

II. LÓGICA DE PROGRAMAÇÃO COM PASCAL



Leitura:

“*Engenharia de Software*” - Roger S. Pressman

Cap 16 - “Linguagens de Programação e Codificação” (exceto 16.2)

“*Introdução à Programação com Pascal*” - Sérgio E. R. Carvalho

“*Informatica – Novas Aplicacoes com Microcomputadores*” - Antonio Meirelles
(pag. 72 a 79)

4. As Linguagens de Programação

A **linguagem de programação** é o meio pelo qual podemos indicar os “passos” que devem ser realizados pelo computador para resolver problemas. Utilizando as linguagens de programação, colocamos algoritmos numa forma que o computador possa interpretá-los, ou seja, na forma de programas computacionais.

Para que o computador execute o algoritmo proposto, as operações devem ser transcritas para uma linguagem que a máquina consiga compreender. Na realidade, os computadores só podem executar algoritmos expressos em **linguagem de máquina** que constitui-se de um conjunto de instruções capazes de ativar diretamente os dispositivos eletrônicos do computador.

Características da Linguagem de Máquina:

- diferente para cada tipo de computador, dependendo de sua arquitetura;
- extremamente rudimentar, onde até as operações mais simples têm que ser expressas em termos de *registros*, *acumuladores* e outros dispositivos de máquina;
- totalmente expressa em forma numérica - sistema de numeração binário (0s e 1s) ou hexadecimal.

Consequentemente, é uma linguagem de difícil aprendizado e pouco expressiva para as pessoas.

Para tornar a atividade de programação mais acessível, foram desenvolvidas outras linguagens, denominadas de “Linguagens de Programação”, que funcionam como uma forma alternativa de se comunicar com o computador.

Como Funcionam as Linguagens de Programação?

As linguagens de programação são compostas por um grupo de elementos e regras que permitem a construção das instruções utilizadas para resolver os problemas computacionais. Com elas construímos programas que devem ser, posteriormente, transformados em instruções em Linguagem de Máquina. Para realizar a transformação, cada linguagem de programação possui um programa-suporte denominado, genericamente, de TRADUTOR.

4.1 Tipos de Linguagens de Programação

As linguagens de programação podem ser divididas em dois grupos básicos:

- ✓ Linguagens de Programação de Baixo Nível
- ✓ Linguagens de Programação de Alto Nível

4.1.1 Linguagem de Programação de Baixo Nível

Conhecida como Linguagem Assembler ou Linguagem de Montagem, ou ainda, Linguagem Simbólica.

Utiliza números binários, hexadecimais, alguns símbolos e letras para compor os programas. Está muito próxima da Linguagem de Máquina, onde cada instrução simbólica corresponde, praticamente, a uma instrução de máquina.

Para transformar o programa escrito em Linguagem Assembler em código de máquina executável, é utilizado um programa-suporte denominado de MONTADOR.

4.1.2 Linguagens de Programação de Alto Nível

São linguagens de programação que utilizam notações matemáticas e grupos de palavras para representar as instruções de máquina, tornando o processo de programação mais próximo do entendimento humano.

Muitas destas linguagens foram desenvolvidas para atender os problemas de áreas de aplicação específicas, como, por exemplo, linguagens para aplicações comerciais, científicas, administrativas, de ensino, etc.

A primeira linguagem de alto nível foi desenvolvida em 1957 - denominada de FORTRAN (Formula Translator) - e destina-se a aplicações científicas e de engenharia.

De acordo com seu período de surgimento e características particulares adotadas em sua composição, as ling. de alto nível são divididas em GERAÇÕES (Linguagens de 2ª Geração, 3ª Geração e 4ª Geração).

Vantagens das linguagens de Alto Nível:

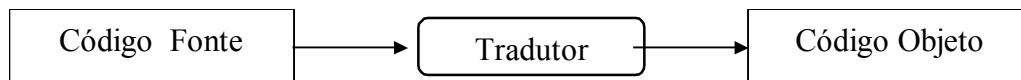
- facilidade de entendimento e uso;
- independência de máquina (é praticamente a mesma, não importando o computador utilizado).

Para transformar os programas escritos com Linguagens de Alto Nível em códigos de máquina, é usado um programa-suporte denominado TRADUTOR (Compilador ou Interpretador).

4.2 Compilação e Execução de Programas

Para executarmos um programa escrito numa linguagem de alto nível é preciso primeiro traduzir o *código-fonte* para *código-objeto*. O processo de tradução pode dar-se em tempo de execução caso a linguagem use um interpretador (traduz e executa instrução a instrução), ou

todas as instruções podem ser traduzidas antes que se inicie a execução do programa, o que ocorre no caso de linguagens que usam tradutores do tipo compilador.



Compilador: No sentido mais geral, qualquer programa que transforme um conjunto de símbolos em outro obedecendo a uma série de regras sintáticas e semânticas; no sentido mais comum, um programa que traduz todo o código-fonte de programas escritos numa linguagem de alto nível em código-objeto antes da execução do programa. O código-objeto é o código de máquina, ou alguma variação do código de máquina.

Código-fonte: não é executável diretamente pelo processador - permite apenas que o programador consiga definir o programa em uma forma legível aos humanos.

Código-objeto: é o código produzido pelo compilador; é uma forma intermediária, similar a linguagem de máquina do computador. Apesar de estar representado em binário, não é executável diretamente pelo processador, pois normalmente, o código-objeto referencia partes de programa que não estão necessariamente definidas no mesmo arquivo que o gerou, por exemplo, arquivos de bibliotecas de sub-rotinas.

Editores de ligação (ou linkeditores): Um programa que reúne módulos compilados e arquivos de dados para criar um programa executável. Os linkeditores têm também outras funções, como a criação de bibliotecas.

Exercícios de Fixação

F1. Faça um quadro comparativo para:


- b) Linguagem de Alto Nível e Linguagem de Baixo Nível
- c) Programa Fonte e Programa Objeto

F2. Faça um diagrama que descreva o processo de compilação e execução de programas, indicando que "ferramentas" (softwares) são utilizados em cada etapa e suas respectivas funções.

 Pesquise sobre Linguagens de Programação de Alto Nível e responda:

- a) A que geração de linguagens PASCAL pertence?
- b) Quais as características marcantes das linguagens de 3ª geração?
- c) Cite exemplos de linguagens atuais para aplicações comerciais e de ensino.

5. A Linguagem de Programação Pascal

 Leitura: “Programação em Pascal” - Byron Gottfried Cap. 2

5.1 Estrutura Geral dos Programas em Pascal

|----Significado ----- estrutura do programa -----|

Cabeçalho	PROGRAM	<identificador>	;
Bloco			
Declarações: Constantes	CONST	<lista de constantes>;	
Tipos	TYPE	<tipos definidos pelo usuário>;	
Variáveis	VAR	<lista de variáveis>;	
Procedimentos	PROCEDURE	{ <i>descrição do procedimento</i> }	
Funções	FUNCTION	{ <i>descrição da função</i> }	
Instruções	BEGIN		
		{ <i>Bloco principal de comandos</i> }	
	END.		

* PROGRAM, CONST, etc. são palavras reservadas da linguagem Pascal; isto é, são termos que tem um significado pré-definido e, portanto, só devem ser usados da forma especificada na linguagem (por exemplo, você não poderia criar uma variável com o nome “var”).

5.2 Estrutura de Dados em Pascal


A linguagem Pascal tem a capacidade de tratar muitos tipos de dados, permitindo inclusive que o programador defina novos tipos a partir de tipos básicos existentes (na cláusula TYPE), aumentando assim o número de aplicações que podem ser implementadas com a linguagem.

☞ Os Tipos de Dados podem ser divididos em:

1. Tipos Simples
2. Tipos Definidos pelo usuário
 - 2.1 Simples
 - 2.2 Estruturados
3. Tipos Estruturados
4. Tipo Apontador

5.2.1 Tipos de Dados Simples

São elementos individuais associados a um identificador simples. Representam locais de memória individuais ocupados por valores simples (números, cadeia de caracter, etc.)

 **Inteiro** (*integer*)

- * A operação de divisão (/) efetuada com valores do tipo inteiro resultam em um valor do tipo REAL.
- * Não existe operador exponencial !

☒ Real (*real*)

- * Os operadores DIV e MOD não podem ser usados com dados do tipo real.

☒ Caracter (*char*)

São cadeias de caracteres individuais, ou seja, caracteres individuais escritos entre apóstrofos (' ').

- * ' ' representa o caracter *branco*.
- * Quando queremos representar o caracter apóstrofo, devemos escrevê-lo duas vezes (''').
- * Correspondem os caracteres da Tabela ASCII. Assim cada caracter corresponde a um código numérico, podendo ser comparados uns com os outros, baseados em sua ordem relativa dentro do conjunto de caracteres.

☒ Booleano (*boolean*)

A este tipo são atribuídos os valores TRUE ou FALSE, que representam um conjunto ordenado onde o valor falso precede o verdadeiro.

- * Nas operações lógicas compostas devem ser utilizados parênteses (obrigatoriamente) para determinar a precedência de realização das operações.

5.2.2 Tipos Definidos pelo Usuário - Tipos Simples

Podem ser criados a partir de um grupo de valores específicos (Subrange) ou Enumerados. Os tipos estruturados são definidos a partir de tipos primitivos ou tipos definidos anteriormente (tipos de tipos).

- * Uso de cláusula TYPE para declaração de tipos definidos pelo usuário.
- * Deve preceder a declaração de variáveis (*ver esquema geral de programas pascal-item 5.1*)

☒ Subrange (grupo de valores)

É um subconjunto contínuo e ordenado, formado a partir de um tipo original de dados simples e ordenado. Isto inclui os tipos inteiros, booleanos, char e dados enumerados previamente definidos.

- Sintaxe Geral:

TYPE nome_tipo = primeiro_elemento ...último_elemento;
--

Ex1: Type maiuscula = 'A'..'Z';
 Var letra_m: maiuscula;

Ex2: Type Dia = 1..31;
 Var
 dia_data: dia;
 mes_data: 1..12;
 ano_data: 90..95;

☒ Enumerados

Tipos enumerados definem conjuntos ordenados de valores, através da enumeração de identificadores que representam estes valores. Sua ordem segue a sequência na qual os identificadores são enumerados.

- Sintaxe Geral:

```
TYPE nome = (identif, identif, ..., identif);
```

- * Os identificadores na definição do tipo tornam-se constantes do tipo enumerado.
- * A primeira constante é associada à posição 0, a segunda à posição 1 e assim por diante.
- * Os tipos enumerados são uma subclasse dos tipos ordinais.

Ex1: type Naipes = (Ouros, Espada, Paus, Copas);

- * Dada esta declaração, Copas, por exemplo, é uma constante do tipo Naipes.

* A função do Pascal ORD retorna a posição (ou ordinalidade) de uma constante do tipo enumerado.

```
Ord(Ouros)   = 0
Ord(Espada)  = 1
Ord(Paus)    = 2
```

Mais adiante veremos os outros tipos de dados (Tipos Estruturados, Tipo Apontador, etc.)

5.3 Estruturas de Controle

☐ Seleção: IF ... THEN ... ELSE

```
IF <expressão booleana> THEN
    <bloco de instruções>;
```

<bloco de instruções> pode ser somente uma instrução ou um conjunto de instruções delimitadas pelas palavras reservadas BEGIN e END que definem um **bloco**.

```
IF <expressão booleana> THEN
    <bloco de instruções>
ELSE
    <bloco de instruções>;
```

* Não deve haver um terminador (;) antes do ELSE. Caso haja mais de uma instrução antes do ELSE, estas devem ser delimitadas por BEGIN e END (sem o ; após o *end*).

☐ Seleção: CASE

```
CASE
```

```

CASE <expressão> OF
  Label1:    <bloco de instruções1> ;
  Label2:    <bloco de instruções2> ;
  ...
  ELSE
    <bloco de instruções> ;
END;
    
```

* <expressão> : se for uma variável, ela não pode ser do tipo real e se for uma expressão, não pode gerar resultado do tipo real

* Label: deve ser um valor relacionado com o tipo da variável ou resultado da expressão (números inteiros, caracteres, valores booleanos)

* na estrutura CASE, não se coloca o *begin*, mas é preciso colocar o **end**;

□ Repetição: condicional

```

WHILE <expressão booleana> DO
  <bloco de instruções> ;
    
```

```

REPEAT
  <instruções> ;
UNTIL <expressão booleana> ;
    
```

* Não é preciso delimitar as instruções da Estrutura REPEAT com as palavras reservadas BEGIN e END.

□ Repetição: incondicional (número pré-definido de vezes)

```

FOR var_controle := Vi TO / DOWNTO Vf DO
  <bloco de instruções> ;
    
```

* Usando TO : Enquanto $V_i \leq V_f$ executa o laço de repetição

* Usando DOWNTO: Enquanto $V_i \geq V_f$ executa o laço de repetição

* <var_controle>: deve ser do **tipo inteiro e não pode ser alterada dentro do laço.**

Ex:

```

Program ListaNumeros;
VAR lin, col, n : integer;
Begin
  Readln(n);
  For lin := 1 to n do
    begin
      For col:= lin to (n + lin -1) do
        write(col, ' ');
      writeln;
    end;
End.
    
```

Por exemplo, se $N = 4$
O programa vai gerar a seguinte saída de dados:

```

1 2 3 4
2 3 4 5
3 4 5 6
4 5 6 7
    
```

☑ Exemplo: Exercício no. 14 pag. 63 - Cap 03 Forbellone (2a. edição)

```

Program Pesquisa;
Uses crt;           { indicação do uso de biblioteca de funções do Turbo Pascal que
                    manipula o hardware : video, teclado, etc. }

Var
    sexo,olhos,cabelos :char;
    idade, maioridade, conta, total: integer;
    perc: real;

Begin

    total := 0;      { inicialização de variáveis }
    conta := 0;
    maioridade := 0;
    clrscr;         { limpa a tela }
    gotoxy(5,3);    { Posiciona cursor na coluna 5, linha 3 da tela}
    write('Pesquisa de Características Físicas na População ');
    gotoxy(10,5);
    write('digite a idade:');
    readln(idade);
    While idade <> -1 do { laço de repetição - Finaliza quando idade = -1}
        begin
            gotoxy(10,6);
            write('digite o sexo (F/M):');
            readln(sexo);
            gotoxy(10,7);
            write('digite a cor dos olhos (A:azuis/V:verdes/C:castanhos) :');
            readln(olhos);
            olhos := upcase(olhos);
            gotoxy(10,8);
            write('digite a cor dos cabelos (L:louro/C:castanhos/P:pretos) :');
            readln(cabelos);
            cabelos := upcase(cabelos);
            If (upcase(sexo)='F') and (olhos='V') and (cabelos='L') and
                (idade >= 18) and (idade <= 35) then
                conta:= conta + 1;
            If idade > maioridade then
                maioridade := idade;
            total := total + 1;
            gotoxy(10,5);
            write('digite a idade:');
            readln(idade);
        end;
    perc := conta * 100 / total;
    clrscr;
    gotoxy(1,7);    { Apresentação de Resultados }
    writeln('A maior idade encontrada na população foi de ', maioridade,' anos');
    writeln('Perc. mulheres de 18 a 35 anos c/ cabelos louros e olhos verdes=',perc:5:2,'%');
End.

```

Y Exercícios Propostos

☞ Exemplo de programa sem entrada de dados

1. Faça um programa Pascal que calcule e escreva o número de grãos de milho que se pode colocar num tabuleiro de xadrez, colocando um milho no primeiro quadro e, nos quadros seguintes, o dobro do quadro anterior.

☞ Exemplo de programa com repetições (número pré-definido de vezes)

2. Um número primo é aquele que não é divisível por nenhum número menor do que ele exceto a unidade. Deseja-se ler N números e mostrar todos os divisores de cada um deles. Para os primos, imprimir um asterisco(*) do lado.

Ex.: 10 → 1 5 10
 15 → 1 3 5 15
 11 → 1 *

☞ Exemplo de programa com repetições (número indefinido de vezes) e uso de variáveis de apoio a estatísticas

3. Uma certa firma fez uma pesquisa de mercado para saber se as pessoas gostaram ou não de um novo produto lançado. Para isto, forneceu o sexo do entrevistado e sua resposta (sim ou não). Sabendo-se que foram entrevistadas 200 pessoas, fazer um programa que calcule e mostre:
 - a) o número de pessoas que disseram SIM
 - b) o número de pessoas que disseram NÃO
 - c) a percentagem de pessoas do sexo feminino que responderam SIM
 - d) a percentagem de pessoas do sexo masculino que responderam NÃO

☞ Exemplo de programa aplicado a matemática

4. Fazer um programa que calcule o volume e a área de uma esfera usando as fórmulas abaixo:

$$V = 4\pi r^3 / 3$$

$$A = 4\pi r^2$$

☞ Um desafio de Lógica !!!


5. São fornecidos o *início* e o *fim* de uma atividade em termos de DIA, MÊS e ANO. Deseja-se imprimir o número de meses e dias decorridos desde o início até o final da atividade.

✍ Exercícios Complementares

Forbellone (2ª Edição) - pag. 62 a 65

Exercícios Propostos → 1, 9, 13, 16

5.4 Tipos de Dados Estruturados

 Leitura: “Lógica de Programação” - Forbellone Cap. 04

Os tipos estruturados são compostos por múltiplos elementos relacionados entre si. Cada grupo de elementos está relacionado a um identificador. Os elementos do grupo *podem* estar também relacionados a identificadores individuais.

Representam vários locais de memória que guardam vários valores, que podem ser acessados em conjunto ou individualmente. Podem ser:

-  **String**
-  **Vetores**
-  **Registros**
-  **Arquivos**
-  **Conjuntos**

String

São cadeias (ou sequência) de caracteres (letras, dígitos e caracteres especiais) entre apóstrofes (' ').

- * Nas cadeias podem ser usadas letras maiúsculas e minúsculas.
- * O número máximo de caracteres que pode ser incluído numa cadeia varia de acordo com a versão do Pascal, sendo que a maioria permite um comprimento máximo de 255 caracteres.
- * Especificação do tipo String:
`string [n]`, onde *n* é a quantidade de caracteres da cadeia.
- * Cada elemento da cadeia pode ser manipulado separadamente.

Ex:

```

Var cadeia: string[15];
Begin
  read(cadeia); { manipulação de toda a cadeia }
  write('esta foi a informação digitada:', cadeia);
  write('primeiro elemento (ou letra) da cadeia :', cadeia[1] );

```

Exercício Proposto:

Pesquise as Funções do Pascal para Manipulação de Strings. Explique seu objetivo, tipos de parâmetros e resultados e dê um exemplo.

Ex. de funções: Length (), Copy (string, índice, contador)

5.4.1 Vetores (ARRAY)

Definem agregados de dados homogêneos (todos os elementos são do mesmo tipo). Cada elemento ocupa uma posição definida no agregado e pode ser referenciado através dela.

- **Declaração de Vetores**

ARRAY [tipo_índice] OF tipo_elemento ;

tipo_índice é um tipo simples ordenado (inteiro, caracter, booleano, enumerado)
 É formado por: [li..ls] , onde *li*: limite inferior e *ls*: limite superior

Este tipo pode ser utilizado tanto na declaração de variáveis como também na definição de novos tipos (sessão TYPE).

Ex1: Var lista: array [1..100] of real;
 { o identificador do vetor é *lista* e ele poderá conter 100 elementos do tipo real }

Ex2: Type índice = 1..100; { tipo definido pelo usuário - tipo *subrange* }
 Var lista: array [índice] of real;

Ex3: Type max = 300;
 vetor = array[1..max] of string[20];

 Var endereço: vetor; { a variável *endereço* está associada ao tipo *vetor* }

Ex4: Var dados: array['A'..'Z'] of integer;

- **Definição de Vetores como Constantes (Const)**

Sintaxe geral: *identificador_constante* : *tipo_array* = (*lista de valores*)

Ex1: Const
 vetconst : array [1..3] of integer = (0, 1, 2);

Ex2: Type
 vetsemana: array [1..7] of string [3];
 Const
 dias_semana: vetsemana = ('DOM', 'SEG', 'TER', 'QUA', 'QUI',
 'SEX', 'SAB');

↳ Obs: Os valores das constantes array de tipo *Char* podem ser especificadas ou como valores caracteres simples ou como um string .

Ex:
 Const dígito: array [0..9] of char = ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9');
 ou
 Const dígito: array [0..9] of char = '0123456789';

- **Manipulação de vetores nos programas Pascal**

identificador_array[posição_elemento]

posição_elemento (ou índice) pode ser expresso com uma constante, variável ou expressão, devendo ser do tipo correto (correspondente à definição do array) e estar dentro do intervalo pré-definido.

Ex: lista[5]

lista[i], onde i é uma variável inteira com valores variando de 1 a 100.

lista[i+j], onde i e j são variáveis inteiras cuja soma não ultrapassa 100.

Os elementos do ARRAY podem ser usados em expressões, instruções de atribuição, instruções read e write, etc.

- **Programa Exemplo:**

```

Program ExVetor;

Type
  VetorNotas: array [1..40] of real;
Var
  Nota: vetorNotas;
  i: integer;

Begin
  i := 1; media := 0;
  Repeat
    write ('Digite a Nota :');
    read(Nota[i]);
    media:= media + Nota[i];
    i:= succ(i); { função que incrementa a variável ⇒ i ← i + 1 }
  until i > 40;
  writeln ('Notas dos Alunos ');
  For i:= 1 to 40 do
    writeln (Nota[i]);
  writeln ('Media da Turma:', media/40:5:2);
End.
    
```

Exercícios de Fixação

1. Elabore um programa que leia dois vetores inteiros de 20 elementos cada, depois some seus elementos, gerando um terceiro vetor. Ao final, mostre o novo vetor gerado.
2. Considere um vetor VET com 30 elementos. Verificar se existe um elemento igual a K no vetor. Se existir mostrar a posição em que se encontra, senão imprimir "*não encontrei K no vetor*".
3. Elabore um programa que leia um conjunto A com 50 números reais e construa um conjunto B, onde os elementos de ordem (posição) par são os elementos correspondentes de A divididos por 2 e os de ordem (posição) ímpar correspondem aos elementos de A multiplicados por 3. Ao final, mostre os dois conjuntos de números.
4. Fazer um programa Pascal que, ao ser fornecida uma data no formato DD/MM/AA, mostre-a por extenso.
Ex: Entrada Data: 12/ 06 / 95
 Saída 12 de junho de 1995
5. Defina um vetor de 100 elementos inteiros e elabore um programa que preencha VETOR[i] com 1 , se i é um quadrado perfeito e, com 0 , nos demais casos.
6. Elabore um programa que utilize dois vetores V1 e V2, formados de números reais com 20 posições cada um, e efetue neles as operações indicadas no vetor OP, cujos elementos são caracteres que indicam as quatro operações aritméticas básicas (+, -, *, /) . O resultado obtido das operações devem ser colocados num vetor resultante VR e mostrado ao final do programa.

5.4.2 Vetores Multidimensionais (Matrizes)

- Sintaxe Geral:

nome_array [dim₁, dim₂, dim₃...] of tipo_elemento

Onde as dimensões são definidas como: $dim_1 = li_1..ls_1$, $dim_2 = li_2..ls_2$, $dim_3 = li_3..ls_3$, etc.

Ex1: TRIDIMENSIONAL: array [1..10, 1..20, 1..40] of real; \Rightarrow define matriz tridimensional onde a 1ª dimensão tem 10 elementos, a 2ª tem 20 elementos e a 3ª tem 40 elementos.

Ex2: BIDIMENSIONAL: array [1..5,1..9] of char; \Rightarrow define matriz bidimensional com 5 elementos do tipo *char* na 1ª dimensão e 9 na 2ª dimensão.

- Manipulação de matrizes nos programas Pascal

identificador_array[posição_dim1, posição_dim2, ...]

posição_dim1, posição_dim2... podem ser expressos com constantes, variáveis ou expressões, devendo ser do tipo correto (correspondente à definição do array) e estar dentro do intervalo pré-definido.

Ex: tridim[5,4,8]
bidim[i,9], onde *i* é uma variável inteira com valores variando de 1 a 5.

Os elementos da MATRIZ podem ser usados em expressões, instruções de atribuição, instruções read e write, etc.

- Programa Exemplo :

```

Program Matrizes;

Var Matriz: array [1..20, 1..10] of integer;
    lin, col : integer;
Begin
    For lin := 1 to 20 do
        For col := 1 to 10 do
            read (matriz[lin,col]);
        ...
    
```

Exercícios de Fixação

1. Escreva um programa que leia duas matrizes bidimensionais reais MAT1 e MAT2 de dimensões 3x5 cada, calcule e imprima a matriz soma MSOMA.
2. Calcule e imprima a soma dos elementos situados abaixo da diagonal principal da matriz A (dimensões 10x10), incluindo os elementos da própria diagonal.
3. Escreva um programa que leia duas matrizes reais A e B de dimensões 3x5 e 5x3, respectivamente, calcule e imprima o produto delas.
4. Dada uma matriz A de dimensões 5x4 formada de elementos numéricos reais, calcule e mostre sua matriz transposta T.
5. Dada uma matriz B formada por números inteiros com 10 linhas por 15 colunas, determinar o elemento de maior valor algébrico. Mostre tal elemento e sua posição na matriz (linha e coluna).

5.5 Algoritmos de Classificação e Busca

☒ Algoritmos de Classificação

Existem diversos métodos para classificar (ou ordenar) uma estrutura de dados, dentre eles temos:

- ✓ Método da Bolha (Bubble Sort)
- ✓ Método da Seleção Direta
- ✓ Método Quick Sort

Exemplo: Método da Bolha na ordenação de um vetor de “nomes de objetos”

```

Program BubbleS;
Uses crt;
Const   N=5;
Type
  letras = string[10];
  vet_letras= array[1..N] of letras;
Var
  objetos :vet_letras;
  aux     :letras;
  I,J,cont:integer;
Begin
  clrscr;
  Gotoxy(22,2);
  Write('>>> Exercicio - Ordenacao de Vetores com metodo da Bolha<<<<');
  Gotoxy(7,4);
  write('Digite 'n,' nomes de objetos para compor o conjunto');
  For i:=1 to N do
    begin
      gotoxy(7,6+i);
      write ('Digite o elemento Objetos('i,') : ');
      readln( objetos[i] );
      gotoxy(33,6);
      write(' ');
    end;
  {***** Ordenação do Vetor *****}
  For i:= 2 to N do
    for j:= N downto i do
      if objetos[j] < objetos[j-1] then
        begin
          aux     := objetos[j];
          objetos[j] := objetos[j-1];
          objetos[j-1]:= aux
        end;
  {***** Saida Ordenada *****}
  gotoxy(7,15);
  writeln ('Vetor Ordenado :');
  for i:=1 to N do
    begin
      gotoxy(20,15+i);
      writeln( objetos[i] );
    end;

```

```
    end;
End.
```

⊗ Algoritmo de Busca Sequencial

Exemplo: Considere um vetor A com 50 elementos. Verificar se existe um elemento igual a **K** no vetor. Se existir mostrar a posição em que se encontra, senão imprimir "*não encontrei K no vetor*".

```
Program Procura_K;
Uses crt;
CONST
    Max=10; {Indica o numero maximo de elementos do ARRAY NUM}
VAR
    I,K : integer;
    achou : boolean;
    NUM : Array [1..Max] of integer;
Begin
    clrscr;
    Gotoxy(10,7);
    Writeln('Digite ',Max,' numeros inteiros');
    For i:=1 to Max do
        begin
            Gotoxy(10,9);
            Write('Digite o elemento ('i,') : ');
            read(NUM[i]);
            gotoxy(34,9); { posiciona no local onde esta o numero digitado}
            write(' '); { limpa o campo de digitacao do numero }
        end;
    Gotoxy(10,12);
    Write('Digite o numero que deseja procurar no conjunto: ');
    read(k);
    achou:=false;
    i:=1;
    While (not achou) and (i<=Max) do
        if Num[i]=k then
            achou:=true
        else
            i:=i+1;
    gotoxy(12, 16);
    If achou then
        write('Achei o numero ',k,' na posicao ('i,') do vetor !!!')
    else
        write('Nao achei o numero ',k,' no vetor !!!');
End.
```

⊗ Pesquise outro algoritmo de busca e faça um exemplo em Pascal.
(Ex. Algoritmo de Pesquisa Binária)