APOSTILA DE TÉCNICAS DE PROGRAMAÇÃO E LINGUAGEM PASCAL

SUMÁRIO

1 INTRODUÇÃO À PROGRAMAÇÃO

1.1 Definição de Algoritmo

É a descrição, de forma lógica, dos passos a serem executados no cumprimento de determinada tarefa.

É a forma pela qual descrevemos soluções de problemas do nosso mundo, afim de serem implementadas utilizando os recursos do mundo computacional. Como este possuí severas limitações em relação ao nosso mundo, exige que sejam impostas algumas regras básicas na forma de solucionar os problemas para que possamos utilizar os recursos de hardware e software disponíveis.

1.1.1 Algoritmo x Programa

Um algoritmo é uma seqüência lógica de ações a serem executadas para se executar uma determinada tarefa. Um programa é a formalização de um algoritmo em uma determinada linguagem de programação, segundo suas regras de sintaxe e semântica, de forma a permitir que o computador possa entender a seqüência de ações.

1.2 Linguagem de Programação

Uma linguagem de programação é um conjunto de símbolos (comandos, identificadores, caracteres ASCII, etc. ...) e regras de sintaxe que permitem a construção de sentenças que descrevem de forma precisa ações compreensíveis e executáveis para o computador.

```
LINGUAGEM DE PROGRAMAÇÃO = SÍMBOLOS + REGRAS DE SINTAXE
```

Uma linguagem de programação é uma notação formal para descrição de algoritmos que serão executados por um computador. Como todas as notações formais, uma linguagem de programação tem dois componentes: Sintaxe e Semântica. A sintaxe consiste em um conjunto de regras formais, que especificam a composição de programas a partir de letras, dígitos, e outros símbolos. Por exemplo, regras de sintaxe podem especificar que cada parênteses aberto em uma expressão aritmética deve corresponder a um parênteses fechado, e que dois comandos quaisquer devem ser separados por um ponto-e-vírgula. As regras de semântica especificam o “significado” de qualquer programa, sintaticamente válido, escrito na linguagem.

1.2.1 Tipos de Linguagens de Programação

Existem diversas linguagens de programação, cada uma com suas características específicas e com níveis de complexidade e objetivos diferentes, como pode ser visto na tab. 1.1.

LINGUAGEM	CARACTERÍSTICAS
Linguagem de Máquina	Única compreendida pelo computador. Específica de cada computador.
Linguagens de Baixo Nível	Utiliza mnemônicos para representar instruções elementares Ex.: Assembly
Linguagens de Alto Nível	Utiliza instruções próximas da linguagem humana de forma a facilitar o raciocínio. Ex.: Uso Científico : Fortran Propósito Geral : Pascal, C, Basic Uso Comercial : Cobol, Clipper Uso específico : Lisp, Prolog

TABELA 1.1 – TIPOS DE LINGUAGENS DE PROGRAMAÇÃO E SUAS CARACTERÍSTICAS

1.2.2 Processo de Criação e Execução de um Programa

Embora seja teoricamente possível a construção de computadores especiais, capazes de executar programas escritos em uma linguagem de programação qualquer, os computadores, existentes hoje em dia são capazes de executar somente programas em linguagem de baixo nível, a *Linguagem de Máquina*.

Linguagens de Máquina são projetadas levando-se em conta os seguintes aspectos :

- rapidez de execução de programas;
- custo de sua implementação; e
- flexibilidade com que permite a construção de programas de nível mais alto.

Por outro lado, linguagens de programação de alto nível são projetadas em função de :

- facilidade de construção de programas; e
- confiabilidade dos programas.

O PROBLEMA É: Como a linguagem de nível mais alto pode ser implementada em um computador, cuja linguagem é bastante diferente e de nível mais baixo ?

SOLUÇÃO: Através da tradução de programas escritos em linguagens de alto nível para a linguagem de baixo nível do computador.

Para isso existem três tipos de programas tradutores : Montadores, Interpretadores e Compiladores. Vejamos o que cada um deles representa.

MONTADOR

Efetua a tradução de linguagem de montagem (Assembly) para a linguagem de máquina, da seguinte forma:

- obtém a próxima instrução do Assembly;
- traduz para as instruções correspondentes em linguagem de máquina;

- executa as instruções em linguagem de máquina; e
- repete o passo 1 até o fim do programa.

INTERPRETADOR

Efetua a tradução a de uma linguagem de alto nível para linguagem de máquina da seguinte forma:

- obtém próxima instrução do código-fonte em linguagem de alto nível;
- traduz para as instruções correspondentes em linguagem de máquina;
- executa as instruções em linguagem de máquina; e
- repete o passo 1 até o fim do programa

COMPILADOR

Efetua a tradução de todo o código-fonte em linguagem de alto nível para as instruções correspondentes em linguagem de máquina, gerando o código-objeto do programa. Em seguida é necessário o uso de um outro programa (Link-Editor) que é responsável pela junção de diversos códigos-objeto em um único programa executável.

Possibilidades de Erros no Programa:

- Erros de Compilação : erros de digitação e de uso da sintaxe da linguagem.
- Erros de Link-Edição : erro no uso de bibliotecas de sub-programas necessárias ao programa principal.
- Erros de Execução : erro na lógica do programa (algoritmo).

1.3 Critérios de Qualidade de um Programa

Vejamos alguns critérios para escrevermos um programa com qualidade:

Integridade: refere-se à precisão das informações manipuladas pelo programa, ou seja, os resultados gerados pelo processamento do programa devem estar corretos, caso contrário o programa simplesmente não tem sentido;

Clareza: refere-se à facilidade de leitura do programa. Se um programa for escrito com clareza, deverá ser possível a outro programador seguir a lógica do programa sem muito esforço, assim como o próprio autor do programa entendê-lo após ter estado um longo período afastado dele. O Pascal favorece a escrita de programas com clareza e legibilidade;

Simplicidade: a clareza e precisão de um programa são normalmente melhoradas tornando seu entendimento o mais simples possível, consistente com os objetivos do programa. Muitas vezes torna-se necessário sacrificar alguma eficiência de processamento, de forma a manter a estrutura do programa mais simples;

Eficiência: refere-se à velocidade de processamento e a correta utilização da memória. Um programa deve ter performance SUFICIENTE para atender às necessidade do problema e do usuário, bem como deve utilizar os recursos de memória de forma moderada, dentro das limitações do problema;

Modularidade: consiste no particionamento do programa em módulos menores bem identificáveis e com funções específicas, de forma que o conjunto desses módulos e a interação entre eles permite a resolução do problema de forma mais simples e clara; e

Generalidade: é interessante que um programa seja tão genérico quanto possível de forma a permitir a reutilização de seus componentes em outros projetos.

1.4 A Linguagem Pascal

1.4.1 Histórico

Origem: desenvolvida nos anos entre 1968 e 1970 por Nicklaus Wirth na Universidade Técnica de Zurique, Suíça. Em 1970 é disponibilizado o primeiro compilador para a linguagem.

Objetivo: desenvolver uma linguagem de programação disciplinada de alto nível para ensinar programação estruturada. Esta linguagem foi batizada com o nome de Pascal, em homenagem a Blaise Pascal, filósofo e matemático francês que viveu entre 1623 e 1662.

Padronização: ANSI (American National Standars Institute) e IEEE (Institute of Eletrical and Eletronics Engineers)

Padrão de Fato: Borland International cria em 1983 o Turbo Pascal.

Atualizações: durante os últimos anos foram lançadas diversas variações da linguagem Pascal-Padrão, implementando recursos que não são encontrados na estrutura padrão da linguagem. Nas mãos da Borland, já foram lançadas as versões 3.0, 4.0, 5.0 e 5.5 na década de 80. Durante a década de 90 foram lançadas as versões 6.0, 7.0 e o lançamento da linguagem Delphi, para programação em ambiente Windows.

1.4.2 O Turbo Pascal

O Turbo Pascal é mais que um simples compilador da linguagem Pascal, ele é um Ambiente Integrado de Desenvolvimento (IDE - Integrated Development Environment), consistindo de um conjunto de ferramentas de desenvolvimento integradas. Entre as ferramentas que compõem o Turbo Pascal temos:

- Editor de Código-Fonte
- Compilador
- Link-Editor
- Depurador
- Ajuda On-Line da Linguagem e do próprio IDE

Podemos citar alguns comandos principais do Turbo Pascal :

- Compilar o programa : ALT + F9
- Compilar e Executar o Programa : CTRL + F9
- Acessar o Menu : ALT + Letra realçada.
- Criar um novo programa : menu FILE | NEW
- Salvar o programa atual : F2
- Sair do Turbo Pascal : ALT + X
- Ver tela com os resultados do programa executado : ALT + F5

2 ESTRUTURA DE UM PROGRAMA EM PASCAL

Todo programa escrito em Pascal é subdividido em três áreas distintas: cabeçalho do programa, área de declarações e corpo do programa.

2.1 Cabeçalho do Programa

Esta área é utilizada para se fazer a identificação do programa com um nome. O cabeçalho de um programa é atribuído pela instrução **program** seguida de um nome. Ao final do nome deve-se colocar o símbolo ponto-e-vírgula (;). Não pode existir nenhuma variável no programa que tenha o mesmo nome dado ao programa.

Vejamos um exemplo:

```
program CALCULA_AREA;
```

Caso você tenha atribuído ao programa o nome *Soma* e também tenha atribuído este nome a uma variável no programa, quando for executado apresentará um mensagem de erro.

2.2 Área de Declarações

Esta área é utilizada para validar o uso de qualquer tipo de identificador que não seja predefinido, estando subdividida em sete sub-áreas: **uses, label, const, type, var, procedure** e **function.**

Inicialmente, vamos estudar apenas a sub-área **var.** As demais serão estudadas em capítulos posteriores. A sub-área **var** é utilizada na declaração das variáveis que serão utilizadas durante a execução de um programa, bem como, também o seu tipo. Desta forma, a linguagem Pascal efetua a reserva de espaço na memória para que as variáveis sejam utilizadas.

A declaração das variáveis é atribuída pela instrução **var** seguida da relação de variáveis. Após os nomes de cada variável deverá ser utilizado o símbolo dois-pontos (:), e após estes é mencionado o tipo de dado que a variável irá receber, seguido de ponto-e-vírgula.

Vejamos um exemplo:

```
var
  NOME   : string;
  IDADE  : integer;
  ALTURA : real;
```

Caso as variáveis sejam de mesmo tipo, estas poderão ser relacionadas separadas por vírgula. Vejamos um exemplo:

```
A, B, C : integer;
```

2.3 Corpo do Programa

O programa propriamente dito em Pascal está escrito na área denominada corpo do programa. esta área tem início com a instrução **begin** e é finalizada pela instrução **end** seguida do símbolo ponto (.). O uso destas instruções caracteriza o que é chamado de bloco, como indicado abaixo:

```
begin
  instruções;
  (...)
end.
```

Na área denominada corpo do programa, poderão existir, dependendo da necessidade, vários blocos.

2.4 Exemplo de um Programa em Pascal

Para se colocar em prática o que foi explicado até este momento, considere o seguinte exemplo de um problema:

"Desenvolver um programa que efetue a leitura de dois valores numéricos. Faça a operação de adição entre os dois valores e apresente o resultado obtido."

Note que sempre estaremos diante de um problema, e que este deverá ser resolvido primeiro por nós, para que depois seja resolvido por um computador. Primeiramente, você deve entender bem o problema, para depois buscar a sua solução dentro de um computador, ou seja, você deverá "ensinar" a máquina a resolver seu problema, através de um programa. Desta forma, o segredo de uma boa lógica está na compreensão adequada do problema a ser solucionado. Vejamos nosso problema citado acima:

Algoritmo:

1. Ler um valor para a variável A;
2. Ler outro valor para a variável B;
3. Efetuar a soma das variáveis A e B, colocando o resultado na variável X;
4. Apresentar o valor da variável X após a operação de soma dos dois valores fornecidos.

Completada a fase de interpretação do problema e da definição das variáveis a serem utilizadas passa-se para a fase de codificação de nosso programa para a linguagem Pascal.

Programa em Pascal

```
program ADICIONA_NUMEROS;

var
   X : integer;
   A : integer;
   B : integer;

begin
   readln(A);
   readln(B);
   X := A + B;
   writeln(X);
end.
```

Entrada e Saída dos Dados

Você deve ter observado os comandos **readln** e **writeln.** Estes comandos são chamados de entrada e saída dos dados, respectivamente. A instrução **readln** permite a entrada de dados via teclado e a instrução **writeln** gera a saída de dados via vídeo.

Poderiam também ser utilizadas as instruções **write** e **read.** Mas qual a diferença entre elas?

É muito simples: a colocação da **ln** indica **line new**, ou seja, nova linha. Será realizado um "pulo" de linha após a leitura ou exibição dos dados. As instruções **write** e **read** permitem manter o cursor na mesma linha, após a exibição das mensagens ou leitura de dados.

3 VARIÁVEIS e CONSTANTES

3.1 Identificadores

Os identificadores são nomes a serem dados a variáveis, tipos definidos, procedimentos, funções e constantes nomeadas.

Devem seguir as seguintes regras de construção:

- iniciar sempre por uma letra (`a - z , A - Z`) ou um *underscore* (`_`);
- o restante do identificador deve conter apenas letras, *underscores* ou dígitos (`0 - 9`). **Não pode** conter outros caracteres; e
- pode ter qualquer tamanho, desde que os primeiros 63 caracteres sejam significativos.

Não existe distinção entre letras maiúsculas e minúsculas no nome de um identificador. Por exemplo, os nomes ALPHA, alpha e Alpha são equivalentes. Atenção para o fato de que identificadores muito longos são mais fáceis de serem lidos pelas as pessoas quando se usa uma mistura de letras maiúsculas e minúsculas; por exemplo, SalarioMinimo é mais fácil de se ler do que SALARIOMINIMO.

Vejamos alguns exemplos de identificadores válidos:

```
PAGAMENTO
Soma_Total
MaiorValor
Media1
_Media
```

e alguns exemplos de identificadores inválidos:

```
%Quantidade   O símbolo % não é permitido
4Vendedor     Não pode começar com um número
Soma Total    Não pode ter espaços entre as letras
```

Observação : Um identificador deverá ficar inteiramente contido em uma linha do programa, ou seja você não pode começar a digitar o nome do identificador numa linha e acabar em outro.

3.2 Palavras Reservadas

Pascal reconhece certo grupo de palavras como sendo reservadas. Essas palavras tem significado especial e não podem ser usadas como identificadores em um programa. A tab. 3.1 apresenta todas as palavras reservadas do Pascal Padrão:

and	downto	In	packed	to
array	else	inline	procedure	type
asm	End	interface	program	unit
begin	File	Label	record	until
case	For	mod	repeat	until
const	Foward	nil	set	uses
constructor	Function	not	shl	var
destructor	Goto	object	shr	while
div	If	of	string	with
do	implementation	or	then	xor

TABELA 3.1 – PALAVRAS RESERVADAS DO PASCAL

3.3 Comentários

Comentários são textos escritos dentro do código-fonte para explicar ou descrever alguns aspectos relativos ao mesmo. Os comentários podem ser colocados em qualquer lugar do programa onde um espaço em branco possa existir.

Você pode colocar comentários de duas formas: ou envolvendo o texto entre chaves "{..}" ou entre " (* .. *)". Quando o compilador encontra o símbolo "{" ele salta todos os caracteres até encontrar um "}". Da mesma forma, todos os caracteres que seguem "(*" são pulados até ser detectado o símbolo "*)". Como resultado disso, qualquer uma das formas pode ficar dentro da outra; por exemplo {...(*...*)...} é um comentário.

3.4 Tipos de Dados

Os dados são representados pelas informações a serem processadas por um computado. Um tipo de dados especifica as características, ou seja os valores e operações possíveis de serem utilizados com um dado desse tipo. Toda variável e constante usada em um programa tem um tipo associado a ela. A linguagem Pascal fornece ao programador um conjunto de tipos de dados predefinidos.

3.4.1 Tipos de Dados Inteiros

São caracterizados tipos inteiros, os dados numéricos positivos ou negativos, excluindo-se qualquer número fracionário. Em Pascal, este tipo de dado pode ser referenciado por um dos seguintes identificadores:

Tipo de dado inteiro	Faixa de abrangência	Tamanho(bytes)
shortint	de –128 até 127	1 byte
integer	de –32.768 a 32.767	2 bytes
longint	de –2.147.483.648 a 2.147.483.647	4 bytes
byte	de 0 até 255	1 byte
word	de 0 até 65535	2 bytes

Vejamos um exemplo:

```
var
  NumAlunos    : integer;
  Cont,cont1   : integer;
```

3.4.2 Tipos de Dados Reais

O tipo de dado real permite trabalhar com números fracionários, tanto positivos como negativos, sendo sua capacidade de armazenamento maior que dos números inteiros. Vejamos os tipos:

Tipo de dado real	Faixa de abrangência	Tamanho(bytes)
real	de 2.9 e-39 até 1.7 e38	6 bytes
single	de 1.5 e-45 até 3.4 e38	4 bytes
double	de 5.0 e-324 até 1.7e308	8 bytes
extended	de 3.4 e-4.932 até 1.1 e4.932	10 bytes
comp	de –9.2 e18 até 9.2 e18	8 bytes

Vejamos um exemplo:

```
var
  Nota           : real;
  Salario, media : real;
```

3.4.3 Tipos de Dados Caracteres

São considerados tipos caracteres, as seqüências contendo letras, números e símbolos especiais. Uma seqüência de caracteres, em Pascal, deve ser representada entre apóstrofos (‘’). Este tipo de dado é referenciado pelo identificador **string**, podendo armazenar de 1 até 255 caracteres. Podemos ainda especificar um tamanho menor do que os 255 caracteres permitidos. Vejamos a sintaxe para criarmos uma variável do tipo string com tamanho limitado.

Sintaxe:

```
variável : string[tamanho];
```

Vejamos um exemplo do uso de strings:

```
var
  Frase : string;
  Nome  : string[45];
```

Existe ainda o tipo **char**, utilizado da mesma forma que o tipo **string**, porém com uma pequena diferença: é usado para strings de apenas um caracter. Vejamos um exemplo do tipo de dado **char**:

```
var
  Sexo : char;
```

3.4.4 Tipos Lógicos

São caracterizados tipos lógicos, os dados com valores **true** (verdadeiro) e **false** (false). Este tipo de dado também e chamado de tipo boleano. Ele é representado pelo identificador **boolean.**

Vejamos um exemplo da utilização do tipo de dado **boolean:**

```
var
  Aprovado : boolean;
  Confirma : boolean;
```

3.5 Variáveis

Variável, no sentido de programação, é uma região previamente identificada, que tem por finalidade armazenar informações (dados) de um programa temporariamente. Uma variável armazena apenas um valor por vez. Sendo considerado como valor o conteúdo de uma variável, este valor está associado ao tipo de dado da variável.

Sintaxe:

```
var
  identif [ , identif ]... : tipo-de-dado;
  [identif [ , identif ]... : tipo-de-dado; ] ...
```

onde: *tipo-de-dado* é um dos tipos predefinidos ou um tipo definido pelo usuário:

Vejamos um exemplo:

```
var
  Soma, Total, Salario   : real;
  Idade, Contador        : integer;
```

3.6 Constantes

Uma constante é um valor que não pode ser alterado durante a execução do programa, sendo que seu tipo é definido por seu conteúdo.

Sintaxe:

```
const
  identificador = expressão;
  ...
  identificador = expressão;
```

Vejamos um exemplo:

```
program Area_Circulo;
{ Programa para calcular a área de um círculo. }

const
  PI = 3.141519265;
var
  Area, Comprimento, Raio   : real;

begin
  writeln( ´Digite o Raio : ´ );
  readln( Raio );
  Area := PI * Raio * Raio;
  Comprimento := 2 * PI * Raio;
  writeln( ´Área = ´, Area );
  writeln( ´Comprimento da Circunferencia = ´,    Comprimento );
end.
```

4 OPERADORES E EXPRESSÕES

Um programa tem como característica fundamental a capacidade de processar dados. Processar dados significa realizar operações com estes dados. O uso de operadores permite a realização de tais operações.

Exemplo: o símbolo + é um *operador* que representa a operação aritmética de adição.

Uma **expressão** é um arranjo de operadores e operandos. A cada expressão válida é atribuído um valor numérico.

Exemplo:

4 + 6 é uma expressão cujo valor é *10*.

Os operandos podem ser variáveis, constantes ou valores gerados por funções. Os operadores identificam as operações a serem efetuadas sobre os operandos. Cada tipo de dados possui um conjunto de operadores relacionados. Os operadores classificam-se em Unários e Binários, conforme tenham um ou dois operandos, respectivamente.

4.1 Prioridade das Operações

Se vários operadores aparecerem em uma expressão, a ordem de execução das operações será dada segundo os critérios abaixo :

- pelo emprego explícito de parênteses;
- pela ordem de precedência existente entre os operadores; e
- se existirem operadores de mesma ordem de precedência, a avaliação será feita da esquerda para a direita..

Vejamos a ordem de precedência dos operadores (da maior para a menor):

Operadores	Categoria
( )	Parêntesis
not	Operador Lógico Unário
* / div mod	Operadores Multiplicativos
and or	Operadores Lógicos
= <> < > <= >=	Operadores Relacionais

4.2 Tipos de Expressões

Existem três tipos básicos de expressões:

Numéricas: uma expressão é numérica quando os seus operandos são numéricos (inteiros ou reais) e o resultado também é um valor numérico.

Literais: Uma expressão é literal quando os seus operandos são literais do tipo **string** (não pode ser char) e o resultado também é um valor literal. Só existe um único operador para se usar em expressões literais, o operador Concatenação(+).

Boolenas: Uma expressão é booleana quando seu resultado é do tipo boolean (TRUE ou FALSE), podendo seu operando serem de qualquer tipo de dados. Nas expressões relacionais são usados os operadores Lógicos e os Relacionais.

4.3 Tipos de Operadores

Existem vários operadores para a realização de expressões em Pascal. Veremos agora algumas das mais importantes.

4.3.1 Operador de Atribuição

O operador de atribuição é utilizado para atribuir o valor de uma expressão a uma variável.

Sintaxe:

```
identificador_variável := expressão;
```

Vejamos um exemplo:

```
A := 10;
Nome := ´Josias´;
```

4.3.2 Operadores Aritméticos

Os operadores aritméticos são utilizados para efetuar operações aritméticas com número inteiros e reais. A tab. 4.1 apresenta os operadores aritméticos existentes em Pascal.

Operador	Símbolo
Subtração	-
Adição	+
Multiplicação	*
Divisão Real	/
Divisão Inteira (truncada)	div
Resto da Divisão Inteira	mod
Inverte Sinal	-
Mantém Sinal	+

TABELA 4.1 – OPERADORES ARITMÉTICOS

Vejamos um exemplo:

```
var
  A, B : integer;
  C, D : real;

begin
  A := 1;
  B := 3;
  C := 5;
  D := 10;
  A := 1 + B;
  A := B + D;  { errado, D é real }
  B := 10 div 3;
  A := 10 mod 3;
  C := D / C;
  D := 10 div C;    { errado, o operado div é só para
     inteiros }
  A := -1;
  B := 5 + A;
  B := -A;
  C := D * A;
  B := C * B;  { errado, C é real }
end.
```

4.3.3 Operador de Concatenação

O operador de concatenação efetua a junção de duas variáveis ou constantes do tipo string.

Vejamos um exemplo:

```
var
  PreNome, SobreNome, NomeCompleto : string[ 30 ];

begin
{ Suponhamos o nome Josias Lima Santos }
   PreNome     := ´Josias´;
   SobreNome   := ´Santos´;

   NomeCompleto := PreNome + SobreNome;
   writeln( NomeCompleto );

   NomeCompleto := ´Jose´ + ´Maria´;
   writeln( NomeCompleto );
   ...
end.
```

4.3.4 Operadores Relacionais

Os operadores relacionais são utilizados para efetuar a comparação entre dados de mesmo tipo. Os operadores relacionais existentes em Pascal podem ser vistos na tab.4.2.

Operador	Símbolo
Maior que	>
Menor que	<
Maior ou igual	>=
Menor ou igual	<=
Igual	=
Diferente	<>

TABELA 4.2 – OPERADORES RELACIONAIS

Vejamos um exemplo:

```
var
  Nota1, Nota2           : real;
  NomeAluno1, NomeAluno2 : string[ 30 ];
  A, B, C                : integer;

begin
  A := 2;
  B := 3;
  C := 1;
  if B = A + C then
    writeln( B );
  Nota1 :=  5.0;
  Nota2 := 10.0;
  if Nota1 < Nota2 then
    writeln( Nota1 );
  NomeAluno1 := ´Maria Jose´;
  NomeAluno2 := ´MariaJose´;
  if NomeAluno1 < NomeAluno2 then
    writeln( NomeAluno1 );
end.
```

4.3.5 Operadores Lógicos

Os operadores lógicos são utilizados para se analisar duas ou mais expressões interrelacionadas. A tab. 4.3 exibe os operadores lógicos existentes em Pascal.

Operador	Símbolo
E	and
OU	or
NÃO	not

TABELA 4.3 – OPERADORES LÓGICOS

Vejamos um exemplo:

```
var
  Nota1, Nota2 : real;
  NomeAluno1, NomeAluno2 : string[ 30 ];
  A, B, C : integer;

begin
  A := 2;
  B := 3;
  C := 1;
  NomeAluno1 := ´Maria Jose´;
  NomeAluno2 := ´MariaJose´;

  if ( B = A + C ) and ( NomeAluno1 <> NomeAluno2 ) then
     writeln( NomeAluno1, B );
  if ( A = C ) or ( NomeAluno1 = NomeAluno2 ) then
     writeln( NomeAluno1 );
  if not( A = C ) then
     writeln( NomeAluno1 );
end.
```

Abaixo, segue três tabelas, chamadas *tabela-verdade*, contendo o resultado do uso dos operadores lógicos sobre dois operandos.

OPERADOR AND

OPERANDO 1	OPERANDO 2	RESULTADO
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

OPERADOR OR

OPERANDO 1	OPERANDO 2	RESULTADO
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

OPERADOR NOT

OPERANDO	RESULTADO
TRUE	FALSE
FALSE	TRUE

4.4 Funções predefinidas

O Pascal oferece um conjunto de funções predefinidas (built-in functions), que são usadas com vários tipos de dados simples. As funções, na maioria das vezes, necessitam de dados como parâmetro (dados de entrada). Vejamos algumas dessas funções:

Funções Matemáticas:

Nome Função	Objetivo	Tipo Parâmetro	Tipo do Retorno
abs(x)	Calcula o valor absoluto de x.	inteiro ou real	o mesmo que x
cos(x)	Calcula o coseno de x em radianos	inteiro ou real	real
exp(x)	Calcula e^x, em que e=2.7182818 é sistema natural de logaritmos neperianos.	inteiro ou real	real
ln(x)	Calcula o logaritmo natural de x (x>0)	inteiro ou real	real
exp(ln(x)*y)	Retorna x elevado a y {utilizando regras de logaritmos}.	Inteiro ou real	real
sin(x)	Calcula o seno de x em radianos	inteiro ou real	real
sqr(x)	Calcula o quadrado de x	inteiro ou real	o mesmo que x
sqrt(x)	Calcula a raiz quadrada de x (x>=0)	inteiro ou real	real
odd(x)	Determina se x é par ou impar TRUE, X é par / FALSE, X é impar	inteiro	boolean
random(x)	Retorna um número pseudo-aleatório entre 0 e x. Se x não for especificado retorna um valor entre 0.0 e 1.0	inteiro	real
pi	Retorna o valor de PI (3.1415...)	Nenhum	real

Funções Literais:

Nome Função	Objetivo	Tipo Parâmetro	Tipo Retorno
length(x)	Determina o número de caracteres de x	string	inteiro
concat(x1, x2, x3,...)	Concatena duas ou mais strings (máx 255 caracteres)	string	string
copy(x, y, z)	Retorna uma subcadeia da cadeia x, com z caracteres, começando no caracter y.	string, inteiro, inteiro	string
UpCase(x)	Retorna x convertido para maiúscula	char	char

Funções para Conversão:

Nome Função	Objetivo	Tipo Parâmetro	Tipo Retorno
trunc(x)	Trunca x para um número inteiro	real	inteiro
int(x)	Retorna a parte inteira de x	real	real
frac(x)	Retorna a parte fracionária de x	real	real
round(x)	Arredonda x para um inteiro	real	inteiro
chr(x)	Determina o caracter ASCII representado por x	inteiro	char

Funções e Procedimentos de Uso Geral:

Nome Função	Objetivo	Tipo Parâmetro	Tipo do Retorno
sizeof(x)	Retorna o número de byte de x	qualquer tipo	inteiro
gotoxy(x,y)	Move o curso para a coluna x e linha y	inteiro	

5 ESTRUTURAS DE DECISÃO

Imagine um programa que apresente a média escolar de um aluno. Até aqui, muito simples; mas além de calcular a média, o programa deve apresentar uma mensagem indicando se o aluno foi aprovado ou reprovado. Esta variável, que até então servia para cálculos, agora passa a ser uma variável de controle, onde partir dela poderemos tomar uma decisão sobre o *status* do aluno. Em Pascal existem duas instruções para efetuar tomadas de decisão e desvios de operações:

- a instrução de decisão simples **if...then**; e
- a instrução de decisão composta **if...then...else.**

5.1 A instrução if..then

A instrução **if...then** tem por finalidade tomar uma decisão e efetuar um desvio no processamento, dependendo, é claro, da condição atribuída ser *verdadeira* ou *falsa*. Sendo a condição *verdadeira*, será executada a instrução que estiver escrita após a instrução **if..then**. Se a instrução for *falsa*, serão executadas as instruções que estejam após as instruções consideradas verdadeiras.

Sintaxe:

```
  if <condição> then
    <instrução para condição verdadeira>;
  <instrução para condição falsa ou após condição ser
verdadeira>;
```

Vejamos um exemplo de um trecho de programa com o uso da instrução **if...then**:

```
...
if (x > 10) then
  writeln("O valor da variavel X e 10");
...
```

Caso venha a existir mais de uma instrução verdadeira para uma determinada condição, estas deverão estar inseridas em um bloco. Um bloco é o conjunto de instruções estar entre **begin** e **end.**

Sintaxe:

```
  if <condição> then
  begin
    <instrução para condição verdadeira>
    <instrução para condição verdadeira>
    <instrução para condição verdadeira>
    <instrução para condição verdadeira>
   end;
  <instrução para condição falsa ou após condição ser
verdadeira>
```

Observe que o **end** é finalizado com um ponto-e-vírgula (;) e não apenas ponto. O uso de ponto ao final de um **end** ocorre somente na finalização de um programa.

Vejamos um exemplo de um trecho de programa com a utilização de um bloco de instruções:

```
...
if (x > 10) then
begin
  writeln("O valor da variavel X e 10");
  x : = x + 1;
  writeln("O valor da variavel X agora  e11");
end;
...
```

Vejamos um exemplo completo de um programa com o uso da instrução **if...then.**

"Ler dois valores inteiros e independentemente da ordem em que foram inseridos, estes deverão ser exibidos em ordem crescente, ou seja, se forem fornecidos 5 e 3 respectivamente, deverão ser apresentados 3 e 5. O programa em questão deverá efetuar a troca dos valores entre as duas variáveis"

Algoritmo

1. Ler dois valores inteiros (estabelecer variáveis A e B);
2. Verificar se o valor de A é maior que o valor de B:
   a. se for verdadeiro, efetuar a troca de valores entre as variáveis;
   b. se for falso, pedir para executar o que está estabelecido no passo 3;
3. Apresentar os valores das duas variáveis.

Programa em Pascal

```
program ORDENA;
var
  X, A, B : integer;

begin
  write ('Informe uma valor para a variavel A: ');
  readln(A);
  write ('Informe uma valor para a variavel B: ');
  readln(B);
  writeln;
  if (A > B) then
  begin
    X := A;
    A := B;
    B := X;
  end;
  writeln('Os valores ordenados são: ');
  write(A, '  ', B);
end.
```

5.2 A instrução if...then...else

Assim como a instrução **if...then,** a instrução **if...then...else** tem por finalidade tomar uma decisão e efetuar um desvio no processamento. Se a condição for *verdadeira* será executada a instrução logo abaixo do **if**. Sendo a condição *falsa*, será executada a instrução que estiver posicionada logo após a instrução **else**. O conceito de blocos de instruções vale também para esta instrução.

Sintaxe:

```
if  <condição> then
  <instruções para condição verdadeira>
else
  <instruções para condição falsa>;
```

Caso venha a existir mais de uma instrução verdadeira ou falsa para uma condição, estas deverão estar inseridas em um bloco.

Sintaxe:

```
if  <condição> then
begin
  <instruções para condição verdadeira>;
  <instruções para condição verdadeira>;
end
else
begin
  <instruções para condição falsa>;
  <instruções para condição falsa>;
end;
```

Observe que nos dois casos abordados acima, qualquer instrução que antecede a instrução **else** está escrita sem o ponto-e-vírgula (;). Isto ocorre pelo fato de a instrução **else** ser uma extensão da instrução **if...then**, e sendo assim, o final da condição somente ocorre após o processamento da instrução **else.**

Vejamos um exemplo:

"*Ler dois valores numéricos e efetuar a adição. Caso o valor somado seja maior ou igual a 10, este deverá ser apresentado somando-se a ele mais 5. Caso o valor somado não seja maior ou igual a 10, esta deverá ser apresentado subtraindo-se 7.*"

Algoritmo

1. Ler dois valores (variáveis A e B);
2. Efetuar a soma dos valores A e B, atribuindo o resultado da soma a uma variável X;
3. Verificar se X é maior ou igual a 10:
   a. se for verdadeiro, mostrar X+5;
   b. se for falso, mostrar X-7.

Programa em Pascal

```
program ADICIONA_NUMEROS;
var
   X, A, B : integer;
begin
   write(‘Informe um valor para a variavel A: ‘);
   readln(A);
   write(‘Informe um valor para a variavel B: ‘);
   readln(B);
   writeln;
   X := A + B;
   write (‘O resultado equivale a:  ‘);
   if  (X>=10) then
      writeln(X+5)
  else
     writeln(X-5)
  end.
```

Podemos trabalhar com o relacionamento de duas ou mais condições dentro da instrução **if...then**. Para estes casos, é necessário trabalhar com os operador lógicos, vistos anteriormente. Vejamos alguns exemplos :

Operador AND

```
program TESTA-LOGICA-AND;
var
   NUMERO : integer;
begin
   write(‘Informe um numero: ’);
   readln(NUMERO);
   writeln;
   if (NUMERO >= 20) and  (NUMERO <=90) then
      writeln(‘O numero esta na faixa de 20 a 90’)
   else
      writeln(‘O numero esta fora da faixa de 20 a 90’);
end.
```

Operador OR

```
program TESTA-LOGICA-OR;
var
  A, B : integer;
begin
  write(‘Informe um valor para a variavel A: ‘);
  readln(A);
  write(‘Informe um valor para a variavel B: ‘);
  readln(B);
  writeln;
  if  (A>0)  or (B>0) then
    writeln(‘Um dos numeros e positivo’);
```

Operador NOT

```
program TESTA-LOGICA-NOT;
var
  A, B, C, X : integer;
begin
  write(‘Informe um valor para a variavel A: ‘);
  readln(A);
  write(‘Informe um valor para a variavel B: ‘);
  readln(B);
  write(‘Informe um valor para a variavel X: ‘);
  readln(X);
  if not (X>5) then
    C := (A + B) * X
  else
    C := (A - B) * X;
  writeln(‘O resultado da variavel C corresponde a:’, C);
end.
```

6 ESTRUTURAS DE REPETIÇÃO (LOOPS)

Existem situações onde é necessário repetir um determinado trecho de um programa um certo número de vezes. Imagine que você tenha que executar um determinado bloco de instruções 5 vezes. Com os conhecimentos que você tem até agora, seria necessário repetir as instruções CINCO vezes, tornando seu trabalho muito cansativo. Assim, existem as estruturas de repetição, que permitem que você execute estas tarefas de forma mais simplificada. Podemos chamar as estruturas de repetição de **laços** ou **loops**, sendo que podem existir dois tipos de laços de repetição:

- *laços finitos:* neste tipo de laço se conhece previamente o número de repetições que serão executadas; e
- *laços infinitos:* neste tipo de laço não se conhece previamente o número de repetições que serão executadas. São também chamados de **condicionais**, pelo fato de encerrarem sua execução mediante uma determinada condição.

6.1 Instrução for

O comando FOR executa repetitivamente um comando enquanto é atribuído uma série de valores a uma **var**iável de controle (contador do FOR).

Sintaxe :

```
for variavel := <início> to/downto <fim> do
  instrução;

        ou

for variavel := <início> to/downto <fim> do
begin
  instrução1;
  instrução2;
  instrução3;
end;
```

Vejamos um exemplo da utilização da estrutura de repetição **for.**

""Ler dois valores inteiros. Realize a adição destes números, armazenando o resultado em uma terceira variável. Apresente o valor encontrado. Repita esta operação 5 vezes."

Algoritmo

1. Criar uma variável para servir como contador, que irá variar de 1 até 5;
2. Ler dois valores (variáveis A e B);
3. Efetuar o cálculo, armazenando o resultado em R;
4. Apresentar o valor calculado contido na variável R;
5. Repetir os passos 2, 3 e 4 até que o contador seja encerrado.

Programa em Pascal

```
program EXEMPLO_FOR;
var
  A, B, R, I : integer;
begin
   for I := 1 to 5 do
    begin
      write('Entre um valor para A: ');
      readln(A);
      write('Entre um valor para B: ');
      readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ',R);
      writeln;
    end;
end.
```

Quando executado o programa, o conjunto de instruções situadas abaixo da instrução **for** serão executadas durante 5 vezes, pois a variável **I** (variável de controle) inicializada com o valor 1 é incrementada com mais 1 a cada vez que o processamento passa pela instrução **for**. Este tipo de estrutura de repetição pode ser utilizado todas as vezes que se tiver necessidade de repetir trechos finitos, ou seja, quando se conhece o número de vezes que se deseja repetir tais instruções.

6.2 Instrução while...do

Esta estrutura de repetição se caracteriza por efetuar um teste lógico no início de um *loop*, verificando se é permitido executar o trecho de instruções abaixo dela. A estrutura **while...do** tem o seu funcionamento controlado por condição. Desta forma, poderá executar um determinado conjunto de instruções enquanto a condição verificada permanecer verdadeira. No momento em que a condição se torna falsa, o processamento da rotina é desviado para fora do *loop.* Sendo a condição falsa logo no início do *loop*, as instruções contidas nele são ignoradas.

Sintaxe:

```
while <condição> do
  <instrução para condição verdadeira>;

     ou

while <condição> do
begin
  <instruções para condição verdadeira>;
end;
```

Vejamos novamente o exemplo da adição de dois números, utilizado agora a instrução **while...do.**

""Ler dois valores inteiros. Realize a adição destes números, armazenando o resultado em uma terceira variável. Apresente o valor encontrado. Repita esta operação 5 vezes."

Algoritmo

1. Criar uma variável para servir como contador, com valor inicial igual a1;
2. Enquanto o valor do contador for menor ou igual a 5, processar os passos 3, 4 e 5;
3. Ler dois valores (variáveis A e B);
4. Efetuar o cálculo, armazenando o resultado em R;
5. Apresentar o valor calculado contido na variável R;
6. Acrescentar ao contador mais 1; e
7. Quando o contador for maior que 5, encerrar o processamento.

Programa em Pascal

```
program EXEMPLO_WHILE_DO1;
var
  A, B, R, I : integer;
begin
   while (I <= 5) do
    begin
      write('Entre um valor para A: ');
      readln(A);
      write('Entre um valor para B: ');
      readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ',R);
      writeln;
      I := I + 1;
    end;
end.
```

Além da utilização das variáveis **A**, **B** e **R**, foi necessário criar uma terceira variável, no caso **I**, para controlar a contagem do número de vezes que o trecho de programa deverá ser executado.

Imagine ainda uma outra situação, onde o usuário deseja executar a rotina do programa várias vezes, mas este não sabe quantas vezes ao certo deverá executar o trecho de programa. Neste caso não seria conveniente manter um contador para controlar o looping, e sim, seria melhor que o programa fizesse ao usuário uma pergunta, solicitando se o mesmo deseja ou não continuar executando o programa.

Vejamos o mesmo exemplo, agora sob esta nova situação:

Algoritmo

1. Criar uma variável para ser utilizada como resposta;
2. Enquanto a resposta for sim, executar os passos 3, 4 e 5;
3. Ler dois valores (variáveis A e B);
4. Efetuar o cálculo, armazenando o resultado em R;
5. Apresentar o valor calculado contido na variável R;
6. Quando resposta for diferente de *sim*, encerrar o processamento.

Programa em Pascal

```
program EXEMPLO_WHILE_DO2;
var
   A, B, R : integer;
   RESP    : string;
begin
    RESP := 'SIM';
    while (RESP = 'SIM') or (RESP = 'S') do
     begin
        write('Entre um valor para A: ');
        readln(A);
        write('Entre um valor para B: ');
        readln(B);
        writeln;
        R := A + B;
        writeln('O resultado corresponde a: ',R);
        writeln;
        write('Deseja continuar ??');
        readln(RESP);
    end;
end.
```

No programa acima, o contador foi substituído pela variável **RESP**, que enquanto tiver o seu valor igual a *SIM* ou igual a *S*, executará as instruções contidas entre as instruções **begin** e **end** do comando **while...do**.

6.3 Instrução repeat...until

Esta estrutura caracteriza-se por efetuar um teste lógico no final de um looping, sendo parecida com a estrutura **while...do**. Seu funcionamento é controlado também por decisão. Esta instrução irá efetuar a execução de um conjunto de instruções pelo menos uma vez antes de verificar a validade da condição estabelecida.

Desta forma, **repeat** irá processar um conjunto de instruções, no mínimo uma vez, até que a condição se torne *verdadeira*. Para a estrutura **repeat** um conjunto de instruções é executado enquanto a condição se mantém *falsa* e até que se torne *verdadeira*.

Sintaxe:

```
repeat
  <instrução1 até que a condição seja verdadeira> ;
  <instrução1 até que a condição seja verdadeira> ;
  <instrução1 até que a condição seja verdadeira> ;
until <condição>;
```

Vejamos novamente nosso exemplo da adição, agora utilizando a instrução **repeat...until**:

Algoritmo

1. Criar uma variável para servir como contador, com valor inicial igual a 1;
2. Ler dois valores (variáveis A e B);
3. Efetuar o cálculo, armazenando o resultado em R;
4. Apresentar o valor calculado contido na variável R;
5. Acrescentar o contador com mais 1;
6. Repetir os passos 2, 3 e 4 até que o contador seja maior que 5.

Programa em Pascal

```
program EXEMPLO_REPEAT1;
var
  A, B, R, I : integer;
begin
  I := 1;
  repeat
    write('Entre um valor para A: ');
    readln(A);
    write('Entre um valor para B: ');
    readln(B);
    writeln;
    R := A + B;
    writeln('O resultado corresponde a: ',R);
    writeln;
    I := I + 1;
  until (I > 5);
end.
```

Assim que o programa é executado, a variável contador é inicializada com valor igual a 1. Em seguida a instrução **repeat** indica que todo trecho de instruções situado até a instrução **until** será executado repetidas vezes até que sua condição se torne verdadeira, ou seja, até que **I** seja maior que 5.

Vejamos o mesmo exemplo, agora considerando que o usuário encerrará a execução segundo sua vontade.

Algoritmo

1. Criar uma variável para ser utilizada como resposta;
2. Ler dois valores (variáveis A e B);
3. Efetuar o cálculo, armazenando o resultado em R;
4. Apresentar o valor calculado contido na variável R;
5. Perguntar ao usuário se deseja continuar executando o programa; e
6. Repetir o passos 2, 3 e 4 até que a resposta do usuário seja *não*.

Programa em Pascal

```
program EXEMPLO_REPEAT2;
var
   A, B, R : integer;
   RESP    : string;
begin
    RESP := 'SIM';
    repeat
        write('Entre um valor para A: ');
        readln(A);
        write('Entre um valor para B: ');
        readln(B);
        writeln;
        R := A + B;
        writeln('O resultado corresponde a: ',R);
        writeln;
        write('Deseja continuar Sim/Não ??');
        readln(RESP);
        writeln;
    until (<RESP <> 'SIM') and (<RESP <> 'sim');
end.
```

Assim que o programa é executado, a variável **RESP** é inicializada com o valor *SIM*. Em seguia, a instrução **repeat** indica que todas as instruções situadas até a instrução **until** deverão ser executadas até que sua condição se torne verdadeira, ou seja que o usuário responda algo diferente de *SIM*, e também diferente de *sim.*

7 VETORES, MATRIZES E REGISTROS

Ao utilizamos variáveis, podemos armazenar apenas um valor por vez. Agora, imagine um programa onde precisa-se armazenar as notas de 5 provas realizadas por um aluno. Com os conhecimentos que você possui até agora, seria necessário criar cinco variáveis para armazenar as notas. Desta forma:

```
NOTA1, NOTA2, NOTA3, NOTA4, NOTA5 : integer;
```

Em Pascal existem estruturas que nos permitem agrupar várias informações dentro de uma mesma variável. Estas estruturas são chamadas de **vetores** e **matrizes.**

7.1 Vetores

Este tipo de estrutura é também chamado de **matriz unidimensional**. Um vetor é representado por seu nome, tamanho e seu tipo.

Sintaxe:

```
<vetor> : array[tamanho]  of  <tipo_de_dado>;
```

onde: *vetor* é o nome atribuído ao vetor, *tamanho* é o tamanho atribuído ao vetor, em número de elementos e *tipo_de_dado* é o tipo de elemento armazenado (inteiro, real...).

Como já visto, uma variável somente pode conter um valor por vez. No caso dos vetores, estes poderão armazenar mais de um valor por vez, pois são dimensionados exatamente para este fim.

Vejamos como poderíamos declarar um vetor para armazenar as cinco notas do aluno:

```
NOTAS : array[1..5]  of  real;
```

A leitura e exibição de vetores são feitas indicando individualmente quais elementos se deseja ler ou imprimir. Vejamos como inserir as notas do aluno:

```
NOTA[1] := 5.2;
NOTA[2] := 8.0;
NOTA[3] := 9.2;
NOTA[4] := 7.5;
NOTA[5] := 8.3;
```

Observe que o nome é um só, o que muda é a informação indicada dentro dos colchetes. A esta informação dá-se o nome de *índice,* sendo este o endereço onde o valor está armazenado, ou seja, a nota do aluno. Podemos imaginar o vetor como uma tabela com cinco colunas:

NOTA

5.2	8.0	9.2	7.5	8.3

Vamos analisar um programa com a utilização de **vetores:**

"Desenvolver um programa que efetue a leitura de 10 elementos de um vetor A Construir um vetor B de mesmo tipo, observando a seguinte lei de formação: se o valor do índice for par, o valor deverá ser multiplicado por 5; sendo ímpar, deverá ser somado com 5. Ao final, mostrar os conteúdos dos dois vetores. "

Algoritmo

1. Iniciar o contador do índice de um vetor (A), com valor 1 até 10;
2. Ler os 10 valores, um a um;
3. Verificar se o índice é par:
   a. Se for, multiplica por 5 e atualiza no vetor B;
   b. Se não for par, soma 5 e atualiza no vetor B.
4. Apresentar o conteúdo dos dois vetores.

Programa em Pascal

```
program EXEMPLO_VETOR;
var
  A, B : array[1..10] of integer;
  I    : integer;
begin
   { *** Entrada dos Dados ***}
   for I := 1 to 10 do
   begin
      write('Informe o valor', I);
      readln(A[I]);
   end;

   { *** Teste Par ou Ímpar ***}
   for I := 1 to 10 do
     if (I mod 2 = 0) then
        B[I] := A[I] * 5
     else
        B[I] := A[I] + 5;
   writeln;

   { *** Apresentação das matrizes ***}
    for I := 1 to 10 do
       writeln('A[', I:2,']  = ', A[I]:2, '       ', 'B[', I:2, '] = ', B[I]:2 );
    writeln;
    writeln('Tecle <ENTER> para encerrar: ');
    readln;
end.
```

Vejamos um exemplo do uso de vetores de strings:

"Ler e exibir o nome de 10 pessoas."

Algoritmo

1. Definir a variável I do tipo inteira para controlar a repetição;
2. Definir a matriz NOME do tipo string para 10 elementos;
3. Iniciar o programa, fazendo a leitura dos 10 nomes; e
4. apresentar, após a leitura, os 10 nomes.

Programa em Pascal

```
program LISTA_NOME;
var
   NOME : array[1..10] of string;
   I:  integer;
begin
    writeln('Listagem de nomes');
    writeln;

    { ***  Entrada dos Dados  *** }

     for I := 1 to 10 do
     begin
        write('Digite o ', I:2, ' o. nome: ');
        readln(NOME[I]);
     end;
     writeln;

    { ***  Apresentação dos Nomes  *** }

     for I := 1 to 10 do
        writeln('Nome: ' , I:2, ' -->  ', NOME[I]);
      writeln;
      writeln('Tecle <ENTER> para encerrar: ');
       readln;
end.
```

Uma forma mais organizada de apresentar o programa acima, seria exibir os nomes das pessoas em ordem alfabética, contribuindo para uma pesquisa mais fácil. Vejamos como ficaria o programa desta forma:

Programa em Pascal

```
program LISTA_NOME_ORDENADO;
var
  NOME : array[1..10] of string;
  I, J:  integer;
  X : string;
begin
   writeln('Listagem de nomes');
   writeln;

   { ***  Entrada dos Dados  *** }

    for I := 1 to 10 do
    begin
       write('Digite o ', I:2, ' o. nome: ');
       readln(NOME[I]);
    end;

   { ***  Ordenação dos Nomes  *** }

    for I := 1 to 9 do
      for J := I +1 to 10 do
           if (NOME[I] > NOME[J])  then
          begin
             X := NOME[I];
             NOME[I] := NOME[J];
             NOME[J] := X;
          end;

         { ***  Apresentação dos Nomes  *** }

      writeln;
      for I := 1 to 10 do
         writeln('Nome: ' , I:2, ' -->  ', NOME[I]);
      writeln;
      writeln('Tecle <ENTER> para encerrar: ');
      readln;
end.
```

Observe o algoritmo da troca, utilizado junto da instrução **if** NOME[I] > NOME[J] **then**. Após a verificação desta condição, sendo o primeiro nome maior que o segundo, efetua-se então a sua troca com a seqüência:

```
X := NOME[I];
NOME[I] := NOME[J];
NOME[J] := X;
```

Considere o vetor NOME[I] com o valor “CARLOS” e o vetor NOME[J] com o valor “ALBERTO”. Ao final, NOME[I] deverá estar com “ALBERTO” e NOME[J] deverá estar com o valor “CARLOS”. Para se conseguir este efeito é necessária a utilização de uma variável de apoio, no caso **X.**

Para que o vetor NOME[I] fique livre para receber o valor do vetor NOME[J], X deverá receber o valor de NOME[I], assim sendo X passa a ter o valor “CARLOS”. Neste momento passa-se o valor de NOME[J] para NOME[I], no caso “ALBERTO”. Po fim, NOME[J] pode receber o valor de X para receber o valor “CARLOS”.

7.2 Matrizes

Vamos imaginar que queremos rescrever o programa das notas do aluno, agora mantendo o controle para cinco alunos ao invés de apenas um. Com os conhecimentos adquiridos até agora seria necessário criar cinco vetores (matrizes unidimensionais), um para cada aluno. Esta é uma solução, e funciona, porém, torna nosso trabalho mais cansativo.

Para facilitar o trabalho com estruturas deste porte, existem as chamadas matrizes (ou matrizes multidimensionais, se você pensar que vetores são matrizes unidimensionais).A mais comum é a matriz de duas dimensões.

Uma matriz de duas dimensões estará sempre fazendo menção a linhas e colunas e será representada por seu nome e seu tamanho.

Sintaxe:

```
  <matriz> : array[dimensão_linha,dimensão_coluna] of
<tipo_de_dado>;
```

onde: *matriz* é o nome atribuído à matriz, *dimensão_linha* é o tamanho da matriz em número de linhas, *dimensão_coluna* é o tamanho da matriz em número de colunas e *tipo_de_dado* é o tipo do elemento armazenado(inteiros, reais...).

Vejamos como ficaria a matriz das notas dos nossos 5 alunos:

Pedro	8.5	9.0	7.8	8.9
Ana	5.0	6.8	8.7	6.5
Joana	7.0	7.5	7.8	6.5
Joao	8.5	8.0	9.2	7.9
Mauro	5.5	8.0	7.2	7.0

E como ficaria sua declaração em Pascal:

NOTAS : **array**[1..5,1..4] **of real;**

Vamos ver o programa em Pascal para este exemplo para entenderemos melhor o conceito de matriz:

Programa em Pascal

```
program NOTA_ALUNO;
var
   NOTAS :  array[1..5,1..4] of real;
   I, J        :  integer;
begin
  writeln('Leitura e Apresentacao da Notas');
   writeln;
   for I := 1 to 5 do
   begin
      writeln;
      writeln('Informe as notas do ', I:2, 'o. aluno: ');
      for J:= 1 to 4 do
      begin
         write('Nota', J:2, ':');
         readln(NOTAS[I, J]);
      end;
    end;
    writeln;
    for I:= 1 to 5 do
    begin
        write('As notas do aluno ', I:2, ' são: ');
         for J := 1 to 4 do
             write(NOTAS[I, J] :2 :2, ' ');
         writeln;
    end;
    writeln;
    writeln('Tecle <ENTER> para encerrar: ');
    readln;
end.
```

Em um exemplo anterior, foi utilizada a variável **I** para controlar as posições dos elementos dentro do vetor, ou seja, a posição em nível de linha. Neste exemplo, a variável **I** continua tendo o mesmo efeito e a segunda variável, **J**, está controlando a posição da coluna.

Analisando o programa, temos a inicialização das variáveis **I** e **J** como 1, ou seja, a leitura será efetuada na primeira linha da primeira coluna. Primeiro é iniciado o looping das linhas e após, para cada linha é feito o looping para todas as colunas de uma linha. Quando todas as notas (colunas) de um aluno forem preenchidas, passa-se para o próximo aluno (linha), e novamente são preenchidas todas as notas(colunas) para o segundo aluno (linhas). Assim até o final.

Vamos ver mais um exemplo para fixarmos nosso conhecimento:

"Desenvolver um programa de agenda que cadastre o nome, endereço, CEP, bairro e telefone de 10 pessoas. Ao final, o programa deverá apresentar seus elementos dispostos em ordem alfabética, independentemente da ordem em que foram digitados"

Algoritmo

Para resolver este problema, você deverá possuir uma tabela com 10 linhas (pessoas) e 5 colunas (dados pessoais). Assim sendo, imagine esta tabela:

	Nome	Endereço	CEP	Bairro	Telefone
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

Nesta tabela, serão utilizados dois elementos numéricos, O CEP e o telefone, mas como não são executados cálculos com estes números, eles serão armazenados como caracteres.

Depois de cadastrar todos os elementos, será iniciado o processo de classificação alfabética, pelo nome de cada pessoa. Este método já foi anteriormente estudado, bastando aplicá-lo neste contexto. Porém, após a comparação do primeiro nome com o segundo, sendo o primeiro maior que o segundo, estes deverão ser trocados, bem como os elementos relacionados ao nome também. Ao final, as informações são apresentadas.

Programa em Pascal

```
program AGENDA;
var
   DADO   : array[1..10,1..5] of string;
   I, J       : integer;
   X          : string;
begin
   { *** Rotina de Entrada ***}
   writeln('Programa agenda');
   writeln;
   for I := 1 to 10 do
   begin
      write('Nome......: ');    readln(DADO[I,1]);
      write('Endereço.: ');     readln(DADO[I,2]);
      write('CEP.........: ');  readln(DADO[I,3]);
      write('Bairro.....: ');   readln(DADO[I,4]);
      write('Telefone..: ');    readln(DADO[I,5]);
```

```
        writeln;
    end;
    { *** Rotina de ordenação e troca de elementos *** }
    for I := 1 to 9 do
        for J := I + 1 to 10 do
            if (DADO[I,1] )> (DADO[J,1]) then
            begin
                {Troca Nome}
                X := DADO[I,1];
                DADO[I,1] := DADO[J,1];
                DADO[J,1] := X;

                {Troca Endereço}
                X := DADO[I,2];
                DADO[I,2] := DADO[J,2];
                DADO[J,2] := X;

                {Troca CEP}
                X := DADO[I,3];
                DADO[I,3] := DADO[J,3];
                DADO[J,3] := X;

                {Troca Bairro}
                X := DADO[I,4];
                DADO[I,4] := DADO[J,4];
                DADO[J,4] := X;

                {Troca Telefone}
                X := DADO[I,5];
                DADO[I,5] := DADO[J,5];
                DADO[J,5] := X;
            end;

    { *** Rotina de Saida ***}
    for I := 1 to 10 do
        for J := 1 to 5 do
            writeln(DADO[I, J]);
    writeln;
    writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

7.3 Registros

Quando trabalhamos com matrizes percebemos que somente foi possível agrupar informações com o mesmo tipo de dados. Caso fosse necessário trabalhar com mais de um tipo de dado, precisaríamos criar matrizes diferentes. Para solucionar esta deficiência podemos utilizar uma estrutura de dados chamada de **Registro**.

Em um registro poderemos utilizar uma estrutura que agrupe várias informações, que podem ser de tipos de dados diferentes. Por esta razão, este tipo de dado é considerado heterogêneo. Em Pascal, os tipos r**egistro** devem ser declarados ou atribuídos antes das definições das variáveis, pois é muito comum ocorrer a necessidade de se declarar uma variável com o tipo de registro atribuído. Um tipo **registro** é declarado em Pascal com a instrução **type** em conjunto com a instrução **record**.

Sintaxe:

```
type
  <identificador> = record
                      <lista de campos e seus tipos>
                    end;
var
  <variavel> : <identificador_registro>
```

onde: *identificador* é o nome do tipo registro e *lista de campos e seus tipos* é a relação de variáveis que serão usadas como campos, bem como seu tipo(real, integer...).

Após a instrução **var**, deverá ser indicada uma variável **tipo registro** e a declaração do seu tipo de acordo com um identificador definido anteriormente. Perceba que a instrução **type** deverá ser utilizada antes da instrução **var**, pois, ao definir um tipo de variável, pode-se fazer uso deste tipo definido.

Vamos novamente analisar o exemplo do aluno e suas 4 notas, considerando, agora, que devemos armazenar o nome deste aluno e suas 4 notas em uma mesma estrutura.

Programa em Pascal

```
program EX_REGISTRO1;
type
  CAD_ALUNO = record
                NOME : string;
                NOTA1 : real;
                NOTA2 : real;
                NOTA3 : real;
                NOTA4 : real;
              end;
var
  ALUNO : cad_aluno;
begin
  writeln('Cadastro de Aluno');
  writeln;
  write('Informe o nome.....: ');   readln(ALUNO.NOME);
```

```
   write('Informe a primeira nota..: ');    readln(ALUNO.NOTA1);
   write('Informe a segunda nota...: ');    readln(ALUNO.NOTA2);
   write('Informe a terceira nota....: ');  readln(ALUNO.NOTA3);
   write('Informe a quarta nota.....: ');   readln(ALUNO.NOTA4);
   writeln;
   writeln('Nome..: ' , ALUNO.NOME);
   writeln('Nota 1.: ' , ALUNO.NOTA1 :2:2);
   writeln('Nota 2.: ' , ALUNO.NOTA2 :2:2);
   writeln('Nota 3.: ' , ALUNO.NOTA3 :2:2);
   writeln('Nota 4.: ' , ALUNO.NOTA4 :2:2); writeln;
   writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Perceba que o registro está sendo denominado como CAD_ALUNO, o qual será um conjunto de dados heterogêneos (um campo tipo string e quatro do tipo real). Desta forma, é possível guardar em uma mesma estrutura, vários tipos diferentes de dados.

Para armazenar as notas do aluno foram necessárias 4 variáveis. E se utilizássemos um vetor para armazenar as notas, como já fizemos anteriormente? Vejamos como ficará:

Programa em Pascal

```
program EXEMPLO_REGISTRO2;
type
  NOTAS          =  array[1..4] of real;
  CAD_ALUNO  =  record
                     NOME : string;
                     NOTA  : notas;
                   end;
var
  ALUNO : cad_aluno;
  I : integer;
begin
   writeln('Cadastro de Aluno');
   writeln;
   write('Informe o nome.....: ');     readln(ALUNO.NOME);
   writeln;
   for I := 1 to 4 do
   begin
      write('Informe a ' , I:2, 'a. nota..: ');
      readln(ALUNO.NOTA[I]);
   end;
   writeln;
   writeln('Nome..: ' , ALUNO.NOME);
   for I := 1 to 4 do
      writeln('Nota ' ,I, '...: ', ALUNO.NOTA[I]:2:2);
   writeln;
   writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Até agora, os dois exemplos apresentados só servem para a leitura e exibição das notas de um aluno. E se quiséssemos apresentar as notas de cinco alunos, como fizemos utilizando matrizes? Vamos ver como ficaria um programa assim:

Programa em Pascal

```
program EXEMPLO_REGISTRO3;
type
  NOTAS          =  array[1..4] of real;
  CAD_ALUNO  =  record
                     NOME : string;
                     NOTA  : notas;
                   end;
var
  ALUNO : array[1..5] of cad_aluno;
  I , J: integer;
begin
   writeln('Cadastro de Aluno');
   writeln;
   for J := 1 to 5 do
   begin
      write('Informe o nome do ', J:2, 'o. aluno...: ');
      readln(ALUNO[J].NOME);
      writeln;
      for I := 1 to 4 do
      begin
         write('Informe a ' , I:2, 'a. nota.............: ');
         readln(ALUNO[J].NOTA[I]);
      end;
      writeln;
   end;
   writeln;
   writeln;
   for J := 1 to 5 do
   begin
      writeln('Nome aluno  ' , J:2, '...: ', ALUNO[J].NOME);
      writeln;
      for I := 1 to 4 do
         writeln('Nota ' ,I, '.....: ', ALUNO[J].NOTA[I]:2:2);
      writeln;
    end;
    writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

8 PROCEDURES E FUNCTIONS

Até agora temos desenvolvido programas que englobam a lógica completa do algoritmo para a solução de um determinado problema. É comum, em programação, decompor a lógica de programas complexos em programas menores e, depois, juntá-los para compor o programa final. Essa técnica de programação é denominada *programação modular*.

A programação modular consiste num método para facilitar a construção de grandes programas, através de sua divisão em pequenas etapas, que são os módulos ou rotinas e para possibilitar o reaproveitamento de código, já que podemos utilizar um módulo quantas vezes for necessário, eliminando assim a necessidade de escrever o mesmo código do programa em situações repetitivas. Outra importância da modularização é que ela permite que diferentes programadores trabalhem simultaneamente na solução de um mesmo problema, através da codificação separada dos diferentes módulos.

A modularização, em Pascal, pode ser feita através de *procedimentos (**procedures**) e funções (**functions**)*. Isso é feito associando-se um nome a uma seqüência de comandos através do que chamamos *Declaração do Procedimento ou da Função*. Pode-se, então, usar o nome do procedimento ou da função dentro do corpo do programa, sempre que desejarmos que o seu bloco de comandos seja executado, isso é o que chamamos de *Chamada do Procedimento ou da Função*.

Além das rotinas desenvolvidas pelo usuário, existe na linguagem Pascal, um conjunto de rotinas embutidas. Este tipo de rotina embutida é conhecido pelo nome de **unidade** (do inglês, *unit*).

8.1 Utilização de Units

Vejamos de forma básica as **units** embutidas no Turbo Pascal. As **units** são conjuntos de rotinas prontas para serem usadas pelo programador. Uma **unit** é, na verdade, uma biblioteca de funções e procedimentos. Vejamos a lista das unidades do Pascal:

- **CRT**: esta unidade é a mais utilizada na programação Pascal. Ela possui a maior parte das rotinas e variáveis de geração de som, controle de vídeo e teclado;
- **DOS:** esta unidade possui as rotinas que envolvem a utilização do sistema operacional, na maior parte das vezes permitindo controles de baixo nível;
- **GRAPH:** esta unidade possui rotinas destinadas à manipulações gráficas;
- **OVERLAY:** esta unidade possibilita gerenciar as atividades de um programa, desta forma, é possível aproveitar uma mesma área de memória para rodar várias rotinas diferentes, economizando memória;
- **PRINTER:** esta unidade permite declarar um arquivo tipo texto com o nome LST e associá-lo à impressora; e
- **SYSTEM:** esta unidade possui a maior parte das rotinas padrão da linguagem Pascal, não necessitando ser citada para ser usada, pois o turbo Pascal já a executa de forma automática.

Para se fazer uso deste recurso é necessário o uso da instrução **uses** situada antes da declaração da instrução **var.**

Sintaxe:

```
uses <unidade>;
```

À medida que formos estudando procedimentos e funções veremos a utilização das **units**.

8.2 Procedures

Um *procedimento* é uma estrutura de programa autônoma que está incluída num programa em Pascal. Nele podem ser colocados todos os elementos da linguagem Pascal, como se fosse um programa completo; isso é feito através de sua declaração.

Um procedimento pode ser referido escrevendo simplesmente o seu nome seguido de um lista opcional de parâmetros. Quando um procedimento é referenciado, o controle de execução do programa é automaticamente transferido para o início do procedimento. As instruções de execução dentro do procedimento são então executadas, tendo em conta quaisquer declarações especiais que sejam únicas para o procedimento. Quando todas as instruções de execução tiverem sido executadas, o controle passa automaticamente para a instrução imediatamente a seguir à da chamada do procedimento.

Sintaxe:

```
procedure Nome( parametros )
var
 <variáveis>
begin
 <instruções>
end;
```

onde: *nome* é o nome atribuído ao procedimento e *parametros* são informações adicionais que serão vistas mais tarde.

A linguagem Pascal exige que os procedimentos sejam definidos antes do programa principal Para entendermos melhor o conceito de procedimento vamos fazer um exemplo.

"Desenvolver um programa calculadora que apresente um menu de seleções no programa principal. Este menu deverá dar ao usuário a possibilidade de escolher uma entre duas operações aritméticas. Escolhida a opção desejada, deverá ser solicitada a entrada de dois números, e processada a operação, deverá ser exibido o resultado."

Algoritmo

Note que este programa deverá ser um conjunto de cinco rotinas sendo uma principal e 2 secundárias. A rotina principal efetuará o controle sobre as duas rotinas secundárias, que por sua vez efetuarão o pedido de leitura de dois valores, farão a operação e apresentarão o resultado obtido. Tendo-se uma idéia da estrutura geral do

programa, será escrito em separado cada algoritmo com os seus detalhes de operação. Primeiro o programa principal e depois as outras rotinas.

Programa principal

1. Apresentar um menu de seleção com cinco opções:
   a. Adição
   b. Multiplicação
   c. Fim de Programa
2. Ao ser selecionado um valor, procedimento correspondente deverá ser executado; e
3. Ao se escolher o valor 3 (c), o programa deverá ser encerrado.

Rotina 1 – Adição

1. Ler dois valores, no caso, variáveis A e B;
2. Efetuar a soma das variáveis A e B, armazenando o resultado na variável X;
3. Apresentar o valor da variável X; e
4. Voltar ao programa principal.

Rotina 1 – Multiplicação

1. Ler dois valores, no caso, variáveis A e B;
2. Efetuar a multiplicação das variáveis A e B, armazenando o resultado na variável X;
3. Apresentar o valor da variável X; e
4. Voltar ao programa principal.

Programa em Pascal

```
program CALCULADORA;
uses
   Crt;
var
   OPCAO : char;

{ **  Procedimento de Soma ** }
procedure ADICAO;
var
   X, A, B    : real;
   TECLA    : char;
begin
   clrscr;     gotoxy(32,1);
   write('Procedimento de Adicao');
   gotoxy(5,6);
   write('Informe um valor para A: ');
   readln(A);
   gotoxy(5,7);
   write('Informe um valor para B: ');
   readln(B);
   X := A + B;
   gotoxy(5,10);
   write('O resultado e: ', X:4:2);
```

```
   gotoxy(25,24);
   writeln('Tecle algo para voltar ao menu');
   TECLA := readkey;
end;

{ **  Procedimento de Multiplicacao ** }
procedure MULTIPLICA;
var
   X, A, B    : real;
   TECLA    : char;
begin
   clrscr;     gotoxy(32,1);
   write('Procedimento de Multiplicacao');
   gotoxy(5,6);
   write('Informe um valor para A: ');
   readln(A);
   gotoxy(5,7);
   write('Informe um valor para B: ');
   readln(B);
   X := A * B;
   gotoxy(5,10);
   write('O resultado e: ', X:4:2);
   gotoxy(25,24);
   writeln('Tecle algo para voltar ao menu');
   TECLA := readkey;
end;
{ ** Programa Principal **}
begin
   OPCAO := '0'
   while (OPCAO <> 3) do
   begin
      clrscr;
      gotoxy(33,1);
      write('Menu Principal');
      gotoxy(28,6);
      write('1. Soma.....');
      gotoxy(28,8);
      write('2. Multiplicacao.....');
      gotoxy(28,10);
      write('3. Fim do Programa.');
      gotoxy(28,12);
      write('Escolha uma opcao:');
      readln(OPCAO);
      if (OPCAO = '1') then
        Adicao;
      if (OPCAO = '2') then
        Multiplica;
   end;
end.
```

Como você já deve ter percebido, uma das unidades foi utilizada neste programa: a **CRT**, e com ela apareceram algumas novas instruções, tais como *gotoxy*, *clrscr* e *readkey*. Vamos ver a funcionalidade destas instruções:

Instrução	Tipo	Funcionalidade
clrscr	Procedure	Efetua a limpeza da tela, posicionando o cursor do lado esquerdo superior.
gotoxy (coluna,linha)	Procedure	Permite posicionar o cursor em ponto da tela. Para utilizá-lo é necessário informar dois parâmetros: a coluna e linha para posicionamento.
readkey	Function	Permite retornar o valor da tecla pressionada, sendo este valor do tipo caractere. Quando utilizada em um programa, e esta função faz a leitura de apenas um caractere. Por esta razão, não necessita ser usada a tecla <ENTER> para confirmar a entrada da informação

Podemos simplificar nosso programa com a utilização da instrução condicional, semelhante ao **if...then**, chamada **case...of**

Sintaxe:

```
case <variavel> of
  <opcao1> : <instrucao1>;
  <opcao2> : <instrucao2>;
  <opcao3> : <instrucao3>;
else
   <instrução4>;
end;
```

onde: *variavel* é o nome da variável a ser controlada na decisão, *opcao* é o conteúdo da variável a ser verificado e *instrucao* é a execução de um comando ou bloco de comandos.

Lembre-se que você pode inserir um bloco de instruções após uma instrução condicional. Para isto basta colocar as instruções entre **begin** e **end.**

Vejamos um trecho do programa da calculadora, com o uso da instrução **case...of**.

Programa em Pascal

```
{ ** Programa Principal **}
begin
   OPCAO := '0'
   while (OPCAO <> 3) do
   begin
      clrscr;
      gotoxy(33,1);
      write('Menu Principal');
      gotoxy(28,6);   write('1. Soma.....');
      gotoxy(28,8);   write('2. Multiplicacao.....');
      gotoxy(28,10);  write('3. Fim do Programa.');
      gotoxy(28,12);  write('Escolha uma opcao:');
      readln(OPCAO);
      if (OPCAO <> '3') then
        case OPCAO of
          '1' :  Adicao;
          '2' :  Multiplica;
        else
          gotoxy(27,24);
          writeln('Opcao invalida – Tecle algo: ');
          OPCAO := readkey;
        end;
   end;
end.
```

No programa da calculadora você deve ter observado que utilizamos algumas variáveis dentro dos procedimentos (A, B e X), e também uma variável fora das rotinas (OPCAO). Estas variáveis são chamadas de *locais* e *globais*, respectivamente.

8.2.1 Variáveis Globais e Locais

Uma variável é considerada *Global* quando é declarada no início de um programa escrito em Pascal, podendo ser utilizada por qualquer procedimento ou função. Assim sendo, este tipo de variável passa a ser visível a todas as funções ou procedimentos. No programa anterior, a variável OPCAO é uma variável global.

Uma variável é considerada *Local* quando é declarada dentro de uma função ou procedimento, sendo somente válida dentro da qual está declarada. Desta forma, as demais rotinas e o programa principal não poderão fazer uso daquelas variáveis como *Global*, pois não visualizam a existência das mesmas. No programa anterior, as variáveis A, B e X são locais. Por este motivo estão sendo declaradas repetidas vezes dentro de cada procedimento.

Dependendo da forma como se trabalha com as varáveis, é possível economizar espaço em memória, tornando o programa mais eficiente.

8.3 Parâmetros

Os parâmetros têm por finalidade servir como um ponto de comunicação entre uma rotina e o programa principal, ou com outra rotina de nível mais alto. Desta forma, é possível passar valores de uma rotina a outra rotina, através do uso de parâmetros que poderão ser *formais* ou *reais.*

Serão considerados parâmetros *formais* quando forem declarados através de variáveis juntamente com a identificação do nome da rotina, os quais serão tratados exatamente da mesma forma que são tratadas as variáveis globais ou locais.

Vejamos um exemplo:

```
procedure CALC_ADICAO(A, B : integer);
var
   Z : integer;
begin
   Z := A + B;
   writeln(Z);
end;
```

Observe que **Z** é uma variável local e está sendo usada para armazenar a soma das variáveis **A** e **B** que representam os parâmetros formais da rotina CALC_ADICAO

Serão considerados parâmetros *Reais* quando estes substituírem os parâmetros formais, quando da utilização da rotina por um programa principal ou por uma rotina chamadora.

Vejamos um exemplo do programa principal que chama o procedimento CALC_ADICAO

```
begin
   readln(X);
   readln(Y);
   Calc_Adicao(X,Y);
   readln(W);
   readln(T);
   Calc_Adicao(W,T);
   Calc_Adicao(6, 4);
end.
```

No trecho acima, toda vez que a rotina CALC_ADICAO é chamada, faz-se uso de parâmetros reais. Desta forma, são parâmetros reais as variáveis **X**, **Y**, **W** e **T**, pois são fornecidos seus valores para as rotinas.

Vejamos agora o programa completo:

Programa em Pascal

```
procedure CALC_ADICAO(A, B : integer);
var
  Z : integer;
begin
   Z := A + B;
   writeln(Z);
end;

var
   X, Y, W, T : integer;
begin
   readln(X);
   readln(Y);
   Calc_Adicao(X,Y);
   readln(W);
   readln(T);
   Calc_Adicao(W,T);
   Calc_Adicao(6, 4);
end.
```

A passagem de parâmetro ocorre quando é feita uma substituição dos parâmetros formais pelos reais no momento da execução da rotina. Estes parâmetros serão passados de duas formas: por valor e por referência.

8.3.1 Passagem por Valor

A passagem de parâmetro por valor caracteriza-se pela não alteração do valor do parâmetro real, quando este é manipulado dentro da rotina. Assim sendo, o valor passado pelo parâmetro real é copiado para o parâmetro formal, que no caso assume o papel de variável local da rotina. Desta forma, qualquer modificação que ocorra na variável local da rotina não afetará o valor do parâmetro real correspondente.

Vejamos um exemplo utilizando passagem de parâmetro por valor:

Programa em Pascal

```
program FATORIAL;
uses
   Crt;

procedure FATOR(N : integer);
var
   I, FAT  : integer;
begin
   FAT := 1;
   for 1 to N do
      FAT := FAT * I;
   writeln('A fatorial de ', N, ' equivale a: ', FAT);
end;

var
   LIMITE : integer;
begin
   clrscr;
   writeln('Programa Fatorial');
   writeln;
   write('Informe um valor inteiro: ');
   readln(LIMITE);
   Fatorial (LIMITE);
   writeln('Tecle <ENTER> para encerrar o programa');
   readln;
end.
```

8.3.2 Passagem por Referência

A passagem de parâmetro por referência caracteriza-se pela ocorrência da alteração do valor do parâmetro real quando o parâmetro formal é manipulado dentro da rotina chamada. Desta forma, qualquer modificação feita no parâmetro formal, implica em alteração no parâmetro real correspondente. A alteração efetuada é devolvida para a rotina chamadora.

Vejamos novamente o exemplo do programa FATORIAL agora com passagem de parâmetros por referência:

```
program FATORIAL2;
uses
   Crt;

procedure FATOR(N : integer;  var FAT : integer);
var
   I :  integer;
begin
   FAT := 1;
   for 1 to N do
      FAT := FAT * I;
end;

var
   LIMITE, RETORNO : integer;
begin
   clrscr;
   writeln('Programa Fatorial');
   writeln;
   write('Informe um valor inteiro: ');
   readln(LIMITE);
   Fatorial (LIMITE, RETORNO);
   writeln('A fatorial de ', LIMITE, ' equivale a: ', RETORNO);
   writeln('Tecle <ENTER> para encerrar o programa');
   readln;
end.
```

Neste exemplo, é indicado o uso da passagem de parâmetro por referência. A variável **N** continua sendo uma variável do tipo *passagem por valor*, pois esta receberá o valor de LIMITE. Este valor estabelece quantas vezes o *looping* deve ser executado. Dentro da rotina é encontrada a variável FAT que é do tipo *passagem de parâmetro por referência*. Ao término do *looping* o valor da variável FAT é transferido para fora da rotina, ou seja, seu valor é transferido para a variável RETORNO.

8.4 Functions

Uma **function**, assim com uma **procedure**, é um bloco de programa, ao qual são válidas todas as regras estudadas de programação. Sua diferença em relação a uma **procedure** está no fato de retornar sempre um valor. O valor de uma função é retornado no próprio nome da função. Quando se diz valor, devem ser levados em conta os valores numéricos, lógicos ou literais.

Sintaxe:

```
function <nome> [(parâmetros)] : <tipo>;
var
  <variaveis> ;
begin
  <instrucoes>;
end;
```

onde: *nome* é o nome atribuído a uma função, *parâmetro* é uma informação opcional e *tipo* é o tipo de dado a ser retornado pela função.

Vejamos o exemplo da **procedure** do Fatorial agora, na forma de função:

Programa em Pascal

```
function FATOR(N : integer) : integer;
var
  I, FAT  : integer;
begin
   FAT  :=  1;
   for  1 to N do
      FAT  :=  FAT  *  I;
    FATOR := FAT;
end;
```

Observe que o nome da função, no caso FATOR, é também o nome da variável interna que recebe o valor acumulado da variável FAT, caso o número fornecido para o parâmetro não seja zero. É muito simples, o resultado da função é atribuído ao próprio nome da função.

A definição de tipo de dado fora do parênteses, na declaração da função, indica o tipo de dado que retornará à rotina chamadora.

Vejamos um exemplo completo do uso de funções:

"Desenvolver um programa que faça uso de uma rotina de função que retorne o valor da adição de dois valores fornecidos como parâmetros."

Algoritmo

Programa Principal:
1. Ler dois valores, no caso as variáveis NUM1 e NUM2;
2. Chamar a função de soma, fornecendo como parâmetros as duas variáveis; e
3. Apresentar o resultado retornado.

Rotina para Adição:
1. Receber dois valores fornecidos através de parâmetros;
2. Efetuar o cálculo entre os dois valores, no caso a adição; e
3. Retornar para a função o resultado obtido.

Programa em Pascal

```
program CALC_ADICAO;
uses
   Crt;

function ADICAO (A, B : real)  :  real;
begin
   ADICAO  :=  A + B;
end;

var
   NUM1, NUM2  : real;
begin
    clrscr;
    write('Informe o 1º. valor : ');
    readln(NUM1);
    write('Informe o 2º. valor : ');
    readln(NUM2);
    writeln;
    writeln('O resultado da adicao = ', Adicao(NUM1, NUM2) :2:2 );
    writeln('Tecle <ENTER> para encerrar o program');
    readln;
end.
```

9 ARQUIVOS

Anteriormente foi estudado o conceito de **vetores** e **matrizes** para armazenar informações. Depois foi estudado o tipo **registro.** Porém, estas informações são armazenadas na memória RAM de nosso computador, permanecendo lá apenas até que nosso programa encerre a execução.

Para que possamos armazenar nossas informações fisicamente, de forma a recuperarmos tais informações quando quisermos surge o conceito de arquivos, que permite que gravemos nossas informações em dispositivos físicos, tais como o winchester e os disquetes.

A principal vantagem na utilização de um arquivo está no fato de as informações poderem ser utilizadas a qualquer momento. Outra vantagem encontrada na utilização de arquivos é o fato de poder armazenar um número maior de informações do que as de memória, estando apenas limitado ao tamanho do meio físico para sua gravação.

9.1 Definição de um Arquivo

A manipulação de um arquivo em Pascal ocorre com a definição do tipo **file**, o qual se caracteriza por ser uma estrutura formada por elementos do mesmo tipo dispostos de forma seqüencial, tendo como finalidade fazer a comunicação entre a memória principal e a memória secundária.

Sintaxe:

```
type
   <arquivo> = [text][file [of <tipo>]];
var
   <variavel> : <arquivo>;

        ou

<variavel> : [text][file [of <tipo>]];
```

onde: *arquivo* é o nome de um arquivo com tipo definido, *tipo* é o tipo de um arquivo (text, string, real, record, etc...) e *variavel* é a variável que será usada para representar o arquivo.

9.2 Operações de um Arquivo

Um arquivo tem a capacidade de executar algumas operações, tais como: abertura, leitura ou escrita e fechamento de um arquivo, sendo que estas operações, em Pascal, possuem algumas instruções apropriadas. Vejamos:

Instrução	Tipo	Sintaxe	Descrição
assign	procedure	**assign** (<variavel>, <arquivo>)	Tem por finalidade associar um nome lógico de arquivo ao arquivo físico. o parâmetro <variavel> é a indicação da variável do tipo arquivo, e <arquivo> é o nome do arquivo a ser manipulado
rewrite	procedure	**rewrite** (<variavel>)	Tem por finalidade criar um arquivo para uso, utilizando o nome associado ao parâmetro <variavel>. Caso o arquivo já exista, esta instrução o apagará para criá-lo novamente.
reset	procedure	**reset** (<variavel>)	Tem por finalidade abrir um arquivo existente, colocando-o disponível par leitura e gravação, utilizando o nome associado ao parâmetro <variavel>.
write	procedure	**write**(<variavel>,<dado>)	Tem por finalidade escrever a informação <dado> no arquivo indicado pelo parâmetro <variavel>.
read	procedure	**read**(<variavel>,<dado>)	Tem por finalidade ler a informação <dado> no arquivo indicado pelo parâmetro <variavel>.
close	procedure	**close**(<variavel>)	Tem por finalidade fechar um arquivo em uso dentro de um programa. nenhum programa deve ser encerrado sem antes fechar os arquivos abertos.

9.3 Formas de Acesso em um Arquivo

Os arquivos criados poderão ser acessados para leitura e gravação na forma seqüencial, direta ou indexada.

9.3.1 Acesso Seqüencial

O acesso seqüencial ocorre quando o processo de gravação e leitura é feito de forma contínua, um após o outro. Desta forma, para se gravar a informação é necessário percorrer todo o arquivo a partir do primeiro registro, até localizar a primeira posição vazia. A leitura também ocorre desta maneira.

9.3.2 Acesso Direto

O acesso direto ocorre através de um campo *chave* previamente definido. Desta forma, passa a existir um vínculo existente entre um dos campos do registro e sua posição de armazenamento, através da *chave.* Assim sendo, o acesso a um registro tanto para leitura como para gravação poderá ser feito de forma instantânea.

9.3.3 Acesso Indexado

O acesso indexado ocorre quando se acessa de forma direta um arquivo seqüencial. Na sua maioria, todo arquivo criado armazena os registros de forma seqüencial. A forma seqüencial de acesso se torna inconveniente, pois à medida que o arquivo aumenta de tamanho, aumenta também o tempo de acesso ao mesmo.

Por isso, cria-se um índice de consulta, passando a ser esse arquivo indexado. Existirão dois arquivos, um seqüencial e outro indexado. Os acessos serão feitos com em um livro: primeiro se consulta o índice de localização, e depois procura-se o conteúdo desejado.

9.4 Arquivos do Tipo Texto

Os arquivos do tipo texto estão capacitados a armazenar palavras, frases e também dados numéricos. Os números, entretanto, serão armazenados com um caractere do tipo alfanumérico. Ao serem lidos de um arquivo e passados para a memória, os dados numéricos são convertidos para seu formato original.

Vamos criar um arquivo do tipo texto:

```
program CRIA_ARQUIVO_TEXTO;
var
   ARQUIVO_TXT  : text;
begin
   assign(ARQUIVO_TXT, 'ARQTXT.XXX');
   rewrite(ARQUIVO_TXT);
   close(ARQUIVO_TXT);
end.
```

O programa acima estabelece para a variável ARQUIVO_TXT o identificador **text**, que tem por finalidade definir para a variável indicada o tipo de arquivo texto.

Tendo sido o arquivo criado, este poderá ser agora utilizado para a gravação das informações que irá guardar. Vejamos:

```
program GRAVA_ARQUIVO_TEXTO;
var
   ARQUIVO_TXT   : text;
   MENSAGEM      : string[50];
begin
   assign(ARQUIVO_TXT, 'ARQTXT.XXX');
   append(ARQUIVO_TXT);
   readln(MENSAGEM);
   writeln(ARQUIVO_TXT, MENSAGEM);
   close(ARQUIVO_TXT);
end.
```

Observação: quando a instrução **write** ou **writeln** é utilizada para escrever um dado em um arquivo, esta não apresenta o dado no vídeo, ficando disponível apenas para a operação de gravação do arquivo.

A seguir, é necessário podermos ler nosso arquivo texto. O programa abaixo mostra como fazemos isto:

```
program LER_ARQUIVO_TEXTO;
var
   ARQUIVO_TXT   : text;
   MENSAGEM      : string[50];
begin
   assign(ARQUIVO_TXT, 'ARQTXT.XXX');
   reset(ARQUIVO_TXT);
   readln(ARQUIVO_TXT , MENSAGEM);
   writeln( MENSAGEM);
   close(ARQUIVO_TXT);
end.
```

9.5 Arquivos com Tipo Definido

Os arquivos com tipo definido caracterizam-se por serem do tipo binário, diferente dos arquivos do tipo texto. Este arquivo permite armazenar específicos, podendo ser: **integer, real, record,** entre outros. Um arquivo com tipo definido executa as operações de gravação e leitura mais rápido que os arquivos texto. Os arquivos de tipo definido estão capacitados a armazenar dados na forma de registro.

Vamos criar um arquivo que seja capacitado a receber dados numéricos inteiros:

```
program CRIA_ARQUIVO_INTEIRO;
var
   ARQUIVO_INT   : file of integer;
begin
   assign(ARQUIVO_INT, 'ARQINT.XXX');
   rewrite(ARQUIVO_INT);
   close(ARQUIVO_INT);
end.
```

O programa acima estabelece para a variável ARQUIVO_INT o identificador **file of integer**, que tem a finalidade de definir para a variável indicada de arquivo como sendo inteiro. Todas as operações são semelhantes às operações usadas para se criar um arquivo texto

Estando o arquivo criado, este agora poderá ser utilizado para a gravação de dados de tipo inteiro. Vejamos

```
program CADASTRA_ARQUIVO_INTEIRO;
uses
   Crt;
var
   ARQUIVO_INT    :  file of integer;
   NUMERI         : integer;
   RESP           :char;
begin
   assign(ARQUIVO_INT, 'ARQINT.XXX');
   reset(ARQUIVO_INT);
   RESP := 'S';
   while (RESP = 'S') or (RESP = 's') do
   begin
       clrscr;
       writeln('Gravacao de Registros Inteiros');
       writeln;
       seek(ARQUIVO_INT, filesize(ARQUIVO_INT));
       write('Informe um numero inteiro: ');
       readln(NUMERO);
       write(ARQUIVO_INT, NUMERO);
       writeln;
       write('Deseja continuar ? [S/N] ');
      readln(RESP);
   end;
   close(ARQUIVO_INT);
end.
```

O programa acima utiliza a função **assign** para associar o arquivo à variável de controle. Faz uso da instrução **reset** para abrir o arquivo. Um pouco abaixo é apresentada uma linha com a instrução **seek**(ARQUIVO_INT, **filesize**(ARQUIVO_INT)); que executa em um arquivo definido o mesmo efeito que a instrução **append** em um arquivo texto, ou seja, se o arquivo já possuir algum registro, o próximo registro será sempre colocado após o último cadastrado. A instrução **filesize** retorna o número de registros do arquivo e a instrução **seek** posiciona o ponteiro de registro em uma determinada posição do arquivo.

Vamos agora ver um programa, que irá efetuar a leitura em um arquivo do tipo definido:

"Ler um arquivo de números inteiros, mostrando no vídeo a informação armazenada com o programa anterior."

```
program CADASTRA_ARQUIVO_INTEIRO;
uses
   Crt;
var
   ARQUIVO_INT    :  file of integer;
   NUMERI         : integer;
   RESP           :char;
begin
   assign(ARQUIVO_INT, 'ARQINT.XXX');
   reset(ARQUIVO_INT);
   RESP := 'S';
   while not eof(ARQUIVO_INT) do
   begin
      read(ARQUIVO_INT, NUMERO);
      writeln(NUMERO);
   end;
close(ARQUIVO_INT);
end.
```

9.6 Arquivo com Tipo Definido de Registro

Em Pascal, existe um arquivo do tipo **record**, que nos permite desenvolver uma aplicação de forma mais profissional e com acessos mais rápidos.

Sintaxe:

```
type
  <nome_arquivo> = record
                     <campos do arquivo>
                   end;
var
  <variavel> : <nome_arquivo>;
  <variavel> : file of <nome_arquivo>;
```

Vejamos um exemplo:

```
type
   BIMESTRE      =   array[1..4] of real;
   REG_ALUNO   =   record
                      FLAG          :  char;
                      MATRICULA  : integer;
                      NOME          :  string[30];
                      NOTAS         :  bimestre;
                   end;
var
   ALUNO    : reg_aluno;
   ARQALU  : file of reg_aluno;
```