Formato Básico Programa

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

1. Um cabeçalho, que dá nome ao algoritmo;
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;

A definição de cada uma dessas partes é opcional, mas deve seguir a ordem estabelecida. Por exemplo, uma função não pode ser definida antes da declaração de uma variável.

Em seguida, deve ser elaborada 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.

Seguindo essa ideia, o formato genérico de um programa Pascal tem a seguinte estrutura:

```
Program  NomePrograma ;

Seção de definições e declarações

Begin
  Comandos
End.
```

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

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  = constante
;
```

onde *constante* deve ser uma constante inteira, real, uma cadeia de caracteres ou um único caractere.

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

```
const dez  = 10 ;
```

Definicao 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* ou outro tipo de dados simples.

Exemplo

```
type
  intList = array[1..100] of integer ;
  matrix = array[0..9, 0..9] of real ;
  pInt = ^integer ;
```

Declaração de Variáveis

A declaração de uma variável faz com que o compilador reserve uma quantidade de espaço em memória suficientemente grande para armazenar um tipo de dados, além

de associar também um “nome” a esta posição de 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;
  caracter:  char;
  booleano: boolean;
```

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 seqüência 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).

ComandosAuxiliares

O compilador reconhece os seguintes comandos *Pascal*:

- Break
- Clrscr
- Continue
- Dec
- Delay
- Delete
- Exit
- Gotoxy
- Inc
- Insert
- Readkey
- Randomize
- Str
- TextBackground
- Textcolor
- Val

clrscr

Limpa a tela de impressão.

Sintaxe

```
clrscr ;
```

Exemplo

```
Program PascalZIM;
 begin
   clrscr;
   writeln( 'Olá, mundo.' );
 end.
```

readkey

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

Sintaxe

```
readkey ;
```

Exemplo

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

```
    readkey;
  end.
```

Como função, sua sintaxe é:

```
  readkey: integer;
```

Exemplo

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

randomize

Inicializa o gerador de números randômicos do compilador.

Sintaxe

```
  randomize;
```

Exemplo

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

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

      until i>10 ;
   end.
```

str

Usado para converter uma expressão numérica em uma cadeia.

Sintaxe

```
str( expressãoAritmética , variável ) ;
```

Onde:

- expressãoAritmética é um expressão do do tipo **integer**.

- *variável* é uma variável do tipo **string**.

Funcionamento

variável receberá o valor proveniente da conversão.

Exemplo

```
Program PascalZIM ;
 var s: string[2] ;
 Begin
   str( 26+3, s );
   writeln( s );
 End.
```

val

Usado para converter uma cadeia de caracteres em um inteiro ou real.

Sintaxe

```
val( expressãoLiteral , variável , codigoErro ) ;
```

Onde:

- expressãoLiteral 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.
```