextXY(10, 10, 'Estou usando a cor ' + intToStr(getColor));

  setcolor(red);
  outTextXY(10, 30, 'Estou usando a cor ' + intToStr(getColor));

  setcolor(green);
  outTextXY(10, 50, 'Estou usando a cor ' + intToStr(getColor));

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetDefaultPalette

Recupera a palheta inicial do modo gráfico, configurada de acordo com a chamada a InitGraph.

Sintaxe

```
procedure GetDefaultPalette ( var palheta: PalleteType ) ;
```

O argumento especifica um registro do tipo PalleteType, que armazena as cores e o tamanho da palheta de desenho.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo, i: integer ;
    palheta, palhetaPadrao: PaletteType;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Recupera a palheta padrao
  GetDefaultPalette(palhetaPadrao);

  // Escreve linhas de texto com as cores da palheta padrao
  for i := 0 to palhetaPadrao.size - 1 do
  begin
    SetColor(palhetaPadrao.colors[i]);
    OutTextXY(10, I * 10, 'Pressione uma tecla para continuar, outra para
sair');
  end;
  readkey ;

  // Muda as cores da palheta ate que uma tecla seja pressionada
  palheta := palhetaPadrao;
  repeat
    palheta.colors[Random(palheta.size)] := Random(palheta.size + 1);
    delay(1000);
    SetAllPalette(palheta);
  until KeyPressed;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetDriverName

Retorna o nome do driver utilizado pelo sistema gráfico.

Sintaxe

```
function GetDriverName : string ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;

// Funcao utilizada para converter um inteiro em string
function intToStr(valor: integer) : string ;
var s : string ;
begin
  str(valor, s);
  intToStr := s ;
End;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Escreve mensagem
  setcolor(lightgreen);
  outTextXY(10, 10, 'Estou usando o modo ' + intToStr(getGraphMode));
  outTextXY(10, 30, 'O nome desse modo eh ' + getModeName(getGraphMode)) ;
  outTextXY(10, 50, 'Estou usando o driver ' + getDriverName) ;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetFillSettings

Recupera o padrão e cor que estão sendo utilizados para preenchimento de formas.

Sintaxe

```
procedure GetFillSettings ( var dadosPreenchimento : FillSettingsType ) ;
```

O argumento especifica um registro do tipo FillSettingsType, que vai armazenar os dados desejados.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    dadosPreenchimento: FillSettingsType;
```

```
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Define um estilo para preenchimento
  SetFillStyle(XHatchFill, yellow);
  Bar(0, 0, 50, 50);

  // Guarda o estilo
  GetFillSettings(dadosPreenchimento);

  // Define novo estilo para preenchimento
  SetFillStyle(SlashFill, green);
  Bar(50, 0, 100, 50);

  // Recupera o primeiro estilo
  SetFillStyle(dadosPreenchimento.Pattern, dadosPreenchimento.Color);
  Bar(100, 0, 150, 50);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetFillPattern

Recupera o padrão de preenchimento que está sendo utilizado para preenchimento de formas.

Sintaxe

```
procedure GetFillPattern ( var padraoPreenchimento : FillPatternType ) ;
```

O argumento especifica um registro do tipo FillPatternType, que vai armazenar os dados desejados.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;

// Cria novo padrao de preenchimento
const
   padraoPontilhado : FillPatternType = ($AA, $55, $AA, $55, $AA, $55, $AA,
$55);
```

```
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Guarda o padrao de preenchimento inicial
  GetFillPattern(padraoInicial);

  // Define novo padrao para preenchimento de formas
  SetFillPattern(padraoPontilhado, lightgreen);

   // Desenha retangulo com novo padrao de preenchimento
   Bar(0, 0, 100, 100);
   readkey;

   // Retorna ao padrao de preenchimento inicial
   SetFillPattern(padraoInicial, yellow);

   // Desenha retangulo com antigo padrao de preenchimento
   Bar(0, 101, 100, 200);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetGraphMode

Retorna o modo gráfico definido na inicialização do sistema gráfico.

Sintaxe

```
function GetGraphMode : integer ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;

// Funcao utilizada para converter um inteiro em string
function intToStr(valor: integer) : string ;
var s : string ;
begin
  str(valor, s);
  intToStr := s ;
```

```
End;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Escreve mensagem
  setcolor(lightgreen);
  outTextXY(10, 10, 'Estou usando o modo ' + intToStr(getGraphMode));
  outTextXY(10, 30, 'O nome desse modo eh ' + getModeName(getGraphMode)) ;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetImage

Salva, na variável bitmap, uma cópia dos pixels armazenados na região que está delimitada pelas coordenadas (x1, y1) e (x2, y2).

Sintaxe

```
procedure GetImage( x1, y1, x2, y2: integer; var bitmap ) ;
```

A variável bitmap, nesse caso, corresponde a um vetor de inteiros no qual:

- A primeira posição guarda o comprimento da região informada
- A segunda posição guarda a altura da região informada
- As posições seguintes guardam, cada uma, a cor de um pixel da região

Exemplo

```
Program Pascalzim ;
uses graph ;

var driver, modo: integer ;
```

```
    tamanhoImagem: integer ;
    bitmap : ^integer ;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Desenha retangulo nas coordenadas (10, 40) e (50,100)
  bar(10, 40, 50, 100);

  // Guarda imagem do retangulo no bitmap
  tamanhoImagem := imagesize(10, 40, 50, 100);
  getmem(bitmap, tamanhoImagem);
  getimage(10, 40, 50, 100, bitmap^);

  // Desenha o bitmap na coordenada (70, 40)
  putimage(70, 40, bitmap^, NormalPut);
  freemem(bitmap, tamanhoImagem);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetLineSettings

Recupera as configurações do modo gráfico para desenho de linhas.

Sintaxe

```
procedure GetLineSettings ( var configLinhas : LineSettingsType ) ;
```

O argumento especifica um registro do tipo LineSettingsType, que vai armazenar os dados desejados.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    estiloInicial: LineSettingsType;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');
```

```
  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Exibe linha com o estilo inicial do modo grafico
  Line(0, 20, getmaxx, 20);

  // Guarda estilo inicial
  GetLineSettings(estiloInicial);

  // Exibe linha com novo estilo de linha
  SetLineStyle(Dashedln, 0, NormWidth);
  Line(0, 50, getmaxx, 50);

  // Exibe linha com o estilo inicial
  with estiloInicial do
  begin
    SetLineStyle(LineStyle, Pattern, Thickness);
  end;
  Line(0, 80, getmaxx, 80) ;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetMaxColor

Retorna a maior cor suportada pelo modo gráfico.

Sintaxe

```
function GetMaxColor : integer ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
```

```
  end;

  // Desenha circulos de cores aleatórias até pressionar alguma tecla
  Randomize;
  repeat
    SetColor(Random(GetMaxColor)+1);
    circle(Random(GetMaxX), Random(GetMaxY), 10);
    delay(5);
  until KeyPressed;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetMaxMode

Retorna o maior modo gráfico suportado pelo driver gráfico carregado.

Sintaxe

```
function GetMaxMode : integer ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo, i: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Mostra o nome de todos os modos gráficos suportados
  setColor(lightGreen);
  for i := 0 to GetMaxMode do
  begin
    OutTextXY(10, 10 * Succ(I), GetModeName(I));
  end ;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```

GetMaxX

Retorna o número de colunas (largura) da tela.

Sintaxe

```
function GetMaxX : integer ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo, i: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo verde
  setbkcolor(green);
  cleardevice;

  // Desenha mil linhas em posicoes aleatorias
  for i:= 1 to 1000 do
  begin
    setcolor(1 + random (15));
    line(random(getmaxx()), random(getmaxy()), random(getmaxx()),
random(getmaxy()));
    delay(10);
  end;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetMaxY

Retorna o número de linhas (altura) da tela.

Sintaxe

```
function GetMaxY : integer ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo, i: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo verde
  setbkcolor(green);
  cleardevice;

  // Desenha mil linhas em posicoes aleatorias
  for i:= 1 to 1000 do
  Begin
    setcolor(1 + random (15));
    line(random(getmaxx()), random(getmaxy()), random(getmaxx()),
random(getmaxy()));
    delay(10);
  End;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetModeName

Retorna o nome do modo gráfico informado.

Sintaxe

```
function GetModeName(GraphMode : integer) : string ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;

// Funcao utilizada para converter um inteiro em string
function intToStr(valor: integer) : string ;
var s : string ;
```

```
begin
  str(valor, s);
  intToStr := s ;
End;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Escreve mensagem
  setcolor(lightgreen);
  outTextXY(10, 10, 'Estou usando o modo ' + intToStr(getGraphMode));
  outTextXY(10, 30, 'O nome desse modo eh ' + getModeName(getGraphMode)) ;
  outTextXY(10, 50, 'Estou usando o driver ' + getDriverName) ;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GetModeRange

Retorna o maior e o menor modo gráfico suportado pelo driver informado.

Sintaxe

```
procedure GetModeRange( driver: integer; var modoMinimo, modoMaximo :
integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var menorModo, maiorModo: integer;
Begin
  GetModeRange(VGAHI, menorModo, maiorModo);
  writeln('Menor modo gráfico = ', menorModo);
  writeln('Maior modo gráfico = ', maiorModo);
End.
```


GetPalette

Recupera a palheta atual e seu tamanho.

Sintaxe

```
procedure GetPalette ( var palheta: PalleteType ) ;
```

O argumento especifica um registro do tipo PalleteType, que armazena as cores e o tamanho da palheta de desenho.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    palheta: PaletteType;
    posicao, cor: integer ;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Recupera a palheta utilizada pelo programa
  GetPalette(palheta);

  // Desenha linhas com as cores da palheta
  for posicao := 0 to pred(GetPaletteSize) do
  begin
     setColor(palheta.colors[posicao]);
     Line(0, posicao * 5, 100, posicao * 5);
  end;

  // Modifica as cores da palheta
  randomize;
  repeat
    posicao:= Random(palheta.size) ;
    cor := Random(palheta.size);
    SetPalette(posicao, cor);
    delay(500);
  until keyPressed;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```

GetPaletteSize

Retorna o tamanho da palheta de desenho.

Sintaxe

```
function GetPaletteSize: integer ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    palheta: PaletteType;
    posicao, cor: integer ;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Recupera a palheta utilizada pelo programa
  GetPalette(palheta);

  // Desenha linhas com as cores da palheta
  for posicao := 0 to pred(GetPaletteSize) do
  begin
     setColor(palheta.colors[posicao]);
     Line(0, posicao * 5, 100, posicao * 5);
  end;

  // Modifica as cores da palheta
  randomize;
  repeat
    posicao:= Random(palheta.size) ;
    cor := Random(palheta.size);
    SetPalette(posicao, cor);
    delay(500);
  until keyPressed;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```

GetPixel

Retorna a cor do pixel que da coordenada (x, y).

Sintaxe

```
function GetPixel ( x, y : integer ) : integer ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;

// Funcao utilizada para converter um inteiro em string
function intToStr(valor: integer) : string ;
var s : string ;
begin
  str(valor, s);
  intToStr := s ;
End;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(blue);
  cleardevice;

  // Pinta pixels nas coordenadas (30,40) e (35,40)
  putpixel(30, 40, yellow);
  putpixel(35, 40, lightgreen);
  setColor(green);
  outTextXY(30, 60, 'O pixel na coordenada (30,40) tem cor ' +
intToStr(getpixel(30,40)));
  outTextXY(30, 80, 'O pixel na coordenada (35,40) tem cor ' +
intToStr(getpixel(35,40)));

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```

GetTextSettings

Recupera as configurações do modo gráfico para escrita de texto.

Sintaxe

```
procedure GetTextSettings ( var configText : TextSettingsType ) ;
```

O argumento especifica um registro do tipo TextSettingsType, que vai armazenar os dados desejados.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    estiloInicial: TextSettingsType;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Exibe texto com o estilo inicial do modo grafico
  setcolor(lightgreen);
  SetTextJustify(LeftText, CenterText) ;
  OutTextXY(10, 10, 'Texto com estilo inicial');

  // Guarda estilo inicial
  GetTextSettings(estiloInicial);

  // Define novo estilo de texto
  SetTextStyle(SansSerifFont, VertDir, 4);
  OutTextXY(10, 220, 'Novo estilo de texto');

  // Exibe texto com o estilo inicial
  with estiloInicial do
  begin
    SetTextJustify(Horiz, Vert);
    SetTextStyle(Font, Direction, CharSize);
  end;
  OutTextXY(10, 30, 'Outro texto com estilo inicial');

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```

GetViewSettings

Recupera as configurações do modo gráfico para a janela (viewport) atual.

Sintaxe

```
procedure GetViewSettings ( var configView : ViewPortType ) ;
```

O argumento especifica um registro do tipo ViewPortType, que vai armazenar os dados desejados.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    estiloJanela : ViewPortType ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Define janela nas coordenadas (10,10) (300,100)
  setViewPort(10, 10, 300, 100, ClipOn);

  // Recupera as configurações da janela atual
  GetViewSettings(estiloJanela);

  // Desenha moldura tomando como base a janela corrente
  setcolor(lightgreen);
  with estiloJanela do
  begin
    Rectangle(x1+20, Y1+20, X2-20, Y2- 20);
    if Clip then
      OutTextxy(x1 + 30, y1 + (y2-y1) div 2,'Clip ativado')
    else
      OutTextxy(x1 + 30, y1 + (y2-y1) div 2,'Clip desativado');
  end;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```

GetX

Retorna a coluna (coordenada x) onde está posicionado o cursor gráfico.

Sintaxe

```
function GetX : integer ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    xs, ys: string;

// Funcao utilizada para converter um inteiro em string
function intToStr(valor: integer) : string ;
var s : string ;
begin
  str(valor, s);
  intToStr := s ;
End;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(blue);
  cleardevice;

  // Escreve mensagem na coordenada (30, 40)
  moveTo(30,40);
  xs := intToStr(getx);
  ys := intToStr(gety);
  setcolor(lightgreen);
  outText('Estou escrevendo na coordenada (' + xs + ',' + ys + ')');

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```

GetY

Retorna a linha (coordenada y) onde está posicionado o cursor gráfico.

Sintaxe

```
function GetY : integer ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    xs, ys: string;

// Funcao utilizada para converter um inteiro em string
function intToStr(valor: integer) : string ;
var s : string ;
begin
  str(valor, s);
  intToStr := s ;
End;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(blue);
  cleardevice;

  // Escreve mensagem na coordenada (30, 40)
  moveTo(30,40);
  xs := intToStr(getx);
  ys := intToStr(gety);
  setcolor(lightgreen);
  outText('Estou escrevendo na coordenada (' + xs + ',' + ys + ')');

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GraphDefaults

Inicializa todas as variáveis do sistema gráfico. Essas variáveis incluem:

- viewport
- palhetas
- cores de desenho e background
- estilo e padrão de linha
- estilo, cor e padrão de preenchimento
- fonte ativa, estilo e alinhamento de texto

Sintaxe

```
procedure GraphDefaults ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Escreve texto em amarelo
  SetColor(yellow);
  OutText('Texto escrito em cor amarela');
  readkey;

  // Limpa o modo grafico e escreve novamente
  ClearViewPort;
  GraphDefaults;
  OutText('Texto escrito na cor padrao');

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GraphErrorMsg

Retorna a mensagem de erro correspondente ao código informado.

Sintaxe

```
function GraphErrorMsg ( codigoErro : integer; ) : string ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end
  else
  begin
    writeln('Modo grafico iniciado com sucesso');
  end;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


GraphResult

Retorna um código de erro para a última operação gráfica.

Sintaxe

```
function GraphResult : integer ;
```

Os valores possíveis de retorno para essa função são os seguintes:

Valor retornado	Constante
0	grOk
1	grNoInitGraph
2	grNotDetected
3	grFileNotFound
4	grInvalidDriver
5	grNoLoadMem
6	grNoScanMem
7	grNoFloodMem
8	grFontNotFound
9	grNoFontMem

10	grInvalidMode
11	grError
12	grIOerror
13	grInvalidFont
14	grInvalidFontNum

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end
  else
  begin
    writeln('Modo grafico iniciado com sucesso');
  end;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


ImageSize

Retorna o número de bytes necessários para armazenar a imagem no retângulo definido pelas coordenadas (x1, y1) e (x2, y2).

Sintaxe

```
function ImageSize( x1, y1, x2, y2: integer ) : integer ;
```

Exemplo

```
Program Pascalzim ;
```

```
uses graph ;

var driver, modo: integer ;
    tamanhoImagem: integer ;
    bitmap : ^integer ;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Desenha retangulo nas coordenadas (10, 40) e (50,100)
  bar(10, 40, 50, 100);

  // Guarda imagem do retangulo no bitmap
  tamanhoImagem := imagesize(10, 40, 50, 100);
  getmem(bitmap, tamanhoImagem);
  getimage(10, 40, 50, 100, bitmap^);

  // Desenha o bitmap na coordenada (70, 40)
  putimage(70, 40, bitmap^, NormalPut);
  freemem(bitmap, tamanhoImagem);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


InitGraph

Inicializa o sistema gráfico.

Sintaxe

```
procedure InitGraph ( var driver, modo : integer; caminhoDriver : string ) ;
```

Se driver armazena o valor dado pela constante Detect, o sistema gráfico detecta automaticamente o driver e o modo.

Se driver armazena um valor diferente de Detect, esse valor é assumido como o número do driver a ser utilizado. Esse driver é então selecionado, e o sistema é colocado no modo especificado.

Os seguintes valores são válidos para a variável driver:

- DETECT
- VGA

- CUSTOM

Os seguintes valores são válidos para a variável modo:

- VGALO
- VGAMED
- VGAHI
- GM_640x480
- GM_800x600
- GM_1024x768
- GM_NOPALETTE

Depois de chamar o procedimento, driver passa a armazenar o driver gráfico atual, e modo passa a armazenar o modo gráfico atual.

Se um erro ocorrer, a função GraphResult retornará um valor diferente de grOk.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(BLUE);
  cleardevice;

  // Define a cor de desenho em amarelo
  setColor(YELLOW);

  // Desenha circulo de raio 5 em (20, 10)
  circle(50, 50, 20);

  // Desenha ellipse de raio 20, com angulo entre 0 e 270, na coordenada (100,
50)
  ellipse(100, 50, 0, 270, 20, 20);

  // Desenha pizza verde, de raio 20, com angulo entre 45 e 315, na coordenada
(50, 100)
  setColor(green);
  pieSlice (50, 100, 45, 315, 20);

  // Desenha pizza amarela, de raio 20, com angulo entre 45 e 315, na
coordenada (100, 100)
  setColor(yellow);
  sector(100, 100, 45, 315, 20, 20);
```

```
  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


Line

Desenha uma linha que vai da coordenada (x1, y1) até a coordenada (x2, y2).

Sintaxe

```
procedure Line ( x1, y1, x2, y2 : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo, i: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo verde
  setbkcolor(green);
  cleardevice;

  // Desenha mil linhas em posicoes aleatorias
  for i:= 1 to 1000 do
  begin
    setcolor(1 + random (15));
    line(random(getmaxx()), random(getmaxy()), random(getmaxx()),
random(getmaxy()));
    delay(10);
  end;

  // Fecha o modo grafico
  readkey;
  closegraph;
```

```
End.
```


LineRel

Desenha uma linha que vai da coordenada atual (x, y) até a coordenada (x + dx, y + dy).

Sintaxe

```
procedure LineRel ( dx, dy : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(blue);
  cleardevice;

  // Desenha um triangulo amarelo
  setcolor(yellow);
  moveTo(50, 50);
  lineRel(0, 150);
  lineRel(150, 0);
  lineRel(-150, -150);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


LineTo

Desenha uma linha que vai da coordenada atual até a coordenada (x, y).

Sintaxe

```
procedure LineTo ( x, y : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo amarela
  setbkcolor(yellow);
  cleardevice;

  // Desenha um triangulo azul
  setcolor(blue);
  moveTo(50, 50);
  lineTo(50, 200);
  lineTo(200, 200);
  lineTo(50, 50);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


MoveRel

Move o cursor para a coordenada (x + dx, y + dy), assumindo que (x, y) é a coordenada atual.

Sintaxe

```
procedure MoveRel ( dx, dy : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');
```

```
  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(blue);
  cleardevice;

  // Escreve tres linhas em texto verde
  setColor(lightgreen);
  OutTextXY(25,20,'Bem vindo ao modo grafico do Pascalzim');
  OutTextXY(25,40,'Experimente o desenho de circulos e linhas');

  // Move o cursor para a coordenada (25, 60)
  MoveTo(25, 40);
  MoveRel(0, 20);
  OutText('Deixe sua imaginacao fluir');

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


MoveTo

Move o cursor para a coordenada (x, y).

Sintaxe

```
procedure MoveTo ( x, y : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo amarela
  setbkcolor(yellow);
```

```
  cleardevice;

  // Desenha um triangulo azul
  setcolor(blue);
  moveTo(50, 50);
  lineTo(50, 200);
  lineTo(200, 200);
  lineTo(50, 50);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


OutText

Escreve o texto na posição corrente da tela, usando as configurações atuais de fonte e texto.

Sintaxe

```
procedure OutText ( texto : string ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(blue);
  cleardevice;

  // Escreve tres linhas em texto verde
  setColor(lightgreen);
  OutTextXY(25,20,'Bem vindo ao modo grafico do Pascalzim');
  OutTextXY(25,40,'Experimente o desenho de circulos e linhas');

  // Move o cursor para a coordenada (25, 60)
  MoveTo(25, 40);
  MoveRel(0, 20);
  OutText('Deixe sua imaginacao fluir');

  // Fecha o modo grafico
```

```
  readkey;
  closegraph;
End.
```


OutTextXY

Escreve o texto na coordenada (x, y), usando as configurações atuais de fonte e texto.

Sintaxe

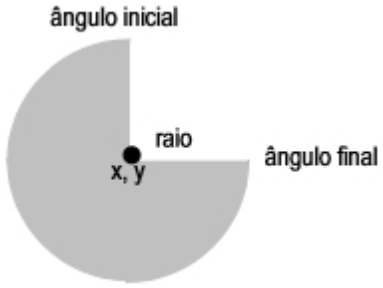
```
procedure OutTextXY ( x, y : integer ; texto : string ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(blue);
  cleardevice;

  // Escreve tres linhas em texto verde
  setColor(lightgreen);
  OutTextXY(25,20,'Bem vindo ao modo grafico do Pascalzim');
  OutTextXY(25,40,'Experimente o desenho de circulos e linhas');

  // Move o cursor para a coordenada (25, 60)
  MoveTo(25, 40);
  MoveRel(0, 20);
  OutText('Deixe sua imaginacao fluir');

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


PieSlice

Desenha o setor de um círculo de raio r na coordenada (x, y), iniciando no angulo anglnicio e terminando no angulo angFim. O setor é preenchido com a cor atual de primeiro plano.



Sintaxe

```
procedure PieSlice ( x, y, angInicio, angFim, r : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(BLUE);
  cleardevice;

  // Define a cor de desenho em amarelo
  setColor(YELLOW);

  // Desenha circulo de raio 5 em (20, 10)
  circle(50, 50, 20);

  // Desenha ellipse de raio 20, com angulo entre 0 e 270, na coordenada (100,
50)
  ellipse(100, 50, 0, 270, 20, 20);

  // Desenha pizza verde, de raio 20, com angulo entre 45 e 315, na coordenada
(50, 100)
  setColor(green);
  pieSlice (50, 100, 45, 315, 20);

  // Desenha pizza amarela, de raio 20, com angulo entre 45 e 315, na
coordenada (100, 100)
  setColor(yellow);
```

```
  sector(100, 100, 45, 315, 20, 20);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


PutImage

Desenha o bitmap na coordenada (x1, y1).

Sintaxe

```
procedure PutImage( x1, y1: integer; var bitmap; modo: integer ) ;
```

A variável bitmap, nesse caso, corresponde a um vetor de inteiros no qual:

- A primeira posição guarda o comprimento da região informada
- A segunda posição guarda a altura da região informada
- As posições seguintes guardam, cada uma, a cor de um pixel da região

O modo, por sua vez, pode assumir os seguintes valores:

- NormalPut
- CopyPut
- XORPut
- ORPut
- AndPut
- NotPut

Exemplo

```
Program Pascalzim ;
uses graph ;

var driver, modo: integer ;
    tamanhoImagem: integer ;
    bitmap : ^integer ;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
```

```
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Desenha retangulo nas coordenadas (10, 40) e (50,100)
  setFillStyle(BkSlashFill, lightcyan) ;
  bar(10, 40, 50, 100);

  // Guarda imagem do retangulo no bitmap
  tamanhoImagem := imagesize(10, 40, 50, 100);
  getmem(bitmap, tamanhoImagem);
  getimage(10, 40, 50, 100, bitmap^);

  // Desenha o bitmap na coordenada (70, 40)
  putimage(70, 40, bitmap^, NormalPut);
  freemem(bitmap, tamanhoImagem);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


PutPixel

Desenha um pixel na coordenada (x, y) com a cor c.

Sintaxe

```
procedure PutPixel ( x, y, c : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo, lin, col: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(blue);
  cleardevice;
```

```
  // Pinta pixels das linhas de 10 a 100
  for lin := 10 to 100 do
    for col:= 1 to 500 do
      putpixel(50+col, 50+lin, random(16));

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


Rectangle

Desenha um retângulo com borda nas coordenadas (x1, y1) e (x2, y2), usando a cor e estilo atuais.

Sintaxe

```
procedure Rectangle ( x1, y1, x2, y2 : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(blue);
  cleardevice;

  // Desenha um retangulo verde com borda em (50, 50) e (300, 15)
  setcolor(green);
  rectangle(50, 50, 300, 150);

  // Fecha o modo grafico
  readkey;
```

```
  closegraph;
End.
```


RestoreCrtMode

Restaura a tela para o modo que estava ativo antes da inicialização do modo gráfico.

Sintaxe

```
procedure RestoreCrtMode ;
```

Na implementação do compilador, essa função não faz nada.

No Turbo Pascal essa função era utilizada para alternar entre o modo gráfico e o modo console.

Assim, no modo gráfico podia-se chamar essa função para alternar para o modo console e utilizar comandos de impressão em console (writeln).

No Pascalzim não é necessário chamar essa função, pois ele mantém duas janelas separadas: a janela de console e a janela gráfica.

Assim, no compilador, os comandos para leitura e impressão em console são direcionados para aquela janela, enquanto os comandos gráficos são direcionados para a janela gráfica.

Sector

Desenha com preenchimento o setor de uma elipse de raio horizontal rx, raio vertical ry, centrado na coordenada (x, y), com início no angulo anglnicio e término no angulo angFim.

Sintaxe

```
procedure Sector ( x, y, angInicio, angFim, rx, ry : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
```

```
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(BLUE);
  cleardevice;

  // Define a cor de desenho em amarelo
  setColor(YELLOW);

  // Desenha circulo de raio 5 em (20, 10)
  circle(50, 50, 20);

  // Desenha ellipse de raio 20, com angulo entre 0 e 270, na coordenada (100,
50)
  ellipse(100, 50, 0, 270, 20, 20);

  // Desenha pizza verde, de raio 20, com angulo entre 45 e 315, na coordenada
(50, 100)
  setColor(green);
  pieSlice (50, 100, 45, 315, 20);

  // Desenha pizza amarela, de raio 20, com angulo entre 45 e 315, na
coordenada (100, 100)
  setColor(yellow);
  sector(100, 100, 45, 315, 20, 20);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


SetActivePage

Define qual página será utilizada como saída (estará ativa) para os comandos de desenho informados em seguida.

Sintaxe

```
procedure SetActivePage( pagina: integer ) ;
```

A tela inicial do modo gráfico corresponde à página de número zero.

Nessa implementação, o compilador suporta somente duas páginas, zero e um, respectivamente.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Escreve uma mensagem na tela oculta
  SetActivePage(1);
  setcolor(LIGHTGREEN) ;
  OutTextXY(10, 10, 'Essa mensagem esta na pagina 1');

  // Escreve uma mensagem na tela
  SetActivePage(0);
  setcolor(YELLOW) ;
  OutTextXY(10, 10, 'Essa mensagem esta na pagina 0');

  // Exibe tela oculta
  SetVisualPage(1);
  readkey ;

  // Exibe tela inicial
  SetVisualPage(0);
  readkey ;

  // Escreve outra mensagem na tela oculta
  SetActivePage(1);
  OutTextXY(10, 30, 'Essa outra mensagem esta na pagina 1');
  // Exibe novamente a tela oculta
  SetVisualPage(1);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


SetAllPalette

Altera as cores da palheta de desenho atual para as cores especificadas na nova palheta.

Sintaxe

```
procedure SetAllPalette ( var novaPalheta: PalleteType ) ;
```

O argumento especifica um registro do tipo PalleteType, que armazena as cores e o tamanho da palheta de desenho.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    posicao, cor: integer ;
    novaPaleta, palheta: PaletteType;

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Define nova paleta de cores
  with novaPaleta do
  begin
    Size := 4;
    Colors[0] := black ;
    Colors[1] := lightmagenta ;
    Colors[2] := lightgreen ;
    Colors[3] := lightcyan ;
    SetAllPalette(novaPaleta);
  end;

  // Recupera a palheta utilizada pelo programa
  GetPalette(palheta);

  // Desenha linhas com as cores da palheta
  SetLineStyle(DashedLn, 0, NormWidth);
  for posicao := 0 to pred(GetPaletteSize) do
  begin
     cor := palheta.colors[posicao];
     setColor(cor);
     Line(0, 10 + posicao * 10, 100, 10 + posicao * 10);
  end;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


SetAspectRatio

Define a proporção efetiva da tela.

Sintaxe

```
procedure SetAspectRatio( Xasp, Yasp: integer ) ;
```


SetBkColor

Define a cor de background da tela.

Sintaxe

```
procedure SetBkColor ( cor : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(blue);
  cleardevice;

  // Define a cor de desenho em amarelo
  setColor(yellow);

  // Desenha circulo de raio 5 em (20, 10)
  circle(50, 50, 20);

  // Desenha ellipse de raio 20, com angulo entre 0 e 270, na coordenada (100,
50)
  ellipse(100, 50, 0, 270, 20, 20);

  // Desenha pizza verde, de raio 20, com angulo entre 45 e 315, na coordenada
(50, 100)
  setColor(green);
  pieSlice (50, 100, 45, 315, 20);

  // Desenha pizza amarela, de raio 20, com angulo entre 45 e 315, na
coordenada (100, 100)
```

```
  setColor(yellow);
  sector(100, 100, 45, 315, 20, 20);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


SetColor

Define a cor de primeiro plano da tela.

Sintaxe

```
procedure SetColor ( cor : integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(BLUE);
  cleardevice;

  // Define a cor de desenho em amarelo
  setColor(YELLOW);

  // Desenha circulo de raio 5 em (20, 10)
  circle(50, 50, 20);

  // Desenha ellipse de raio 20, com angulo entre 0 e 270, na coordenada (100,
50)
  ellipse(100, 50, 0, 270, 20, 20);

  // Desenha pizza verde, de raio 20, com angulo entre 45 e 315, na coordenada
(50, 100)
  setColor(green);
  pieSlice (50, 100, 45, 315, 20);

  // Desenha pizza amarela, de raio 20, com angulo entre 45 e 315, na
coordenada (100, 100)
```

```
  setColor(yellow);
  sector(100, 100, 45, 315, 20, 20);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


SetFillStyle

Define o estilo para preenchimento de formas.

Sintaxe

```
procedure SetFillStyle ( padrao, cor : integer ) ;
```

Os seguintes valores são válidos para a variável padrao:

- EmptyFill
- SolidFill
- LineFill
- ltSlashFill
- SlashFill
- BkSlashFill
- LtBkSlashFill
- HatchFill
- XHatchFill
- InterLeaveFill
- WideDotFill
- CloseDotFill
- UserFill

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela
```

```
  SetBkColor(white) ;
  cleardevice ;

  // Desenha linha e dois retangulos vermelhos
  SetColor(red);
  Line(0, 0, 400, 400);
  Rectangle(300, 300, 400, 400);
  Rectangle(320, 320, 380, 380);

  // Define padrao de preenchimento amarelo
  SetFillStyle(SolidFill, yellow);

  // Preenche regiao do ponto (315, 310) delimitada pela cor vermelha
  FloodFill(315, 310, red);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


SetFillPattern

Define o padrão de preenchimento que será utilizado para preenchimento de formas.

Sintaxe

```
procedure SetFillPattern ( padraoPreenchimento : FillPatternType, cor:
integer ) ;
```

O argumento especifica um registro do tipo FillPatternType, que armazena o padrão de preenchimento.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    padraoInicial : FillPatternType;

// Cria novo padrao de preenchimento
const
   padraoPontilhado : FillPatternType = ($AA, $55, $AA, $55, $AA, $55, $AA,
$55);

Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
```

```
    exit;
  end;

  // Guarda o padrao de preenchimento inicial
  GetFillPattern(padraoInicial);

  // Define novo padrao para preenchimento de formas
  SetFillPattern(padraoPontilhado, lightgreen);

   // Desenha retangulo com novo padrao de preenchimento
   Bar(0, 0, 100, 100);
   readkey;

   // Retorna ao padrao de preenchimento inicial
   SetFillPattern(padraoInicial, yellow);

   // Desenha retangulo com antigo padrao de preenchimento
   Bar(0, 101, 100, 200);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


SetGraphBufSize

Muda o tamanho do buffer utilizado pelo modo gráfico.

Sintaxe

```
procedure SetGraphBufSize( tamanho: integer ) ;
```

Na implementação do compilador, essa função não faz nada.

Ela aparece na biblioteca somente para compatibilidade de código.

SetGraphMode

Seta o modo gráfico e limpa a tela.

Sintaxe

```
procedure SetGraphMode( modo: integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
```

```
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Desenha um circulo
  circle(50, 50, 30);
  readkey;

  // Limpa a tela, desenha retangulo
  setgraphmode(getgraphmode);
  rectangle(10,10, 100, 100);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


SetLineStyle

Define o estilo para desenho de linhas.

Sintaxe

```
procedure SetLineStyle ( estilo, padrao, comprimento : integer ) ;
```

Os seguintes valores são válidos para a variável *estilo*:

- SolidIn (constante com o valor 0, define uma linha sólida)
- DottedIn (constante com o valor 1, define uma linha pontilhada)
- CenterIn (constante com o valor 2, define uma linha centralizada)
- DashedIn (constante com o valor 3, define uma linha com pontilhado extendido)
- UserBitIn (constante com o valor 4, define uma linha com padrão definido pelo usuário)

A variável *padrão* é ignorada se o *estilo* é diferente de *UserBitIn*.

Se o estilo UserBitIn for utilizado, padrao contém o caractere usado para desenhar a linha. Para os outros estilos, o valor informado em padrao será ignorado

Os seguintes valores são válidos para a variável *comprimento*:

- NormWidth (constante com o valor 1)
- ThickWidth (constante com o valor 3)

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo azul
  setbkcolor(BLUE);
  cleardevice;

  // Desenha retangulo
  SetLineStyle(DottedLn, 0, NormWidth);
  Rectangle(10, 10, 200, 60);

  // Desenha retangulo
  SetLineStyle(DashedLn, 0, NormWidth);
  Rectangle(10, 70, 200, 120);

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


SetPalette

Altera a cor que está armazenada em uma dada posição da palheta.

Sintaxe

```
procedure SetPalette ( posicao: integer; cor: integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    palheta: PaletteType;
    posicao, cor: integer ;

Begin
  // Inicializa o modo grafico
  driver := Detect;
```

```
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Recupera a palheta utilizada pelo programa
  GetPalette(palheta);

  // Desenha linhas com as cores da palheta
  for posicao := 0 to pred(GetPaletteSize) do
  begin
     setColor(palheta.colors[posicao]);
     Line(0, posicao * 5, 100, posicao * 5);
  end;

  // Modifica as cores da palheta
  randomize;
  repeat
    posicao:= Random(palheta.size) ;
    cor := Random(palheta.size);
    SetPalette(posicao, cor);
    delay(500);
  until keyPressed;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


SetRGBPalette

Altera a entrada da palheta, na posição especificada, para a cor com valores RGB dados por r, g e b.

Sintaxe

```
procedure SetRGBPalette ( posicao, r, g, b: integer ) ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;

const maxcor=255;

var driver, modo: integer ;
    r, g, b: integer ;
```

```
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Desenha retangulo piscando
  setFillStyle(SolidFill, blue);
  bar(100,100,300,300);
  for r := 5 to maxcor do
    for g := 0 to maxcor do
      for b := 0 to maxcor do
      begin
        SetRGBPalette(1,r,g,b);
        SetRGBPalette(2,r,0,0);
        SetRGBPalette(3,0,g,0);
        SetRGBPalette(4,0,0,b);
        if keypressed then exit;
      end;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


SetTextJustify

Controla a impressão de um novo texto, com relação ao cursor corrente.

Sintaxe

```
procedure SetTextJustify( horizontal, vertical: integer ) ;
```

Os seguintes valores são válidos para a variável *horizontal*:

- LeftText (constante com o valor 0)
- CenterText (constante com o valor 1)
- RightText (constante com o valor 2)

Os seguintes valores são válidos para a variável *vertical*:

- BottomText (constante com o valor 0)
- CenterText (constante com o valor 1)
- TopText (constante com o valor 2)

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    estiloInicial: TextSettingsType;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Exibe texto com o estilo inicial do modo grafico
  setcolor(lightgreen);
  SetTextJustify(LeftText, CenterText) ;
  OutTextXY(10, 10, 'Texto com estilo inicial');

  // Guarda estilo inicial
  GetTextSettings(estiloInicial);

  // Define novo estilo de texto
  SetTextStyle(SansSerifFont, VertDir, 4);
  OutTextXY(10, 220, 'Novo estilo de texto');

  // Exibe texto com o estilo inicial
  with estiloInicial do
  begin
    SetTextJustify(Horiz, Vert);
    SetTextStyle(Font, Direction, CharSize);
  end;
  OutTextXY(10, 30, 'Outro texto com estilo inicial');

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


SetTextStyle

Define o estilo para impressão de texto no modo gráfico.

Sintaxe

```
procedure SetTextStyle ( fonte, direcao, tamanho : integer ) ;
```

Os seguintes valores são válidos para a variável *fonte*:

- DefaultFont (constante com o valor 0)
- TriplexFont (constante com o valor 1)
- SmallFont (constante com o valor 2)
- SansSerifFont (constante com o valor 3)
- GothicFont (constante com o valor 4)

Os seguintes valores são válidos para a variável *direção*:

- HorizDir (constante com o valor 0)
- VertDir (constante com o valor 1)

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    linhaTexto, tamanhoTexto: integer;
    texto: string;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo vermelha
  setbkcolor(lightred);
  cleardevice;

  // Define tamanho inicial do texto
  tamanhoTexto := 10;
  SetTextStyle(DefaultFont, HorizDir, tamanhoTexto);
  texto := 'Pascalzim Graphics';

  // Escreve o texto enquanto cabe horizontalmente na tela
  linhaTexto := 0;
  while TextWidth(texto) < GetMaxX do
  begin
    OutTextXY(0, linhaTexto, texto);
    // Determina qual a próxima linha de impressão
    linhaTexto := linhaTexto + TextHeight(texto) ;
    // Aumenta o tamanho do texto
            tamanhoTexto := tamanhoTexto + 4 ;
    SetTextStyle(DefaultFont, HorizDir, tamanhoTexto);
  end;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```

SetUserCharSize

Define o comprimento e a largura de fontes vetoriais.

Sintaxe

```
procedure SetUserCharSize( Xasp1, Xasp2, Yasp1, Yasp2: integer ) ;
```

O tamanho horizontal é dado por Xasp1/Xasp2, enquanto o tamanho vertical é dado por Yasp1/Yasp2.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Define cor de fundo branca
  setbkcolor(white);
  cleardevice;

  // Exibe texto com fonte de tamanho normal
  SetTextStyle(TriplexFont, HorizDir, 4);
  OutTextXY(10,10,'Texto Normal');

  // Exibe texto com fonte de comprimento reduzido (1/3)
  SetUserCharSize(1, 3, 1, 1);
  OutTextXY(10,50,'Texto Curto');

  // Exibe texto com fonte de comprimento aumentado (3x)
  SetUserCharSize(3, 1, 1, 1);
  OutTextXY(10,90,'Texto Largo');

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```

SetViewPort

Define uma janela no retângulo com borda nas coordenadas (x1, y1) e (x2, y2).

Sintaxe

```
procedure SetViewPort (x1, y1, x2, y2 : integer ; clip : boolean ) ;
```

Se a variável clip receber o valor true, todos os desenhos subsequentes serão realizados na janela.

Os seguintes valores são válidos para a variável clip:

- ClipOn, que é equivalente a **true**
- ClipOff, que é equivalente a **false**

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela
  SetBkColor(white) ;
  cleardevice ;

  // Cria janela com retangulo e texto
  SetViewPort(20, 20, 200, 100, ClipOn);
  rectangle(0, 0, 150, 50);
  outtextxy(0,0, 'primeiro retangulo');

  // Cria janela com retangulo e texto
  SetViewPort(20, 120, 200, 200, ClipOn);
  rectangle(0, 0, 150, 50);
  outtextxy(0,0, 'segundo retangulo');

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```

SetVisualPage

Define qual página será apresentada na tela.

Sintaxe

```
procedure SetVisualPage( pagina: integer ) ;
```

A tela inicial do modo gráfico corresponde à página de número zero.

Nessa implementação, o compilador suporta somente duas páginas, zero e um, respectivamente.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Escreve uma mensagem na tela oculta
  SetActivePage(1);
  setcolor(LIGHTGREEN) ;
  OutTextXY(10, 10, 'Essa mensagem esta na pagina 1');

  // Escreve uma mensagem na tela
  SetActivePage(0);
  setcolor(YELLOW) ;
  OutTextXY(10, 10, 'Essa mensagem esta na pagina 0');

  // Exibe tela oculta
  SetVisualPage(1);
  readkey ;

  // Exibe tela inicial
  SetVisualPage(0);
  readkey ;

  // Escreve outra mensagem na tela oculta
  SetActivePage(1);
  OutTextXY(10, 30, 'Essa outra mensagem esta na pagina 1');
  // Exibe novamente a tela oculta
  SetVisualPage(1);

  // Fecha o modo grafico
  readkey;
  closegraph;
```

```
End.
```


SetWriteMode

Controla a maneira como será feito o desenho de linhas na tela.

Sintaxe

```
procedure SetWriteMode ( modo : integer ) ;
```

Os seguintes valores são válidos para a variável *modo*:

- CopyPut
- XORPut

Usando o modo *CopyPut*, a linha é normalmente desenhada na tela.

Usando o modo *XORPut* é feito um xor entre os pixels da linha e os pixels da tela.

Assim, em termos práticos, quando uma linha é desenhada duas vezes em uma mesmo local, com o modo *XORPut* setado, no segundo desenho a tela será restaurada para cor original.

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    x, y: integer ;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Desenha um retângulo na tela até pressionar tecla
  repeat
    x := Random(GetMaxX - 100);
    y := Random(GetMaxY - 100);

    // Desenha retângulo
    SetWriteMode(CopyPut);
    Rectangle(x, y, x + 100, y + 100);
    Delay(500);

    // Apara retângulo
    SetWriteMode(XORPut);
```

```
    Rectangle(x, y, x + 100, y + 100);
  until KeyPressed;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


TextHeight

Retorna a altura do texto, em pixels, de acordo com a fonte e tamanho de texto definidos.

Sintaxe

```
function TextHeight ( texto : string ) : integer ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    linhaTexto, tamanhoTexto: integer;
    texto: string;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo vermelha
  setbkcolor(lightred);
  cleardevice;

  // Define tamanho inicial do texto
  tamanhoTexto := 10;
  SetTextStyle(DefaultFont, HorizDir, tamanhoTexto);
  texto := 'Pascalzim Graphics';

  // Escreve o texto enquanto cabe horizontalmente na tela
  linhaTexto := 0;
  while TextWidth(texto) < GetMaxX do
  begin
    OutTextXY(0, linhaTexto, texto);
    // Determina qual a próxima linha de impressão
    linhaTexto := linhaTexto + TextHeight(texto) ;
    // Aumenta o tamanho do texto
            tamanhoTexto := tamanhoTexto + 4 ;
    SetTextStyle(DefaultFont, HorizDir, tamanhoTexto);
```

```
  end;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


TextWidth

Retorna o comprimento do texto, em pixels, de acordo com a fonte e tamanho de texto definidos.

Sintaxe

```
function TextWidth ( texto : string ) : integer ;
```

Exemplo

```
Program Pascalzim ;
uses graph ;
var driver, modo: integer ;
    linhaTexto, tamanhoTexto: integer;
    texto: string;
Begin
  // Inicializa o modo grafico
  driver := Detect;
  initgraph(driver, modo, '');

  // Verifica se a inicializacao foi feita com sucesso
  if(graphResult <> grOk) then
  begin
    writeln('Erro ao inicializar o modo grafico:',
GraphErrorMsg(graphResult));
    exit;
  end;

  // Limpa a tela com a cor de fundo vermelha
  setbkcolor(lightred);
  cleardevice;

  // Define tamanho inicial do texto
  tamanhoTexto := 10;
  SetTextStyle(DefaultFont, HorizDir, tamanhoTexto);
  texto := 'Pascalzim Graphics';

  // Escreve o texto enquanto cabe horizontalmente na tela
  linhaTexto := 0;
  while TextWidth(texto) < GetMaxX do
  begin
    OutTextXY(0, linhaTexto, texto);
    // Determina qual a próxima linha de impressão
    linhaTexto := linhaTexto + TextHeight(texto) ;
    // Aumenta o tamanho do texto
            tamanhoTexto := tamanhoTexto + 4 ;
```

```
    SetTextStyle(DefaultFont, HorizDir, tamanhoTexto);
  end;

  // Fecha o modo grafico
  readkey;
  closegraph;
End.
```


Unidade Padrão

O compilador implementa as seguintes funções na unidade padrão:

- abs
- arctan
- chr
- concat
- copy
- cos
- eof
- eoln
- exp
- frac
- FreeMem
- GetMem
- int
- keypressed
- length
- ln
- odd
- ord
- pos
- pred
- random
- readkey
- round
- sin
- sqr
- sqrt
- succ
- trunc
- upcase
- wherex
- wherey

O compilador reconhece os seguintes comandos Pascal na unidade padrão:

- chdir
- clreol
- clrscr

- cursoroff
- cursoron
- dec
- delay
- delete
- delline
- erase
- getdate
- getdir
- gettime
- gotoxy
- highvideo
- inc
- insert
- insline
- lowvideo
- mkdir
- normvideo
- randomize
- rename
- rmdir
- str
- textbackground
- textcolor
- val
- window
- with

abs

Retorna o valor absoluto de um argumento numérico.

Sintaxe

```
function abs ( x : integer ): integer ;

function abs ( x : real ): 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.
```

arctan

Retorna o arco tangente do argumento numérico (onde x é um ângulo, em radianos).

Sintaxe

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

Exemplo

```
Program Pascalzim ;
var r: real ;
Begin
   r:= arctan( 3.14 );
End.
```


chr

Recebe como parâmetro um inteiro e retorna o caractere ASCII 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.
```


chdir

Modifica o diretório de trabalho atual.

Sintaxe

```
procedure chdir( nomeDiretorio: string);
```

Exemplo

```
Program PascalZIM ;
```

```
var nomePasta: string ;
Begin
   // Recupera a pasta de trabalho atual
   getdir(0, nomePasta);
   writeln('A pasta de trabalho atual é ', nomePasta);

   // Cria a nova pasta, depois entra nela
   mkdir('novaPasta');
   chdir('novaPasta');
   getdir(0, nomePasta);
   writeln('Agora estamos em ', nomePasta);

   // Exclui a pasta criada anteriormente
   chdir('..');
   getdir(0, nomePasta);
   writeln('Agora estamos em ', nomePasta);
   rmdir('novaPasta');
End.
```


clreol

Limpa o texto da linha onde está posicionado o cursor do teclado.

Sintaxe

```
procedure clreol ;
```

Exemplo

```
Program PascalZIM ;
uses crt ;
Begin
   // Escreve três linhas de texto
   writeln( 'primeira linha' );
   writeln( 'segunda linha' );
   writeln( 'terceira linha' );

   // Posiciona o cursor na segunda linha, depois limpa o texto dessa linha
   gotoxy(1,2);
   clreol ;
   readkey ;
End.
```


clrscr

Limpa a tela de impressão.

Sintaxe

```
procedure clrscr ;
```

Exemplo

```
Program PascalZIM ;
Begin
   clrscr ;
   writeln( 'Olá, mundo.' );
End.
```


concat

Concatena uma sequencia de cadeias de caracteres.

Sintaxe

```
function concat ( s1, s2, s3, ... : string ): string ;
```

Onde s1, s2, s3 são expressões do tipo **string**. As reticências indicam que mais de uma expressão pode ser informada para a função **concat**.

O número mínimo de expressões informadas para a função **concat** é um.

As expressões informadas podem também ter o tipo **char**.

Exemplo

```
Program Pascalzim ;
var s1, s2, cadeia: string ;
    vetor: array[1..4] of char ;
Begin
   // Exibe Compilador Pascalzim
   s1:= 'Compilador';
   s2:= 'Pascalzim';
   cadeia:= concat(s1, ' ', s2);
   writeln( cadeia );

   // Exibe pzim
   vetor[1]:= 'p' ;
   vetor[2]:= 'z' ;
   vetor[3]:= 'i' ;
   vetor[4]:= 'm' ;
   writeln( concat(vetor[1],vetor[2],vetor[3],vetor[4]));
End.
```


copy

Retorna parte de uma cadeia de caracteres.

Sintaxe

```
function copy( cadeia: string ; posInicio, quantidade: integer ) : 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 corresponde 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.
```


cos

Retorna o cosseno do argumento (x é um ângulo, em radianos).

Sintaxe

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

Exemplo

```
Program Pascalzim ;
var r: real ;
Begin
   r:= cos( 3.14 );    // Imprime o cosseno de pi
End.
```


cursoroff

Esconde o cursor do teclado.

Sintaxe

```
procedure cursoroff ;
```

Exemplo

```
Program PascalZIM ;
Begin
   write( 'mostrando o cursor' );
   readkey ;
   cursoroff ;
   write( 'cursor escondido' );
   readkey ;
   cursoron ;
   write( 'agora o cursor voltou' );
End.
```


cursoron

Deixa visível o cursor do teclado.

Sintaxe

```
procedure cursoron ;
```

Exemplo

```
Program PascalZIM ;
Begin
   write( 'mostrando o cursor' );
   readkey ;
   cursoroff ;
   write( 'cursor escondido' );
   readkey ;
   cursoron ;
   write( 'agora o cursor voltou' );
End.
```

dec

Diminui o valor armazenado em uma variável.

Sintaxe

```
procedure dec( var variavel: integer/char/ponteiro );

procedure dec( var variavel: integer/char/ponteiro; decremento: integer );
```

Nesse caso, a variável deve ter o tipo *integer*, *char* ou *ponteiro*.

O *decremento*, quando informado, informa o valor que será subtraido da variável.

Exemplo

```
Program PascalZIM ;
var intVar: integer ;
Begin
  dec( intVar );   { Equivalente a IntVar:= IntVar - 1 }
End.
```

No caso de ponteiros, o decremento no endereço de memória ali armazenado ocorre de acordo com o tamanho do objeto apontado.

Como exemplo, se o ponteiro guarda o endereço do último elemento de um vetor, o decremento de um faz o ponteiro apontar para o *penúltimo* elemento do vetor.

Um novo decremento de dois, por sua vez, faz o ponteiro apontar para dois elementos antes no vetor.

Exemplo

```
Program Pzim ;
  // Declara um vetor
  const vetor: array[1..5] of integer = ( 10, 20, 30, 40, 50 ) ;
  // Guarda o endereco de um inteiro
var p: ^integer ;

Begin
   // O ponteiro guarda o endereco do quinto elemento do vetor
   p:= @vetor[5] ;
   writeln(p^); // Mostra 50

   // Move o ponteiro para o elemento anterior do vetor
   dec(p) ;
   writeln(p^); // Mostra 40

   // Move o ponteiro para dois elementos antes no vetor
   dec(p, 2) ;
   writeln(p^); // Mostra 20
End.
```

delay

Suspende a execução do programa durante X milissegundos.

Sintaxe

```
procedure delay( tempo: integer );
```

Onde tempo é uma expressão do tipo integer que indica, em milissegundos, quanto tempo a execução do programa será suspensa.

Exemplo

```
Program PascalZIM ;
var milissegundos: integer ;
Begin
   write( 'Quanto tempo (em milissegundos) o programa ficará inativo?' );
   readln( milissegundos );
   writeln( 'Parando a execução por um tempo...' );
   delay( milissegundos );
   writeln( 'De volta à ativa! ' );
End.
```


delete

Usado para remover parte de uma cadeia.

Sintaxe

```
procedure delete( var variável: string, posInicio, quantos: integer );
```

Funcionamento

- O comando remove quantos caracteres da cadeia armazenada em variável, começando da posição posInicio.
- A posição do primeiro caractere da cadeia é 1.
- Se posInicio é menor ou igual a zero, nenhum caractere é removido da cadeia.
- Se quantos é menor ou igual a zero, nenhum caractere é removido da cadeia.
- Se posInicio é maior que o tamanho da cadeia nenhum caractere é removido da cadeia.
- Se a soma de posInicio e quantos é maior que o tamanho da cadeia, então quantos é assumido como igual ao tamanho da cadeia - posInicio + 1.

Exemplo

Assumindo que a variável cadeia armazena "1234567":

- Ao executar o comando delete(cadeia, 3, 2) a variável cadeia fica armazenando 12567
- Ao executar o comando delete(cadeia, 1, 3) a variável cadeia fica armazenando 4567

- Ao executar o comando delete(cadeia, 5, 10) a variável cadeia fica armazenando 1234
- Ao executar o comando delete(cadeia, 7, 3) a variável cadeia fica armazenando 123456

- Ao executar o comando delete(cadeia, -3, 3) a variável cadeia fica armazenando 1234567
- Ao executar o comando delete(cadeia, 0, 3) a variável cadeia fica armazenando 1234567

- Ao executar o comando delete(cadeia, 7, 0) a variável cadeia fica armazenando 1234567
- Ao executar o comando delete(cadeia, 5, -2) a variável cadeia fica armazenando 1234567
- Ao executar o comando delete(cadeia, 9, 5) a variável cadeia fica armazenando 1234567

Exemplo

```
Program PascalZIM ;
var cadeia: string ;
Begin
   cadeia:= '1234567' ;
   writeln( 'Valor de cadeia: ', cadeia );
   delete( cadeia, 3, 4 );
   writeln( 'Depois do delete: ', cadeia );   // Mostra 127

End.
```


delline

Remove do console a linha onde está posicionado o cursor do teclado.

Sintaxe

```
procedure delline ;
```

Exemplo

```
Program PascalZIM ;
uses crt ;
Begin
   // Escreve três linhas de texto
   writeln( 'primeira linha' );
   writeln( 'segunda linha' );
   writeln( 'terceira linha' );
   // Posiciona o cursor na segunda linha, depois remove essa linha
   gotoxy( 1,2 );
   delline ;
   readkey ;
```

```
End.
```


eof

Usada para verificar se o final de um arquivo foi alcançado durante uma leitura de valores.

Sintaxe

```
function eof ( var arquivo: text ): boolean ;
```

A função recebe como argumento uma variável do tipo text, retornando true se o cursor de leitura do arquivo referenciado por F se encontra no seu fim, false em caso contrário.

Exemplo

```
var arq: text ;
    caractere: char ;
Begin
   assign( arq, 'teste.pas' );
   reset( arq );
   while not eof( arq ) do
   Begin
      read( arq, caractere );
      write( caractere );
   End ;
End.
```


eoln

Usada para verificar se o final de uma linha em um arquivo do tipo texto foi alcançado durante uma leitura de valores.

Sintaxe

```
function eoln ( var arquivo: text ): boolean ;
```

A função recebe como argumento uma variável do tipo text, retornando true se o cursor de leitura do arquivo referenciado por F se encontra no fim de uma linha, false em caso contrário.

Exemplo

```
var arq: text ;
    caractere: char ;
Begin
   assign( arq, 'teste.pas' );
   reset( arq );
   while not eoln( arq ) do
```

```
   Begin
      read( arq, caractere );
      write( caractere );
   End ;
End.
```


erase

Apaga um arquivo em disco.

Sintaxe

```
procedure erase( var arquivo: file);

procedure erase( var arquivo: text);
```

Exemplo

```
Program PascalZIM ;
var arq: text ;
begin
   // Cria o arquivo de teste
   assign(arq, 'c:\teste.txt');
   rewrite(arq);
   close(arq);

   // Apaga o arquivo criado
   erase(arq);
end.
```


exp

Retorna o exponencial do argumento.

Sintaxe

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

Exemplo

```
Program Pascalzim ;
   Begin
   writeln( 'e = ', exp(1.0) );
End.
```


frac

Retorna a parte fracionária de um valor numérico.

Sintaxe

```
function frac( valor: integer ): real ;

function frac( valor: real ): 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.
```


FreeMem

Libera uma determinada quantidade de memória (em bytes) que foi alocada para um ponteiro.

Sintaxe

```
procedure FreeMem ( var p: ponteiro; quantidadeMemoria: integer ) ;
```

Como alternativa, pode-se utilizar também os métodos new e dispose para alocar e liberar memória para um ponteiro.

Exemplo

```
Program Pascalzim ;
var p: ^integer ;
    v: integer ;
Begin
   GetMem( p, sizeof(integer) );  // Aloca espaço para armazenar um inteiro
   p^:= 10 ;   // Guarda um inteiro na posição apontada por p
   writeln( 'Valor armazenado na posicao de memoria: ', p^ );
   v:= p^ ;   //Guarda em v o valor apontado por p
   writeln( 'Valor armazenado em v: ', v );
   FreeMem( p, sizeof(integer) );   // Libera a memoria associada a p
   readln ;
End.
```


getdate

Recupera a data corrente.

Sintaxe

```
procedure getdate( var ano: integer; var mes: integer; var dia: integer; var
diaSemana: integer );
```

Onde:

- ano é o ano corrente, com quatro dígitos decimais
- mês é o mês corrente, entre 1 e 12, assumindo que Janeiro é o mês 1
- dia é o dia corrente do mês, entre 1 e 31
- diaSemana é o dia corrente da semana, entre 0 e 6, assumindo que Domingo é o dia 0

Exemplo

```
Program PascalZIM ;
const
  DiaDaSemana: array[0..6] of
string=('Domingo','Segunda','Terça','Quarta','Quinta','Sexta','Sábado');

  MesDoAno: array[1..12] of
string=('Janeiro','Fevereiro','Marco','Abril','Maio','Junho',
                                    'Julho','Agosto','Setembro','Outubro','Nov
embro','Dezembro');
var
  dia, mes, ano, diaSemana: integer ;
Begin
  // Recupera a data corrente
  GetDate(ano, mes, dia, diaSemana);

  // Exibe a data corrente
  Write('Hoje é ', DiaDaSemana[diaSemana], ', ', dia, ' de ', MesDoAno[mes], '
de ', ano);
End.
```


getdir

Recupera o diretório de trabalho atual.

Sintaxe

```
procedure getdir( drive: integer; var nomeDiretorio: string);
```

Onde:

- drive é igual a zero para o diretório corrente no disco rígido.

- nomeDiretorio é uma variável que receberá o diretório de trabalho atual.

Exemplo

```
Program PascalZIM ;
var nomePasta: string ;
Begin
   // Recupera a pasta de trabalho atual
   getdir(0, nomePasta);
   writeln('A pasta de trabalho atual é ', nomePasta);

   // Cria a nova pasta, depois entra nela
```

```
   mkdir('novaPasta');
   chdir('novaPasta');
   getdir(0, nomePasta);
   writeln('Agora estamos em ', nomePasta);

   // Exclui a pasta criada anteriormente
   chdir('..');
   getdir(0, nomePasta);
   writeln('Agora estamos em ', nomePasta);
   rmdir('novaPasta');
End.
```


GetMem

Aloca uma determinada quantidade de memória (em bytes) para um ponteiro.

Sintaxe

```
procedure GetMem ( var p: ponteiro; quantidadeMemoria: integer ) ;
```

Ao especificar a quantidade de memória, é uma boa prática utilizar a função sizeof para dizer ao compilador o tamanho do tipo de dados que será armazenado. Exemplo:

```
   GetMem( p, 10*sizeof(char) );
```

Isso porque o tamanho dos tipos, no Pascalzim não segue o tamanho dos tipos de acordo com a definição da linguagem Pascal.

Como alternativa, pode-se utilizar também os métodos new e dispose para alocar e liberar memória para um ponteiro.

Exemplo

```
Program Pascalzim ;
var p: ^integer ;
    v: integer ;
Begin
   GetMem( p, sizeof(integer) );  // Aloca espaço para armazenar um inteiro
   p^:= 10 ;   // Guarda um inteiro na posição apontada por p
   writeln( 'Valor armazenado na posicao de memoria: ', p^ );
   v:= p^ ;   //Guarda em v o valor apontado por p
   writeln( 'Valor armazenado em v: ', v );
   FreeMem( p, sizeof(integer) );   // Libera a memoria associada a p
   readln ;
End.
```


gettime

Recupera a hora corrente.

Sintaxe

```
procedure gettime( var hora: integer; var minuto: integer; var segundo:
```

```
integer; var milisegundo: integer );
```

Onde:

- hora é a hora corrente, entre 0 e 23
- minuto é o minuto corrente, entre 0 e 59
- segundo é o segundo corrente, entre 0 e 59
- milisegundo é o milisegundo corrente, entre 0 e 59

Exemplo

```
Program PascalZIM ;
var
  hora, minuto, segundo, msegundo: integer ;
Begin
  // Recupera a hora corrente
  GetTime(hora, minuto, segundo, msegundo);

  // Exibe a hora corrente
  Write('Agora são ', hora, ' horas, ', minuto, ' minutos, ', segundo, '
segundos');
End.
```


gotoxy

O comando gotoxy define a posição do cursor do teclado na tela de console.

Essa tela possui várias linhas, e cada linha possui um conjunto de colunas.

Pensando em termos de eixos cartesianos, as colunas correspondem ao eixo x, e as linhas ao eixo y.

As linhas aumentam de cima para baixo na tela, enquanto as colunas aumentam da esquerda para a direita.

Sintaxe

```
procedure gotoxy( coluna, linha: integer );
```

Exemplo

```
Program PascalZIM ;

Begin
   gotoxy(10,2);
   textcolor( lightcyan );
   textbackground( red );
   write( 'Olá, mundo!' );
End.
```

highvideo

Faz com o texto impresso em console apareça em alta intensidade.

Sintaxe

```
procedure highvideo ;
```

Exemplo

```
Program PascalZIM ;
uses crt ;
Begin
   highvideo ;
   writeln( 'Texto em alta intensidade' );
   lowvideo ;
   writeln( 'Texto em baixa intensidade' );
   normvideo ;
   writeln( 'Texto com intensidade padrao' );
End.
```


inc

Aumenta o valor armazenado em uma variável.

Sintaxe

```
procedure inc( var variavel: integer/char/ponteiro );

procedure inc( var variavel: integer/char/ponteiro; incremento: integer );
```

Nesse caso, a variável deve ter o tipo *integer*, *char* ou *ponteiro*.

O *incremento*, quando informado, informa o valor que será adicionado à variável.

Exemplo

```
Program PascalZIM ;
var intVar: integer ;
Begin
  inc( intVar );   { Equivalente a IntVar:= IntVar + 1 }
End.
```

No caso de ponteiros, o incremento no endereço de memória ali armazenado ocorre de acordo com o tamanho do objeto apontado.

Como exemplo, se o ponteiro guarda o endereço do primeiro elemento de um vetor, o incremento de um faz o ponteiro apontar para o *segundo* elemento do vetor.

Um novo incremento de dois, por sua vez, faz o ponteiro apontar para o *quarto* elemento do vetor.

Exemplo

```
Program Pzim ;
  // Declara um vetor
  const vetor: array[1..5] of integer = ( 10, 20, 30, 40, 50 ) ;
  // Guarda o endereco de um inteiro
var p: ^integer ;

Begin
   // O ponteiro guarda o endereco do primeiro elemento do vetor
   p:= @vetor ;
   writeln(p^); // Mostra 10

   // Avanca o ponteiro para o proximo elemento do vetor
   inc(p) ;
   writeln(p^); // Mostra 20

   // Avanca o ponteiro para mais dois elementos no vetor
   inc(p, 2) ;
   writeln(p^); // Mostra 40
End.
```


insert

Usado para adicionar uma subcadeia a uma cadeia.

Sintaxe

```
procedure insert( subcadeia: string ; var cadeia: string ; posInicio:
integer );
```

Onde:

- subcadeia é uma expressão.
- cadeia é uma variável.

Funcionamento

- O comando adiciona subcadeia em cadeia, na posição posInicio.
- A posição do primeiro caractere de cadeia é 1.
- Se posInicio é menor ou igual a 1, o comando adiciona subcadeia no início de cadeia.
- Se posInicio é maior que o tamanho da cadeia, o comando adiciona subcadeia no fim de cadeia.
- Se a cadeia resultante tem mais de 255 caracteres, ela é truncada para 255 caracteres.

Exemplo

Assumindo que a variável cadeia armazena "1234567":

- Ao executar o comando insert(' abcd', cadeia, 2) na variável cadeia fica armazenando 1abcd234567

- Ao executar o comando insert(' abcd', cadeia, 7) na variável cadeia fica armazenando

123456abcd7

- Ao executar o comando insert(' abcd', cadeia, 1) na variável cadeia fica armazenando abcd1234567
- Ao executar o comando insert(' abcd', cadeia, -1) na variável cadeia fica armazenando abcd1234567
- Ao executar o comando insert(' abcd', cadeia, 8) na variável cadeia fica armazenando 1234567abcd

Exemplo

```
Program PascalZIM ;
var cadeia: string ;
   Begin
      cadeia:= '1234567' ;
      writeln( 'Valor de cadeia: ', cadeia);
      insert( 'abcd', cadeia, 4);
      writeln( 'Depois do insert: ', cadeia);   // Mostra 123abcd4567
  End.
```


insline

Insere uma linha em branco onde está posicionado o cursor do teclado.

Sintaxe

```
procedure insline ;
```

Exemplo

```
Program PascalZIM ;
uses crt ;
Begin
   // Escreve três linhas de texto
   writeln( 'primeira linha' );
   writeln( 'segunda linha' );
   writeln( 'terceira linha' );
   // Posiciona o cursor na segunda linha, insere linha em branco em cima
dessa linha
   gotoxy(1,2);
   insline ;
   readkey ;
End.
```


int

Retorna a parte inteira de um valor numérico.

Sintaxe

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

function int( valor: real ): 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.
```


keypressed

Verifica se foi pressionada alguma tecla.

Sintaxe

```
function keypressed : boolean ;
```

Exemplo

```
Program PascalZIM ;
Begin
   while not keypressed do
      Begin
         write( 'x' );
      End ;
End.
```


length

Retorna o comprimento de uma cadeia de caracteres.

Sintaxe

```
function length( cadeia: string ): integer ;
```

Exemplo

```
Program Pascalzim ;
var s: string ;
Begin
```

```
   write( 'Digite uma cadeia: ' );
   readln( s );
   writeln( ' O comprimento da cadeia lida = ', length( s ));
End.
```


ln

Retorna o logaritmo neperiano do argumento.

Sintaxe

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

Exemplo

```
Program PascalZIM ;
var e: real ;
Begin
   e:= exp( 1.0 );
   writeln( 'ln(e) = ', ln( e ) );
End.
```


lowvideo

Faz com o texto impresso em console apareça em baixa intensidade.

Sintaxe

```
procedure lowvideo ;
```

Exemplo

```
Program PascalZIM ;
uses crt ;
Begin
   highvideo ;
   writeln( 'Texto em alta intensidade' );
   lowvideo ;
   writeln( 'Texto em baixa intensidade' );
   normvideo ;
   writeln( 'Texto com intensidade padrao' );
End.
```


mkdir

Cria um novo diretório.

Sintaxe

```
procedure mkdir( nomeDiretorio: string);
```

Exemplo

```
Program PascalZIM ;
var nomePasta: string ;
Begin
   // Recupera a pasta de trabalho atual
   getdir(0, nomePasta);
   writeln('A pasta de trabalho atual é ', nomePasta);

   // Cria a nova pasta, depois entra nela
   mkdir('novaPasta');
   chdir('novaPasta');
   getdir(0, nomePasta);
   writeln('Agora estamos em ', nomePasta);

   // Exclui a pasta criada anteriormente
   chdir('..');
   getdir(0, nomePasta);
   writeln('Agora estamos em ', nomePasta);
   rmdir('novaPasta');
End.
```


normvideo

Faz com o texto impresso em console apareça na intensidade padrão.

Sintaxe

```
procedure normvideo ;
```

Exemplo

```
Program PascalZIM ;
uses crt ;
Begin
   highvideo ;
   writeln( 'Texto em alta intensidade' );
   lowvideo ;
   writeln( 'Texto em baixa intensidade' );
   normvideo ;
   writeln( 'Texto com intensidade padrao' );
End.
```


odd

Verifica a paridade do argumento, retornando true se o argumento é ímpar, false em caso

contrário.

Sintaxe

```
function odd ( x: integer ): boolean ;
```

Exemplo

```
Program PascalZIM ;
Begin
   if odd( 5 ) then
      writeln( '5 é impar!' )
   else
      writeln( '5 não é impar...' );
End.
```


ord

Recebe como parâmetro um caractere e retorna o inteiro correspondente ao código ASCII referente ao caractere.

Sintaxe

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

Exemplo

```
Program Pascalzim ;
Begin
   writeln( 'O codigo ASCII para "c" = ', ord( 'c' ), ' decimal' );
End.
```


paramcount

Retorna o número de parâmetros informados na linha de comando para o programa.

Sintaxe

```
function paramcount: integer ;
```

Exemplo

```
Program PascalZIM ;
  var i: integer ;
Begin
   writeln( 'Nome do programa: ', paramstr(0) ) ;
   writeln( 'Número de parâmetros informados na linha de comando: ',
paramcount ) ;
```

```
   if(paramcount > 0) then
     for i:= 1 to paramcount do
     Begin
       writeln('Parâmetro ', i, ': ', paramstr(i) );
     End;
End.
```


paramstr

Retorna o valor de um parâmetro informado na linha de comando para o programa.

O parâmetro 0, nesse caso corresponde ao nome do arquivo executável, contendo o caminho no sistema de arquivos.

O parâmetro 1 corresponde ao primeiro parâmetro, o parâmetro 2 corresponde ao segundo parâmetro, e assim por diante.

Sintaxe

```
function paramstr ( parametro : integer ): string ;
```

Exemplo

```
Program PascalZIM ;
  var i: integer ;
Begin
   writeln( 'Nome do programa: ', paramstr(0) ) ;
   writeln( 'Número de parâmetros informados na linha de comando: ',
paramcount ) ;
   if(paramcount > 0) then
     for i:= 1 to paramcount do
     Begin
       writeln('Parâmetro ', i, ': ', paramstr(i) );
     End;
End.
```


pos

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

Sintaxe

```
function pos( subcadeia, cadeia: string ): 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.
```


pred

Retorna o número/caractere que antecede o argumento.

Sintaxe

```
function pred( valor: integer ) : integer ;

function pred( valor: char ) : char ;
```

Onde variável tem o tipo **inteiro** ou **char**.

Exemplo

```
Program PascalZIM ;
Begin
   writeln( 'O predecessor de 5 = ', pred(5) );
   writeln( 'O sucessor de 10 = ', succ(10) );
End.
```


random

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

Sintaxe

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

Exemplo

```
Program PascalZIM ;
var i: integer ;
Begin
   randomize ;
   repeat
      i:= i + 1 ;
      writeln( random(1000) )
   until i>10 ;
End.
```


randomize

Inicializa o gerador de números randômicos do compilador.

Sintaxe

```
procedure randomize ;
```

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 caractere do teclado. Pode ser utilizado como um comando ou como uma função.

Sintaxe

```
procedure readkey ;
```

Exemplo

```
Program PascalZIM ;
Begin
   writeln( 'O programa vai terminar...' );
   readkey ;
End.
```

Como função, sua sintaxe é:

```
function readkey: integer ;
```

Exemplo

```
Program PascalZIM ;
var umCaractere: char ;
Begin
   writeln( 'Digite um caractere:' );
   umCaractere:= readkey ;
   writeln( 'Você digitou: ', umCaractere );
End.
```


rename

Muda o nome de um arquivo em disco.

Sintaxe

```
procedure rename( var arquivo: file; novoNome: string );

procedure rename( var arquivo: text; novoNome: string );
```

Exemplo

```
Program PascalZIM ;
var arq: text ;
begin
   // Cria o arquivo de teste
   assign(arq, 'c:\teste.txt');
   rewrite(arq);
   close(arq);

   // Muda o nome do arquivo criado
   rename(arq, 'c:\testeRenomeado.txt');
end.
```


rmdir

Remove um diretório.

Sintaxe

```
procedure rmdir( nomeDiretorio: string);
```

Exemplo

```
Program PascalZIM ;
var nomePasta: string ;
```

```
Begin
   // Recupera a pasta de trabalho atual
   getdir(0, nomePasta);
   writeln('A pasta de trabalho atual é ', nomePasta);

   // Cria a nova pasta, depois entra nela
   mkdir('novaPasta');
   chdir('novaPasta');
   getdir(0, nomePasta);
   writeln('Agora estamos em ', nomePasta);

   // Exclui a pasta criada anteriormente
   chdir('..');
   getdir(0, nomePasta);
   writeln('Agora estamos em ', nomePasta);
   rmdir('novaPasta');
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.
```


sin

Retorna o seno do argumento (x é um ângulo, em radianos).

Sintaxe

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

Exemplo

```
Program PascalZIM ;
var r: real ;
Begin
   r:= sin( 3.14 );
   writeln( 'O seno de Pi = ', r );
```

```
   readln ;
End.
```


SizeOf

Retorna o número de bytes ocupados pelo argumento.

Sintaxe

```
function SizeOf ( argumento ) : integer ;
```

O argumento, nesse caso, denota o nome de um tipo de dados.

Exemplo

```
Program Pascalzim ;
var p: ^integer ;
    v: integer ;
Begin
   GetMem( p, sizeof(integer) );  // Aloca espaço para armazenar um inteiro
   p^:= 10 ;   // Guarda um inteiro na posição apontada por p
   writeln( 'Valor armazenado na posicao de memoria: ', p^ );
   v:= p^ ;   //Guarda em v o valor apontado por p
   writeln( 'Valor armazenado em v: ', v );
   FreeMem( p, sizeof(integer) );   // Libera a memoria associada a p
   readln ;
End.
```


sqr

Retorna o quadrado do argumento.

Sintaxe

```
function sqr ( x : integer ): integer ;

function sqr ( x : real ): 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.
```


str

Usado para converter uma expressão numérica em uma cadeia.

Sintaxe

```
procedure str( expressão: integer, var variavel: string );

procedure str( expressão: real, var variavel: string );
```

Funcionamento

variável receberá o valor proveniente da conversão.

Pode-se também informar parâmetros de formatação para a conversão:

```
procedure str( expressão:par1:par2, var variavel: string );
```

Nesse caso:

- par1 informa o tamanho da cadeia obtida na conversão
- par2 informa o número de casas decimais usada na conversão do real

Exemplo

```
Program PascalZIM ;
  var s: string ;
Begin
  str(1234, s);
  writeln(s);   // Mostra 1234

  str(789.123456:5:3, s);
  writeln(s);   // Mostra 789.123
End.
```


succ

Retorna o número/caractere que sucede o argumento.

Sintaxe

```
function succ( valor: integer ) : integer ;

function succ( valor: char ) : char ;
```

Onde variável tem o tipo **inteiro** ou **char**.

Exemplo

```
Program PascalZIM ;
Begin
   writeln( 'O predecessor de 5 = ', pred(5) );
   writeln( 'O sucessor de 10 = ', succ(10) );
End.
```


textbackground

O comando textbackground define a cor de fundo usada na impressão de textos.

Sintaxe

```
procedure textbackground( cor: integer );
```

Onde cor pode ser uma constante inteira ou qualquer uma dentre as cores seguintes:

- black
- blue
- green
- cyan
- red
- magenta
- brown
- lightgray

Nesse caso, as constantes acima assumem os valores de 0 (black) a 7 (lightgray).

Para outros valores inteiros é aplicada a seguinte regra:

cor = cor % 8

Exemplo. Programa que move o cursor do teclado para a linha 15, coluna 10, e imprime "Olá, mundo!" na tela.

```
Program PascalZIM ;

Begin
   gotoxy(15,10);
   textcolor( lightcyan );
   textbackground( red );
```

```
   write( 'Olá, mundo!' );
End.
```


textcolor

O comando textcolor define a cor da fonte usada para impressão de texto na tela.

Sintaxe

```
procedure textcolor( cor: integer );
```

Onde cor pode ser uma constante inteira ou qualquer uma dentre as cores seguintes:

- black
- blue
- green
- cyan
- red
- magenta
- brown
- lightgray
- darkgray
- lightblue
- lightgreen
- lightcyan
- lightred
- lightmagenta
- yellow
- white

Nesse caso, as constantes acima assumem os valores de 0 (black) a 15 (white).

Para outros valores inteiros é aplicada a seguinte regra:

cor = cor % 16

Exemplo. Pode ser utilizada uma combinação de cores, como em:

```
textcolor( red + blue );
```

Exemplo. Programa que move o cursor do teclado para a linha 15, coluna 10, e imprime "Olá, mundo!" na tela.

```
Program Pascalzim ;
Begin
   gotoxy(15,10);
   textcolor( lightcyan );
   textbackground( red );
   write( 'Olá, mundo!' );
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.
```


val

Usado para converter uma cadeia de caracteres em um inteiro ou real.

Sintaxe

```
procedure val( cadeia: string ; var variavel: integer ; var codigoErro:
integer );
```

```
procedure val( cadeia: string ; var variavel: real ; var codigoErro:
integer );
```

Onde:

- cadeia é uma cadeia de caracteres ou uma expressão envolvendo a concatenação de várias cadeias.
- variável é uma variável do tipo **integer** ou **real**.
- codigoErro é uma variável do tipo **integer**.

Funcionamento

- Se a cadeia de caracteres puder ser convertida, variável receberá o valor proveniente da conversão, e codigoErro armazenará o valor zero.
- Se a cadeia de caracteres não puder ser convertida, variável receberá o valor zero, e codigoErro armazenará a posição na cadeia em que foi encontrado um caractere inválido.

Exemplo

- A conversão da cadeia "123" armazena em variável o valor 123 e armazena em codigoErro o valor 0.
- A conversão da cadeia "abc" armazena em variável o valor 0 e armazena em codigoErro o valor 1.
- A conversão da cadeia "123v5" armazena em variável o valor 0 e armazena em codigoErro o valor 4.

Exemplo

```
Program PascalZIM ;
var cadeia: string ;
    nro, codigoErro: integer ;
Begin
   write( 'Digite um número inteiro: ' );
   readln( cadeia );
   val( cadeia, nro, codigoErro );
   if ( codigoErro = 0 ) then
      writeln( 'O número lido e convertido foi: ' , nro )
   else
      writeln( 'Inteiro inválido, e o código de erro foi: ' , codigoErro );
End.
```


wherey

Retorna a linha onde está posicionado o cursor do teclado na tela de console.

Sintaxe

```
function wherey : integer ;
```

Exemplo

```
Program PascalZIM ;
Begin
   gotoxy(10,20);
   write( wherex, ' ', wherey );
End.
```


wherex

Retorna a coluna onde está posicionado o cursor do teclado na tela de console.

Sintaxe

```
function wherex : integer ;
```

Exemplo

```
Program PascalZIM ;
Begin
   gotoxy(10,20);
   write( wherex, ' ', wherey );
End.
```


with

Usado para simplificar o acesso aos campos de um registro.

Por exemplo, dada a seguinte declaração de registro:

```
var aluno: record
              nome: string ;
              idade: real ;
           End ;
```

Podemos escrever os seguintes comandos de atribuição de dados:

```
aluno.nome = 'Joaozinho' ;
aluno.idade = 18 ;
```

Ou podemos escrever comandos equivalentes usando o with:

```
with aluno do
   Begin
      nome = 'Joaozinho' ;
      idade = 18 ;
   End ;
```

Sintaxe

```
with listaVariáveis do
    comando ;
```

Onde:

- listaVariáveis é uma sequência de uma ou mais variáveis separadas por vírgula.
- comando denota um comando simples ou um bloco de comandos.

As variáveis informadas na lista de variáveis devem ser do tipo registro.

Nos comandos restringidos pelo with é realizada uma análise para verificar se as variáveis ali encontradas correspondem a campos de alguma das variáveis presentes na lista de variáveis.

Essa análise é feita tomando-se em consideração a última variável da lista, depois a penúltima, e assim por diante até a primeira.

Se alguma variável encontrada nos comandos restringidos pelo with não for um campo de alguma das variáveis da lista, é verificado se a variável foi declarada no escopo corrente.

De acordo com essas regras, o trecho de código abaixo imprime o resultado 2 2. Isso acontece porque o compilador assume que os campos X e Y pertencem ao registro T.

```
Program testw ;
type Ponto = record
                X, Y: integer ;
             End ;
var S,T: Ponto ;
Begin
   S.X:= 1 ;
   S.Y:= 1 ;
   T.X:= 2 ;
   T.Y:= 2 ;
   With S,T do
      writeln(X,' ',Y);    // imprime 2 2
End.
```

O exemplo apresentado a seguir mostra como comandos with podem ser aninhados. Nesse caso, a variável do with mais interno pode ainda ser um campo da variável de um with mais externo.

Exemplo

```
Program PascalZIM ;
var registro: record
                  campo1: real ;
    campo2: record
       campo21: real ;
       campo22: char ;
       End ;
              End ;
Begin
   with registro do
      with campo2 do                          // campo2 é um campo da
variável registro
         campo22:= 'M' ;                      // campo22 é um campo da
```

```
variável campo2
         writeln(registro.campo2.campo22);      // Mostra M
         with registro, campo2 do               // campo2 é um campo da
variável registro
           campo22:= 'P' ;
           writeln(registro.campo2.campo22);    // Mostra P
End.
```


window

Cria uma nova janela em console. Os comandos de escrita em console são feitos dentro dos limites dessa janela.

Sintaxe

```
procedure window( col1, lin1, col2, lin2: integer);
```

Onde:

- col1 e lin1 indicam, respectivamente, a coluna e linha correspondente às coordenadas do canto superior esquerdo da janela.

- col2 e lin2 indicam, respectivamente, a coluna e linha correspondente às coordenadas do canto inferior direito da janela.

Exemplo

```
Program PascalZIM ;
uses crt ;
Begin
   // Pinta o fundo da tela de azul
   textbackground(lightblue);
   clrscr ;

   // Cria uma janela com fundo vermelho
   window(10,2, 15, 6);
   textbackground(lightred);
   clrscr ;

   // Escreve texto dentro da janela
   writeln( 'PascalzimPascalzim' );
   readkey ;
End.
```


Tratamento de overflow

O tratamento de overflow, no compilador, é realizado para constantes inteiras, reais e literais.

O tratamento consiste em verificar se, durante uma determinada operação, uma constante ultrapassa o valor máximo permitido para constantes do tipo em questão.

O intervalo de valores válidos para constantes numéricas é:

- Para constantes reais: 3.4 * (10^-38) à 3.4 * (10^+38)
- Para constantes inteiras: 32767 à -32767.

O tratamento de overflow dado às cadeias de caracteres depende do tamanho da cadeia.

Comentários

Os comentários são usados dentro de um programa com a finalidade de documentar trechos de código, e não afetam a execução do programa.

A definição de um comentário, na linguagem Pascal, é feita com o uso de pares de chaves { } ou * *.

O texto comentado é ignorado pelo compilador.

Exemplo

```
Program Pascalzim ;    { Esse é meu programa de teste }
Begin
   Write( 'Olá, mundo!' );    { Imprime a mensagem 'Olá, mundo!' }
End.
```

Exemplo

```
Program PascalZim ;    (* Esse é meu outro programa de teste *)
Begin
   Write( 'Olá, mundo!' );    (* Imprime a mensagem 'Olá, mundo!' *)
End.
```

O Pascalzim possui ainda um outro tipo de comentário, o comentário de linha. Um comentário de linha é iniciado por // e todos os caracteres que seguem o // na linha são automaticamente ignorados pelo compilador, da mesma forma que em C e Java.

Exemplo

```
Program PascalZIM ;
Begin
   Write( 'Olá, mundo!' );    // Esse é um comentário de linha
End.
```


Formato básico de uma unidade Pascal

Uma unidade consiste em um módulo contendo constantes, tipos, variáveis, funções e procedimentos que podem ser utilizados por um programa Pascal. Unidades são utilizadas

para criar bibliotecas de código reutilizável, e também para dividir um programa em partes menores.

Uma unidade pode ser, basicamente, estruturada em quatro regiões significativas:

1. Um cabeçalho, que dá nome à unidade;

2. Uma parte de interface, ou seção de definição e declaração de dados públicos;

3. Uma parte de implementação, ou seção de definição e declaração de dados privados;

4 . Uma parte de inicialização, que contém comandos de inicialização da interface.

O cabeçalho de uma unidade é iniciado com a palavra reservada **Unit**, seguido de um nome que identifica a unidade e um ponto e vírgula. O nome do arquivo que armazena a unidade deve ter o mesmo nome que identifica a unidade.

A parte de interface segue o cabeçalho da unidade, e é iniciada pela palavra reservada **interface**. A parte de interface é o local onde são declarados constantes, tipos e variáveis que estarão acessíveis a todas as funções e procedimentos definidos na unidade. Além disso, essas estruturas de dados estarão também acessíveis a todos os programas que fizerem uso da unidade, por meio da declaração **uses**.

Nessa seção são também listados os cabeçalhos das funções e procedimentos que estarão acessíveis a todos os programas que fizerem uso da unidade. A definição completa dessas funções e procedimentos deve aparecer na parte de implementação da unidade.

A parte de implementação segue a parte de interface da unidade, e é iniciada pela palavra reservada **implementation**. A parte de implementação interface é o local onde são declarados constantes, tipos e variáveis que estão estarão acessíveis somente às funções e procedimentos definidos na unidade. Essas estruturas de dados não estão acessíveis aos programas que fizerem uso da unidade, por meio da declaração **uses**.

A parte de inicialização é iniciada com a palavra reservada **Begin** e terminada com a palavra reservada **End**, seguida de ponto. Entre as palavras **Begin** e **End** devem ser colocados os comandos que são executados antes dos comandos de um programa que faz uso da unidade.

Se a parte de inicialização não tiver comandos ela pode também ser declarada com a palavra reservada **End**, seguida de ponto.

De maneira geral, o formato de uma unidade Pascal possui a seguinte estrutura:

```
unit NomeUnidade ;

interface

Seção de definição e declaração de dados públicos

implementation

Seção de definição e declaração de dados privados

Begin
  Comandos
```

```
End.
```

Exemplo. A unidade minmax define as funções min e max, usadas para calcular o menor e o maior de dois valores inteiros.

```
unit minmax ;

interface
   function min(a, b: integer) : integer ;
   function max(a, b: integer) : integer ;

implementation
   function min(a, b: integer) : integer ;
   Begin
      if a < b then
         min:= a
      else
         min:= b ;
   End ;

   function max(a, b: integer) : integer ;
   Begin
      if a > b then
         max:= a
      else
         max:= b ;
   End ;
End.
```

Exemplo. O programa abaixo faz uso da unidade definida minmax.

```
Program extremos ;
uses minmax ;
var x, y: integer ;
Begin
   x:= 20 ;
   y:= 30 ;
   writeln( min(x,y):6, max(x,y):6 );
End.
```