A Linguagem Turbo Pascal ....................2
Estrutura de um programa em Pascal: ....................2
Comentários ....................2
Declaração de Bibliotecas (Uses): ....................3
Variáveis do Turbo Pascal ....................3
Constantes ....................4
Operadores do Turbo Pascal ....................5
Entrada de dados ....................7
  Comandos Read e Readln ....................7
Saída de dados ....................7
  Comandos Write e Writeln ....................7
Exercícios ....................9
Manipulação de Variáveis Numéricas ....................10
  Função ABS ....................10
  Função DEC ....................11
  Função INC ....................11
  Função EXP ....................12
  Função FRAC ....................12
  Função INT ....................12
  Função RANDOM ....................13
  Função RANDOMIZE ....................13
  Função ROUND ....................13
  Função TRUNC ....................14
  Função SQR ....................14
  Função SQRT ....................14
Biblioteca CRT – Recursos na Tela ....................14
Estruturas de Decisão: ....................18
  SE...ENTÃO...SENÃO (IF...THEN...ELSE) ....................18
  Exercícios usando IF ....................23
  CASE...OF (Caso...Seja) ....................23
  Exercícios usando Case ....................28
Estrutura de Repetição ....................29
  While...Do (Enquanto...Faça) ....................29
  Exercícios usando While Do ....................30
  Repeat...Until (Repita...Até que) ....................30
  Exercícios usando Repeat / Until ....................32
  FOR...TO...DO (PARA..ATÉ..FAÇA) ....................33
  Exercícios de For/Do ....................35
Estruturas de Dados Homogêneas ....................35
  Matrizes Unidimensionais ou Vetores ou Arrays ....................35
  Declaração de uma Matriz Unidimensional ....................39
  Ordenação Vetores ....................42
Métodos de Pesquisa em uma Matriz ....................46
Método de Pesquisa Seqüencial ou Linear ....................46
Método de Pesquisa Binária (Manzano) ....................48

## A Linguagem Turbo Pascal

O Pascal, cujo nome é uma homenagem ao filósofo Blaise Pascal, foi inventado por Niklaus Wirth no início da década de 70. Originalmente, a linguagem de programação Pascal foi criada para ser uma linguagem educacional, para ajudar programadores iniciantes a desenvolver bons hábitos, permitindo a elaboração de programas claros, concisos e estruturados. Antes do Pascal, a introdução à programação se fazia, em geral, através do Fortran, uma linguagem desestruturada e bem mais antiga. Wirth acreditava que muitos dos erros mais comuns de programação poderiam ser evitados com o uso de uma linguagem por blocos, e que trouxesse, embutido, um severo controle de tipos.

Apesar de todas as dificuldades iniciais, de seu propósito educacional e a facilidade de programação, o Pascal começou a ser utilizado por programadores de outras linguagens. Contudo, somente no final de 1983, quando a empresa Borland International lançou o Turbo Pascal é que esta linguagem ganhou maior fama.

## Estrutura de um programa em Pascal:

Um programa em pascal tem a seguinte estrutura:

```
Program NomePrograma;
{------- Área de declarações de bibliotecas, variáveis, constantes e conjuntos}
Uses ... { utilização das bibliotecas }
Const... { definição de constantes }
Type ... { definição de novos tipos }
Var ... { declaração das variáveis }
Begin
{instruções e comandos};
End.
```

## Exemplo de um programa simples em Pascal:

```
Program teste;  {nome do programa}
Var A : integer; {Estamos declarando a variável A como sendo do tipo inteiro}
Begin  {Início}
    Readln(A); {Leia A}
    Writeln('O valor de A é: ', A); {Escreva na tela: O valor de A é:  e coloque na frente o
valor de A}
End. {Fim}
```

**OBS**: O Pascal não faz distinção entre letras minúsculas e letras maiúsculas

## Comentários

Usam-se os símbolos '{' e '}' ou '(*' e '*)' para indicar que o texto entre os delimitadores é um comentário, o qual não tem sentido para o ***compilador***.
Perceba que as linhas entre chaves e em azul colocadas na frente de cada linha são apenas um comentário !

**Declaração de Bibliotecas (Uses):**

Permite a utilização de bibliotecas de subprogramas existentes na linguagem Pascal ou criadas pelo programador. Estas bibliotecas são 'importadas' para o programa quando declaradas.
***Ex***:Uses crt, dos, graph, printer, overlay;
**CRT**: Rotinas de tratamento de vídeo e som, torna disponível todos os comandos para formatação de tela.
**DOS**: Manipulação do sistema operacional (dos), rotinas que permitem controle de baixo nível.
**GRAPH**: Rotinas para tratamento gráfico.
**PRINTER**: Rotinas para tratamento de impressões
**OVERLAY**: Rotinas para tratamento de Overlays (sobreposição)

**Variáveis do Turbo Pascal**

Existem 4 grupos de variáveis no Turbo Pascal que são divididas em:

**NUMÉRICAS** - utilizadas para trabalharmos com números. Ex: 0,1,2,3...

**ALFANUMÉRICAS** - Utilizadas para trabalharmos com caracteres. Ex: a,b,c,A,B,C,!,@,#, 0, 1,2, ...

**LÓGICAS** - São variáveis que assumem apenas dois valores: TRUE ou FALSE, ou seja, verdadeiro e falso.

**Variáveis Numéricas:**

| **tipo** | **Faixa** | **bytes** |
|---|---|---|
| Shortint | -128..127 | 1 |
| Integer | -32768..32767 | 2 |
| Longint | -2147483648.. 2147483647 | 4 |
| Byte | 0..255 | 1 |
| Word | 0..65535 | 2 |
| real | 2.9e-39..1.7e38 | 6 |
| single | 1.5e-45..3.4e38 | 4 |

| | | |
|---|---|---|
| double | 5.0e-324..1.7e308 | 8 |
| extended | 3.4e-4932..1.1e4932 | 10 |

**Variáveis Alfanuméricas**

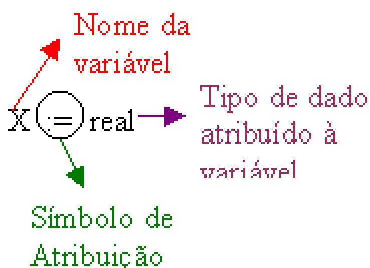
| Tipo | Capacidade |
|---|---|
| Char | Ocupa 1 byte. Capacidade para armazenar um caracter por vez. |
| String | Ocupa de 2 a 256 bytes. Capacidade para armazenar de 1 a 256 caracteres |

**Variáveis Lógicas**

| Tipo | Capacidade |
|---|---|
| Boolean | Ocupa 1 byte. Assume os valores TRUE ou FALSE |

**Declarando uma Variável no Pascal:**

**A atribuição de valores para as variáveis é feita através do comando.**




```
var1 := 1;                  { recebe um valor - inteiro, real...}
var2 := X + Y;              { recebe uma expressão }
varBool := false;           { atribuição lógica }
varStr := 'atribui string'; { recebeu um string }
varChar := a;               { recebe um caracter }
```

Como em Pascal as variáveis não são inicializadas, temos a necessidade de as inicializar no começo de um programa, sempre lembrando que o que for atribuído à variável deve ter o tipo compatível com o da variável.

**Ex: Program Atrib;**

```
var V : integer;
begin
  V := 0;
  ...
end.
```

**Constantes**

A declaração const

Nesta subárea, podemos definir tantas constantes quantas quisermos.

**Sintaxe:**

**Const**
**Meu_telefone = '(16)3384-1111';**
**Minha_Faculdade = 'Politécnica Matão';**

(* e assim por diante *)

Toda vez que nos referirmos às constantes acima, o Turbo Pascal substitui-la-á pelos seus respectivos valores.

**Operadores do Turbo Pascal**

Operadores Aritméticos

| | |
|---|---|
| **+** adição | **/** divisão entre números reais |
| **-** subtração | **DIV** divisão entre números inteiros |
| * multiplicação | **MOD** resto da divisão |

**Exemplos:**

(a+b) / (a-b)
a div b

Operadores Lógicos

**AND** E lógico
**OR** OU lógico
**XOR** OU EXCLUSIVO lógico

Estes operadores só aceitam como operandos, valores lógicos, ou seja : **TRUE** e **FALSE** .

A operação **AND** resulta em TRUE se e somente se todos os operandos forem TRUE, se um deles ou mais de um for FALSE então o resultado será FALSE.

A operação **OR** resulta TRUE quando pelo menos um dos operandos for TRUE.

A operação **XOR** resulta TRUE quando os operandos forem diferentes entre si, isto é, quando um for TRUE o outro deverá ser FALSE.

**Exemplos:**

(a=b*2) or (x<>y)

not (a=b)

**Operadores Relacionais**

| = | igual |
|---|---|
| <> | diferente |
| > | maior que |
| < | menor que |
| >= | maior ou igual que |
| <= | menor ou igual que |
| IN | testa se um elemento está incluso em um conjunto |

**Exemplos**

Se A=30 e B=50 então

( A = B ) FALSE
( A < B ) TRUE

Se A=50 , B=35, C='A' , D='B'

( ( A < B ) OR ( C < D ) ) TRUE

Prioridade dos Operadores:

| **not** | **1** |
|---|---|
| ***, /, div, mod, and** | **2** |
| **+, -, or** | **3** |
| **=,<>, <, <=, >** | **4** |

Lembre-se de que o uso de parêntese prioriza a execução de qualquer operador!

**Entrada de dados**

**Comandos Read e Readln**

**Read** - Comando que permite a entrada de dados através de dispositivos que não são o teclado. Esses dispositivos são geralmente arquivos em disco.

**Read(<lista de variáveis>)**

**Readln** -Comando que permite a entrada de dados através de variáveis oriundas de diversos dispositivos, principalmente o teclado.

**Readln(<lista de variáveis>)**

**Saída de dados**

**Comandos Write e Writeln**

São as principais procedures destinadas a exibir todos os tipos dados no vídeo.

**Write -** Comando que permite escrever em um dispositivo de saída (o vídeo, normalmente).

**Writeln -** Comando que permite escrever em um dispositivo de saída (vídeo), avançando o cursor para o início da próxima linha.

O comando Write (Writeln), permite ao usuário um certo controle com relação à formatação dos dados de saída. Podemos imprimir variáveis com justificação à direita e se o tipo da variável for real, double ou extended, podemos também definir a quantidade de casas decimais. As formatações seguem a seguinte forma:

Primeira forma:

Write(parâmetro_1,Parâmetro_2, ...);

Exemplo:

```
Program formatacao;
Uses Crt;
Var
  A: real;
  B: integer;
  C: String;
Begin
  clrscr; {apaga a tela}
  A:=2;
  B:=2;
  C:='Teste' ;
  Writeln(A);
  Writeln(B);
  Writeln;  {apenas pula uma linha}
  Writeln(C);
End.
```

Este programa resultaria na seguinte tela:

```
2.0000000000E+00
2

Teste
```

<u>Segunda forma</u>:

Write(parâmetro : n);

onde n é um número inteiro que determina quantas colunas o cursor deve ser deslocado à direita, antes do parâmetro ser escrito. Além disso, o parâmetro é escrito da direita para a esquerda, exemplo:

```
Program formatacao;
Uses Crt;
Var
  A: real;
  B: integer;
  C: String;
Begin
  clrscr; {apaga a tela}
  A:=2;
  B:=2;
  C:='Teste' ;
  Writeln(A);
  Writeln(B:5);
  Writeln;  {apenas pula uma linha}
  Writeln(C:10);
End.
```

Resultaria a seguinte tela:

```
2.0000000000E+00
....2

..............Teste
```

Os pontos representam espaços em branco.

Terceira forma:

Write(parâmetro : n : d);

Neste caso, n tem a mesma função que o caso anterior sendo que d representa o número de casas decimais. Obviamente, parâmetro terá que ser do tipo Real.
Exemplo:

```
Program formatacao;
Uses Crt;
Var
  A: real;
  B: integer;
  C: String;
Begin
  clrscr; {apaga a tela}
  A:=2;
  B:=2;
  C:='Teste' ;
  Writeln(A:5:2);
  Writeln(B:5);
  Writeln; {apenas pula uma linha}
  Writeln(C:10);
End.
```

resultaria a seguinte tela:

```
....2.00
....2

..............Teste
```

Enviar dados para a Impressora:
Podemos enviar dados para a impressora através das procedures Write e Writeln. Para tanto, devemos colocar, antes dos parâmetros a serem enviados à impressora, o nome lógico **LST**.
**Exemplo**: **Writeln**('apresentar no vídeo')
**Writeln**(**lst**,'imprimir este parágrafo');

**Exercícios**

1. Criar um programa que pergunte o nome do produto, o preço e a quantidade e que ao terminar exiba o valor total do produto.

2. Ler uma temperatura em graus Celsius e apresentá-la convertida em graus Fahrenheit. A fórmula de conversão é : F := (9*C+160)/5, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.

3. Efetuar a leitura de um número inteiro e apresentar o resultado do quadrado desse número

4. Elaborar um programa que efetue a leitura de três valores (A, B e C) e apresente como resultado final a soma dos quadrados dos três valores lidos.

5. Fazer um programa que pegue o valor da compra, divida pelo número das parcelas e calcule o valor das parcelas.

6. Crie um programa que lê a base e a altura de um triângulo e calcule sua área.

7. Crie um programa que lê uma distância percorrida, o tempo gasto para percorrer tal distância e calcule a velocidade. (v=d/t)

8. Fazer um programa que leia dois valores V1 e V2 e escolha a operação a ser executada entre as seguintes.
    - Soma (V1 + V2)
    - Subtração (V1 - V2)
    - Produto (V1 * V2)
    - Divisão ( V1 / V2)

## Manipulação de Variáveis Numéricas

Com o intuito de se desenvolver problemas matemáticos que fogem das operações básicas de adição, subtração, multiplicação e divisão utilizando também nos algoritmos funções que nos auxiliam no cálculo de algumas operações matemáticas. Neste capítulo veremos cada dessas funções com exemplos e a sua correspondência na linguagem Turbo Pascal. Os fluxogramas foram omitidos devido à simplicidade dos algoritmos. Analise cada umas das funções com atenção, pois elas serão amplamente utilizadas nos capítulos posteriores.

## Função ABS

Retorna o valor absoluto de uma expressão numérica. Pode ser utilizado na declaração de constantes. As variáveis podem ser do tipo ***inteiras*** ou ***fracionárias.***

```
PROGRAM EXEMPLO_ABS;
USES CRT;
VAR
        A:REAL;          B:INTEGER;
BEGIN
        CLRSCR;               { limpa tela }
        A:= ABS(-123,43);   { a = 123,43 }
        WRITELN(A);          { Aparecerá 123,43 }
        B:=ABS(-10);          { b = 10 }
        WRITELN(B);          {Aparecerá 10)
        READLN;               { Pausa }
END.
```

**Função DEC**

Decrementa o valor de uma variável numérica inteira, em uma ou mais unidades. Se não for usado parâmetro para decremento será assumido 1. Utilizando somente com ***variáveis inteiras***.

```
PROGRAM EXEMPLO_DEC;
USES CRT;
VAR
        A:REAL;
BEGIN
        CLRSCR;               { limpa tela }
        A:= 10;                {a = 10}
        DEC(A);               { Decrementa 1 unidade; a = 9}
        WRITELN(A);         {Aparecerá 9 }
        DEC(A,2);             { Decrementa 2 unidades; a = 7 }
        WRITELN(A);         { Aparecerá 7 }
        READLN ;              { Pausa }
END.
```

**Função INC**

Mesmo procedimento da função DEC, a diferença é que incrementa a variável utilizada em  vez decrementar. Utilizando somente com ***variáveis inteiras***.

```
PROGRAM EXEMPLO_INC;
USES CRT;
VAR
        A:INTEGER;
BEGIN
        CLRSCR;               { limpa tela }
        A:= 10;                {a = 10}
        INC(A);               { Incrementa 1 unidade; a = 11}
        WRITELN(A);         {Aparecerá 11 }
        INC(A,2);             { Incrementa 2 unidades; a = 13 }
        WRITELN(A);         { Aparecerá 13 }
        READLN ;              { Pausa }
END.
```

**Função EXP**

Retorna o valor exponencial da variável ou do número em uso. Pode ser utilizado com variáveis ***inteiras*** ou ***fracionárias***, mas retornará sempre um valor fracionário.

```
PROGRAM EXEMPLO_EXP;
USES CRT;
VAR
        A:INTEGER;
BEGIN
        CLRSCR;                         { limpa tela }
        A:= 1;                          { A = 1}
        WRITELN(EXP(A):1:3);         { Aparecerá  2.718 }
        READLN ;                       { Pausa }
END.
```

**Função FRAC**

Retorna a parte fracionária de um número. Se for utilizado um número inteiro, retornará 0. Se for utilizado como número negativo a parte fracionária também será negativa. Pode ser utilizado com variáveis ***fracionárias*** ou ***inteiras***.

```
PROGRAM EXEMPLO_FRAC;
USES CRT;
VAR
        A:REAL;
BEGIN
        CLRSCR;                         { limpa tela }
        A:= 1.23;                        { A = 90 graus}
        WRITELN(FRAC(A):1:2);          { Aparecerá  023 }
        READLN ;                        { Pausa }
END.
```

**Função INT**

Retorna a parte inteira de um número. Se for utilizado um número inteiro, retornará o próprio número. Pode ser utilizado com variáveis ***fracionárias*** ou ***inteiras***.

```
PROGRAM EXEMPLO_INT;
USES CRT;
VAR
        A:REAL;
BEGIN
        CLRSCR;                         { limpa tela }
        A:= 1.23;                        { A = 90 graus}
        WRITELN(INT(A):1:2);           { Aparecerá  1.00. }
        READLN ;                        { Pausa }
END.
```

**Função RANDOM**

Retorna um número aleatório que pode variar de 0 a 1, ou por um limite passado como parâmetro.

```
PROGRAM EXEMPLO_RANDOM;
USES CRT;
BEGIN
        CLRSCR;                         { limpa tela }
        WRITELN(RANDOM:1:10);          { Gera entre 0 e 1 }
        READLN ;                        { Pausa }
END.
```

**Função RANDOMIZE**

Se executarmos novamente o programa anterior veremos que o resultado é o mesmo. Para contornar este problema utilize a função randomize. Assim sempre que for executado o programa serão gerados números diferentes.

```
PROGRAM EXEMPLO_RANDOM;
USES CRT;
BEGIN
        RANDOMIZE;                       { Altera sempre }
        CLRSCR;                           { limpa tela }
        WRITELN(RANDOM:1:10);          { Gera entre 0 e 1 }
        READLN ;                          { Pausa }
END.
```

**Função ROUND**

Arredonda um número ***fracionário***. Se o valor decimal for de 0.5 ou maior, o número é arredondado para cima, caso contrário, para baixo.

```
PROGRAM EXEMPLO_ROUND;
USES CRT;
BEGIN
        CLRSCR;                         { limpa tela }
        WRITELN(ROUND(1.5) ;          { Aparece na tela o 2 }
        READLN ;                        { Pausa }
END.
```

**Função TRUNC**

Trunca um número ***fracionário***, retorna sempre sua parte inteira.

```
PROGRAM EXEMPLO_TRUNC;
USES CRT;
BEGIN
        CLRSCR;                         { limpa tela }
        WRITELN(TRUNC(16.5);          { Aparece na tela o16}
        READLN;                         { Pausa }
END.
```

**Função SQR**

Eleva um número ao quadrado. Aceita números ***inteiros*** e ***fracionários***.

```
PROGRAM EXEMPLO_SQR;
USES CRT;
BEGIN
        CLRSCR;                         { limpa tela }
        WRITELN(SQR(2));              { Aparece na tela o 4 }
        READLN ;                        { Pausa }
END.
```

**Função SQRT**

Calcula a raiz quadrada de um número. Aceita números ***inteiros*** e ***fracionários***.

```
PROGRAM EXEMPLO_SQRT;
USES CRT;
BEGIN
        CLRSCR;                         { limpa tela }
        WRITELN(SQRT(4));             { Aparece na tela o 2 }
        READLN ;                        { Pausa }
END.
```

**Biblioteca CRT – Recursos na Tela**

***GOTOXY –*** Permite o posicionamento do cursor em qualquer posição da tela, referenciando pelos eixos X e Y, ou seja a coluna e linha. Seu formato GOTOXY(coluna, linha).

```
PROGRAM EXEMPLO_GOTO;
USES CRT;
BEGIN
        CLRSCR;
        GOTOXY(20,2);
        WRITELN( ' ESTOU ESCREVENDO NA COLUNA 20 E NA LINHA 10');
        READLN;
END.
```

***DELAY -*** Permite executar um atraso no tempo do programa que é cronometrado em milésimos de segundos.

```
PROGRAM EXEMPLO_DELAY;
USES CRT;
BEGIN
        CLRSCR;
        WRITELN( ' AGUARDEM...');
        DELAY(2000);  {TEMPO DE ESPERA NA TELA}
END.
```

***CLREOL –*** Permite apagar todos os caracteres da linha correspondente. Veja o exemplo.

```
PROGRAM EXEMPLO_3;
USES CRT;
BEGIN
        CLRSCR;
        WRITELN('ISTO ESTÁ NA LINHA 1');
        WRITELN('ISTO ESTÁ NA LINHA 2');
        WRITELN('ISTO ESTÁ NA LINHA 3');
        WRITELN('ISTO ESTÁ NA LINHA 4');
        WRITELN('ISTO ESTÁ NA LINHA 5');
        WRITELN('ISTO ESTÁ NA LINHA 6');
        WRITELN('ISTO ESTÁ NA LINHA 7');
        WRITELN('APÓS PRESSIONAR <ENTER> AS LINHAS 2, 4 e 6 SERÃO APAGADAS') ;
        READLN;
        GOTOXY(1,2);CLREOL;
        GOTOXY(1,4);CLREOL;
        GOTOXY(1,6);CLREOL;
        READLN;
END.
```

***HIGHVIDEO – Coloca o texto em cor de alta intensidade. Veja o exemplo.***

```
PROGRAM EXEMPLO_HIGHVIDEO;
USES CRT;
BEGIN
CLRSCR;
WRITELN('ESTE TEXTO ESTÁ EM BAIXA INTENSIDADE, OU SEJA SEU
NORMAL');
HIGHVIDEO;
WRITELN(' ESTE TEXTO ESTÁ AGORA EM ALTA INTENSIDADE');
READLN;
END.
```

***LOWVIDEO –*** Coloca o texto em baixa intensidade.

***NORMVIDEO –*** Faz com que o texto volte ao seu normal; sem cor.

***SOUND*** - O comando sound emite um som em determinada freqüência. Deve ser utilizado em conjunto do comando NOSOUND que o desativa. Se o comando NOSOUND não for utilizado no programa o som continua ate que o programa seja novamente rodado com o NOSOUND. O programa abaixo ilustra o comando SOUND e NOSOUND em conjunto com o comando delay. Uma vez acionado o programa, de forma alguma utilize CTRL-BREAK para travá– lo, deixe que o nosound desative o som para depois uitilizá- lo.

```
PROGRAM EXEMPLO_SOUND;
USES CRT;
VAR
I:INTEGER;
BEGIN
CRLSCR;
WRITELN('ESTE PROGRAMA EMITIRÁ APENAS SONS');
FOR I:=1 TO 4000 DO
BEGIN
SOUND(I);
DELAY(2);
END;
NOSOUND;
END.
```

***WINDOW –*** Este procedimento permite que se defina a area utíl do video que sera utilizada. O default é a partir da coluna 1 linha 1 até coluna 80 e linha 25, para modo de 80 colunas, é coluna 1 até coluna 40 e linha 25, posições definidas pela nova window. Sua sintaxe: window(col1,lin2, col2, lin2). Vejamos um exemplo.

```
PROGRAM EXEMOPLO_WINDOW;
USES CRT;
BEGIN
  WINDOW(10,10,70,20);
  CLRSCR;  {LIMPA SOMENTE A JANELA}
  WRITELN(' TESTE'); {ESCREVE 'TESTE' NA LINA COLUNA 10 E NA LINHA 10}
END.
```

***READKEY –*** Permite efetuar a leitura de uma tecla. Pode ser utilizado também para efetuar uma pausa no final do programa.

```
PROGRAM EXEMPLO_READKEY;
USES CRT;
VAR   TECLA: CHAR;
BEGIN
  WRITELN('DIGITE UMA TECLA');
  TECLA:= READKEY;
  WRITELN('VOCÊ DIGITOU A TECLA   ', TECLA);
END.
```

***DELLINE –*** Este procedimento elimina a linha em que estiver posicionado o cursor, fazendo o rolamento das linhas que estiverem abaixo desta e incrementando uma linha ao final do vídeo. Alinha deletada  não pode ser recuperada. Tanot a linha deletada como a linha que foi introduzida, respeitam a WINDOW ativa no momento.

```
PROGRAM EXEMPLO_INSLINE;
USES CRT;
BEGIN
TEXTBACKGROUND(0);
CLRSCR;
TEXTBACKGROUND(1);
WINDOW(10, 10, 20, 20);
TEXTBACKGROUND(1);
         CLRSCR;
         WRITELN('LINHA 1');
         WRITELN('LINHA 2');
         WRITELN('LINHA 3');
         WRITELN('LINHA 4');
         READLN;
         GOTOXY(1, 2);INSLINE;
         READLN;
END.
```

***INSLINE–*** Este procedimento insere uma linha onde estiver posicionado o cursor, fazendo o rolamento das linhas que estiverem abaixo desta e incrementando uma linha ao final do vídeo. A ultima linha é perdida e não Poe ser recuperada.

```
PROGRAM EXEMPLO_INSLINE;
USES CRT;
BEGIN
TEXTBACKGROUND(0);
CLRSCR;
TEXTBACKGROUND(1);
WINDOW(10, 10, 20, 20);
TEXTBACKGROUND(1);
        CLRSCR;
        WRITELN('LINHA 1');
        WRITELN('LINHA 2');
        WRITELN('LINHA 3');
        WRITELN('LINHA 4');
        READLN;
        GOTOXY(1, 2);INSLINE;
        READLN;
END.
```

**Estruturas de Decisão:**

Um programa é uma lista de seqüência de comandos que são executados seqüencialmente. A menos que seja solicitado o contrário, um programa executa as instruções do início de um programa (Begin) até o fim (End). Conforme aumenta a complexidade, tais programas precisão tomar decisões e alterar este processamento seqüencial. Com estruturas de decisões, o fluxo de instruções seqüenciais é escolhido em função do resultado da avaliação de uma ou mais condições, onde uma condição é uma expressão lógica. Existem dois tipos de estruturas de decisão, aquela que fornece apenas duas possibilidades (se...então...senão...fim), de seqüência para o programa, e outra que possui inúmeras possibilidades (caso...senão...fim) de seqüências.

**SE...ENTÃO...SENÃO (IF...THEN...ELSE)**

Desvio Condicional Simples

Um comando condicional é representado em Pascal através do comando **if - then.**

**If {<condição>} Then**

**{instruções para condição verdadeira}**

Veja o exemplo do programa abaixo:

```
Program Calc_Media;
Var
  N1, N2, Media: real;
Begin
  Readln(N1, N2);
  Media:=(N1+N2)/2;
  If Media >=7 then
     Writeln('Aprovado');
End.
```

Somente se a condição é verdadeira, executamos comando escrever Aprovado.

Note que depois do **THEN** NÃO existe ponto e vírgulo (;) !

## Desvio Condicional Composto

Um comando condicional é representado em Pascal através do comando **if...then...else.**

**If {<condição>} Then**

**{instruções para condição verdadeira}**

**else**

**{instruções para condição falsa}**

Veja o exemplo do programa abaixo:

```
Program Calc_Media;
Var
  N1, N2, Media: real;
Begin
  Readln(N1, N2);
  Media:=(N1+N2)/2;
  If Media >=7 then
     Writeln('Aprovado')
  Else
     Writeln('Reprovado');
End.
```

**Importante:** Note que, se existe o else então não se pode terminar a parte do then com ponto e vírgula (;). A presença do ponto e vírgula terminando a parte do then faz com que o compilador "pense" que não existe a parte do else automaticamente.

**Definição da instrução: uso de begin e end**

Nos trechos acima induziu-se que só se pode executar uma instrução para cada then ou else. Isto não é verdade! Usando os delimitadores begin (começo) e end (fim) faz-se com que um conjunto de instruções sejam interpretados como uma única instrução (na verdade um bloco de instruções):

## Desvios Condicionais Encadeados ou Aninhados

### Aninhamento de if's:

Imagine o seguinte problema: Se eu não tiver dinheiro devo pedir emprestado ao banco. Se eu tiver pouco dinheiro não vou fazer nada, se eu tiver mais ou menos, vou comer uma pizza. Se eu tiver muito dinheiro vou jantar em Paris.

```
Program Oh_dinheiro
Var dinheiro : real;
Begin
Redaln(dinheiro);
if dinheiro <= 0.5 then
  Writeln('Pedir_Empréstimo_ao_Banco')
else
   if dinheiro > 100000 then
      writeln('Jantar_em_Paris')
   else
      if dinheiro > 20 then
         writeln('Comer_uma_Pizza')
      else
         Writeln ('Não_fazer_nada');
End.
```

Observe a estrutura para um programa com desvios condicionais aninhados ou encadeados:

**if** condição1 **then**

**if** condição2 **then** instrução1

**else** instrução2

Quando a instrução2 é executada? Quando a condição2 for falsa ou quando a condição1 for falsa?

Neste caso a instrução2 é executada quando a condição2 for falsa, não executando tarefa alguma, caso a condição1 seja falsa ! Se não fosse essa a intenção dever-se-ia usar a seguinte construção:

```
if  condição1 then
begin
    if  condição2 then instrução1
end  else instrução2;
```

Neste caso, a instrução2 é executada quando a condição1 for falsa ! Não havendo instruções nenhuma caso a condição2 for falsa !

Para que o programa execute tarefas para as condições1 e 2 sendo verdadeiras ou falsas, o programa deveria ser construído da seguinte forma:

```
if condição1 then
 begin
  if condição2 then instrução1
 else
  instrução2
 end
else
 instrução3
```

Neste caso, a instrução1 é executada caso a condição2 for verdadeira. A instrução2 é executada caso a condição2 for falsa e a instrução3 é executada caso a condição1 for falsa e a condição2 é testada caso a condição1 for verdadeira !

Vamos analisar o uso do **BEGIN** e **END** segundo Flávio Mendes:

Quando você tem mais do que uma instrução a ser executada para uma determinada condição, então deve-se usar um begin antes de começar a relacionar as instruções e usar um End logo após a última instrução, desta forma, você está fazendo com que um conjunto de instruções sejam interpretados como uma única instrução (na verdade um bloco de instruções):

**Veja o exemplo:**

```
if condição then
  begin
    instrução1;
```

```
     instrução2;
     instrução3
  end
else
  begin
    outra_instrução1;
    outra_instrução2
  end;
```

Olha o exemplo citado pelo autor:

```
if Filho_mal_educado then
  begin
    Não_deixar_ver_televisão;
    Não_deixar_jogar_no_computador;
    Não_deixar_tomar_coca-cola
  end
else
  begin
    Elogiar;
    Incentivar;
    Deixar_mais_livre
  end
```

## Aninhamento de if's:

Imagine o seguinte problema: Se eu não tiver dinheiro devo pedir emprestado ao banco. Se eu tiver pouco dinheiro não vou fazer nada, se eu tiver mais ou menos, vou comer uma pizza. Se eu tiver muito dinheiro vou jantar em Paris.

```
        if dinheiro <= 0.5 then Pedir_Empréstimo_ao_Banco

                else if dinheiro > 100000 then Jantar_em_Paris

                        else if dinheiro > 20 then Comer_uma_Pizza

                                else Não_fazer_nada;
```

Olhe com cuidado este exemplo e entenda porque o problema anterior está codificado perfeitamente Existem algumas situações em que se deve tomar cuidado na programação:

Pense no assunto...Apesar de alguns livros sugerirem que, na dúvida, deve-se colocar o par begin end sempre, eu (Flávio Mendes) discordo! Deve ser utilizado o par begin end quando necessário! Outro tipo de construção ridícula é a colocação de begin end sem nenhum comando "dentro". Por exemplo:

```
if teste1 then
        if teste2 then
                executa1
        else begin end
```

```
else executa2;
```

## Uso do ponto e vírgula:

A não ser no caso de terminar a parte do then quando se tem o else, pode-se sempre usar o ponto e vírgula para terminar comandos.

Existem situações em que o uso do ponto e vírgula é opcional. Isto é uma questão de estilo e eu (Flávio Mendes), particularmente, uso ponto e vírgula somente quando necessário...(Analise com cuidado os exemplos anteriores).

Uso opcional: quando o comando que segue a instrução presente já é um comando de finalização. Por exemplo:

- **Não é necessário o uso de ponto e vírgula antes de end**
- **Não é necessário o uso de ponto e vírgula antes de until (aliás não é necessário delimitar o corpo de instruções contidas no par repeat until com begin end...O par repeat until já é um delimitador).**

### Exercícios usando IF

1. Fazer um programa que pergunte o nome do aluno, a nota1, a nota2 e a nota 3. Calcular a média e o desempenho (aprovado para média >= 7, reprovado para média <= 4 e recuperação para média entre 4,1 e 6,9).
2. Elaborar um programa que pergunte um número. Caso este número esteja entre 20 e 90, deve ser emitida a mensagem “O número está na faixa dos 20 aos 90”, caso contrário à mensagem deve ser “O número está fora da faixa dos 20 aos 90”.
3. Criar um programa que pergunte uma senha. Se a senha for “ETE” então exiba a mensagem “Senha Correta”, caso contrário exiba a mensagem “Senha Incorreta”
4. Efetuar a leitura de quatro números inteiros e apresentar os números que são divisíveis por dois e três.
5. Pergunte as medidas do lado de um triângulo. De acordo com os valores digitados o programa deverá exibir se o triângulo é eqüilátero (3 lados iguais), isósceles (dois lados iguais) ou escaleno (três lados diferentes).
6. Crie um programa para digitar o valor de uma compra. Se o valor da compra for maior ou igual a 100,00, então, dar desconto de 10% e mostrar o valor a pagar, senão exibir uma mensagem: “Compra sem Desconto” e colocar o mesmo valor da compra para o valor a pagar.
7. Faça um programa para calcular o peso ideal de uma pessoa adulta, conforme o sexo. Se o sexo da pessoa for masculino, o valor do peso ideal será calculado da seguinte forma: $PI=Altura^2 * 23$. Se for feminino, calcule o peso ideal como: $PI=Altura^2 * 22$ . Onde PI = Peso Ideal.

### CASE...OF (Caso...Seja)

Esta estrutura é usada para decisão múltipla, que também poderia ser construída através da sentença If-then-else, porém é muito mais prático e fácil utilizar a estrutura CASE...OF para decições múltiplas. Existem duas sintaxes, a saber:

Sintaxe número 1:

```
Case <expressão ou variável> of
<valor 1> : Comando_1;
<valor 2> : Comando_2;
. . .
<valor n> : Comando_n;
End;
```

Observe neste exemplo como fica muito mais fácil trabalhar com a estrutura Case:

```
Program Uso_Caso;
Var X: integer;
      Y: String;
Begin
   Case X of
      1:  Y:='Janeiro';
      2:  Y:='Fevereiro';
      1:  Y:='Março';
      Writeln(Y);
   End;
End.
```

**Quando devemos usar o Begin...End na estrutura Case ?**

**Resposta:** Quando para cada valor, temos várias instruções ou comandos a serem executados. Veja os exemplos a seguir:

```
Case <expressão ou variável> of
<valor 1> : Begin
comando_1;
comando_2;
. . .
End;
<valor 2> : Begin
comando_1;
comando_2;
```

```
. . .
End;
. . .
<valor n> : Begin
comando_1;
comando_2;
. . .
End;
End;
```

A expressão ou variável no comando Case deve ser do tipo simples, normalmente **Char** ou **Integer**. Após a avaliação da expressão, seu valor ou o valor da variável é comparado com os diversos valores discriminados. Se houver algum que satisfaça, o comando subseqüente será executado.

**Uso do ELSE**

Sintaxe número 2:

```
Case <expressão ou variável> of

<valor 1> : Comando_1;
<valor 2> : Comando_2;
. . .
<valor n> : Comando_n;
Else Comando;
End;
```

Neste caso, se o resultado da expressão ou o valor da variável não satisfizer nenhum dos valores discriminados, então o comando que estiver na frente da cláusula **Else** será executado.

Exemplos:

Faça um programa que leia o número do mês e escreva-o por extenso.

```
Program Uso_Case;
Uses CRT;
Var X: integer;
      Y: string;
Begin
   Clrscr;
   Case X of
      1: Y:='Janeiro';
      2: Y:='Fevereiro';
      3: Y:='Março';
      4: Y:='Abril';
      5: Y:='Maio';
      6: Y:='Junho';
      7: Y:='Julho';
      8: Y:='Agosto';
      9: Y:='Setembro';
      10: Y:='Outubro';
      11: Y:='Novembro';
      12: Y:='Dezembro';
Else
      Y: 'Mês inválido !'
End;
Writeln(Y);
Readln;
End.
```

Observe que temos um END; logo após a instrução CASE ser executada.

Observe também o uso do ELSE. Está sendo usado para contrariar caso não seja inserido um valor que não está sendo pesquisado pelo CASE.

E agora, se para cada caso testado, tivéssemos mais de uma instrução ou comando para ser executado ?

Exemplo: Faça um programa que leia o número do mês, escreva-o por extenso e aponte os feriados nacionais ?

```
Program Uso_Case;
Uses CRT;
Var X: integer;
      Y: string;
Begin
   Clrscr;
   Case X of
      1: Begin
         Y:= 'Janeiro';
         Writeln(' Dia 1º - Confraternização');
       End;
      2: Begin
         Y:= 'Fevereiro';
         Writeln('Carnaval');
      End;
      .
      .
      .
      .
Else
      Y: 'Mês inválido !'
End;
Writeln(Y);
Readln;
End.
```

Perceba agora o uso do Begin...End; , pois para cada caso testado, deve-se executar mais que uma linha de comando !

## Testando Intervalos com o uso do Case..Of :

```
 Program Intervalos;
 Var
 x: integer;

begin
  readln(x);
  case x of
   0:             writeln('O valor digitado é igual a zero');
   1..5:          writeln('O valor digitado é maior ou igual a 1 e menor ou igual a 5')
   6..9, 21..29:  writeln('O valor digitado está entre 6 e 9 ou entre 21 e 29');
  else
     writeln('O valor digitado é menor que 0, ou está entre 10 e 20, ou é maior que 29 !')
  end;
end.
```

**Veja este outro exemplo !**

```
Program Exemplo;
Var
  x: char;
 begin
  readln(x);
  case x of
    '0'..'9':  writeln('O valor digitado é um número´);
    'A'..'Z': writeln('O valor digitado é uma letra maiúscula');
    'a'..'z':  writeln('O valor digitado é uma letra minúscula');
    '+', '-', '*', '/':  writeln('O valor digitado é um operador
aritmético');
  else
    writeln('O valor digitado é um outro caracter');
  end;
end.
```

Perceba que o Pascal consegue enxergar no intervalo de 0 a 9 os números 1, 2, 3, 4, 5, 6,7,8 e 9

*'0'..'9':  writeln('O valor digitado é um número´);*

A mesma coisa acontece no intervalo entre A e Z, qualquer letra maiúscula do nosso alfabeto digitada está incluída neste intervalo

*'A'..'Z': writeln('O valor digitado é uma letra maiúscula');*

... e qualquer letra minúscula digitada está incluída neste intervalo

*'a'..'z':  writeln('O valor digitado é uma letra minúscula');*

**Exercícios usando Case**

1. Desenvolva um programa que pergunte um código e de acordo com o valor digitado seja apresentado o cargo correspondente. Caso o usuário digite um código que não esteja na tabela, mostrar uma mensagem de código inválido. Utilize a tabela abaixo:

| Código | Cargo |
|---|---|
| 101 | Digitador |
| 102 | Operador |
| 103 | Programador |
| 104 | Projetista |
| 105 | Analista de Sistemas |
| 106 | Chefe de CPD |

2. Faça um programa que pergunte o nome do aluno, a quantidade de dias na semana e o tipo de curso (**B** para básico, **I** para intermediário e **A** para Avançado). Calcule o valor total com base nas informações abaixo:
   - Caso a opção escolhida for Básico deverá fazer a seguinte conta:
     Valor Total = (Quantidade de dias na semana * 7)*15
   - Caso a opção escolhida for Intermediária deverá fazer a seguinte conta:
     Valor Total = (Quantidade de dias na semana * 8.5)*20
   - Caso a opção escolhida no RadioGroup1 for Avançado deverá fazer a seguinte conta:
     Valor Total = (Quantidade de dias na semana * 10)*25

## Estrutura de Repetição

### While...Do (Enquanto...Faça)

A estrutura **WHILE..DO** permite controlar o número de vezes que uma instrução ou bloco de instruções será executado. Ela difere da instrução Repeat..Until (que veremos no próximo tópico) porque esta só avalia a expressão lógica no final do primeiro Loop, enquanto que a instrução While..Do avalia a expressão lógica antes da primeira interação, isto significa que, eventualmente, pode não ocorrer sequer a primeira interação.

A sintaxe de **WHILE..DO** é:

```
While <expressão_lógica> Do <comando>;
```

ou

```
While <expressão_lógica> Do
Begin
comando_1;
comando_2;
. . .
End;
```

Exemplos:

```
Program Exemplo_1;
Uses CRT;
{Programa exemplo que escreve na tela de 0 até 10}
Var i : Integer;
Begin
  ClrScr;
  i:=0;
  While (i<11) Do
  Begin
     Writeln(i);
```

```
        i:=i+1;
    End
End.
```

**Exercícios usando While Do**

1. Faça um programa que mostre os valores da tabuada do dois de 0 a 10.
2. Desenvolva um programa que apresente os quadrados dos números inteiros de 1 a 10.
3. Elaborar um programa que apresente no final o somatório dos valores pares existentes na faixa de 1 até 20.
4. Faça um programa para exibir os números de 1 a 100.
5. Criar um Calculador de Tabuadas. O usuário deverá digitar o número desejado e o programa deverá fazer a tabuada desse número de zero a 10.
6. Desenvolva um programa que apresente todos os números divisíveis por 4 na faixa de 20 a 60.
7. Elaborar um programa que apresente o resultado da soma dos valores pares situados na faixa numérica de 50 a 80.
8. Faça um programa que escreva na tela os números de 1 até 20, elevados ao cubo.

**Repeat...Until (Repita...Até que)**

Repete um bloco de instruções até que uma certa condição seja satisfeita. Sua sintaxe é:

```
Repeat
Comando_1;
Comando_2;
Comando_3;
. . .
Until (expressão_lógica);
```

Neste caso, todos os comandos entre as palavras reservadas **REPEAT** e **UNTIL** serão executadas, até que a expressão lógica seja verdadeira (TRUE), obviamente, devemos ter o cuidado para que ela venha a ser TRUE em determinado momento, pois caso contrário, teremos um LOOP INFINITO, (o programa fica preso dentro da estrutura Repeat - Until).

***Os comandos múltiplos dentro de um loop REPEAT..UNTIL não necessitam de marcadores BEGIN e END !***

Exemplos:

```
Program Exemplo_1;
Uses CRT;
{Programa exemplo para mostrar o funcionamento da estrutura Repeat Until}
Var i : Integer;
```

```
Begin
  ClrScr;
  i:=1;
  Repeat
     Writeln(i);
     i:=i+1;
  Until i=10;
End.
```

**Diferenças entre Loops While.Do e Repeat..Until**

Num loop WHILE, você coloca a condição no início do loop, e as iterações continuam desde que a condição seja TRUE (verdadeira). Tendo em vista que a condição é avaliada sempre antes de cada iteração, um loop WHILE resulta em nenhuma iteração se a condição for FALSE (falsa) logo no princípio.

Num loop REPEAT UNTIL, a condição vai para o fim do loop, e o processo de repetição continua até que a condição se torne TRUE (verdadeira). Tendo em vista que a condição é avaliada após cada iteração, um loop REPEAT UNTIL sempre executa pelo menos uma iteração.

A escolha entre usar uma ou outra estrutura, dependerá cada aplicação em particular.

Alguns programadores são cautelosos no uso de estruturas de loop que executam automaticamente a primeira iteração automaticamente a primeira iteração antes de avaliar a condição de controle. Esses programadores podem preferir utilizar os loops WHILE, ao invés dos loops REPEAT UNTIL, na maioria dos programas de repetição. Fora isto, o REPEAT UNTIL é vantajoso, desde que você reserve o seu uso para situações nas quais você sempre deseje que seja executada pelo menos uma iteração, como por exemplo no programa abaixo:

Este programa pede a entrada de números pelo teclado e soma os números digitados até que a resposta do usuário for igual de ´S´ ou ´s´.

```
program resposta;
uses crt;
var resp : string;
    nr, soma : real;
begin
  clrscr;
  repeat
    writeln('digite um numero');
    readln(nr);
    soma:= soma + nr;
    writeln('deseja sair (S/s)');
    readln(resp);
  until
   (resp ='S') or (resp = 's');

  writeln(soma:1:2);
  readln;
end.
```

Se fizéssemos este mesmo programa usando a estrutura WHILE, teríamos, necessariamente, que atribuir o valor ´S´ou ´s´ à variável RESP para que o programa entrasse no loop While, pois ao contrário do REPEAT UNTIL, o WHILE somente executa a primeira iteração se a condição for verdadeira . Veja:

```
program resposta;
uses crt;
var resp : string;
    nr, soma : real;
begin
  clrscr;
  resp:='S';
  while (resp = 'S') or (resp = 's') do
  Begin
    writeln('digite um numero');
    readln(nr);
    soma:= soma + nr;
    writeln('deseja continuar (S/s)');
    readln(resp);
  end;

  writeln(soma:1:2);
  readln;
end.
```

**Exercícios usando Repeat / Until**

1. Desenvolva um programa que mostre na tela o quadrado dos números de 1 até 20.
2. Escreva um programa onde os números lidos serão somados até que a soma destes números seja igual ou maior do que 100.

3. Elaborar um programa que possibilite calcular a área de cada cômodo de uma residência (por exemplo: sala, cozinha, banheiro, quartos, etc..). O programa deve solicitar a entrada do nome, da largura e do comprimento de um determinado cômodo, em seguida deverá apresentar a área do cômodo lido e também a mensagem solicitando ao usuário a confirmação de continuar calculando novos cômodos. A operação deve ser repetida até que o usuário responda "NÃO". **Área = largura * comprimento**

## FOR...TO...DO (PARA..ATÉ..FAÇA)

O comando FOR identifica uma variável de controle para o loop e indica uma faixa de valores (ValorInicial TO (até) ValorFinal) que a variável receberá durante as iterações do loop. A variável de controle **deve pertencer a um tipo ordinal** e o ValorInicial e o ValorFinal devem ser compatíveis com esse tipo. Além disso, o For/DO só pode usar um contador simples que **conta de uma em uma unidade**, somando (for..TO..do) ou subtraindo (for DOWNTO do) uma unidade da variável controladora.

*Funciona dessa forma:*

1- No início da execução, a variável de controle recebe o valor do ValorInicial, e a primeira iteração é executada.

2- Antes de cada iteração subseqüente, a variável de controle recebe o próximo valor da faixa de ValorInicial até ValorFinal.

3- O loop termina após a iteração correspondente ao ValorFinal.

**FOR** variável := ValorInicial **TO** ValorFinal **DO**
...... operação;
(aqui valor_final >= valor_inicial)

```
Program Exibe_Numeros;
Uses CRT;
Var  Nr : integer;
Begin
  Clrscr;
  For Nr:= 1 to 10 do
    Writeln(Nr);
  Readln; {pausa}
End.
```

**Será exibido na tela:**

```
1
2
3
4
5
6
7
8
9
10
```

Num loop FOR que usa a cláusula TO, o valor ordinal do ValorInicial deve ser menor que o valor ordinal do ValorFinal. (Senão o loop não terá ação nenhuma). Alternativamente, você pode usar a palavra reservada DOWNTO para projetar um loop FOR que seja decrementado dentro de uma faixa ordinal:

**FOR** variável := ValorInicial **DOWNTO** ValorFinal **DO**
...... operação;
(aqui valor_final <= valor_inicial)

```
Program Exibe_Numeros;
Uses CRT;
Var Nr : integer;
Begin
  Clrscr;
  For Nr:= 10 downto 1 do
    Writeln(Nr);
  Readln; {pausa}
End.
```

**Será exibido na tela:**

```
10
9
8
7
6
5
4
3
2
1
```

**DICAS:**

A variável do For/Do é incrementada ou decrementada automaticamente, portando você nunca deve alterar o valor desta variável dentro do laço do For/Do.

**Exemplo: (Nunca faça !)**

FOR Contador := 50 TO 100

DO Contador := Contador –1; {Você terá um Laço Infinito - Contador é alterado no laço! }

Você não deve alterar o valor final do For/Do durante a sua execução (usualmente toda a alteração do valor final do For/Do será ignorada).

A variável do For/Do deve ser do tipo inteiro (ordinal), sendo que o valor inicial deve ser menor ou igual ao valor final quando usarmos um For-TO-Do, ou, o valor inicial deve ser maior que o valor final quando usarmos um For-DOWNTO-Do.

Certifique-se de que a variável contadora vai realmente chegar até o valor final indicado no For/Do, pois caso no contrário teremos um laço infinito.

Quando você tem várias instruções a serem executadas dentro do laço For/Do, você deve usar o par de BEGIN/END (da mesma forma como ocorre com os outros comandos do Pascal, como por exemplo no IF/THEN/ELSE).

Usando um Laço For/Do você sabe exatamente o número de execuções que seu programa fará.

**Exercícios de For/Do**

1. Elabore um programa que apresente a soma do intervalo entre os números inteiros de 15 a 30.
2. Faça um programa que efetue a leitura sucessiva de valores numéricos e apresente no final o total do somatório dos valores lidos. O programa deve fazer a leitura dos valores enquanto o usuário estiver fornecendo valores positivos. Ou seja, o programa deve parar quando o usuário digitar um valor negativo (menor que zero).

**Estruturas de Dados Homogêneas**

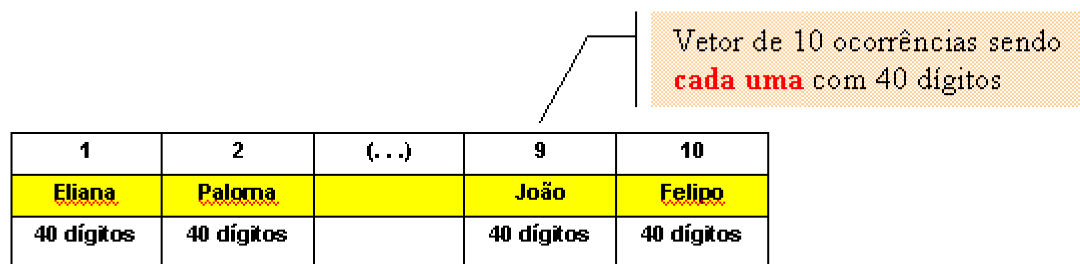
Até então trabalhamos com variáveis simples, que só recebem um dado por vez. A partir deste momento, trataremos sobre variáveis compostas, que são capazes de armazenar mais de um dado ao mesmo tempo.

Vamos aprender a trabalhar com uma técnica de programação que nos permitirá trabalhar com um agrupamento de várias informações dentro de uma mesma variável. Este agrupamento ocorrerá obedecendo sempre o mesmo tipo de dado e por esta razão será chamado de **estrutura de dados homogêneas**. Em nosso próximo assunto estudaremos os agrupamentos de tipos de dados diferentes que por este motivo são chamados de Estruturas de Dados Heterogêneas.

As estruturas de dados homogêneas podem receber diversos nomes: arranjos, vetores, matrizes unidimensionais, tabelas em memória, arrays, variáveis compostas ou variáveis indexadas. Cada autor define com um nome, mas é importante saber todos tem um mesmo significado: são estruturas de dados homogêneas !

**Matrizes Unidimensionais ou Vetores ou Arrays**

Um array é simplesmente um agregado de componentes do mesmo tipo. Imagine um vetor definido para receber 10 nomes, onde cada elemento possui 40 dígitos do tipo caracter:




Como você vê na figura anterior, um vetor é um conjunto de sub-variáveis homogêneas, onde cada uma é intitulada de elemento, e possui as mesmas características do vetor como um todo, à nível de identificador, tipo e tamanho.

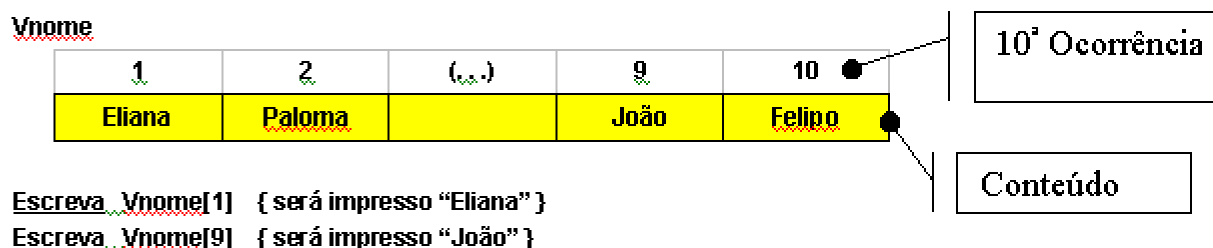
O que difere uma sub-variável de outra é apenas o seu endereço relativo dentro do vetor.

Este endereço relativo é explicitado por um argumento denominado índice , seletor ou subscrito.

O vetor é uma variável unidimensional, pois necessitamos de apenas um índice para nos referenciarmos aos seus diversos elementos.

## Para acessar um dado num vetor o que é necessário fazer?

Muito simples. Informe o valor do índice cujo posicionamento incide na ocorrência que você quer verificar. Caso esta ocorrência exista, será mostrado o conteúdo da célula indicada pelo índice.




### Mas para que servem os Vetores ?

Existem situações nas quais precisamos armazenar uma seqüência de dados semelhantes entre si. Por exemplo: imagine que tenhamos de armazenar 5 números telefônicos para cada cliente de uma determinada empresa. Considerando o que vimos na primeira etapa do curso, teríamos que declarar 5 variáveis literais, cada uma com um identificador próprio.

```
DECLARE FONE1,CARACTER, 10
DECLARE FONE2,CARACTER, 10
DECLARE FONE3,CARACTER, 10
DECLARE FONE4,CARACTER, 10
DECLARE FONE5,CARACTER, 10
```

Como sabemos que cada um dos números telefônicos terá o mesmo tipo e tamanho, poderemos descartar as 5 variáveis acima, e utilizarmos apenas uma variável vetor, com 5 elementos:

```
DECLARE FONES[5],CARACTER, 10
```

Já que sabemos a quantidade de elementos que o vetor irá receber, podemos desenvolver um algoritmo de armazenamento dos dados no vetor. Usaremos, para isto, os comandos que aprendemos até agora

```
program telefone;
uses crt;
var  cont: INTEGER;
     fone: array[1..5] of STRING[10];
begin
 clrscr;
 for cont:=1 to 5 do
 begin
  write('Digite o numero do telefone:  ');
  readln(fone[cont]);
 end;
  clrscr;
  for cont:=1 to 5 do
    writeln('Os numeros dos telefones digitados sao: ', fone[cont]);
  readln;
end.
```

Veja que, pelo exemplo acima, podemos utilizar como índice de um vetor uma outra variável (neste caso, a variável Cont), desde que esta seja do tipo numérica, inteira (sem casas decimais) e positiva (maior que zero).  Não fosse este artifício, teríamos que criar 5 variáveis para receber os telefones e depois escrever uma a uma na ordem de entrada. Veja como ficaria :

Que trabalho, hein !

```
program telefone;
uses crt;
var fone1, fone2, fone3, fone4, fone5 : STRING[10];
begin
  clrscr;
  write('Digite o numero do telefone:  ');
  readln(fone1);
  write('Digite o numero do telefone:  ');
  readln(fone2);
  write('Digite o numero do telefone:  ');
  readln(fone3);
  write('Digite o numero do telefone:  ');
  readln(fone4);
  write('Digite o numero do telefone:  ');
  readln(fone5);
  clrscr;
  writeln('Os numeros dos telefones digitados sao: ');
  writeln(fone1);
  writeln(fone2);
  writeln(fone3);
  writeln(fone4);
  writeln(fone5);
  readln;
end.
```

## Outro Exemplo:

No exemplo abaixo, temos um programa que faz a leitura de 5 nomes de pessoas e mostra estes nomes na tela na ordem em que foram lidos:

```
program nomes;
uses crt;
var i : integer;
   nome: array[1..5] of string[20];
begin
  clrscr;
  for i:=1 to 5 do
  begin
    writeln('digite o ', i , ' º nome');
    readln(nome[i]);
  end;
  clrscr;
  writeln('os nomes digitados foram...');
  for i:= 1 to 5 do
    writeln(nome[i]);
  readln;
end.
```

**Aparecerá na tela :**

```
Digite o 1º nome:
Roberta
Digite o 2º nome:
Giovana
Digite o 3º nome:
Juliana
Digite o 4º nome:
Renata
Digite o 5º nome:
Rogéria

{Limpa a Tela e mostra:}

Os nomes digitados foram:
Roberta
Giovana
Juliana
Renata
Rogéria
```

Perceba que com a técnica de matrizes é possível criar apenas uma variável "Nome" com a capacidade de armazenar 5 valores.

Neste exemplo, estamos utilizando uma **matriz de uma dimensão**, que como visto, pode também ser chamada de **vetor** ou mesmo **array**.

A matriz é chamada de "Nome" e possui tamanho de 1 a 5. Isto significa que poderão ser armazenados em "Nome" até 5 elementos. Perceba que na utilização de variáveis simples existe uma regra: uma variável somente pode conter um valor por vez. No caso das matrizes, poderão armazenar mais de um valor por vez, pois são dimensionadas exatamente para este fim. Desta forma, poderemos manipular uma quantidade maior de informações com pouco trabalho de processamento. Deve-se apenas considerar que com relação à manipulação dos elementos de uma matriz, eles ocorrerão de forma individualizada, pois não é possível efetuar a manipulação de todos os elementos do conjunto ao mesmo tempo.

No caso do exemplo acima, temos uma única variável indexada (a matriz) contendo todos os valores dos 5 nomes. Isto seria representado da seguinte forma:

Nome[1] = Roberta
Nome[2] = Giovana
Nome[3] = Juliana
Nome[4] = Renata
Nome[5] = Rogéria

Observe que o nome é um só, o que muda é a informação indicada entre colchetes. A esta informação dá-se o nome de ***índice***, sendo este o endereço em que o elemento está armazenado. É necessário que fique bem claro que elemento é o conteúdo da matriz, neste caso os nomes das pessoas. No caso de Nome[1] = Roberta, o número 1 é o índice; o endereço cujo elemento Roberta está armazenado.

**Declaração de uma Matriz Unidimensional**

No Pascal a declaração de matrizes unidimensionais é feita através da seguinte forma:

nome da variável: array[ni..nf] of T

onde:

| | |
|---|---|
| **nome da variável** | nome associado às variáveis que se deseja declarar |
| **array of** | são palavras chaves |

| | |
|---|---|
| ni | é o limite inferior do intervalo de variação do índice |
| nf | é o limite superior do intervalo de variação do índice |
| T | é o tipo dos componentes da variável |

No exemplo acima, o vetor nome é declarado da seguinte forma:

Nome : array[1..5] of String[20]

onde:

**NOME** é o nome da variável

**[1..5]** é o intervalo de variação do índice, ou seja irá armazenar 5 locações dentro da variável nome

**STRING[20]** significa que a variável é do tipo texto com um tamanho máximo para armazenar até 20 caracteres.

**A estrutura do tipo Array no Pascal:**

- É uma estrutura homogênea, isto é, formada de elementos do mesmo tipo;
- Todos os elementos da estrutura são igualmente acessíveis, isto quer dizer que o tempo e o tipo de procedimento para acessar qualquer um dos elementos do array são iguais;
- Cada elemento componente desta estrutura tem um nome próprio, que é o nome do vetor seguido do índice.

**Trabalhando dois Vetores utilizando o mesmo índice**

Uma das muitas facetas da informática é a de reaproveitamento de variáveis para tarefas separadas. Um caso típico desses é o de utilização de um mesmo índice em estruturas de vetores diferentes. A medida que percorremos um vetor **A** qualquer com um índice, outro vetor **B** poderá se utilizar deste mesmo índice. Vejamos o caso a seguir.

Ler duas notas N1 e N2 de um grupo de 5 alunos e em seguida calcular a média final de cada aluno.

Veja como ficaria nosso programa:

Teremos que alimentar 2 vetores **N1** e **N2**, cada um com 5 elementos (informados pelo teclado) e guardar, num terceiro vetor (**Media**) o resultado da média as duas notas informadas.

```
Program Notas;
Uses CRT;
var n1 : array[1..5] of real;
     n2 : array[1..5] of real;
     media: array[1..5] of real;
     cont: integer;
Begin

 CLRSCR; {limpa a tela}

 For cont:=1 to 5 do {inserção de elementos nos vetores}
 Begin
   write('Digite a 1ª nota:  ');
   readln(n1[cont]); {alimenta o 1º vetor}
   write('Digite a 2ª nota:  ');
   readln(n2[cont]); {alimenta o 2º vetor}
   media[cont]:=(n1[cont]+n2[cont])/2; {guarda o resultado da conta no 3º vetor}
 End;

 CLRSCR; {limpa a tela}

 For cont:=1 to 5 do {escreve os elemento dos vetores na tela}
   writeln('A media do ', cont, 'º aluno eh:   ', media[cont]:2:2);
  readln;
end.
```

Comparando elementos de um Vetor

Com vetor, a lógica se torna muito simples, pois os dados são digitados e carregados em cada ocorrência do vetor ficando disponível enquanto não for encerrado o processamento. Nesse momento podemos aproveitar para encontrar o maior elemento digitado no vetor:

```
program maior_idade;
uses crt;
var idade : array[1..3] of integer;
     cont  : integer;
     maior: integer;
begin
  clrscr;
  maior:= -1; {a variável maior recebe a menor idade possível !}
  for cont:= 1 to 3 do
  begin
    writeln('digite a idade: ');
    readln(idade[cont]); {alimenta o vetor idade}
    if idade[cont] > maior then {verifica se cada elemento do vetor é maior que a variável maior}
       maior := idade[cont]; {se o elemento for maior que o valor da variável, então maior recebe
o valor do elemento}
 end;
  writeln('A maior idade e: ', maior); {escreve na tela o valor da variável Maior que conterá o
maior dos valores contidos no vetor idade}
  readln;
end.
```

**Ordenação Vetores**

**(Jussara Moreira em Técnicas do Desenvolvimento da Lógica)**

Primeiramente, quero pensar com você o que seria uma ordenação. Podemos afirmar que ordenar é um processo no qual elementos ficarão em uma ordem seqüencial crescente ou decrescente.

Imagine os números:

- 3, 1, 2 (estes dados estão **desordenados**);
- 1, 2, 3 (estes dados estão ordenados **crescentemente**);
- 3, 2, 1 (estes dados estão ordenados **decrescentemente**).

Vamos aprender a ordenar os dados de forma crescente. Existem algumas formas, porém estudaremos duas delas.

Se você tiver um vetor numérico contendo dois elementos (**num[02], numérico, 1**) e quiser que eles fiquem ordenados crescentemente, o que fazer?

- **1° Passo:** Verificar qual dos dois elementos é o maior.

  **Se num[1] > num[2] então (...)**

- **2° Passo:** Se o **primeiro** elemento for **maior** que o **segundo**, precisamos inverter os elementos de posição. Caso contrário, os elementos já estão ordenados e não precisaremos inverter os dados.

```
Se num[1] > num[2] então
   Aux    := num[1]
   Num[1] := num[2]
   Num[2] := Aux
Fim-Se
Escreva num[1] , num[2]
```

Vejamos um segundo caso. Se você tiver três elementos numéricos expostos na seqüência abaixo e quiser ordená-los o que faríamos?

| 1 | 2 | 3 |
|---|---|---|
| 3 | 1 | 2 |

1. Não poderíamos colocar os índices fixos. Como assim? Os valores que ficam valendo como índices de acesso deveriam ser variáveis (mutáveis).

2. No caso em estudo iremos fazer as comparações entre num[1] e num[2], verificando se os elementos já estão em ordem, depois realizaremos a comparação entre os elementos num[2] e num[3], fazendo a mesma comparação, invertendo os elementos caso estes estejam fora da ordem desejada. Com isso temos as seguintes conclusões:

- Estaremos comparando dois elementos ao mesmo tempo, logo, as posições são diferentes [ I ] e [I + 1] ;
- O índice irá até a penúltima posição [n-1];
- Precisamos de uma estrutura de repetição para "varrer" todos os elementos.

```
program ordena1;
uses crt;

var aux: integer;
    int: integer;
    num : array[1..3] of integer;

begin
clrscr;
for int:= 1 to 3 do
 readln(num[int]);

for int:= 1 to 2 do
  begin
    if num[int] > num[int+1] then
      aux:= num[int];
      num[int]:= num[int+1];
      num[int+1]:= aux;
  end;
 clrscr;
for int:= 1 to 3 do
 writeln(num[int]);
readln;
end.
```

Pronto! **Para os elementos dispostos na forma como citados anteriormente, este algoritmo funcionará**. Porém, este programa testou apenas uma seqüência de números, conseguindo colocar em ordem. Se você alterar a seqüência das informações para:

| 1 | 2 | 3 |
|---|---|---|
| 3 | 2 | 1 |

e utilizar-se do mesmo algoritmo, irá perceber que houve uma falha, pois os números após a execução do processo ficaram organizados assim:

| 1 | 2 | 3 |
|---|---|---|
| 2 | 1 | 3 |

Como garantir que os dados ficarão ordenados? Simplesmente, mandando executar esta rotina de comparação de elementos tantas quantas vezes sejam necessárias até nenhum elemento ser trocado de posição, caracterizando assim, a ordenação do vetor. Vamos nos utilizar outra vez de uma chave de controle batizando-a como **mudou**. Ela definirá se houve ou não uma troca de elementos. Chegamos agora a uma das estruturas de ordenação.

```
program ordena;
uses crt;

var mudou : char;
   aux: integer;
   int: integer;
   num : array[1..3] of integer;

begin
clrscr;
for int:= 1 to 3 do
 readln(num[int]);
repeat
mudou:='n';
for int:= 1 to 2 do
  begin
   if num[int] > num[int+1] then
    begin
     aux:= num[int];
     num[int]:= num[int+1];
     num[int+1]:= aux;
     mudou:='s';
    end;
 end;
until mudou = 'n';

 clrscr;
for int:= 1 to 3 do
 writeln(num[int]);
readln;
end.
```

Pronto! Após a execução deste algoritmo, teremos nossos dados organizados de forma crescente.

Outro algoritmo que utilizamos para ordenar os vetores é o exposto a seguir. Ele é bem menor em quantidade de código e é um dos mais utilizados entre os desenvolvedores de softwares.

Observar que **n-1** é igual ao valor do índice da **penúltima posição**, e **n** é o valor do índice da **última posição**.

```
program ordena;
uses crt;

var x: integer;
    i: integer;
    j: integer;
    num : array[1..3] of integer;

begin
clrscr;
x:=0;
for i:= 1 to 3 do
 readln(num[i]);

for i:= 1 to 2 do
begin
  for j:=i+1 to 3 do
  begin
    if num[i] > num[j] then
     begin
       x:=num[i];
       num[i]:=num[j];
       num[j]:=x;
     end;
  end;
 end;
 readln;
for i:= 1 to 3 do
 writeln(num[i]);
readln;
end.
```

**Métodos de Pesquisa em uma Matriz**

*" Quando se trabalha com matrizes, elas poderão gerar grandes tabelas, dificultando localizar um determinado elemento de forma rápida. Imagine uma matriz possuindo 4000 elementos (4000 nomes de pessoas). Será que você conseguiria encontrar rapidamente um elemento desejado de forma manual, mesmo estando a lista de nomes em ordem alfabética? . Certamente que não. Para solucionar este tipo de problema, você pode fazer pesquisas em matrizes com o uso de programação. Serão apresentados dois métodos para efetuar pesquisa em uma matriz, sendo o primeiro o método seqüencial e o segundo o método de pesquisa binária." (Manzano)*

**Método de Pesquisa Seqüencial ou Linear**

Este talvez seja o método de busca mais simples, no qual cada item do vetor é examinado por vez e comparado ao item que se está procurando, até ocorrer uma coincidência. Este método de pesquisa é lento, porém eficiente nos casos em que uma matriz encontra-se com seus elementos desordenados.

**Exemplo de Pesquisa Seqüencial:**

Vamos criar um programa que estabeleça a entrada de 5 nomes em um vetor em seguida faça a pesquisa dos nomes solicitados:

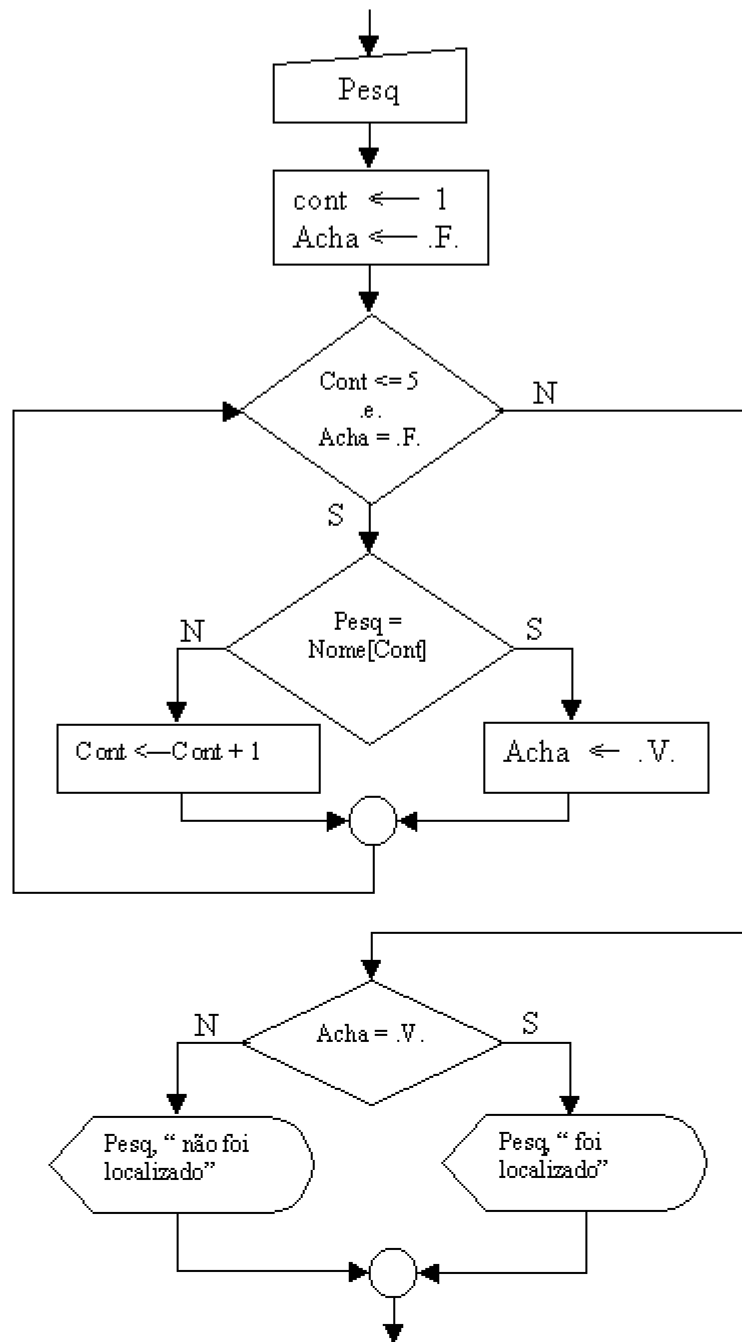
Algoritmo:

1- Iniciar um contador e pedir a leitura de 5 nomes

2- Criar um looping que efetue a pesquisa enquanto o usuário assim desejar. Durante a fase de pesquisa, deverá ser solicitada a informação a ser pesquisada. Essa informação deverá ser comparada com o primeiro elemento; sendo igual mostra, caso contrário, avança para o próximo. Se não achar em toda a lista, informar que não existe o elemento pesquisado; se existir deverá mostrá-lo.

3- Encerrar a pesquisa quando desejado.

***Veja o Fluxograma:***




***Veja o Programa:***

```
program busca_sequenc;
uses crt;
var nome : array[1..5] of string[20];
    cont: integer;
    pesq: string;
    resp: string;
    acha: boolean;
begin
 clrscr;

{entrada de dados no vetor}
 for cont:= 1 to 5 do
  Begin
     writeln('Digite o ', cont, 'º nome');
     readln(nome[cont]);
  end;

 {inicio da rotina de pesquisa}
  resp:= 'sim';
  while resp = 'sim' do
  Begin
     writeln('Digite o nome a ser pesquisado');
     readln(pesq);
     cont:=1;
     acha:=false;
     while (cont<=5) and (acha=false) do
     Begin
       if pesq = (nome[cont]) then
          acha:= true
       else
          cont:=cont + 1;
     end;
     if acha = true then
         writeln(pesq,'  foi encontrado na posicao  ', cont)
     else
         writeln(pesq, '  nao foi encontrado');
 {fim da rotina de pesquisa}

      writeln('Deseja continuar ?');
      readln(resp);
   end;
end.
```

## Método de Pesquisa Binária (Manzano)

*"O método de pesquisa binária é em média mais rápido do que o seqüencial, porém exige que a matriz esteja previamente classificada, pois este método "divide" a lista em duas partes e "procura" saber se a informação a ser pesquisada*

*está acima ou abaixo da linha de divisão. Se estiver acima, por exemplo, toda a metade abaixo é desprezada. Em seguida, se a informação não foi encontrada, é novamente dividida em duas partes, e pergunta se aquela informação está acima ou abaixo, e assim vai sendo executada até encontrar ou não a informação pesquisada. Pelo fato de ir dividindo sempre em duas partes o volume de dados é que o método recebe o nome de pesquisa binária. Para exemplificar a utilização deste tipo de pesquisa, imagine a seguinte tabela:"*

| Índice | Nomes |
|---|---|
| 1 | André |
| 2 | Carlos |
| 3 | Frederico |
| 4 | Golias |
| 5 | Sílvia |
| 6 | Sílvio |
| 7 | Waldir |

A tabela está representando uma matriz do tipo vetor com 7 elementos. Deseja-se neste momento efetuar uma pesquisa de um dos seus elementos. No caso, será escolhido o elemento Wladir, sendo este o último elemento da tabela.

O processo binário consiste em pegar o número de registro e dividir por dois. Sendo assim: a quantidade de elementos do vetor dividido por 2:

**7 div 2 = 3** - o que nos interessa é somente o valor do quociente inteiro. Então a tabela fica dividida em duas partes como em seguida:

Primeira parte após a primeira divisão:

| Índice | Nomes |
|---|---|
| 1 | André |
| 2 | Carlos |
| 3 | Frederico |

Segunda parte após a primeira divisão:

| Índice | Nomes |
|---|---|
| 4 | Golias |
| 5 | Sílvia |
| 6 | Sílvio |
| 7 | Waldir |

Estando a tabela dividida em duas partes, deverá ser verificado se a informação Waldir está na primeira ou na segunda parte. Detecta que Waldir está na segunda parte. Desta forma, despreza-se a primeira e divide-se em duas partes novamente a segunda parte da tabela. Como são 4 elemento, divide-se por 2 e resultam duas tabelas, cada uma com dois elementos:

**4 div 2 = 2**

Veja as tabelas:

Primeira parte após a segunda divisão:

| Índice | Nomes |
|---|---|
| 4 | Golias |
| 5 | Sílvia |

Segunda parte após a segunda divisão:

| Índice | Nomes |
|---|---|
| 6 | Sílvio |
| 7 | Waldir |

Após a segunda divisão, verifica-se em qual das partes Wladir está situado. Veja que está na segunda parte; despreza-se a primeira e divide-se a segunda parte novamente por dois, sobrando um elemento em cada parte:

Primeira parte da terceira divisão

| Índice | Nomes |
|---|---|
| 6 | Sílvio |

Segunda parte da terceira divisão

| Índice | Nomes |
|---|---|
| 7 | Waldir |

Após a terceira divisão, Waldir é encontrado na segunda parte da tabela !

Se estivesse sendo pesquisado o elemento Washington, este não seria apresentado por não existir.

**Exemplo de Pesquisa Binária:**

Vamos criar um programa que estabeleça a entrada de 5 nomes em um vetor em seguida faça a pesquisa dos nomes solicitados:

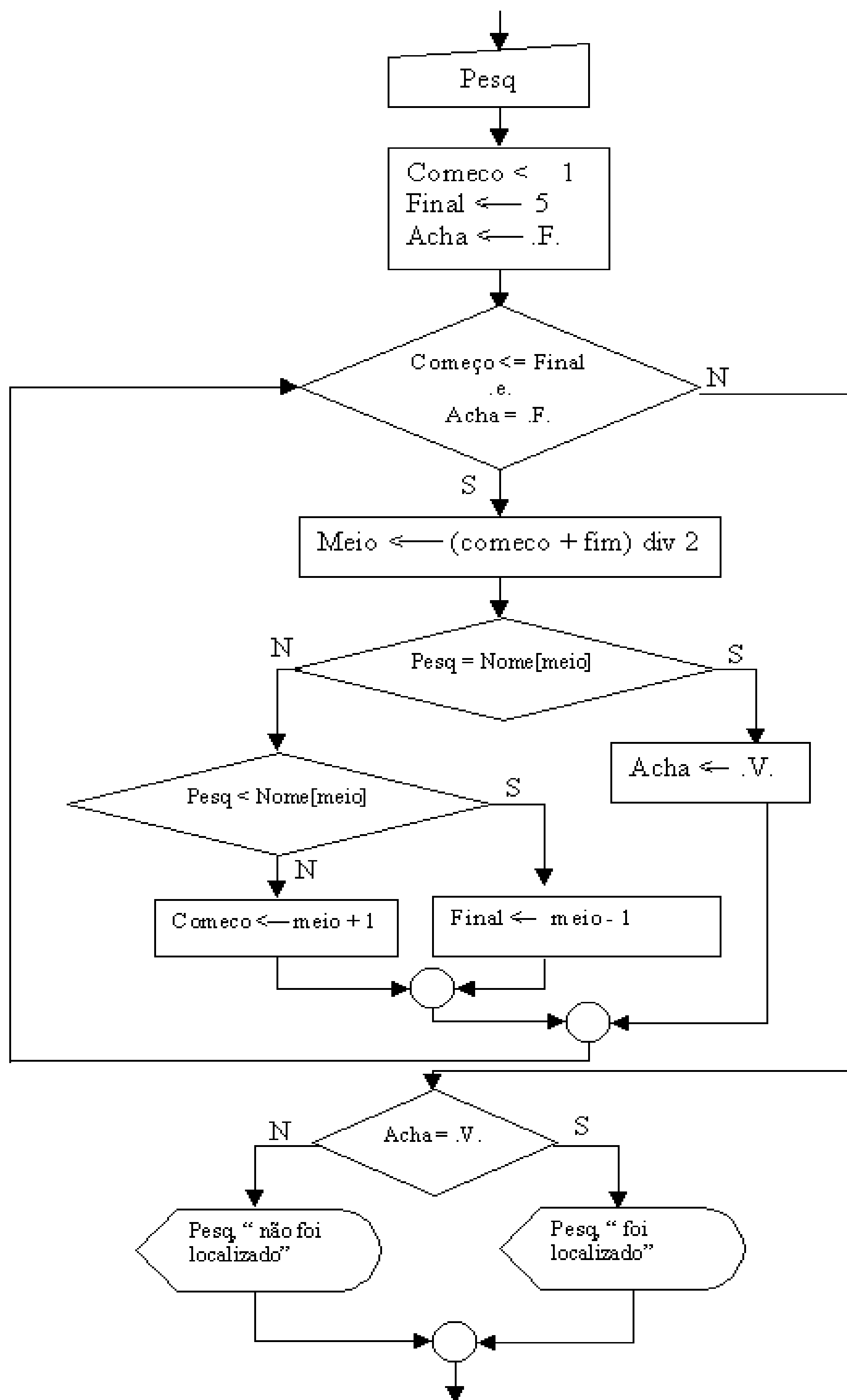
Algoritmo:

1- Iniciar um contador e pedir a leitura de 5 nomes

2- Criar um *looping* que efetue a pesquisa enquanto o usuário assim desejar. Durante a fase de pesquisa, deverá ser solicitada a informação a ser pesquisada. Essa informação deverá ser comparada utilizando-se o método de **pesquisa binária**. Sendo igual mostra, caso contrário, avança para o próximo. Se não achar em toda a lista, informar que não existe o elemento pesquisado; se existir deverá mostrá-lo.

3- Encerrar a pesquisa quando desejado.

***Veja o Fluxograma:***




***Veja o Programa:***

```
program busca_binaria;
uses crt;
var nome : array[1..5] of string[20];
    i, j: integer;
    x: string;
    comeco, meio, fim: integer;
    pesq: string;
    resp: string;
    acha: boolean;
begin
 clrscr;

 {entrada de dados no vetor}
 for i:= 1 to 5 do
  Begin
     writeln('Digite o ', i, '° nome');
     readln(nome[i]);
  end;

  {Ordenacao}
  for i:= 1 to 4 do
  begin
    for j:=i+1 to 5 do
    begin
      if nome[i] > nome[j] then
      Begin
        x:=nome[i];
        nome[i]:=nome[j];
        nome[j]:=x;
     end;
    end;
  end;

  {inicio da rotina de pesquisa}
  resp:= 'sim';
  while resp = 'sim' do
  Begin
     writeln('Digite o nome a ser pesquisado');
     readln(pesq);

     comeco:=1;
     fim:=5;
     acha:=false;

     while (comeco<=fim) and (acha=false) do
     Begin
       meio:=(comeco+fim) div 2;
       if pesq=nome[meio] then
          acha := true
       else
          if pesq < nome[meio] then
            fim:=meio-1
          else
            comeco:= meio + 1
      end;

      if acha = true then
         writeln(pesq,'  foi encontrado na posicao  ', meio)
      else
         writeln(pesq, '  nao foi encontrado');

      {fim da rotina de pesquisa}
      writeln('Deseja continuar ?');
```

**Qual dos dois algoritmos é melhor?**

Não se pode dizer que um é melhor do que o outro. Depende ... Há vários pontos que se deve ter em consideração :

1. Se a quantidade de elementos é pequena (inferior a 100 ou coisa parecida) não vale a pena fazer pesquisas binárias porque o computador é muito rápido para varrer uma coleção de 100 elementos.
2. O algoritmo de pesquisa binária assume que o *array* está ordenado. Ordenar um *array* também tem um custo (quantas comparações são necessárias para ordenar um *array* de dimensão n utilizando o algoritmo da aula passada?)
3. Se for para fazer uma só pesquisa, não vale a pena ordenar o *array*. Por outro lado, se pretendermos fazer muitos pesquisas, o esforço da ordenação já poderá valer a pena.

O estudo de algoritmos e de estrutura de dados é fundamental para se fazer programas eficientes. Por exemplo, os motores de pesquisa da Web (ex: google) utilizam algoritmos semelhantes aos da pesquisa binária de modo a descobrir rapidamente quais são as web Pages que contêm uma determinada palavra. A pesquisa binária é efetuada numa espécie de array gigantesco contendo todas as palavras que existem na Web. Obviamente que eles não fazem uma pesquisa seqüencial. Se assim fosse, as respostas nossas pesquisas demorariam 1 mês em vez de 0.2 segundos!

**Bibliografia**

SCHILDT, HERBERT **Turbo Pascal Avançado**. São Paulo: McGraw-Hill, 1988.

***Elaboração***: Profª. Eliana Cristina Nogueira Barion
**Adaptação:** Profª. Ana Cláudia Câmara Pereira