APRENDENDO A PROGRAMAR EM

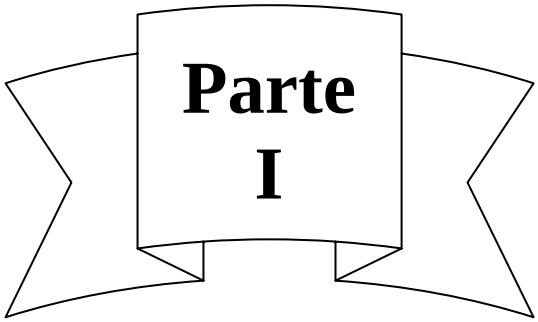
ALFREDO BOENTE

PASCAL

TÉCNICAS DE PROGRAMAÇÃO

Instalação do Turbo Pascal 7.0
Estruturas de dados complexas
Tabela com erros de compilação
Tabela ASCII
Mais de 100 exercícios

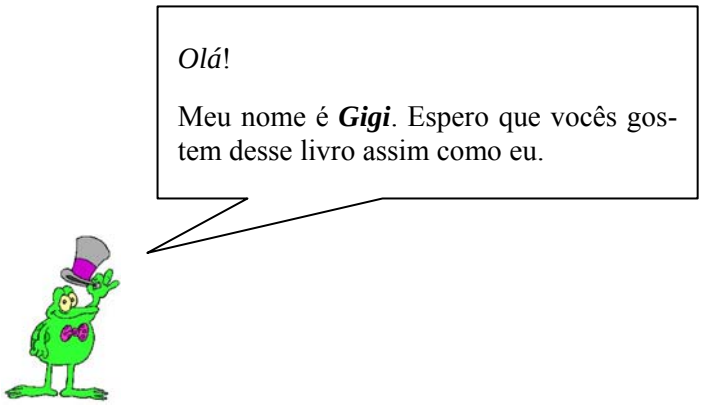
BRASPORT



Conhecendo
A Linguagem Pascal

1
CONCEITOS BÁSICOS DA LINGUAGEM PASCAL

Antes mesmo de falarmos na linguagem de programação Pascal, gostaria de apresentar-lhes o Gigi, um sapinho bem simpático que tem por objetivo chamar sua atenção para os momentos de maior importância deste livro.



A linguagem de programação Pascal é uma linguagem de alto-nível, desenvolvida pelo Prof. Niklaus Wirth da Universidade de Zuric – Suíça, no ano de 1971. A linguagem foi batizada com esse nome em homenagem ao famoso filósofo e matemático francês do século XVII ***Blaise Pascal***.

Hoje em dia existem diversos compiladores para compilares e rodarem um programa Pascal (ZIM, Turbo Pascal etc.) Todos os exemplos abordados neste livro foram compilados no compilador Turbo Pascal 7.0 da Borland Corporation.

Uma das facilidades da linguagem de programação Pascal é fornecer ao iniciante a possibilidade de desenvolver algoritmos computacionais, sistemáticos, de modo facilmente compreensíveis segundo a semelhança existente na notação adotada na escrita de algoritmos computacionais e um programa de computador escrito em Pascal.

A Estrutura Básica de Um Programa Pascal

Um programa desenvolvido na linguagem de programação Pascal consiste no embasamento lógico adotado na confecção do algoritmo computacional que será utilizado como referência para o mesmo. Também, como toda linguagem de programação, o Pascal possui suas próprias regras de sintaxe, com origem na língua Inglesa, para escrever as linhas de programação de um programa de computador.

Um computador não pode entender nem tão pouco executar instruções em linguagens de alto nível. Ele só entende linguagem de máquina (0 e 1). Desta forma, os programas em linguagens de alto nível devem ser traduzidos antes de serem executados pelo computador. Basicamente existem dois (2) tipos de programa tradutor: o interpretador e o compilador; Os dois aceitam como entrada um programa em linguagem de alto nível (fonte) e produzem como saída um programa em linguagem de máquina (objeto). A diferença entre eles está na forma de executar a tarefa de tradução. O interpretador traduz para a linguagem de máquina e roda uma linha por vez, até que todo programa seja executado. Já o compilador traduz para a linguagem de máquina todo o programa fonte e, desde que não apresente nenhum tipo de erro, ele é então executado.

A linguagem de programação Pascal é tradicionalmente compilada e, como já foi dito anteriormente, procuraremos utilizar o compilador Turbo Pascal 7.0 para isto, pois ele facilita todo esse processo, por possuir, na verdade, um editor de textos próprio, um compilador propriamente dito e também um linkeditor.

Na realidade, o Turbo Pascal 7.0 vai muito além disso, pois ele possui inúmeros procedimentos e funções, bem a mais do que aquelas existentes no padrão da linguagem de programação Pascal.

ELEMENTOS BÁSICOS

Todo programa em Pascal é subdividido em três (3) grandes áreas:

- cabeçalho do programa
- área de declarações
- corpo do programa

Na definição padrão da linguagem de programação Pascal, o Cabeçalho do programa é obrigatório, no entanto, no Turbo Pascal 7.0 ele é opcional. A área de declarações é subdividida em seis sub-áreas, a saber:

- Label
- Const
- Type
- Var
- Procedures
- Functions

Na subárea Label, devemos declarar todos os labels que forem utilizados no corpo do programa. Os labels são utilizados em conjunto com a instrução goto. Isso o que eu particularmente, não recomendo para nenhum programa de computador desenvolvido.

Na subárea Const, definimos todas as constantes que iremos utilizar em nossos programas de computador.

O Turbo Pascal 7.0 tem basicamente 6 tipos de variáveis pré-definidas a saber: ***Integer, Real, Byte, Boolean, Char e String***. Contudo, podemos definir novos tipos de variáveis através da subárea Type onde, na realidade, fazemos a definição de tipos inexistentes.

Todas as variáveis utilizadas no programa devem ser devidamente declaradas na subárea Var, pois a alocação de espaço de memória para as variáveis é feita durante o processo de compilação.

Na subárea Procedure, podemos definir todas as sub-rotinas necessárias para nossos programas. Elas são chamadas durante o programa através de seus respectivos nomes.

Na subárea Function podemos definir novas funções que depois poderemos utilizar no programa embora o Turbo Pascal 7.0 possua inúmeras funções pré-definidas.

Todas as sub-áreas só são obrigatórias caso tenhamos a necessidade de utiliza-las.

Finalmente, existe a possibilidade de usarmos a declaração USES, que nos permite utilizar UNITS que nada mais são do que bibliotecas de funções e procedures já pré-definidas.

Observe atentamente a estrutura de um programa Pascal abaixo:

```
Program <nome do programa>;
Uses
        <Biblioteca>;
Label
        { Definição dos labels necessários }
Const
        { Definição das constantes do programa }
Type
        { Definição de tipos inexistentes no Pascal }
Var
        { Declaração de todas as variáveis necessárias para o programa }
Procedure <Nome>;
Begin
        { Corpo do procedimento }
End;
```

```
Function <Nome> : <Tipo>;
Begin
        { Corpo da função }
End;
{ Programa  Principal }
Begin
        { Corpo do Programa Principal }
End.
```

1. Quem foi o criador da linguagem de programação Pascal?
2. Explique a origem da Linguagem de Programação Pascal.
3. Em quantas partes se divide um programa em Pascal?
4. Qual a função da sub-área Type?
5. Os comandos Begin e End. do Pascal são utilizados para que?
6. Qual a função da sub-área Const?
7. Para que serve o USES?
8. Diferencie compilador e interpretador.
9. Quem foi Blaise Pascal?
10. O que são UNITS?

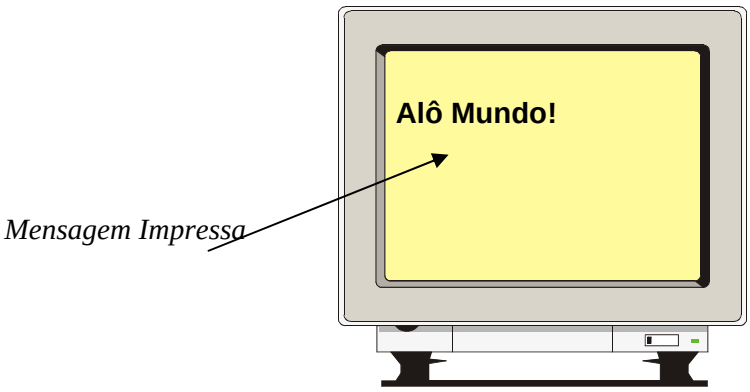
2
SAÍDA DE DADOS

Na verdade, quando falamos em saída de dados, queremos dizer realmente, saída de informações, pois, segundo os conceitos básicos de processamento de dados, tudo aquilo que é inserido no computador, através de um INPUT, é dito dado e, conseqüentemente, tudo aquilo que sai, é dito informação.

Observe um exemplo em Pascal do clássico programa Alô Mundo:

```
Program AloMundo;
Uses
        Crt;
{ Representa uma linha de comentário }
   { Corpo Principal do Programa }
Begin
        Clrscr;     { Comando para linpar a tela do computador }
        Write (´Alô Mundo!´);     { Comando usado para Output }
        Readkey;     { Comando para efetuar uma pausa temporária }
End.
```

Aparecerá então na tela do seu computador a mensagem “Alô Mundo!”, como você pode observar a seguir:



COMANDOS DE SAÍDA – Write e Writeln

O Write e o Writeln são comandos utilizados para saída de informações. Sua utilização equipara-se a instrução Escreva, Exiba, Imprima e Mostre do Pseudo Código. Observe:

```
Program Idade;
Uses
        Crt;
Begin
        Clrscr;
        Write (´Tenho 25 anos de idade´);
        Readkey;
End.
```

Inicialmente, devemos saber que o ponto-e-vírgula é utilizado como um separador de comandos na linguagem de programação Pascal.

A característica do comando Write é escrever uma certa mensagem permanecendo assim, com o cursor na mesma linha da mensagem impressa. Caso você resolvesse substituir a linha de comando: ***Write(´Tenho 25 anos de idade´);*** por **Writeln(´Tenho 25 anos de idade´);** a diferença básica estaria na posição do cursor após a impressão da mensagem, pois com o Writeln o cursor é posicionado na próxima linha disponível. O complemento **ln** utilizado no comando Write, representa *line* – linha.

Também, como já foi comentado anteriormente, o comando Clrscr foi utilizado com o intuito de limpar a tela do computador, e o comando Readkey, foi utilizado para permitir ao usuário uma pausa temporária na execução de um determinado programa. Segue outro exemplo:

```
Program Agora;
Uses
        Crt;
Begin
        Clrscr;
        Gotoxy(10, 5);     { Posicionamento de coluna e linha no vídeo }
        Write (´Tenho´, idade, ´ anos de idade´);
        Readkey;
End.
```

A linha Uses Crt informa ao compilador que ele deve incluir a biblioteca Crt no seu programa. Nesta biblioteca/Unit existem definições para comandos de I/O (entrada e saída).

O uso das chaves representa na linguagem de programação Pascal uma linha de comentário. Sabe-se que toda linha de comentário não oferece efeito visual. O símbolo (* Comentário *), também especifica uma linha de comentário em Pascal.

Nota:

Foi utilizado o comando gotoxy(COL, LIN) para posicionar o cursor em um ponto de coluna e linha na tela do computador.

PALAVRAS-RESERVADAS

Cada palavra indicada, na verdade, especifica uma tarefa ou uma especificação diferente em forma de ***sentença***, ***comando*** ou ***instrução***. Essas palavras servem para indicar basicamente ações a serem executadas pelo seu computador.

As palavras-reservadas Const, Var, Program, Write, Read, Begin, End, Type, If, For, While, Repeat, Until, Integer, Label, Procedure, Function, String, Array, Then, Else, Case dentre outras, não podem ser utilizadas, por exemplo, para a criação do nome de variáveis de um programa.

O programa exemplo abaixo mostra a exibição de uma certa mensagem redirecionada tanto para a tela do computador quanto para o spool de impressora.

```
Programa TelaImpressora;
Uses
        Crt;
Begin
        Clrscr;
        Write(´Mensagem Teste...´);
        Write(lst,´Mensagem Teste...´);
End.
```

O parâmetro **lst** utilizado para redirecionar a saída para impressora, é de uso específico para Outputs via printer. Deve ser sempre utilizado no comando Write ou Writeln, antes da mensagem propriamente dita.

Exercícios

1- Qual a diferença do uso do comando Write e Writeln?

2- Para que serve o comando Clrscr?

3- Qual a função do comando Gotoxy?

4- Qual o comando devo utilizar para imprimir na impressora a seguinte mensagem: TESTE DE IMPRESSÃO?

5- Além de utilizarmos comentários através da abertura e fechamento de chaves, qual a outra forma de fazê-lo?

6- O que são e para que servem as chamadas palavras-reservadas?

7- Escreva um programa que imprima em duas linhas o seu nome e seu sobrenome.

8- Qual a função do ponto-e-vírgula na linguagem de programação Pascal?

3
TIPOS DE DADOS

Os dados são representados pelas informações que serão processadas pelo computador. Estas informações são caracterizadas por dados numéricos, caracteres ou lógicos.

Os chamados dados numéricos, podem ser inteiros, número positivo ou negativo, sem o uso de casas decimais ou reais, número positivo ou negativo, com o uso de casas decimais. Como exemplo tem-se: 24, 0, -5, entre outros.

Os dados caracteres, podem ser representado por um único caracter ou por um conjunto de caracteres, o qual nomeamos de string. Como exemplo tem-se: “JUAN”, “Karine Mara”, “Rua Alfa, nº 66”, “474”, “Z”, entre outros.

Os dados que são conhecidos como lógicos são também chamados de dados boleano por apresentarem apenas dois valores possíveis:

Verdadeiro (true) ou **Falso** (false).

VARIÁVEIS

Uma variável nada mais é do que um endereço de memória que é reservado para armazenar dados do seu programa. Procure utilizar, sempre que possível, as chamadas variáveis significativas, pois seu nome significa, na íntegra, o que elas armazenam (referem-se).

Exemplos:

nome, idade, altura, endereço, data_nasc, salário, cargo etc.

TIPOS DE VARIÁVEIS

TIPO	INTERVALO	TAMANHO
Integer	-32768 a 32767	2 bytes
Real	$2.9*10^{-39}$ a $1.7*10^{38}$	6 bytes
Char	caracteres do teclado	1 byte
Boolean	TRUE e FALSE	-
String	Cadeia de caracteres	até 256 bytes

Cada tipo de dado é associado a uma determinada variável a fim de suprir a necessidade real do programa de computador a ser desenvolvido. Logo, você deve estar bastante atento a este simples porém valioso detalhe.

VARIÁVEL INTEIRA

É uma variável numérica sem o uso do ponto decimal. No exemplo a seguir, iremos demonstrar como usar variáveis do tipo INTEIRA (*Integer*).

```
Program Exemp;
Uses
    Crt;
Var
```

```
        Num : Integer;
Begin
        Num := 2;     { O símbolo utilizado representa o operador de atribuição }
        Write(´Este é o nº ´, Num);
End.
```

Outro exemplo:

```
Programa ExempOutro;
Uses
        Crt;
Var
        Num1, Num2 : Integer;
Begin
        Num2 := 4;
        Num1 := Num2;
        Write(Num1, Num2);
End.
```

Utilizamos o símbolo de := que para a linguagem Pascal é interpretado como atribuição (← no Algoritmo).

VARIÁVEL REAL

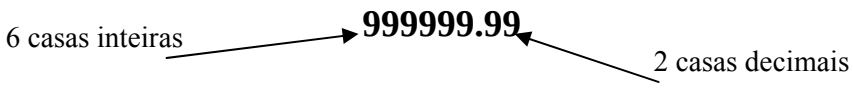
Como já foi comentado anteriormente, as variáveis reais são aquelas que apresentam o uso de casas decimais (valores fracionários). Observe atentamente o exemplo a seguir:

```
Programa ExempReal;
Uses
        Crt;
Var
        Salario : Real;
Begin
        Salario := 965.56;
        Write(´O Salário e: ´, Salario);
End.
```

Aqui utilizamos o ponto para representar o ponto decimal de um número e iremos utilizar a vírgula para separar as casas de milhar. Funciona da mesma forma que a notação Inglesa. Podemos ainda usar os chamados formatadores de casas inteiras e decimais para números reais. Veja o exemplo a seguir:

```
Programa ExempReal;
Uses
        Crt;
Var
        Numero : Real;
Begin
        Numero := 123456.7890;
        Write(´O numero e: ´, Numero:9:2);
End.
```

Aqui o número 9 representa a quantidade de dígitos formatados (inclusive o próprio ponto decimal) e 2 representa a quantidade de casas decimais que serão assumidas pelo número em questão. Ficaria então assumida a seguinte máscara de formatação:



VARIÁVEL CARACTER

No exemplo a seguir, será usada a palavra reservada char para representar o tipo de variável caracter (ocupa 1 byte).

```
Program ExemploPratico1;
Uses
        Crt;
Var
        Letra : Char;
Begin
        Clrscr;
        Letra := 'a';
        Writeln(Letra, ´ e a primeira letra do alfabeto´);
        Readkey;
End.
```

VARIÁVEL CADEIA DE CARACTERES

A variável que representa uma cadeia de caracteres também é chamada de String. Uma variável String ocupa n bytes na memória, dependendo exclusivamente do tamanho definido para ela pelo programador. Observe atentamente o exemplo a seguir:

```
Program XYZ;
Uses
        Crt;
Var
        Nome : String[30];    { a string apresenta 30 caracteres ou posições }
Begin
        Clrscr;
        Nome := ´Kilmer´;
        Gotoxy(2,4);
        Write(´Ola ´, Nome);
        Readkey;
End.
```

VARIÁVEL LÓGICA

A variável lógica é aquela que referencia apenas dois valores possíveis: Verdadeiro ou Falso. Na linguagem de programação Pascal uma variável lógica pe especificada pelo tipo **Boolean**. Observe atentamente a sintaxe de utilização de uma variável lógica:

Var

Achei : **Boolean**;

Certo : **Boolean**;

VARIÁVEL PONTEIRO

Uma variável ponteiro ou apontadora é aquela que ao invés de armazenar um certo conteúdo de dado, guarda em seu espaço de memória, o endereço de memória de outra variável que normalmente apresenta um dado a ser manipulado pelo programa.

Na declaração de uma variável ponteiro ou apontadora em Pascal, devemos utilizar, o símbolo ↑ que indica que uma variável é vista como um ponteiro.

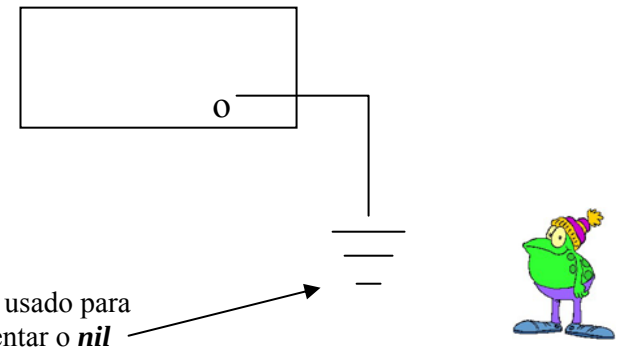
Exemplo:

Var

Pointer : ↑ Terra;

Acima, Pointer é um ponteiro para o tipo Terra. No início do programa o valor de Pointer é indefinido. Em Pascal existe uma constante do tipo ponteiro chamada ***nil*** que não aponta para nenhum lugar. Veja:

Pointer := ***nil***;



A variável dinâmica do tipo Terra é criada através do operador ***new***. Assim, é gerado em Pascal o seguinte procedimento padrão:

```
    new (Pointer);
```

Assim, é gerada uma variável do tipo Terra cujo endereço de memória é alocado para o conteúdo da variável Pointer. Na verdade, quando utilizamos ponteiros ou apontadores, utilizamos a simulação da técnica de gerenciamento de memória denominada, em arquitetura de computadores e sistemas operacionais, como endereçamento indireto.

VARIÁVEIS LOCAIS x VARIÁVEIS GLOBAIS

Uma variável local é aquela declarada dentro do corpo de um determinado procedimento ou função e somente pode ser utilizada por aquela e nenhuma outra mais. Já uma variável global, que poderá ser utilizada por todos os procedimentos e funções existentes em seu programa, é declarada na área de declaração de variáveis do Pascal. Num programa, podem ser apresentadas tanto variáveis locais quanto variáveis globais.

Mais tarde quando estivermos fazendo o estudo da técnica de modularização, no capítulo 8, iremos vivenciar como isso efetivamente ocorre.

CONSTANTES

Uma constante representa um valor fixo, constante, durante todo o processamento de um certo programa. Em Pascal, as constantes são definidas na área especial de declaração de constantes do pascal ***Const***. Observe o programa abaixo cuidadosamente:

```
Program ABC;
Uses
    Crt;
Const
```

```
        PI = 3.14;
Var
        Raio, Area : Real;
Begin
        Clrscr;
        Raio := 4;
        Area := PI * Raio * Raio;
        Write(´Area = ´, Area);
        Delay (500);
End.
```

No nosso programa exemplo utilizamos o commando ***Delay*** que serve para criar um temporizador onde seu computador fica em estado de espera por um dado período fixado como parâmetro do próprio comando. Dependendo do processador, 1000 representa, mais ou menos, 1 minuto.

No programa citado anteriormente não mencionamos nenhuma constante do tipo literal. Para termos essa visão acompanhe atentamente as linhas de código do programa apresentado a seguir:

```
Program Atento;
Uses
        Crt;
Const
        EU = ´Romulo Miranda Pires´;
Begin
        Clrscr;
```

```
        Writeln(´Comunico a quem a turma X que todos foram aprovados
                com louvor´);
        Writeln(´Desejo sucesso profissional a todos os formandos de 2003´);
        Writeln(´Assinado: ´, EU);
        Readkey;
End.
```

Neste exemplo, foi definido EU como constante cujo valor atribuído é "Rômulo Miranda Pires". Logo, toda vez que no programa for referenciada a constante EU, na verdade, será referenciado como conteúdo "Romulo Miranda Pires".

Exercícios

1- Para que utilizamos uma variável?

2- Cite três nomes de variáveis válidas.

3- Diferencie Variável Local e Variável Global.

4- Como declarar uma variável do tipo ponteiro ou apontadora?

5- Como declarar uma constante num programa em Pascal?

6- Quais são os tipos primitivos de variáveis?

7- Diferencie variável String e variável Caracter.

8- Quais os respectivos tamanhos em bytes da variável real, inteira e caracter?

4
ENTRADA DE DADOS

A entrada de dados na linguagem de programação Pascal pode ser feita através do comando Read ou Readln. Geralmente é definida como entrada padrão o input efetuado por meio de um teclado (keyboard).

COMANDO DE ENTRADA – READ E READLN

A diferença básica na utilização ou não do ln é a mesma já ocorrida pelo comando Write/Writeln. Observe atentamente os programas exemplos abaixo:

```
Program Exemp1;
Uses
        Crt;
Const
        Dois = 2;
Var
        Base, Altura, Area : Real;
Begin
        Clrscr;
        Write(´Entre com a base:´);
        Readln(Base);
```

```
        Write(´Entre com a altura:´);
        Readln(Altura);

        Area := Base * Altura / 2;
        Writeln(´Area = ´, Area: 4:1);
        Delay(1000);
End.
```

No exemplo anterior quando é lido o valor de Base e Altura, posteriormente, o cursor fica posicionado na próxima linha (***Readln***).

Outro exemplo:

```
Program Exemp2;
Uses
        Crt;
Var
        Nome : String[35];
Begin
        Write(´Qual o seu nome?´);
        Read(Nome);
        Writeln;
        Writeln(´Bom Dia ´, Nome);
        Delay(1200);
End.
```

Exercícios

1- Crie um programa em Pascal que permita ao usuário fornecer dados suficientes para calcular e imprimir a área de um determinado trapézio.

2- Faça um programa que leia nome, ano de nascimento, altura e peso de uma certa pessoa. No final do processamento, imprima o nome e sua respectiva idade.

3- Escreva um programa em Pascal que leia dois números inteiros pelo teclado. Após o processamento, calcule e imprima o quadrado do primeiro pelo segundo e o cubo do segundo pelo primeiro número.

5
OPERADORES

Os operadores são utilizados com o objetivo de auxiliar na formação de uma certa expressão. Existem diversos tipos diferentes de operadores a saber. Façamos então um estudo mais detalhado sobre eles.

ARITMÉTICOS E FUNÇÕES

Símbolos	Operadores
*	Multiplicação
/	Divisão Real
DIV	Divisão Inteiro
MOD	Módulo
INT	Parte Inteira
+	Adição
-	Subtração

** ou ^	Exponenciação
ABS	Valor Absoluto
ArcTan	Arco Tangencial
Cos	Coseno
Sin	Seno
Exp	Expoente
Frac	Fração
Ln	Logaritmo
MaxInt	Máximo Inteiro
Pi	Valor de π
Round	Arredondamento
Sqr	Quadrado do nº
Sqrt	Raiz Quadrada
Trunc	Parte Inteira

A precedência matemática quanto à utilização de sinais é mantida da mesma forma que em algoritmos computacionais, ou seja, (, *, /, +, -. Do decorrer desse livro

teremos acesso a programas escritos em Pascal que utilizam algumas dessas funções.

LÓGICOS

São também conhecidos como conectivos lógicos de operação, pois objetivam conectar expressões lógicas que geralmente, são apresentadas através de comandos de decisão. São eles:

Operador	Função
And	E lógico
Or	Ou lógico
Not	Não lógico

RELACIONAIS

Símbolos	Significado
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual
=	Igualdade

<>	Diferença
:=	Atribuição simples

Modelos:

```
If ( (estado_civil=´S´) And (idade>17))
        Then
                Writeln(´To dentro...´);

If ((uf=´R´) Or (uf=´S´) Or (uf=´M´) Or (uf=´E´))
        Then
                Writeln(´Região Sudeste´);

If(Not(sexo=´F´))
        Then
                Write(´MASCULINO´);

If((ano<1990) And ((idade=20) OR (idade=30))
        Then
                Writeln(´Mensagem Enviada...´)
        Else
                Write(´Mensagem interrompida...´);
```

Os operadores lógicos And, Or e Not também são conhecidos como conectivos de operação. São usados para permitir o uso de mais de uma condição numa mesma expressão.

PRIORIDADE DOS OPERADORES

1º	Parênteses e Funções
2º	Sinais unários
3º	Exponenciação
4º	Divisão e Multiplicação
5º	Adição e Subtração
6º	Operadores Relacionais
7º	NÃO
8º	E
9º	OU

Exercícios

1. Para que servem os operadores?
2. Qual o símbolo que representa o operador de módulo da linguagem de programação Pascal?
3. O que significa dizer quando falamos que uma variável está sendo decrementada?
4. Diferencie SQR e SQRT.
5. Na expressão A + B * C – F, qual a precedência de operações?
6. Qual a diferença da divisão feita pelo símbolo / daquela feita através da palavra reservada DIV?

6
ESTRUTURAS CONDICIONAIS

Essas estruturas permitem com que o programa execute diferentes tipos de procedimentos baseados em uma determinada decisão. Basicamente, existem dois tipos de estruturas condicionais: Alternativa simples e alternativa composta.

ALTERNATIVA SIMPLES – if... then…

É uma estrutura que através de uma determinada condição, retorna um valor possível, somente se essa condição for verdadeira (debidamente atendida).

```
Program Exemplo;
Uses
        Crt;
Var
        A : Integer;
Begin
        Clrscr;
        Write(´Entre com o valor de A:´);
        Read(A);
```

```
    If ( a > 0 )
        Then
            Writeln(´A e maior que ZERO´);
    Delay(1000);
End.
```

No exemplo, se o valor da variável a for maior do que zero, será impressa a mensagem "*A e maior que ZERO*".

Na verdade, podemos enfatizar tal explicação com diversas modalidades diferentes de exemplos. Contudo, no decorrer de nossos estudos, iremos abordar outras aplicações exemplo e realizar diversos outros tipos de modelos diferentes. Vejamos a seguir como funciona a chamada alternativa composta.

ALTERNATIVA COMPOSTA – if... then… else...

Esta estrutura, diferente da primeira, permite ao usuário retornar dois valores possíveis. O primeiro verdadeiro verdadeiro, se a condição estipulada for satisfeita e o segundo falso, caso a condição não seja devidamente atendida.

```
Program Composta;
Uses
    Crt;
Var
    Sal : Real;
Begin
    Clrscr;
    Write(´Entre com seu salario:´);
    Readln(Sal);
```

```
If ( salario > 1500 )
        Then
            Writeln(´Voce ganha bem´)
        Else
            Writeln(´Voce precisa ganhar mais um pouquinho´);
End.
```

Caso você tenha a necessidade de utilizar dois ou mais testes condicionais vinculados a mesma estrutura condicional, você poderá faze-lo através da técnica de programação denominada encadeamento ou ninho de if´s. Vejamos a seguir como funciona tal estrutura.

ENCADEAMENTO DE if´s

Trata-se de um recurso que permite ao usuário utilizar uma estrutura if dentro de outra obtendo assim, diversas respostas possíveis. Contudo, a última resposta será sempre a negativa da última alternativa levantada.

```
Programa Ninho;
{ Este programa testa o uso de uma estrutura condicional encadeada }
Uses
    Crt;
Var
    Num : Integer;
Begin
    Clrscr;
    Write(´Entre com um numero:´);
```

```
Readln(Num);

If ( num = 0 )
        Then
                Writeln(´Numero ZERO´)
        Else
        If ( num < 0 )
                Then
                        Writeln(´Numero Negativo´)

                Else
                        Writeln(´Numero Positivo´);
        Readkey
End.
```

Não usa ponto-e-vírgula pois a instrução If ainda não terminou.

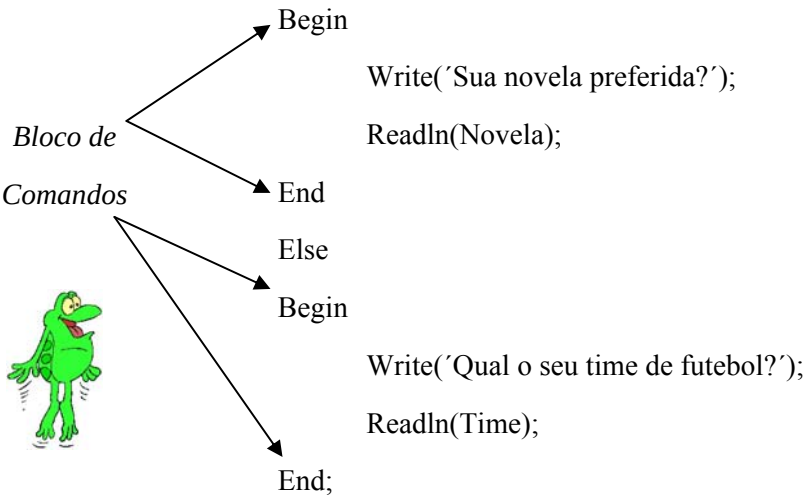
Não usou o ponto-e-vírgula porque no Pascal no último comando antecessor do **End.** é facultativa sua utilização.

Aqui podemos realmente constatar que o comando If tem ordem matemática N-1 que significa dizer...

Para cada N respostas que eu precise obter utilizarei N-1 comando If a utilizar.

Outro Exemplo:

```
Program OutroExempNinho;
Uses
        Crt;
Var
        Sexo : Char;
        Nome : String[40];
        Time : String[12];
        Novela : String[15];
Begin
        Clrscr;
        Write(´Entre com seu nome:´);
        Readln(Nome);
        Write(´Sexo?´);
        Readln(Sexo);

        If ((Sexo <> ´F´) And (Sexo <> ´M´))
                Then
                        Writeln(´O que e isso… Companheiro!?´)
                Else

                If (Sexo = ´F´)
                        Then
```



End.

Um Bloco de Comandos é caracterizado pelo uso do Begin... End; em qualquer estrutura que desejarmos expressar mais de uma instrução e esta só suporte uma instrução como resposta.

Note que sempre ao utilizarmos um bloco de comandos todos os comandos que estiverem vinculados à ele têem que, obrigatoriamente, ser finalizados com o separador de comandos da linguagem de programação Pascal, ponto-e-vírgula (;).

MÚLTIPLA ESCOLHA – Case ... of

É conhecido como estrutura condicional de múltiplas escolhas ou estudo de casos. Torna-se vantajoso ao ninho de if's quanto a utilização de uma expressão bastante longa pois, facilita na escrita do programa, quanto ao comando de tomada de decisão, se o mesmo apresentar muitas alternativas.

Observe atentamente o exemplo apresentado abaixo:

```
Program MultiplaEscolha;
Uses
        Crt;
Var
        Opcao : Char;
Begin
        Clrscr;
        Write(´Entre com uma letra:´);
        Readln(Opcao);

        Case Opcao of
                ´A´ : Writeln(´Letra A´);
                ´B´ : Writeln(´Letra B´);
                ´C´ : Writeln(´Letra C´);
                Else
                        Writeln(´Nao e A, B nem C´);
        End;
        Readkey;
End.
```

Observação:

Na estrutura condicional de múltipla escolha ou estudo de casos, é opcional a utilização do comando Else para representar uma alternativa que significa "EM NENHUM DOS CASOS ANTERIORES".

Também, é importante saber que a execução do comando Case segue os seguintes passos:

1- A expressão é avaliada.

2- Se o resultado da expressão não for igual a nenhuma das constantes e já estiver sido incluída no comando Case a opção Else, o comando associado ao Else será executado. Caso contrário, isto é, se a opção Else não estiver presente, o processamento continuará a partir do comando seguinte ao comando Case.

IMPORTANTE:

Pode haver uma ou mais instruções seguindo cada case. Se isso ocorrer precisaremos especificar para cada uma delas um bloco de comandos.

A expressão em Case (<expressão>) deve ter um valor compatível com um inteiro, isto é, podem ser usadas expressões do tipo char e integer com todas as suas variações. Você não pode usar reais, ponteiros, strings ou estruturas de dados.

Atividade Prática 1

Faça um programa em PAscal que permita cadastrar dois números interiros distintos através

do teclado. Ao final do processamento, imprima

qual o maior e o menor desses números.

Atividade Prática 2

Escreva um programa em Pascal que permita ao usuário ler três números distintos pelo teclado listando, ao final do programa, qual o maior, menor e o mediano deles.

Atividade Prática 3

Faça um programa em Pascal que permita ao usuário ler uma letra qualquer através do teclado. No final do processamento, o programa deverá informar se a letra digitada é uma vogal ou uma consoante.

Atividade Prática 4

Escreva um programa em Pascal que leia um número informando ao final do processamento se esse número é primo ou não.

1- Qual a diferença do uso da múltipla escolha para o ninho de if´s?

2- O que significa dizer que o comando if apresenta ordem matemática N-1?

3- Explique como funciona a chamada alternativa simples. Exemplifique.

4- Explique como funciona a chamada alternativa composta. Exemplifique.

5- Qual a função do "comando" Else na estrutura de múltipla escolha?

6- Para que utilizamos, em nossos programas de computador, as chamadas estruturas condicionais?

7- O que quer dizer a seguinte linha de comando abaixo:

```
if( a > b )
```

```
        Then
                Writeln(A)
        Else
                Writeln(B);
```

8- Qual a função dos chamados conectivos lógicos de operação?

9- Qual o conectivo de operação que expressa o contrário do que estamos querendo referir?

10- O que faz a seguinte linha de comando:

```
If( Not (Sexo = ´M´))
        Then
                Write(´Feminino…´);
```

7
ESTRUTURAS DE ITERAÇÃO

São utilizadas para que uma parte de seu programa possa ser repetida n vezes sem a necessidade de reescrevê-lo. Essas estruturas também são conhecidas como LOOP ou laços.

Iremos estudar as três estruturas possíveis conhecidas em Pascal: FOR (para/variando), WHILE... DO (enquanto/faça) e REPEAT... UNTIL (repita/até). Vamos analisá-las nessa ordem.

LOOP FOR

É encontrado na maioria das linguagens de programação, incluindo Pascal. No entanto, como vamos ver, a versão Pascal, em particular, é mais flexível e dispõe de recursos do tipo crescente (TO) e descrescente (DOWNTO).

A idéia básica do comando for é que você execute um conjunto de comandos, um número fixo de vezes, enquanto uma variável de controle, é incrementada ou decrementada a cada passagem pelo laço. Vejamos o exemplo a seguir:

```
Program ProgTO;

Uses

        Crt;

Var

I : Integer;
```

```
Begin
        For I :=1 to 10 do
                WriteIn(I);
End.
```

Crescente

do Menor para o Maior

Outro Exemplo:

```
Program ProgDescesce;
Uses
        Crt;
Var
   I : Integer;
Begin
        For I := 10 downto 1 do
                Writeln(I);
 End.
```

Descescente

do Maior para o Menor

Observe que quando desejarmos utilizar uma estrutura crescente usamos TO e quando desejamos utilizar uma estrutura decescente usamos então DOWNTO.

Veja a seguir, outro exemplo da estrutura for tendo como resultado um outro comando for resultando assim, em um for dentro de outro for.

```
Program DESC1;
Uses
        CRT;
Var
        A, B : Real;
        CONT : Integer;
Begin
        Clrscr;
        Write (' Informe o 1º valor: ');
        Readln (A);
        Write (' Informe o 2º valor: ');
        Readln (B);

        For  CONT := 1 to 3 do
        Begin
                A := A + B;

                If (A > 0) And (A < 10)
                        Then
                                B := A + 3
                        Else
                                B := A + 1;
        End;
```

```
        Writeln (A:3:1);
        Writeln (B:3:1);
        Readkey
End.
```

LOOP WHILE... DO

É o mais genérico dos três e pode ser usado para substituir os outros dois; em outras palavras, o laço while supre todas as necessidades. Já os outros dois, são usados por uma questão de comodidade. Vamos analisar o exemplo a seguir:

```
Program ExempPrat;
Uses
        Crt;
Var
        A, B, R, I : Integer;
Begin
        I := 1;

        While (I <= 5) do
        Begin
                Write('Entre um valor para A: ');
                Readln(A);
                Write('Entre um valor para B: ');
```

```
            Readln(B);
            Readkey;
            R := A + B;
            Writeln('O resultado corresponde a: ', R);
            Readkey;
            I := I + 1;
        End;
End.
```

LOOP REPEAT ... UNTIL

O comando repeat ... until é semelhante ao comando while. A diferença está no momento da avaliação da expressão, o que sempre ocorre sempre após a execução do comando. Isto faz com que o comando do laço repeat ... until sempre execute pelo menos uma vez antes de realizar tal teste. Observe atentamente o exemplo a-baixo:

```
Program ExemploLoop;
Uses
        Crt;
Var
        A, B, R, I : Integer;
Begin
        I := 1;

        Repeat
```

```
        Write('Entre um valor para A: ');
        Readln(A);
        Write('Entre um valor para B: ');
        Readln(B);
```

```
        Writeln;
        R := A + B;
        Writeln('O resultado corresponde a: ', R);
        Writeln;
        I := I + 1;
    Until (I > 5);
End.
```

Pula linha

Note que no loop Repeat… Until, embora sejam utilizados diversos comandos dentro de sua estrutura, não há necessidade do uso de bloco de comandos, pois já fica implícito para o comando.

1- Qual o uso prático das chamadas estruturas de iteração?

2- Qual a diferença básica em utilizar a estrutura de repetição repita...até e enquanto...faça?

3- Exemplifique o uso da estrutura para...variando com um contador incrementado.

4- Exemplifique o uso da estrutura para...variando com um contador decrementado.

5- Escreva um programa em Pascal que escreva todos os números pares compreendidos na seguinte seqüência: 10 até 20. Para tal, utilize a estrutura de repetição...

   a) Para...variando

   b) Repita...até

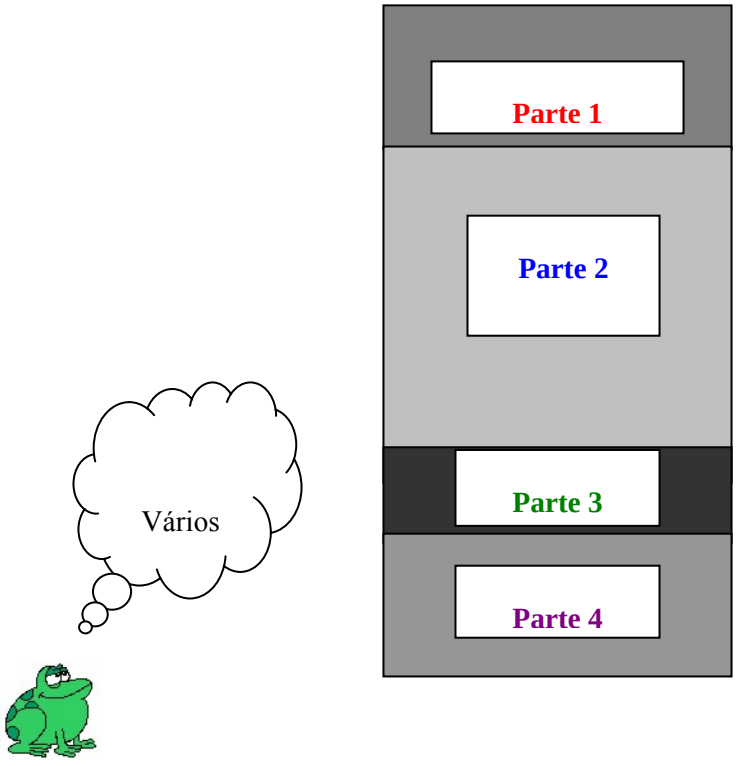
   c) Enquanto...faça

6- Qual a vantagem de elaborarmos um programa cujo teste seja feito no início?

7- Qual a vantagem de elaborarmos um programa cujo teste seja feito no final?

8
Modularização

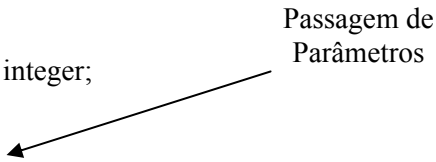
A modularização nada mais é que separar em partes um determinado programa objetivando a facilidade de manutenção do mesmo. Essas partes podem ser identificadas como procedimentos (PROCEDURES) ou funções (FUNCTIONS).



PROCEDURES x FUNCTIONS

Vejamos então, exemplos práticos com o uso de Procedimentos e Funções descritos em Pascal, logo a seguir:

```
Program Exemp01;
uses
        Crt;
var
        X1, X2, X3: integer;

procedure S (var A: integer);
var
        T: integer;
begin
        T:= 1;
        T:= T + A;
        A:= T;
end;

begin
        clrscr;
        X1:= 2;
        S(X1);
        X2:= 5;
```



```
        S(X2);
        X3:= 3;
        S(X3);
        writeln ( X1 );
        writeln ( X2 );
        writeln ( X3 );
        readkey;
end.
```

Note que no Pascal, sempre iniciamos os procedimentos e funções e depois escrevemos o corpo principal do programa. Isso porque, por padrão, a linguagem de programação Pascal assume o método Bottom Up, já estudado em Algoritmos Computacionais.

Outro Exemplo:

```
Program Exemp02;
uses
        Crt;
var
        X, Y: integer;

Procedure procA;
begin
        writeln ('Passou pelo A');
end;
```

```
Procedure procB;
begin
        writeln (X);
end;

Procedure procC;
begin
        writeln ('Valor de Y = ', Y);
        X:= Y;
        procB;
end;

{ Corpo Principal }
begin
        clrscr;
        X:= 20;
        Y:= 30;
        procB;
        procA;
        writeln (Y);
        procC;
        Y:= Y - 3;
        procC;
```

```
        procA;
        readkey;
end.
```

Você observa que no primeiro exemplo foi passado o valor de A através da Procedure **S,** criadas no programa. Quando isso ocorre dizemos ter uma passagem de parâmetros que pode ser interpretada de duas maneiras distintas: por valor e por referência.

Logo, a passagem de parâmetros nada mais é que a substituição dos chamados parâmetros formais pelos parâmetros reais durante a execução de uma certa subrotina. Essa substituição pode ocorrer de duas formas como já foi dito anteriormente. Por valor, quando o parâmetro passado não tem seu valor alterado durante um certo processamento.

Por referência, ocorre quando existe uma alteração do valor do parâmetro real quando o parâmetro formal estiver sendo manipulado por um dado processamento.

```
PROGRAM EXEPROCED;
VAR
  A,B,X,Y:INTEGER;

PROCEDURE CALCULOS;
BEGIN
  X:=A*A+B*B;
  Y:=A+B;
END;
```

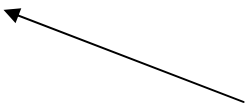
```
{ Corpo Principal }
BEGIN
  WRITELN('INFORME DOIS NUMEROS:');
  READLN(A,B);
  CALCULOS;
  A:=A+1;
  B:=B-1;
  CALCULOS;
  WRITELN(A,B);
  READLN;
END.
```

Mais um Exemplo:

```
Program Processamento;
uses
        crt;
var
        n1,n2:real;
        opcao:char;

procedure soma(a,b:real);
var
        xresultado:real;
```

```
begin
   xresultado:=a+b;
   write('Soma = ',xresultado:8:2);
   readkey;
end;

procedure leitura(var valor1,valor2:real);
begin
   write('1o. valor: ');readln(valor1);
   write('2o. valor: ');readln(valor2);
end;

{ Corpo Principal }
begin
   clrscr;
   while opcao <> '0' do
   begin
      clrscr;
      writeln('      MENU        ');
      writeln;
      writeln('(1) Entrada de Dados    ');
      writeln('(2) Soma                ');
      writeln('(0) Saida               ');
```

```
      writeln;
      opcao := Readkey;

      case opcao of
         '1' : leitura(n1,n2);
         '2' : soma(n1,n2);
      end;
   end
end.
```



Opcao receberá o valor da tecla pressionada pelo usuário.

No exemplo anterior enquanto o usuário não pressionar o número zero (0) o programa não chega ao seu final. Porém, se ele escolher a opção 1, ele realiza a entrada de dados onde serão fornecidos os valores das variáveis valor1 e valor2. E, se e somente se, a opção 2 for escolhida, a procedure é devidamente processada.

Outro Exemplo Prático:

```
program VERIFIQUE1;
uses CRT;
var
        A, B, C: integer;

procedure SOMA ( M: integer; var  N: integer );
begin
        M:= M + 1;
        N:= 2*M+3;
```

```
end;


function CALCULO ( T, K: integer) : integer;
begin
        T:= 8 * T;
        K:= K + T;
        CÁLCULO:= K;
end;


begin
        clrscr;
        A:= 4;
        B:= 3;
        C:= 2;
        writeln ( CALCULO ( A, B ) );
        SOMA ( B, C );
        writeln ( CALCULO ( B, C ));
        readkey;
end.
```

Note que precisamos definir um determinado tipo para a função que está sendo declarada. Isso porque, na verdade, quando declaramos uma função, estamos declarando uma "variável". Outra característica de diferenciação importante entre Procedimentos e Funções é que no uso de Procedimentos podem ser retornados **n** parâmetros como resposta já na função, somente um parâmetro Serpa retornado como resposta.

Outro Exemplo:

```
PROGRAM PROCEDIMENTOS;
USES
  CRT;
VAR
 NUM1,NUM2:INTEGER;
 OP,RESP:CHAR;

PROCEDURE ENTRADA;
BEGIN
  WRITE(' ****  ENTRE COM O 1§ VALOR ==>');
  READLN(NUM1);
  WRITE(' ****  ENTRE COM O 2§ VALOR ==>');
  READLN(NUM2);
  WRITE('QUAL O OPERADOR ?');
  READLN(OP);
END;

PROCEDURE SOMAR;
VAR
 SOMA:INTEGER;
BEGIN
 SOMA:=NUM1+NUM2;
```

```
 WRITELN('A SOMA E:',SOMA);
 READLN;
END;

PROCEDURE SUBTRAIR;
VAR
 SUB:INTEGER;
BEGIN
 SUB:=NUM1-NUM2;
 WRITELN('A SUBTRACAO E:',SUB);
 READLN;
END;

{**** PROGRAMA PRINCIPAL ****}

BEGIN
 REPEAT
   ENTRADA;
   CASE OP OF
   '+' : SOMAR;
   '-' : SUBTRAIR;
   ELSE
    WRITELN('**** OPERADOR INVALIDO ! ****');
```

```
   END;

   WRITELN(' === MAIS [S/N] ?');
   READLN(RESP);
   RESP:=UPCASE(RESP);
 UNTIL (RESP='N');
END.
```

Mais um exemplo:

```
program VERIF;
uses CRT;
var
        A, B, C: integer;

procedure SOMA ( M: integer; var  N: integer );
begin
                M:= M + 1;
                N:= 2*M+3;
end;

function CALCULO ( T, K: integer) : integer;
begin
                T:= 8 * T;
```

```
                K:= K + T;
                CÁLCULO := K;
end;

begin
        clrscr;
        A:= 4;
        B:= 3;
        C:= 2;
        writeln ( CALCULO ( A, B ) );
        SOMA ( B, C );
        writeln ( CALCULO ( B, C ));
        readkey;
end.
```

Outro Exemplo Mais:

```
PROGRAM  Teste;
VAR
        X : INTEGER;

PROCEDURE PorValor (A :  INTEGER);
BEGIN
        A : = 5;
```

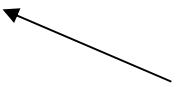
```
END;


BEGIN
        X : = 10;
        PorValor(X);
        WRITE(X);
END.
```

No Exemplo acima, o conteúdo da variável "X" não será alterado após o retorno do procedimento ao programa principal.

FUNÇÃO RECURSIVA

Uma função é dita recursiva quando existe dentro de uma certa função uma chamada para ela mesma por diversas vezes. Logo, a técnica de recursividade cria consecutivos "espelhos" para refletir n vezes a chamada da função que está sendo referenciada.

```
Function Mdc (P1, P2 : Integer) : Integer;
Begin
        If P2 = 0
                Then Mdc := P1
                Else Mdc := Mdc (P2, P1 MOD P2);
End;
```



Chamada da Função
por ela mesma

Outro Exemplo:

```
Function Fat (Num : Integer) : Integer;
Begin
        If Num > 0
                Then Fat := Num * Fat (Num – 1)
                Else Fat := 1;
End;
```

Note que o interessante disso tudo é que nos exemplos apresentados acima, um processo é repetido várias vezes mesmo sem estar escrito fisicamente uma certa estrutura de repetição.

FUNÇÕES PARA COTROLE DE TELA

O Turbo Pascal 7.0 possui algumas funções e procedimentos pré-declarados para controle da tela. São eles:

clrscr - Comando para limpar a tela.

clreol - Este comando apaga todos os caracteres da linha que estão à direito do cursor.

delline - Apaga a linha onde está o cursor e causa scroll nas linhas seguintes de tal modo que preencha a linha deletada.

insline - Este comando funciona de modo oposto ao delline. Insere uma linha vazia na posição onde está o cursor. Provoca scroll nas linhas seguintes.

gotoxy(X,Y) - Movimenta o cursor para coluna X, linha Y da tela. A contagem das colunas e linhas começa no 1, ou seja, o canto superior esquerdo possui coordenadas (1 ,1).

lowvideo - Diminui a luminosidade dos caracteres na tela.

highvideo - Aumenta a luminosidade dos caracteres na tela.

norrnvideo - Retoma a luminosidade normal.

OUTROS RECURSOS:

delay(X) - Provoca uma pausa no processamento equivalente a X millissegundos. X deve ser definido como número inteiro.

exit - Abandona o bloco corrente. Estando numa sub-rotina, retoma ao bloco onde foi feita a sua chamada. Se usado no programa principal, termina a execução do mesmo.

halt - Interrompe o processamento do programa e retoma ao nível de sistema operacional.

randomize - Inicializa o gerador de números aleatórios com um valor aleatório.

UTILIZAÇÃO DE UNITS

Além das rotinas definidas pelo programador, existe na linguagem de programação Pascal, um conjunto de rotinas embutidas, ou seja, fornecidas pela Borland. Este tipo de rotina embutida é conhecido pelo nome de **unidade**, em inglês **unit**, e será apresentado de forma básica a seguir:

CRT: Esta unidade é a mais utilizada na programação do Turbo Pascal. Por esta razão, ela possui a maior parte das rotinas e variáveis de som, controle de vídeo e teclado.

DOS: Esta unidade possui as rotinas que envolvem a utilização operacional, na maior parte das vezes permitindo controle de baixo nível.

GRAPH: Esta unidade possui rotinas destinadas à manipulação da capacidade gráfica de um PC.

OVERLAY: Esta unidade possibilita gerenciar as atividades de um programa, desta forma, é possível aproveitar uma mesma área de memória para rodar várias rotinas diferentes, economizando memória.

PRINTER: Esta unidade permite declarar um arquivo tipo texto com o nome LST e, desta forma, associá-lo à impressora.

SYSTEM: Esta unidade possui a maior parte das rotinas padrão da linguagem Pascal, não necessitando ser citada para ser usada, pois o Turbo Pascal já a executa de forma automática.

Para se fazer uso deste recurso é necessário o uso da instrução **uses** situada antes da declaração da instrução **var** como já foi comentado em capítulos anteriores.

1- O que vem a ser a técnica de Recursividade?

2- Para que serve a passagem de parâmetros?

3- Diferencie passagem de parâmetro por valor e passagem de parâmetros oir referência.

4- Defina modularização.

5- Na linguagem Pascal, qual a diferença existente entre procedimentos e funções, caso realmente exista?

6- Escreva um programa em Pascal que permita ao usuário fornecer dois números inteiros através do teclado imprimindo, no final do processamento, a soma e o produto desses números. Obrigatoriamente seu programa deve adotar a técnica de modularização.

7- Qual a função de utilizarmos Units em nossos programa em Pascal?

8- Para que seve a Unit Crt?

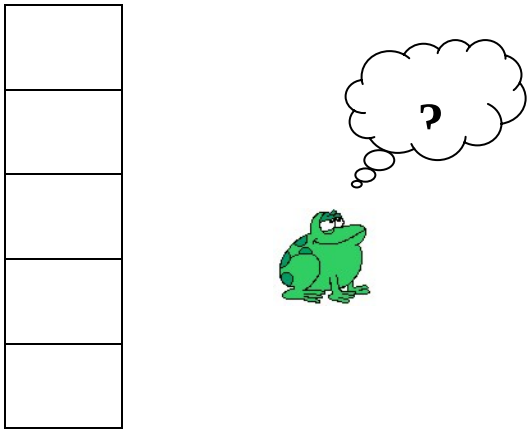
9- Escreva um programa em Pascal que permita ao usuário entrar com dois números para processar e imprimir o quadrado do primeiro pelo segundo número e também, o cubo do segundo pelo primeiro número.

10- Qual a função da Unit DOS em seu programa Pascal?

9
MANIPULAÇÃO COM VETORES

Primeiramente vamos tirar uma dúvida cruel. O que é vetor? Bem, vetor representa um endereço de memória onde serão armazenados diversos dados, de acordo com o dimensionamento dado a ele, na própria definição (criação) da variável vetor. Isso é possível na maioria das linguagens de programação. Um vetor representa então conjuntos indexados de elementos de um mesmo tipo. Na linguagem de programação Pascal, devemos utilizar a palavra reservada **Array** para podemos especificar um dado vetor ou matriz.

Nota



Var

TAB : Array [0..99] of Integer;

Define uma tabela em Pascal de 100 elementos inteiros.

UNIDIMENSIONAIS

Como o próprio nome já diz um vetor unidimensional é aquele que apresenta apenas uma dimensão. Vejamos então um simples exemplo de aplicação de vetores unidimensionais. Note na sintaxe de utilização:

Sintaxe:

< Variável > : **Array** [LimiteInicial..LimiteFinal] **of** < Tipo >;

Vejamos então um simples exemplo de aplicação de vetores unidimensionais.

```
Program Cadast;
Uses
        Crt;
Var
        Nome : Array[1..10] of String[30];
        Idade : Array[1..10] of Integer;
        Cont : Integer;
Begin
        For Cont := 1 to 10 do
        Begin
                Clrscr;
                Write(´Nome:´);
                Readln(Nome[Cont]);
```

```
            Write(´Idade?´);
            Readln(Idade[Cont]);
        End;

        Cont := 1;

        While (Cont <= 10) do
        Begin
            If (Idade [Cont] >= 18)
                Then
                    Writeln(´Seu nome e ´, Nome[Cont]);
            Cont := Cont + 1;
        End;
        Readkey;
End.
```

No exemplo anterior é criado um vetor para a variável Nome e também para a variável Idade cujo dimensionamento determinado para ambas é dez (10) espaços (dimensões) disponíveis. Já no nosso próximo exemplo, apenas a variável NOTA tem um dimensionamento de cinco (5) posições. Veja então como funciona o programa abaixo:

```
Program NOTAS;
Uses
   Crt;
```

```
Var
   NOTA : Array[1..5] of real;
   SOMA, MEDIA : real;
   X : integer;

Begin
   SOMA := 0;

   For X:= 1 to 5 do
   Begin
      Clrscr;
      Gotoxy(5, 4);
      Write('Entre com a nota:');
      Readln(NOTA[X]);
      SOMA := SOMA + NOTA[X];
   End;

   MEDIA := SOMA / 5;

   Write('As notas dos alunos foram:');

   For X := 1 to 5 do
      Writeln(NOTA[X]);
```

```
    Writeln('A media da turma foi', MEDIA:4:1);
    Readkey;
End.
```

Outros Exemplo:

```
Program Unidim;
uses CRT;
type
        vet = array  [1..10] of char;
var
        V: vet;

Procedure TROCA ( var  C: vet );
var
        I : integer;
        AUX : char;
begin
        AUX:= C[6];
        C[6]:= C[9];
        C[9]:= AUX;
        for I := 1 to 4 do
          begin
                AUX:= C [ I ];
```

```
                C [ I ]:= C [ 9 - I ];
                C [ 9 - I ]:= AUX;
          end;
          C[6]:= C[2];
        end;

Procedure EXIBE ( X : vet );
var
        I : integer;
begin
        for I:= 1 to 10 do
                writeln ( X [ I ] );
end;

{ Corpo Principal do Programa }
begin
        clrscr;
        TROCA ( V );
        EXIBE ( V );
        Readkey;
end.
```

COM MAIS DE UMA DIMENSÃO

Caracteriza-se por ter mais de uma dimensão, como é o caso dos programas de matrizes. Observe a sintaxe abaixo:

Sintaxe:

< Variável > : **Array** [LimiteInicial**1**..LimiteFinal**1**,

LimiteInicial**n**..LimiteFinal**n**] **of** < Tipo >;

Observe a seguir a utilização de um vetor com mais de uma dimensão.

```
Program Turma;
Uses
   Crt;
Var
   Coluna, Linha : Integer;
   NotasMedia : Array[1..50, 1..4] of Real;

Begin
   Clrscr;
   Soma := 0;
   For Linha := 1 to 50 Do
      For Coluna := 1 to 4 Do
      Begin
         Write('Entre com as 4 Notas do Aluno:');
         Readln(NotasMedia[Linha, Coluna]);
```

```
        End;
End.

    For Linha := 1 to 50 Do
      For Coluna := 1 to 4 Do
          Soma := Soma + NotasMedia[Linha, Coluna];
    NotasMedia[Linha, 5] := Soma / 4;
    Writeln(NotasMedia[Linha, 5]);
    Readkey;
End.
```

Outro Exemplo:

```
Program MatrizQuad;
Uses
        Crt;
Var
        Mat : Array[1..4, 1..4] of Integer;
        Lin, Col : Integer;

Begin
        For Lin := 1 to 4 do
                For Col := 1 to 4 do
                Begin
```

```
                Clrscr;
                Write(´Entre com o elemento:´);
                Readln(Mat[Lin, Col]);
            End;
End.
```

No programa exemplo anterior é criado uma matriz quadrada de ordem 4 para receber como entrada de dados elementos do tipo inteiro. Só pra relembrar, uma matriz quadrada é aquela que apresenta o mesmo número de linhas e colunas. Logo, uma matriz 4x4 apresenta quatro linhas e quatro colunas.

ALGORITMOS DE ORDENAÇÃO

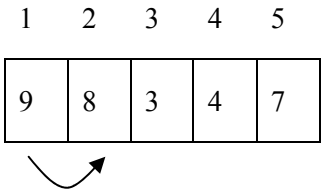
Ordenar ou simplesmente classificar uma tabela consiste em fazer com que seus elementos sejam armazenados de acordo com um critério de classificação determinado.

Os ditos critérios de classificação podem variar muito, dependendo do tipo dos elementos que estejam sendo ordenados e também, do propósito final do programa.

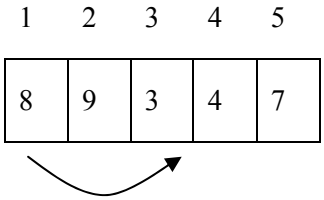
Nos casos mais simples, como os que serão examinados aqui, os elementos da tabela são números inteiros, e os critérios de ordenação se resumem à ordem crescente (ascendente – do menor para o maior) ou decrescente (descendente – do maior para o menor) dos valores expressos na tabela.

Podemos até trabalharmos com tabelas não-ordenada contudo, não é aconselhável pois dificultaria um processo de busca dinâmica, como é o caso da pesquisa binária, por exemplo. A seguir, veremos a ordenação de forma crescente de um vetor numérico do tipo inteiro de 5 elementos.

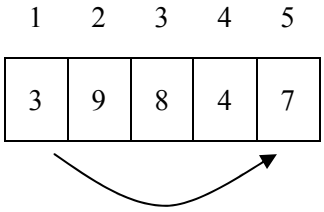
1º)



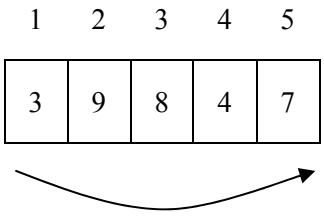
2º)



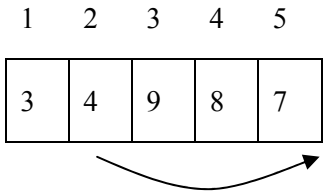
3º)



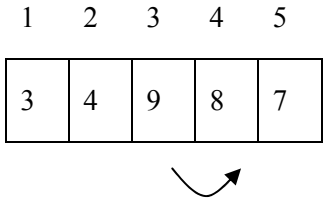
4º)



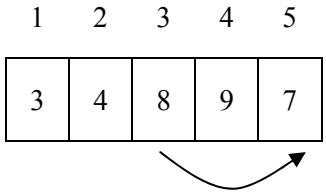
7º)



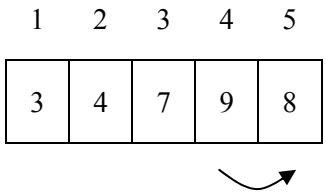
8º)



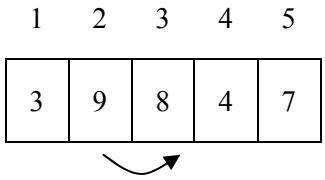
9º)



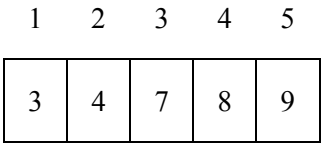
10º)



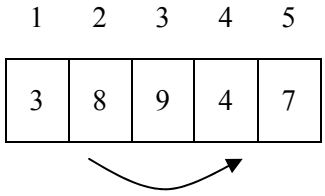
5º)



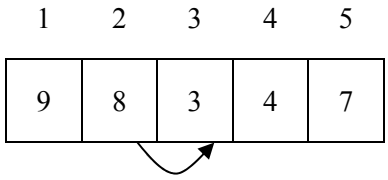
11º)

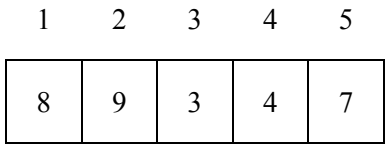


6º)



A estrutura em destaque mostra como ficará o vetor após as comparações e devidas devidas trocas. Quando você consegue acompanhar passo-a-passo o algoritmo desenvolvido, torna-se muito mais fácil seu processo de aprendizagem de classificação. Trocando em miúdo a primeira troca. Ou melhor passando para algoritmo.





Digamos que o vetor chame-se Numeros, para realizar esta troca eu irei precisar de uma variável auxiliar, pois, se eu disser que a posição 1 recebe o conteúdo da posição 2, então, tanto a posição 1 quanto a posição 2 irão conter o mesmo valor como mostrado abaixo:

NUMEROS

1	2	3	4	5
8	8	3	4	7

E agora, onde vou busca o 9 que estava na posição 2? Por isso precisamos de uma variável auxiliar, vamos chama-la de Aux, não é sugestivo ?

NUMEROS

1	2	3	4	5
9	8	3	4	7

Primeiro, eu guardo a conteúdo da posição 1 na variável auxiliar;

{AUX ← NUMEROS[1]}

AUX

9

Depois, coloco o conteúdo da posição 2 na posição;

{NUMEROS[1] ← NUMEROS[2]}

NUMEROS

1	2	3	4	5
8	8	3	4	7

E por último, coloco na posição 2, o valor da posição 1 que guardei na variável auxiliar. {NUMEROS[2] ← AUX}

NUMEROS

1	2	3	4	5
8	9	3	4	7

Analisando o que ocorre nas trocas, chegamos seguinte conclusão:

Pegamos a posição	E verificamos com as demais para saber se devemos ou não fazer a troca
1	2, 3, 4, 5
2	3, 4, 5
3	4, 5
4	5

Analisando o quadro anterior, **então** podemos chegar a seguinte conclusão:

Quando I for	J será
1	2, 3, 4, 5
2	3, 4, 5
3	4, 5
4	5

Observe que o primeira posição a ser verificada será sempre a primeira subseqüente, ou seja, se a posição em questão é I, a primeira subseqüente é I + 1.

Logo a seguir, vamos analisar cada um deles detalhadamente para que possamos então fixar melhor toda essa nova idéia.

```
Program ORDENAVET;
uses crt;
type
   vet = array[1..20] of integer;
var
```

```
  v1, v2: vet;
  i: integer;

Procedure ENTRADA (var x: vet);
begin
      for i:=1 to 20 do
         begin
            write('Informe o ',i, '§ elemento do vetor: ');
            readln(x[i]);
            writeln;
         end;
end;

Procedure ORDENACAO (var y: vet);
var
    aux, atual, proximo: integer;
begin
      for atual:=1 to 19 do
         begin
            for proximo:=atual + 1 to 20 do
               begin
                  if (y[atual] > y[proximo]) then
                     begin
```

```
                    aux:= y[atual];
                    y[atual] := y[proximo];
                    y[proximo] := aux;
                end;
            end;
        end;
end;

Procedure SAIDA (z:vet);
begin
    writeln ('VETORES ORDENADOS');
    for i:= 1 to 20 do
      begin
        writeln (z[i]);
        writeln;
      end;
end;

(*** Programa Principal ***)

begin
  clrscr;
  ENTRADA (v1);
```

```
    ENTRADA (v2);
    clrscr;
    ORDENACAO (v1);
    ORDENACAO (v2);
    clrscr;
    SAIDA (v1);
    SAIDA (v2);
    readln;
end.
```

MÉTODO DE SELEÇÃO - Quick Sort

Ordem crescente: tab[i] >= tab[j] se i > j

Ordem decrescente: tab[i] <= tab[j] se i < j

```
Procedure QuickSort;
Begin
        For p := 1 to N – 1 do
        Begin
                Menor := p;
                For s :=  (p+1) to N do
                        If (Tabela[s].Num < Tabela[Menor].Num)
                                Then
                                        Menor := s;
```

```
                    If Menor > p
                        Then
                            Troca (p, Menor);
        End;
End;
```

Chama o Procedimento de Troca para trocar os valores de p e Menor.

A seguir será abordado o método da bolha, tecnicamente denominado “Bubble Sort”. Observe atentamente e faça um comparativo entre eles:

MÉTODO DA BOLHA - Bubble Sort

O nome método bolha deriva-se da maneira com que os maiores valores "afundam" em direção ao fim do vetor, enquanto os menores valores "borbulham" em direção a "superfície" ou topo da tabela.

No programa a seguir usa-se a variável troquei para indicar se durante um passo foi executada alguma troca.

```
Procedure BubbleSort;
Uses
        Crt;
Var
        I, J, AUX : Integer;
Begin
        For I := 1 to 5 - 1 do
                For J := I + 1 to 5 do
```

```
                If Num[I] > Num[J]
                        Then
                        Begin
                            { Realiza a Troca de Valores }
                                AUX := Num[I];
                                Num[I] := Num[J];
                                Num[J] := AUX;
                        End;
End;
```

Não podemos esquecer de que quando temos a necessidade de classificar dados de uma certa tabela, não basta trocar apenas o campo-chave utilizado como referência para o processo de ordenação.

Devemos então, trocar de posição, todos os campos pertencentes aos registros da tabela pois senão haverá diversos registros com informações misturadas.

Lembre-se que Num[I] > Num[J], refere-se a ordenação / Classificação crescente e, Num[I] < Num[J], refere-se a ordenação decrescente.

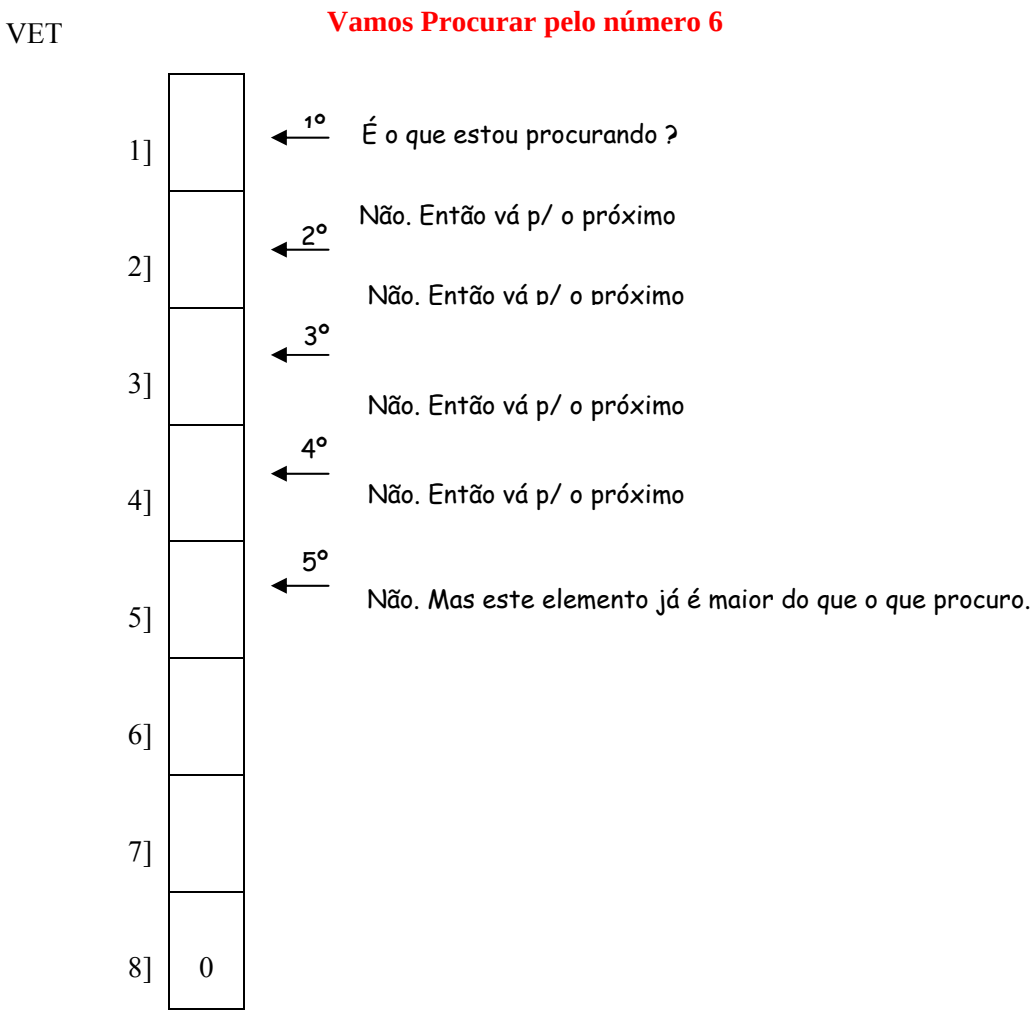
ALGORITMOS DE BUSCA

Os algoritmos de busca existem com o objetivo de realizar uma busca dentro de uma determinada tabela objetivando assim localizar um determinado registro.

Irei abordar e esclarecer a diferença básica existente entre os dois métodos de busca mais utilizados nos programas tradicionais: a busca seqüencial ou pesquisa seqüencial e a busca binária ou pesquisa binária.

PESQUISA SEQUENCIAL

Neste método, o processo de busca pesquisa a tabela seqüencialmente, desde o início até seu fim. Cada elemento da tabela é comparado com a chave. Se eles forem iguais o índice do elemento é retornado, e a busca termina. Esse tipo de busca pode ser realizada em tabelas ordenadas ou não. A princípio, a maioria dos programadores de computadores, realizam suas operações de busca em tabelas ordenadas. Vamos então verificar no exemplo abaixo, a eficiência desse método de busca:



Observe a seguir o código-fonte em Pascal de como funciona na realidade todo esse processo de busca:

```
PROGRAM PESQSEQ;
VAR
  NOME:ARRAY[1..5] OF STRING;
  I:INTEGER;
  PESQ:STRING;
  RESP:CHAR;
  ACHA:BOOLEAN;
PROCEDURE ENTRADA;
BEGIN

  FOR I:=1 TO 5 DO
  BEGIN
    WRITE('INFORME O ',I,' § NOME:');
    READLN(NOME[I]);
  END;
END;

PROCEDURE PESQUISA;
BEGIN
  WRITE('ENTRE COM O NOME A SER PESQUISADO:');
  READLN(PESQ);
```

```
 I:=1;
 ACHA:=FALSE;

 WHILE (I<=5) AND (ACHA=FALSE) DO
 BEGIN
  IF(PESQ=NOME[I])
   THEN
    ACHA:=TRUE
   ELSE
    I:=I+1;
  END;
END;

PROCEDURE SAIDA;
BEGIN
 IF (ACHA=TRUE)
  THEN
   WRITELN(PESQ,' FOI ENCONTRADA NA POSICAO ',I)
  ELSE
   WRITELN(PESQ,' NAO FOI LOCALIZADO')
END;
```

```
{*** PROGRAMA  PRINCIPAL ***}
BEGIN
  ENTRADA;
  RESP:='S';

  WHILE (RESP='S') DO
  BEGIN
    PESQUISA;
    SAIDA;
    WRITE('DESEJA CONTINUAR ?');
    READLN(RESP);
    RESP := UPCASE(RESP);     { LOWCASE tem função contrária }
  END;
END.
```

Aqui a função UPCASE foi utilizada para transformar uma letra em maiúscula; Se quiséssemos transformar a letra em minúscula, deveríamos ter usado a função LOWCASE.

Mais um Exemplo:

```
PROGRAM Exemp002;
VAR
  MATR:ARRAY[1..5] OF INTEGER;
```

```
 NOME:ARRAY[1..5] OF STRING;
 I,PESQ:INTEGER;
 RESP:CHAR;
 ACHA:BOOLEAN;

PROCEDURE ENTRADA;
BEGIN
 FOR I:=1 TO 5 DO
 BEGIN
  WRITE('INFORME A MATRICULA DO ALUNO:');
  READLN(MATR[I]);
  WRITE('INFORME O NOME:');
  READLN(NOME[I]);
 END;
END;

PROCEDURE PESQUISA;
BEGIN
 WRITE('ENTRE COM O N§ DA MATRÍCULA A SER PESQUISADA:');
 READLN(PESQ);
 I:=1;     ACHA:=FALSE;
 WHILE (I<=5) AND (ACHA=FALSE) DO
 BEGIN
```

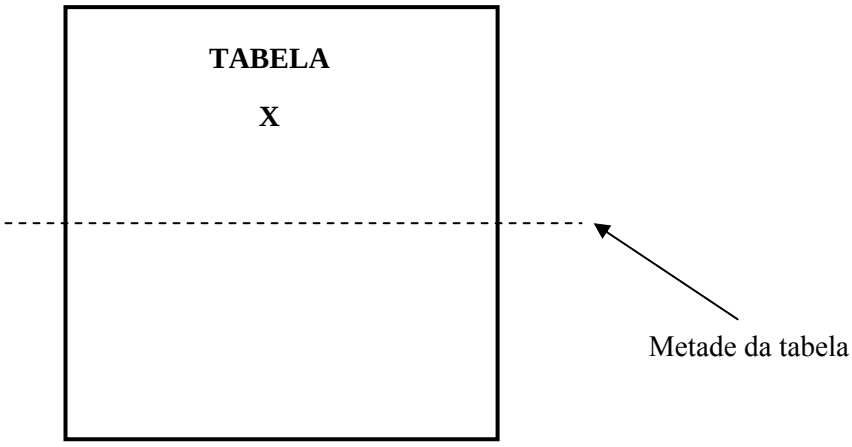
```
   IF(PESQ=MATR[I])
    THEN
     ACHA:=TRUE
    ELSE
     I:=I+1;
  END;
END;

PROCEDURE SAIDA;
BEGIN
 IF (ACHA=TRUE)
  THEN
     WRITELN('O ALUNO CORRESPONDENTE A ',MATR[I],' E  ',NOME[I])
    ELSE
     WRITELN('MATRICULA NAO FOI LOCALIZADA')
END;

{*** PROGRAMA  PRINCIPAL ***}
BEGIN
 ENTRADA;
 RESP:='S';
 WHILE (RESP='S') DO
 BEGIN
```

```
   PESQUISA;
   SAIDA;
   WRITE('DESEJA CONTINUAR ?');
   READLN(RESP);
   RESP := UPCASE(RESP);
 END;
END.
```

Aqui, utilizamos a função UPCASE pois, independentemente do usuário digitar uma letra maiúscula ou minúscula, no nosso caso a letra ‘s’ ou ‘S’, o programa transformará o dado fornecido em maiúsculo.

PESQUISA BINÁRIA

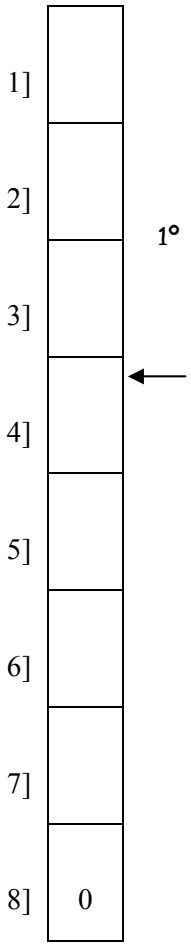
Tem por característica reduzir sempre o espaço da tabela a ser procurado pela metade objetivando assim, um menor tempo de procura de um determinado elemento dentro da tabela. A redução pela metade da tabela deverá ser feita sucessivamente até que todo o processo de busca chegue ao seu final. Para isso, é necessário que os dados já estejam devidamente organizados (ordenados) pois senão, poderá ocorrer falhas nesse tipo de processo. ***Logo, aconselha-se que esse tipo de pesquisa deva ser feito apenas em tabelas ordenadas***.



Vamos Procurar pelo número 7 (Valor existente)

1º Passo

VET



Posição Inicial Atual = 1 e Posição Final Atual = 8

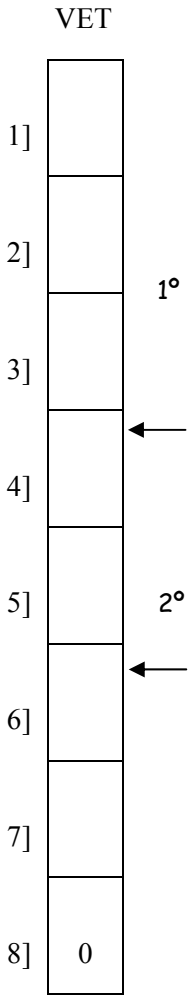
Primeiro devo apontar para o elemento central do vetor, para calcular esta posição soma a posição inicial com a final e divido por dois, ou seja, Quociente(1+ 8, 2).

Com isto concluímos que devemos apontar para o elemento de posição 4.

Em seguida, perguntamos se o conteúdo da posição que estamos apontando é o que estamos procurando. Como 5 não é o que estamos procurando, a resposta é, "NÃO".

Concluímos então que devemos continuar a pes-

2º Passo



A seguir, verifico que o número que estou procurando é maior que o número que estou apontando, logo, concluo que o que procuro só pode está dá posição posterior a que estou apontando até o final do vetor.

Posição Inicial Atual = 5 e Posição Final Atual = 8

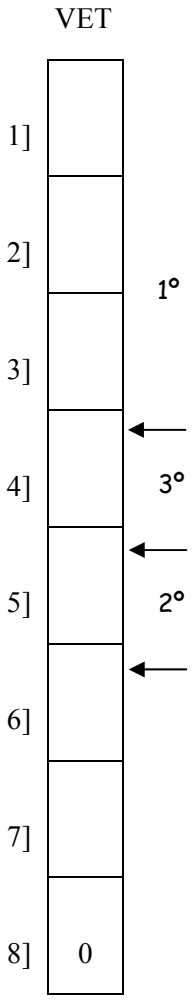
Então pego a minha nova posição inicial que é a posição que estou apontando mais um, soma com a posição final que é oito e divido por dois para encontrar a minha nova posição central, ou seja, Quociente(5 + 8,2).

A seguir aponto para o elemento 6.

Ai faço a celebre pergunta: "É o que estou procurando?", a resposta é "**NÃO**".

Então continuaremos a pesquisa.

3º Passo



A seguir, verifico que o número que estou procurando é menor que o número que estou apontando, logo, concluo que o que procuro só pode está dá posição anterior a que estou apontando até a minha atual posição inicial do vetor.

Posição Inicial Atual = 5 e Posição Final Atual = 5

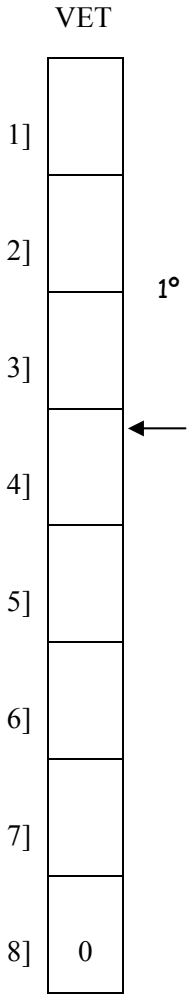
Então pego a minha nova posição final que é a posição que estou apontando menos um, soma com a posição inicial que é cinco e divido por dois para encontrar a minha nova posição central, ou seja, Quociente(5 + 5,2).

A seguir aponto para o elemento 5.

Ai faço a celebre pergunta: "É o que estou procurando?", a resposta é "**SIM**", a seguir paro a pesquisa.

Entendeu? Então, para facilitar ainda melhor o entendimento desse processo, vamos simular a seguir, a busca de um determinado valor armazenado no vetor VET.

Vamos Procurar pelo número 6 (Valor inexistente)



Posição Inicial Atual = 1 e Posição Final Atual = 8

Primeiro devo apontar para o elemento central do vetor, para calcular esta posição soma a posição inicial com a final e divido por dois, ou seja, Quociente(1+ 8, 2).

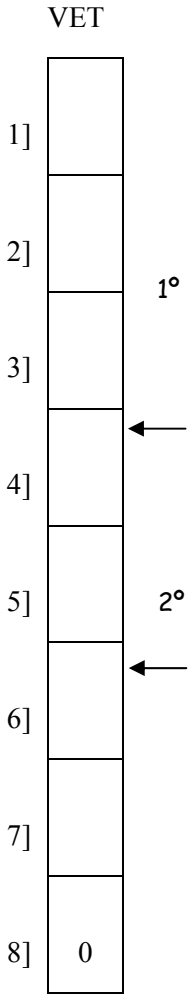
Com isto concluímos que devemos apontar para o elemento de posição 4.

Em seguida, perguntamos se o conteúdo da posição que estamos apontando é o que estamos procurando. Como 5 não é o que estamos procurando, a resposta é, "NÃO".

Concluímos então que devemos continuar a pesquisa.

As posições inicial correspondem a posição dita topo da tabela enquanto a posição final corresponde a posição de fim da tabela.

2º Passo



A seguir, verifico que o número que estou procurando é maior que o número que estou apontando, logo, concluo que o que procuro só pode está dá posição posterior a que estou apontando até o final do vetor.

Posição Inicial Atual = 5 e Posição Final Atual = 8

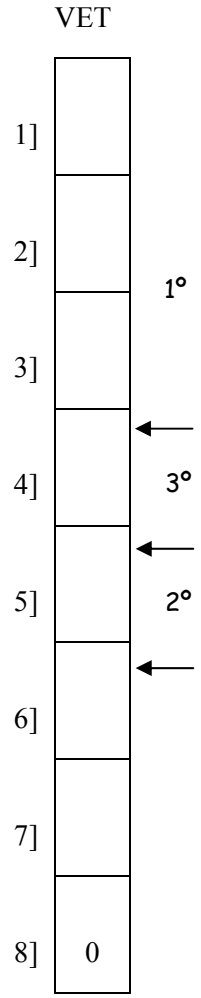
Então pego a minha nova posição inicial que é a posição que estou apontando mais um, soma com a posição final que é oito e divido por dois para encontrar a minha nova posição central, ou seja, Quociente(5 + 8,2).

A seguir aponto para o elemento 6.

Ai faço a celebre pergunta: "É o que estou procurando?", a resposta é "NÃO".

Então continuaremos a pesquisa.

3º Passo



A seguir, verifico que o número que estou procurando é menor que o número que estou apontando, logo, concluo que o que procuro só pode está dá posição anterior a que estou apontando até a minha atual posição inicial do vetor.

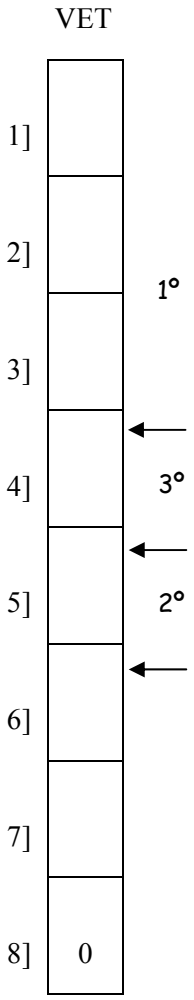
Posição Inicial Atual = 5 e Posição Final Atual = 5

Então pego a minha nova posição final que é a posição que estou apontando menos um, soma com a posição inicial que é cinco e divido por dois para encontrar a minha nova posição central, ou seja, Quociente(5 + 5,2).

A seguir aponto para o elemento 5.

Ai faço a celebre pergunta: “É o que estou procurando?”, a resposta é “NÃO”.

4º Passo



A seguir, verifico que o número que estou procurando é menor que o número que estou apontando, logo, concluo que o que procuro só pode está dá posição anterior a que estou apontando até a minha atual posição inicial do vetor.

Posição Inicial Atual = 5 e Posição Final Atual = 4

Então ele volta a apontar para uma posição que já havíamos visto anteriormente que não poderia está.

Concluo então que toda vez que a posição inicial for maior que a posição final, é sinal que

Segundo a teoria apresentada anteriormente, observe atentamente o próximo exemplo de programa escrito em Pascal para busca ou pesquisa binária:

```
PROGRAM PESQBIN;
VAR
  NOME:ARRAY[1..5] OF STRING;
  I,J,COMECO,MEIO,FINAL:INTEGER;
  PESQ:STRING;
  RESP:CHAR;
  ACHA:BOOLEAN;

PROCEDURE ENTRADA;
BEGIN
  FOR I:=1 TO 5 DO
  BEGIN
    WRITE('INFORME O ',I,' § NOME:');
    READLN(NOME[I]);
  END;
END;

PROCEDURE ORDENACAO;
VAR
 AUX:STRING;
BEGIN
 FOR I:=1 TO 4 DO
   FOR J:=I+1 TO 5 DO
```

```
    IF (NOME[I]>NOME[J])
     THEN
      BEGIN
       AUX:=NOME[I];
       NOME[I]:=NOME[J];
       NOME[J]:=AUX;
      END;
END;

PROCEDURE PESQUISA;
BEGIN
 WRITE('ENTRE COM O NOME A SER PESQUISADO:');
 READLN(PESQ);
 COMECO:=1;
 FINAL:=5;
 ACHA:=FALSE;
 WHILE (COMECO<=FINAL) AND (ACHA=FALSE) DO
 BEGIN
  MEIO := (COMECO+FINAL) DIV 2;
  IF(PESQ=NOME[MEIO])
   THEN
    ACHA:=TRUE
   ELSE
```

```
     IF (PESQ<NOME[MEIO])
      THEN
       FINAL:=MEIO-1
      ELSE
       COMECO:=MEIO+1;
   END;
END;

PROCEDURE SAIDA;
BEGIN
 IF (ACHA=TRUE)
  THEN
   WRITELN(PESQ,' FOI ENCONTRADA NA POSICAO ',MEIO)
  ELSE
   WRITELN(PESQ,' NAO FOI LOCALIZADO')
END;



{*** PROGRAMA  PRINCIPAL ***}
BEGIN
 ENTRADA;
 ORDENACAO;
 RESP:='S';
```

```
 WHILE (RESP='S') DO
 BEGIN
  PESQUISA;
  SAIDA;
  WRITE('DESEJA CONTINUAR ?');
  READLN(RESP);
  RESP:=UPCASE(RESP);
 END;
END.
```

Então, você fica livre para decidir qual o melhor tipo de mecanismo a ser utilizado pelo seu programa. Só não se esqueça que a característica primordial da busca binária é realizar uma divisão sucessiva na tabela reduzindo sempre a mesma em 50% do seu tamanho real. Isto ocorre de forma dinâmica tornando-a assim, bem mais rápida que a busca seqüencial.

1- O que é vetor?

2- Como posso declarar um vetor através da linguagem Pascal?

3- Qual a diferença de vetor unidimensional para um vetor que apresente mais de uma dimensão?

4- O que realmente é uma matriz?

5- Declare uma matriz quadrada de ordem 6, que apresentará elementos do tipo inteiro.

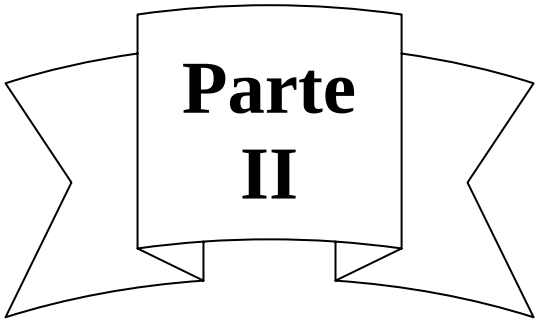
6- Como posso determinar se minha ordenação será crescente ou descrescente?

7- Porque é necessário o uso de variáveis auxiliares durante o processo de troca em uma ordenação?

8- Escreva um algoritmo que expresse o princípio da pesquisa seqüencial.

9- Escreva um algoritmo que expresse o princípio da busca binária.

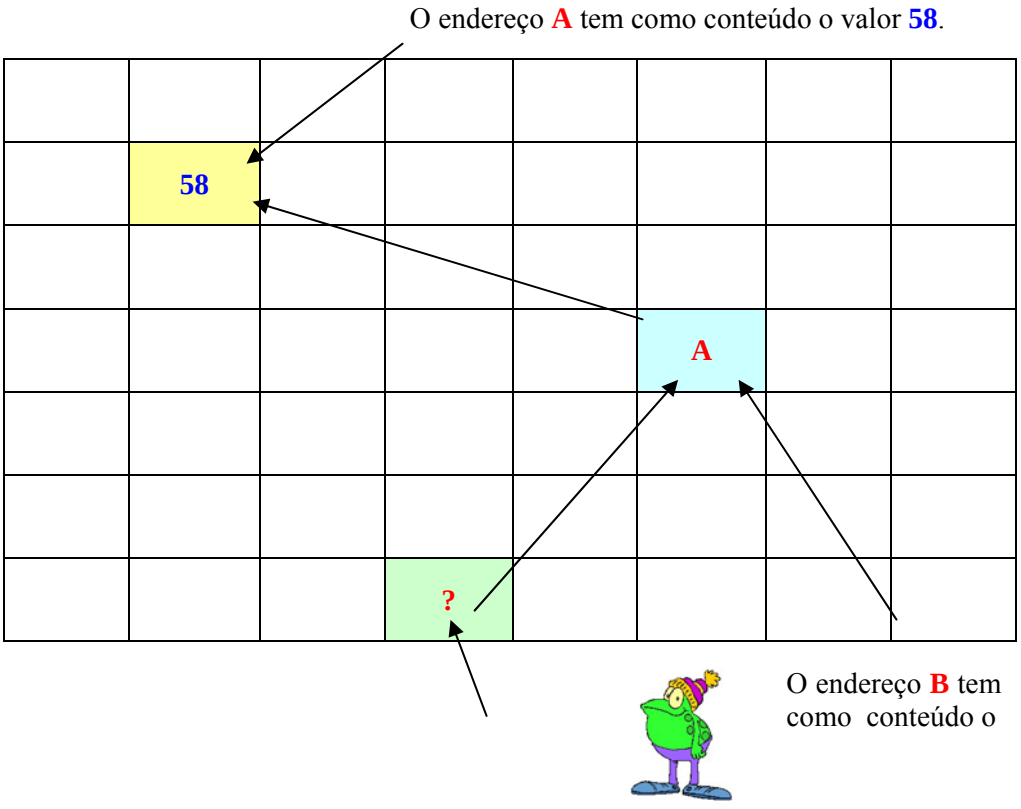
10- Diferencie Busca Seqüencial e Busca Binária.



Tópicos Avançados em Pascal

10
PONTEIROS OU APONTADORES

Um ponteiro ou apontador é uma variável que, no seu espaço de memória, armazena o endereço de uma segunda variável, essa sim, normalmente contendo o dado a ser manipulado pelo programa.



Como o endereço de **C** aponta endereço de **A**

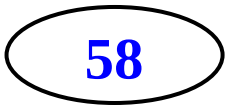
para o endereço de **B** que referencia

o conteúdo do endereço de **A**, portanto

o conteúdo armazenado no endereço **C** será igual ao

endereço da variável **A** cujo valor armazenado em seu conteúdo é 58;

Portanto, o valor apontado pela variável **C** será o conteúdo da variável **A**, ou seja,



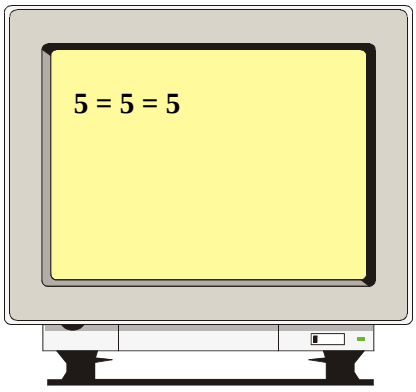
Só para relembrar, uma variável do tipo ponteiro em Pascal, é declarada da seguinte forma:

```
Program TestPoninter;
Uses
        Crt;
Var
        A, B : Integer;
        Pt1 :  ^ Integer;      { Ou também Pointer : ↑Integer; }
        Pt2 : ^Integer;
Begin
        A := 5;
        Pt1 := ^A;
        B := Pt1^;
```

```
    Writeln(A, ´ = ´,  Pt1^, ´ = ´, B);
    Readkey;
    Pt2 := ^B;
    Pt2^ := 24;
    Writeln(A, ´ = ´, Pt2^, ´ = ´, B);
    Readkey;
End.
```

No exemplo anterior, as variáveis Pt1 e Pt2, são variáveis ponteiro do tipo inteiro. Após a variável A receber o valor 5, o ponteiro Pt1 recebe o endereço de memória da variável A passando então a apontar para ele. Logo após a variável B receber o o conteúdo da posição de memória apontado por Pt1, a variável Pt2 recebe o endereço de memória da variável B passando a apontar para ele, desde então. Na linha de comando Pt2^ := 24; é alterado o conteúdo da variável B, devido o Pt2 estar apontando para o endereço de memória dele.

Os dois comandos Writeln existentes no programa geram as seguintes saídas respectivamente:



ALOCAÇÃO DINÂMICA

Esse tipo de alocação de dados que ocorre na memória do computador, que refere-se ora para conteúdo, ora para endereços de uma variável, é tecnicamente denominada ***Alocação Dinâmica***. Para realizar uma alocação dinâmica obrigatoriamente precisamos trabalhar com variáveis do tipo ponteiro ou apontadora. No entanto, uma variável caracterizada como dinâmica é produzida através do procedimento **New** (Pt1), por exemplo.

Toda variável dinâmica quando não é mais utilizada deve ser eliminada da memória do computador para evitar a ocupação de espaço desnecessariamente. Isto ocorre através do procedimento **Dispose** (Pt1), por exemplo.

Lembre-se ainda que se uma variável ponteiro estiver apontando para lugar algum, ela é especificada pela constante pré-definida do Pascal **Nil**. Observe o exemplo prático abaixo atentamente para melhor elucidar a explicação anterior:

```
Program AlocaDim;
Uses
        Crt;
Var
        P, Q : ^Integer;
        X : Integer;
Begin
        New (P);
        P^ := 3;
        Q := P;
        Writeln(Q^, P^);
        X := 7;
        Q^ := X;
```

Cria uma variável dinâmica P

```
Writeln(P^, Q^);
New (P);
P^ := 5;
Writeln(P^, Q^);
Readkey;
Dispose (P);   <-------------  Elimina a variável dinâmica P
End.
```

1- O que você entende por alocação dinâmica?

2- Defina variáveis apontadoras.

3- Escreva um programa em Pascal que permita ao usuário armazenar em uma variável ponteiro valores de forma indireta, ou seja, o conteúdo armazenado deserá ser feito a partir de outra variável, também ponteiro do mesmo tipo.

4- Explique com suas palavras para que serve a constante Dispose?

5- Explique para que serve o New?

6- Declare as seguintes variáveis ponteiro: PT1 e PT2, sabendo-se que PT2 é cracterizada como uma variável dinâmica previamente inicializada no programa principal.

11
ESTRUTURAS DE DADOS

Segundo o professor da COPPE/UFRJ Jayme Luiz Szwarcfiter, em seu livro Estruturas de Dados e seus Algoritmos:

> "*As estruturas diferem uma das outras pela disposição ou manipulação de seus dados. A disposição de dados em uma estrutura obedece a condições preestabelecidas e caracteriza a estrutura.*
>
> *O estudo de estrutura de dados não pode ser desvinculado de seus aspectos algoritmos. A escolha certa da estrutura adequada a cada caso depende diretamente do conhecimento de algoritmos para manipular a estrutura de maneira diferente.*"

As estruturas de dados que aqui serão representadas através de programas escrito através da linguagem de programação Pascal, que objetivam expressar o entendimento concreto do que realmente vem a ser Registro, Fila, Pilha, Lista e Árvore, são aquelas de maior difusão desse estudo. Também, será mostrado um exemplo de aplicação prática que manipula listas encadeadas.

USANDO REGISTROS

Anteriormente, para ser mais exato no capitulo9, você teve contato com a utilização de matrizes e notou que somente foi possível trabalhar com um tipo de dado por matriz. No momento em que se precisou trabalhar com dois tipos de dados diferentes, foi necessária a utilização de duas matrizes, uma de cada tipo.

Para solucionar esta deficiência, poderá ser utilizada estrutura de dados registro, a qual consiste em trabalhar vários dados de tipos diferentes (os campos) em uma mesma estrutura. Por esta razão, este tipo de dado é considerado heterogêneo.

Para tanto, considere que seja informado o nome de um aluno e suas 4 notas bimestrais que deverão ser agrupados em uma mesma estrutura. Note que o registro está formado pelos campos: Nome, Primeira Nota, Segunda Nota, Terceira Nota e Quarta Nota e podendo este ser denominado um registro de aluno. Observe abaixo o Layout do formato de um registro com seus campos

CADASTRO DE NOTAS ESCOLARES

Nome .....: _____

Primeira Nota ....: _____

Segunda Nota .....: _____

Terceira Nota .....: _____

Quarta Nota .....: _____

Na estrutura da linguagem de programação Pascal, os tipos registro devem ser declarados ou atribuídos antes das definições das variáveis, pois é muito comum ocorrer a necessidade de se declarar uma variável com o tipo de registro atribuído.

Um tipo registro é declarado em Pascal com a instrução **type** em conjunto com a instrução **record**, conforme a seguinte sintaxe:

Type

<identificador> = **record**

{ Lista de campos e seus tipos }

end;

Var

<variável> : <identificador> ;

Onde identificador é o nome do tipo registro e lista dos campos e seus tipos é a relação de variáveis que serão usadas como campos, bem como o seu tipo, podendo ser: **real**, **integer**, **boolean**, **string** entre outros existentes na linguagem de programação Pascal.

Na área de tipo podemos também definir um conjunto de dados que será identificado pelo Pascal através da palava reservada **Set**. Observe atentamente o programa exemplo abaixo:

```
Program Conjunto;
Uses
        Crt;
Const
         N=100;
Var
         Crivo, Primos : Set of  2..N;
         Prox, J : Integer;

Begin
```

```
        Crivo :=[2..N];
        Primos := [   ];
        Prox := 2;
        Repeat
           While not (Prox in Crivo) do
              Prox := succ (Prox);
              Primos := Primos + [Prox];
              J := Prox;

              While  J <= N do
                 Begin
                    Crivo := Crivo – [J];
                   J := J+Prox
                 End
       Until  Crivo = [  ]
End.
```

Voltando a explicação do programa anterior ao mostrado acima, podemos observar que após a instrução **var**, deverá ser indicada a variável tipo registro e a declaração do seu tipo de acordo com um identificador definido anteriormente. Note que a instrução **type** deverá ser utilizada antes da instrução **var**, pois ao definir um tipo de variável, pode-se fazer uso deste tipo definido, conforme já foi visto por nós em capítulos anteriores desse livro.

Vamos então pegar como exemplo a proposta de se criar um registro denominado ALUNO, cujos campos são NOME, NOTAI, NOTA2, NOTA3 e NOTA4. Em Pascal, ficaria da seguinte forma:

```
type
CAD_ALUNO = Record
                NOME : string;
                NOTAI : real;
                NOTA2 : real;
                NOTA3 : real;
                NOTA4 : real;
            End;

var
ALUNO : cad_aluno;
```

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

Tanto a leitura quanto escrita de um registro são efetuadas respectivamente com as instruções **read/readln** e **write/writeln** seguidas do nome da variável do tipo registro e de seu campo correspondente separado por um caractere concatenador "." (ponto), como exemplificado a seguir.

```
Program LEITURA_ESCRITA;
Uses crt;
type
CAD_ALUNO = Record
                NOME : string;
```

```
                NOTA1 : real;
                NOTA2 : real;
                NOTA3 : real;
                NOTA4 : real;
            End;

var
ALUNO : cad_aluno;

Begin
        clrscr;
        writeln ('Cadastro de Aluno');
        writeln ;
        write ('Informe o nome .....: ' );
        readln (ALUNO. NOME);
        write ('Informe a primeira nota .....: ' );
        readln (ALUNO.NOTA1);
        write ('Informe a segunda nota .....: ' );
        readln (ALUNO.NOTA2);
        write ('Informe a terceira nota .....: ' );
        readln (ALUNO. NOTA3);
        write ('Informe a quarta nota.....: ' );
        readln (ALUNO. NOTA4);
```

```
        writeln;
        writeln (‘Nome ....: ', ALUNO.NOME);
        writeln ('Nota 1 ..; ', ALUNO.NOTA1:2:2);
        writeln ('Nota 2 ..: ', ALUNO.NOTA2:2:2);
        writeln ('Nota 3 ..: ', ALUNO.NOTA3:2:2);
        writeln ('Nota 4 ..: ', ALUNO.NOTA4:2:2);
        writeln ;
        writeln ('Tecle <ENTER> para encerrar. ');
        readkey;
end.
```

Perceba que no registro definido, existem quatro variáveis do mesmo tipo definidas para armazenar quatro notas (NOTA1, NOTA2, NOTA3 e NOTA4), desta forma, este registro poderá ser definido usando uma matriz do tipo vetor para armazenar as notas.

Pegando como base a proposta de se criar um registro denominado ALUNO, cujas notas serão informadas em uma matriz do tipo vetor, este seria assim declarado:

```
type
BIMESTRE = array [1..4] of real;
CAD_ALDNO = Record
                    NOME : String;
                    NOTA ; Bimestre;
                End;
```

```
var
ALUNO : cad_aluno;
```

Quando especificamos o registro CAD_ALUNO, notamos que existe nele um campo chamado NOTA do tipo bimestre, onde bimestre é a especificação de um tipo de conjunto matricial de uma única dimensão com capacidade para quatro elementos. Veja que o tipo bimestre foi definido acima do tipo CAD_ALUNO. Manter uma ordem na definição de tipos que são dependentes de outros tipos definidos é imprescindível, pois se o tipo BIMESTRE fosse definido abaixo do tipo CAD_ALUNO, geraria um erro na execução do programa, uma vez que o compilador do Turbo Pascal consideraria como sendo tipo desconhecido. A seguir, é apresentado o código-fonte para a leitura e escrita dos dados, usando o registro de matriz.

```
Program LEITURA_ESCRITA;
Uses crt;
type
    BIMESTRE = array[1..4] of real;
    CAD_ALUNO = Record
                    Nome : String;
                    Nota : Bimestre;
                End;

var
    ALUNO : cad_aluno;
    I : byte;
Begin
```

```
clrscr;
writeln ('Cadastro de aluno');
writeln ;
write (‘Informe o nome .......: ' );
readln (ALUNO.NOME);
writeln;

For I :=1 to 4 do
Begin
        write ('Informe a ', I:2, 'a. nota ..:');
        readln (ALUNO.NOTA[I]);
end;

writeln;
writeln;
writeln ('Nome ....: ' , ALUNO.NOME);
writeln;

for I :=1 to 4 do
        writeln ('Nota ', I, ' ..: ', ALUNO.NOTA[I]:2:2);
writeln ;
writeln ('Tecle <ENTER> para encerrar: ');
readkey ;
```

end.

A partir desse momento, seria pertinente construir um programas que permitisse trabalhar com vários registros de alunos. Para melhor exemplificar, considere que você deverá fazer um programa que faça a entrada e a saída de nome e notas de 8 alunos. isto já é familiar. Veja a seguir, a definição do tipo registro e também a definição da matriz de registros para os oito alunos:

```
program LEITURA_ESCRITA;
type
        BIMESTRE = array[1..4] of real;
        CAD_ALUNO = Record
                                NOME : string;
                                NOTA : bimestre;
                        end;
var
        ALUNO : array [1..8] of CAD_ALUNO;
```

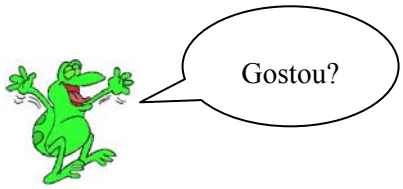
Observe que acabamos efetivamente de criar a variável ALUNO como um vetor de oito posições do tipo CAD_ALUNO. A seguir, é apresentado um programa que fará a entrada e saída dos dados de 8 alunos.

```
program LEITURA_ESCRITA;
type
        BIMESTRE = array[1..4] of real;
        CAD_ALUNO = Record
```

```
                    NOME : string;
                    NOTA : bimestre;
                end;

var
    ALUNO : array[1..8] of cad_aluno;
    I, J : byte;

Begin
    writeln('cadastro de aluno');
    writeln;

    for J := 1 to 8 do
    begin
        write('Informe nome do ', J:2,'o. aluno ...: ');
        readln(ALUNO[J].NOME);
        writeln;

        for I :=1to 4 do
        begin
            write('Informe a ', 1:2, 'a. nota .....:..: ');
            readln(ALUNO[J].NOTA[I]);
        end;
```

```
            writeln;
        end;

        writeln;
        writeln;

        for J := 1 to 8 do
        begin
            writeln('Nome aluno: ', J:2.' ...: ', ALUNO[J].NOME);
            writeln;

            for I :=1to 4 do
                writeln('Nota'. I, ' .....: ', ALUNO[J].NOTA[I]:2:2);
            writeln;
        end;
        writeln('Tecle <ENTER> para encerrar: ');
        readkey;
end.
```

Note que o looping da variável J controla o número de alunos da turma, no caso, 8 e o looping da variável f controla o número de notas, até 4 por aluno. Para cada movimentação de mais um na variável J, existem quatro movimentações na variável I.



Bem, para melhor demonstrarmos a utilização de programas com tabelas de dados heterogêneos, considere um programa que efetue a feitura das 4 notas bimestrais de 8 alunos, apresentando no final, os dados dos alunos classificados por nome.

COMANDO WITH... DO

Note que ao invés de referenciarmos os campos de uma estrutura registro como, por exemplo, ALUNO[i].NOME, ALUNO[i].NOTA, você poderá usar o comando **With <Variáveis Registro> do**, que permite referências abreviadas a campos da estrutura registro mencionada no programa. Observe sintaxe abaixo:

```
With ALUNO[i] do
Begin
        Read(NOME);
        Read(NOTA);
End;
```

A ordenação será efetuada com base no nome de cada aluno e quando estiver fora da ordem, os dados deverão ser então devidamente classificados.

```
Program LEITURA_ORDENACAO_ESCRITA;
type
        BIMESTRE = array[1..4] of real;
        CAD_ALUNO = Record
                                NOME : string;
                                NOTA : bimestre;
                        End;
```

```
var
        ALUNO : array[1..8] of cad_aluno;
        I, J : byte;
        X : cad_aluno;

Begin
        (***  Entrada  ***)
        clrscr;
        writeln('Cadastro de aluno');
        writeln;

        for J := 1 to 8 do
        begin
                write('Informe nome do ‘, j:2,’o.aluno...:’);
                readln(ALUNO[J].NOME);
                writeln;
                for I :=1to 4 do
                begin
                        write('Informe a ', i:2, 'a. nota .....: ');
                        readln(ALUNO[J].NOTA[I]);
                end;
                writeln;
        end;
```

```
        writeln;


        (***  Ordenação  ***)
        for I :=1to 7 do
                for J := I + 1 to 8 do
                        if (ALUNO[I].NOME > ALUNO[J].NOME)
                        then begin
                                X := ALUNO[I];
                                ALUNO[I] := ALUNO[J];
                                ALUNO[J] := X;
                end;
                writeln;
                writeln('Tecle <ENTER> para ver o próximo: ');
                readkey;
        end;


(***  Saída  ***]
        writeln;
        for J :=1to 8 do
        begin
                writeln('Nome aluno: ', J:2,' ...: ', ALUNO[J].NOME);
                writeln;
```

```
            for I :=1 to 4 do
                writeln('Nota', I, ' .....:: ', ALUNO[J].NOTA[1]:2:2);
            writeln;
            writeln('Tecle <ENTER> para encerrar: ');
            readkey;
        end;
end.
```

O programa apresentado anteriormente mostra a comparação efetuada entre os nomes dos alunos. Sendo o nome do aluno atual maior que o nome do próximo aluno, é efetuada a troca não só do nome, mas de todos os elementos que estão armazenados na tabela. isto é possível uma vez que a variável X é do mesmo tipo da tabela ALUNO, no caso CAD_ALUNO.

USANDO LISTAS

Uma lista, como array, pode conter uma sequência ordenada de registros (records) com elementos disponíveis de forma consecutiva (Lista Estática Seqüencial) ou não consecutiva (Lista Estática Encadeada).

A linguagem de programação Pascal permite construir estruturas de dados avançadas (Listas Dinâmicas), mais versáteis utilizando *ponteiros* e *variáveis dinâmicas*, já estudado no capítulo 10 desse livro.

LISTA ESTÁTICA SEQUENCIAL

Uma lista estática sequencial é um arranjo de registros onde estão estabelecidos regras de precedência entre seus elementos ou é uma coleção **ordenada** de componentes do mesmo tipo. O sucessor de um elemento ocupa posição física subsequente. Como exemplo podemos citar uma lista telefônica, lista de alunos, lista de e-mails etc.

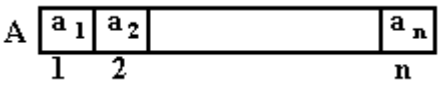
A implementação de operações pode ser feita utilizando *array (vetor)* e *record(registro)*, onde o vetor associa o elemento a(i) com o índice i (mapeamento sequencial).

CARACTERÍSTICAS DE LISTA ESTÁTICA SEQUENCIAL

- elementos na lista estão ordenados;
- armazenados fisicamente em posições consecutivas;
- inserção de um elemento na posição a(i) causa o deslocamento a direita do elemento de a(i) ao último;
- eliminação do elemento a(i) requer o deslocamento à esquerda do a(i+1) ao último;

Mas, absolutamente, uma lista estática sequencial ou é vazia ou pode ser escrita como (a(1), a(2), a(3), ... a(n)) onde a(i) são átomos de um mesmo conjunto S.

Além disso, a(1) é o primeiro elemento, a(i) precede a(i+1), e a(n) é o último elemento.



Assim as propriedades estruturadas da lista permitem responder a questões como:

- qual é o primeiro elemento da lista
- qual é o último elemento da lista
- quais elementos sucedem um determinado elemento
- quantos elementos existem na lista
- inserir um elemento na lista

- eliminar um elemento da lista

Aqui as quatros primeiras operações são feitas em tempo constante. Mas, as operações de inserção e remoção requererão um pouco de mais cuidados.

VANTAGENS

- acesso direto indexado a qualquer elemento da lista
- tempo constante para acessar o elemento *i* - dependerá somente do índice.

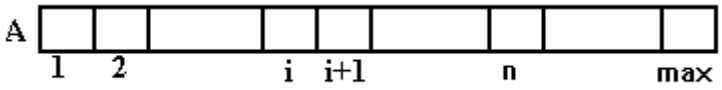
DESVANTAGENS

- movimentação quando eliminado/inserido elemento
- tamanho máximo pré-estimado

DEFINIÇÃO DA LISTA

```
Const
        MAX=100;
Type
```

```
Lista = Record
            A : array[1..MAX] of integer;
            N : 0..MAX;
        End;

Var
    L: lista;
```

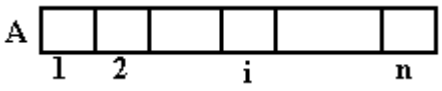


OPERAÇÕES COM LISTAS

1) Acesso a um elemento	Writeln (L.A[i]);
2) Atualização	L.A[i]:= 'Valor'

3) Tamanho da Lista	Begin
	tamanho:=L.n;
	End;

4) Inserção de um elemento na posição i	Requer o deslocamento à direita dos elementos a(i+1)...a(n)

<table>
<tr>
<td>  </td>
<td> 

{ considerando que a posição foi obtida em um procedimento de consulta executado préviamente }
{ também é necessário verificar se a lista não está cheia }
Begin
For j:=L.n+1 downto i+1 do
lista.A[j]:=L.A[j-1];
L.A[i]:='novo valor';
L.n:=L.n+1;
End;</td>
</tr>
<tr>
<td>5) Remoção do i-ésimo elemento</td>
<td>Requer o deslocamento à esquerda dos elementos a(i+1)...a(n)

Begin
For j:=i to L.n-1 do
L.A[j]:=L.A[j+1];
L.n:=L.n-1;
End;</td>
</tr>
</table>

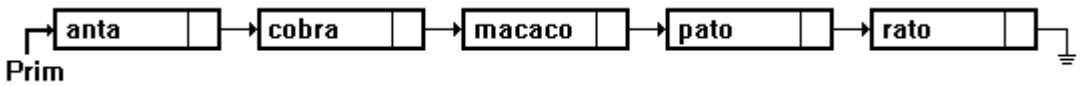
EXERCÍCIOS DE FIXAÇÃO

Dada uma lista sequecial ordenada L1, escreva procedimentos através da linguagem de programação Pascal que:

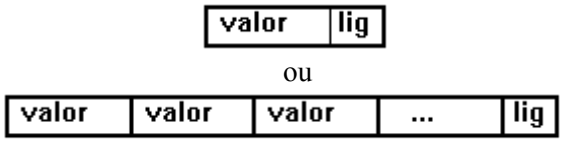
- verifique se L1 está ordenada ou não (a ordem pode ser crescente ou decrescente)
- faça uma cópia da lista L1 em uma outra lista L2;
- faça uma cópia da Lista L1 em L2, eliminando elementos repetidos;
- inverta L1 colocando o resultado em L2;
- inverta L1 colocando o resultado na própria L1;
- intercale L1 com a lista L2, gerando a lista L3. considere que L1, L2 e L3 são ordenadas.
- gere uma lista L2 onde cada registro contém dois campos de informação: *elem* contém um elemento de L1, e *count* contém quantas vezes este elemento apareceu em L1.
- elimine de L1 todas as ocorrências de um elemento dado, L1 ordenada.
- assumindo que os elementos da lista L1 são inteiros positivos, forneça os elementos que aparecem o maior e o menor número de vezes (forneça os elementos e o número de vezes correspondente)

LISTA ESTÁTICA ENCADEADA

Os elementos da lista estática encadeada são registros com um dos componentes destinado a guardar o endereço do registro sucessor. Como exemplo podemos citar as seguinte lista: L = anta, cobra, macaco, pato, rato.

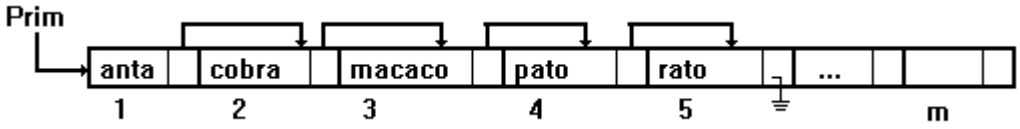


Na lista anterior, cada um resgistro é:

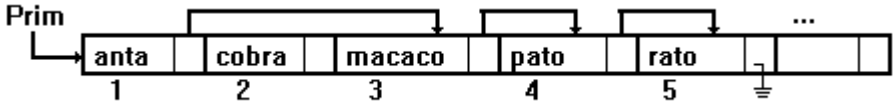


Existem duas alternativas para **implementação de operações** de listas encadeadas: utilizando **arrays** ou variáveis dinâmicas, já vistas no capítulo 10 desse livro.

ENCADEAMENTO EM ARRAYS



Eliminando o elemento "cobra" teremos:



O registro 2 tornou-se disponível para as próximas inserções a serem realizadas. Logo, embora o conteúdo não esteja mais ali fisicamente, o endereço é reservado para o próximo dado a ser fornecido por meio de entrada.

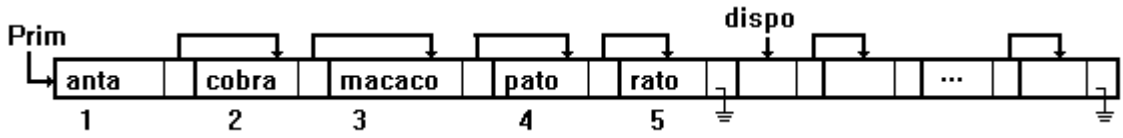
IMPLEMENTAÇÃO UTILIZANDO ARRAY

Após sucessivas inserções e eliminações como descobrir quais registros estão disponíveis? É fácil.

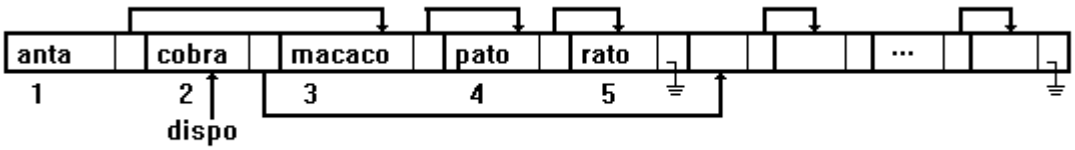
Assim, basta juntá-los numa lista chamada DISPO. Assim, os registros **6**, **7**, ... **n** estariam inicialmente na lista DISPO. Essa lista deve ser capaz de anexar os regis-

tros eliminados da lista principal L. Suponha que queremos inserir algum elemento. Isso implicaria na:

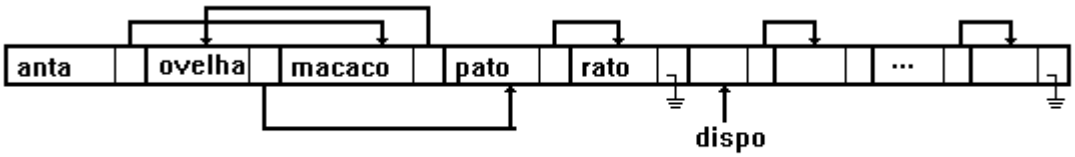
- Eliminação de um elemento da lista principal causa a inserção de um registro na lista Dispo;
- Inserção de um elemento na lista principal causa a utilização de um dos registros da Dispo ;



No nosso exemplo dado, ao eliminar o elemento "cobra" anexamos esse registro à dispo. A princípio podemos utilizar qualquer posição (todos são vazios mesmos!!). A posição mais conveniente é a do primeiro elemento da lista Dispo, uma vez que requer o acesso a poucos ponteiros.



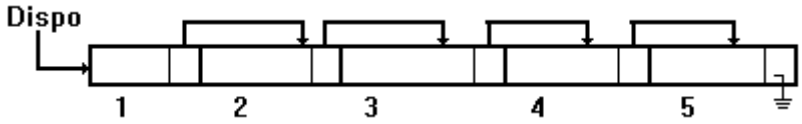
Se a próxima operação é a inserção do elemento ovelha temos:



Com várias inserções e eliminações, os registros da lista principal ficariam espalhados pelo vetor, intermediados por registros disponíveis.

IMPLEMENTAÇÃO DE OPERAÇÕES

Supondo apenas um campo de informação do tipo T:




```
Const
        N = ? ;   { número máximo de elementos }
Type
        Endereço = 0..N;   { elementos armazenados nas posições de
                             1 a N do array; o valor 0 indica fim
                             da lista }
Type
        Rec = record
                info: T
                lig: endereço;
              end;

        Lista = record
                  A: array[1..N] of Rec;
                  Prim, Dispo: endereco;
                end;

Var
        L: lista;
```

1) Qual é o primeiro elemento da lista ?

```
Function Primeiro (L: lista): T;
Begin
        If L.Prim <> 0 Then
        Primeiro := L.A[L.Prim].info;
End;
```

2) Qual é o último ?

```
Function Ultimo(L: Lista): T;
Var
       p: endereco;
Begin
        If L.Prim = 0
                Then {Retorna lista vazia }
                Else
                begin
                        p := L.Prim;
                        While L.A[p].lig <> 0 do
                                p := L.A[p].lig;
                end;
        Ultimo := L.A[p].info;
End;
```

3) Quantos elementos tem a lista ?

```
Function No_elementos(L: Lista): integer;
 Var
          Nelem: integer;
          p: endereco;
 Begin
          If L.Prim = 0
                    Then
                              Nelem := 0 {lista vazia}
                    Else
                    Begin
                              p := L.Prim;
                              Nelem := 1;

                              While L.A[p].lig <> 0 do
                              Begin
                                        Nelem := Nelem + 1;
                                        p := L.A[p].lig;
                              End;
                    End;
          No_elementos := Nelem;
End;
```

INSERÇÃO E ELIMINAÇÃO DE ELEMENTOS

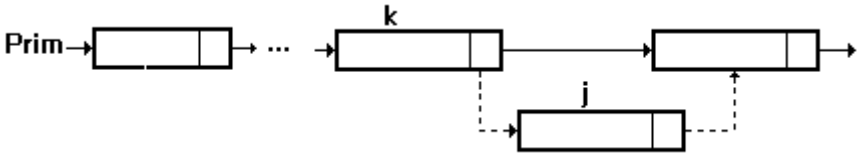
Os algoritmos requerem a mudança de vários ponteiros: do antecessor do registro na lista, do ponteiro da lista Dispo, e do registro a ser inserido/eliminado.

Temos ObterNo(j) e DevolverNo(j), as quais permitem obter um registro de índice j da Dispo, ou devolver o registro de indice j, respectivamente.

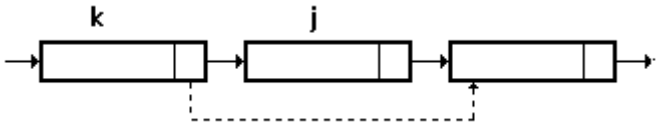
Na inserção podemos contar com o registro j como sendo disponível, e na eliminação podemos contar com o registro j como a ser deixado disponível.

Para inserir ou eliminar um elemento da lista, temos que saber do endereço do predecessor (lembre que a busca é guiada pelo conteúdo do registro, no caso de listas ordenadas). Este predecessor é quem contém o endereço daquele que será o sucessor do elemento inserido/eliminado. Observe atentamente:

INSERÇÃO



ELIMINAÇÃO



1) Inserção após o registro de endereço k

```
Procedure Insere(var L: Lista; k: endereco, valor: T);
 Var
        j: endereco;
```

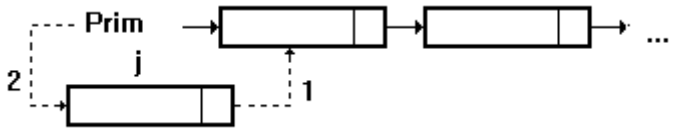
```
Begin
        If L.Dispo <> 0
                Then
                 Begin
                        ObterNo(j);
                        L.A[j].info := valor;
                        L.A[j].lig := L.A[k].lig;
                        L.A[k].lig := j;
                 End
                 Else
                        { Não pode inserir registros ... }
End;
```

2) Eliminação do registro de endereço j precedido por k

```
Procedure Remover(var L: Lista; k,j: endereco);
Begin
        L.A[k].lig := L.A[j].lig;
        DevolverNo(j);
End;
```

3) CASOS ESPECIAIS

INSERÇÃO ANTES DO PRIMEIRO ELEMENTO



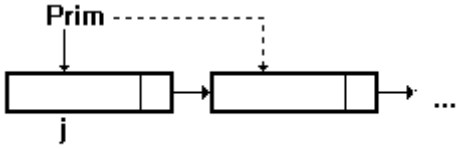

```
Procedure Insere_prim(var L: Lista; valor: T);
Var
        j: endereco;
Begin
        If L.Dispo <> 0
        Then
        Begin
                ObterNo(j);
                L.A[j].info:= valor;
                L.A[j].lig := L.Prim;
                L.Prim := j;
        End
        Else
                { não pode inserir }
End;
```

No caso da lista estar vazia, Prim passa a apontar o único elemento da lista.

INSERÇÃO EM LISTA VAZIA

```
Procedure Insere_ListaVazia(var L: Lista; valor: T);
 Var
        j: endereco;
 Begin
        If L.Dispo <> 0
                Then { lista vazia ou prim = 0 }
                 Begin
                        Obter_No(j);
                        L.A[j].info:=valor;
                        L.A[j].lig:=0; {null}
                        L.Prim:=j;
                 End;
 End;
```

ELIMINAÇÃO DO PRIMEIRO ELEMENTO




```
Procedure Remove_Primeiro(var L: Lista);
 Var
        j: endereco;
```

```
Begin
        j := L.Prim;
        L.Prim := L.A[L.Prim].lig;
        DevolverNo(j);
End;
```

ACESSO AO I-ÉSIMO ELEMENTO A[I].INFO

```
Function Acessa_iesimo(L: Lista; Var dado: T): boolean;
Var
        p,j: endereço;
Begin
        p := L.Prim; { começa do primeiro elemento }
        j := 1;

        While (j < i) and (p <> 0) do
        Begin
                p := L.A[p].lig; { fim := (p=0) }
                j := j + 1; { achou := (j = i) }
        End;

        If p = 0
                Then
                Begin
```

```
                    Acessa_iesimo:= false;
                    dado:=0;
                End
               Else
                Begin
                    Acessa_iesimo:=true;
                    dado:=L.A[p].info;
               End;
End;
```

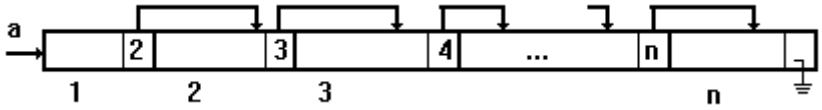
ELIMINAR O REGISTRO DADO PELA VARIÁVEL VALOR

```
Procedure Remove_elem(var L: lista; valor: T);
 Var
        pa, p, endereco;
        acabou: boolean;
 Begin
        pa := 0; { ponteiro anterior }
        p := L.Prim; { busca a partir do primeiro }
        acabou := (p=0); { termina quando fim da lista ]

        While not (acabou) do
        Begin
```

```
                If L.A[p].info <> valor
                        Then
                        Begin
                                pa := p;
                                p := L.A[p].lig;
                                acabou := (p=0);
                        End
                        Else
                                acabou := true;
                End;

        If p=0   Then
                  { não existe o conteúdo `valor' na lista }
                Else
                Begin
                        If pa = 0
                                Then
                                        L.Prim := L.A[L.Prim]lig;
                                Else
                                        L.A[pa].lig := L.A[p].lig;
                        DevolverNo(p);
                End;
End;
```

ALOCAÇÃO DE NÓ EM LISTA ESTÁTICA ENCADEADA

Inicialmente todas as posições do vetor *A* estão disponíveis, portanto fazem parte da lista "Dispo".




```
Procedure Init(L: Lista);
 Begin
        L.Dispo:=1; {primeiro elemento}
        L.Prim:=0; {lista principal está vazia}
        For i:=1 to n-1 do
                L.A[i].lig:=i+1;
        L.A[n].lig:=0; {receber 0 corresponde ao nil}
 End;
```

OBTER UM REGISTRO DE ÍNDICE J DA DISPO

```
Procedure Obter_No(var j: endereco);
 Begin
        j:= L.Dispo; { A dispo passa a apontar para quem ela apontava }
        L.Dispo:= L.A[L.Dispo].lig;
 End;
```

No caso da lista estar vazia, devemos proceder da seguinte maneira:

```
Procedure Devolver_No(j:endereco);
 Begin
        L.A[j].lig := L.Dispo;
       L.Dispo := j;
 End;
```

1- Dada uma lista encadeada ordenada L1, escreva procedimentos na linguagem Pascal que:

a) Verifique se L1 está ordenada ou não (a ordem pode ser crescente ou decrescente)

b) Faça uma cópia da lista L1 em uma outra lista L2;

c) Faça uma cópia da Lista L1 em L2, eliminando elementos repetidos;

d) Inverta L1 colocando o resultado em L2;

e) Inverta L1 colocando o resultado na própria L1;

f) Intercale L1 com a lista L2, gerando a lista L3. considere que L1, L2 e L3 são ordenadas.

g) Gere uma lista L2 onde cada registro contém dois campos de informação: *elem* contém um elemento de L1, e *count* contém quantas vezes este elemento apareceu em L1.

h) Elimine de L1 todas as ocorrências de um elemento dado, L1 ordenada.

i) Assumindo que os elementos da lista L1 são inteiros positivos, forneça os elementos que aparecem o maior e o menor número de vezes (forneça os elementos e o número de vezes correspondente)

2- Escreva os seguintes algoritmos que implementem Listas Encadeadas Estáticas (em array) com **sentinela**:

- Criação de lista;
- Busca em lista ordenada e não ordenada;
- Inserção e eliminação de elementos.

LISTA DINÂMICA

As linguagens de programação modernas tornaram possível explicitar não apenas o acesso aos dados, mas também aos endereços desses dados. Isso significa que ficou possível a utilização de **ponteiros** explicitamente implicando que uma distinção notacional deve existir entre os dados e as referências (endereços) desses dados. Lembra como especificar um ponteiro em Pascal?

```
Type
    Tp = ^T;
```

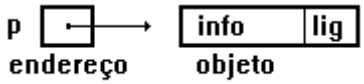
REGISTROS COM PONTEIROS

Dizemos que uma variável do tipo prec *aponta para*, ou é um *ponteiro para* um registro do tipo rec. O ponteiro é o único tipo que pré-referencia outro na linguagem Pascal.

```
Type
    prec = ^rec;
    Lista = prec;
```

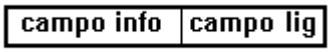
```
rec = Record
        info: T; {seu tipo preferido}
        lig: Lista;
      End;

Var
    p, L: Lista; {nossos ponteiros }
```

Um ponteiro do tipo Lista pode assumir o conjunto de valores que correspondem a endereços reais de memória. Por exemplo, sendo *var p: Lista* podemos ter:



Onde o conteúdo de *p* corresponderia ao endereço do objeto. Esses endereços serão as ligações das listas encadeadas dinâmicas.

Sendo o registro:



O acesso ao registro depende do tipo da alocação. Se compararmos alocação em array com alocação dinâmica com ponteiros, temos que para acessar o conteúdo de uma variável fazemos:

array L.A[p] dinâmica p^

Para *designar* ponteiro, objeto e campos, a notação utilizada é:

ponteiro: p;

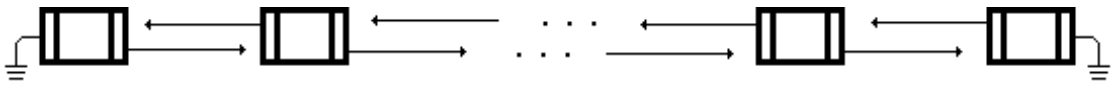
objeto: p^;

campos: p^.info;

p^.lig;

LISTA DUPLAMENTE ENCADEADA

São aquelas listas que são obrigatoriamente percorridas do início ao final. Aqui o ponteiro "anterior" é extremamente necessário para muitas operações. Em alguns casos pode-se desejar percorrer uma lista nos dois sentidos indiferentemente. Nestes casos, o gasto de memória imposto por um novo campo de ponteiro pode ser justificado pela economia em não reprocessar a lista toda.



```
Type
        tpont = ^ trec;
        trec = Record
                    info:Tipoelem;
                    esq, dir: tpont;
                End;
        Lista = tpont;


Var
        pont: Lista;
```

Como consequência, podemos realizar as operações de inserção e eliminação à esquerda ou à direita de um campo no interior de uma lista sem a necessidade de ponteiros "anteriores". Segue abaixo algumas operações de implementação com listas duplamente encadeada com alocação dinâmica:

1) Inserção à direita de pont

```
Procedure ins_dir (Var pont: lista; x: Tipoelem);
Var
        j: Lista;
Begin
        new( j );
        j^.info := x;
        j^.dir := pont^.dir;
        j^.dir^.esq := j;
        j^.esq := pont;
        pont^.dir := j;
End;
```

2) Inserção à esquerda de pont

```
Procedure ins_esq (Var ptlista: Lista; x: Tipoelem);
Var
        j: Lista;
Begin
        new( j );
        j^.info := x;
```

```
        j^.dir := pont;
        j^.esq := pont^.esq;
        j^.esq^.dir := j;
        pont^.esq := j;
End;
```

3) Eliminação à direita de pont

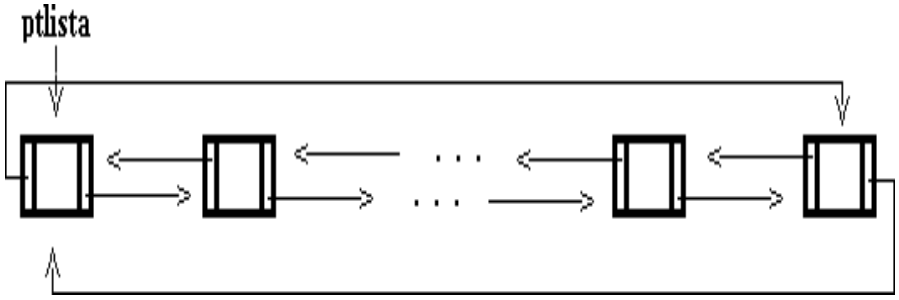
```
Procedure elim_dir (Var pont: Lista);
Var
        j: Lista;
Begin
        j:=pont^.dir;
        pont^.dir:=j^.dir;
        j^.dir^.esq:=pont;
        dispose( j );
End;
```

4) Eliminação do próprio pont

```
Procedure elim (Var pont: Lista);
Begin
        pont^.dir^.esq:=pont^.esq;
        pont^.esq^.dir:=pont^.dir;
        dispose(pont);
```

End;

5) Busca em uma lista circular



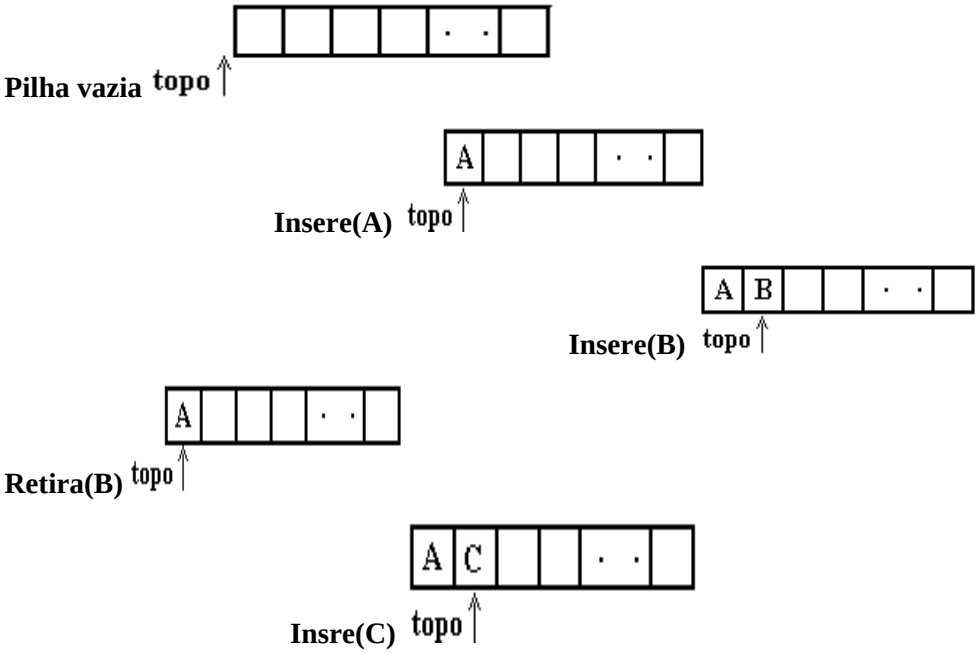

```
Funtion Busca_Dup_Ord(Var ptlista: Lista; x: Tipoelem):Lista;
 Var
        pont, ultimo: Lista;
 Begin
         ultimo:=ptlista^.esq;
         If x <= ultimo^.info
                 then
                 Begin
                         pont:=ptlista^.dir;
                         While pont^.info < x do
                                 pont:=pont^.dir;
                         Busca_Dup_Ord:=pont;
                 End
                 Else
```

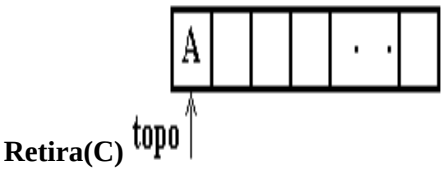
```
Busca_Dup_Ord:=ptlista;
End;
```

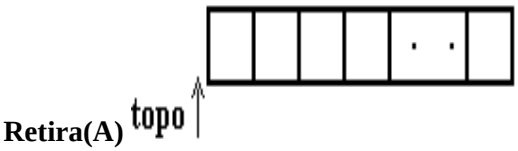
Reescreva os algoritmos de inserção e eliminação de um elemento em uma lista duplamente encadeada de forma a incluir uma chamada à Função Busca_Dup_Ord(x).

USANDO PILHAS

As Pilhas são listas onde a inserção de um novo item ou a remoção de um item já existente se dá em uma única extremidade, no topo.







Dada uma pilha P=(a(1), a(2), ..., a(n)), dizemos que a(1) é o elemento da base da pilha; a(n) é o elemento topo da pilha; e a(i+1) está acima de a(i). Elas são também conhecidas como listas **LIFO** (last in first out).

OPERAÇÕES ASSOCIADAS

1. criar (P) - criar uma pilha P vazia

2. inserir (x, P) - insere x no topo de P (empilha): push(x,P)

3. vazia (P) - testa se P está vazia

4. topo (P) - acessa o elemento do topo da pilha (sem eliminar)

5. elimina (P) - elimina o elemento do topo de P (desempilha): pop(P)

Vejamos a seguir operações de implantação de pilhas:

```
Procedure InicializaPilhas(No:Tno; var T,B:TApont);
var
  i:integer;
begin

  for i:=1 to k+1 do
```

```
    begin
    B[i]:=trunc((i-1)/k*N);
    T[i]:=trunc((i-1)/k*N);
    end;
end;

Procedure RetiraPilha(var No:Tno;var T,B:TApont; i:integer);
begin
  if T[i]=B[i]
     then writeln('Pilha ',i,' vazia')
     else Dec(T[i]);
end;

Procedure InserePilha(var No:Tno;var T,B:TApont;info,i:integer);
var
  achou : boolean;
  L,j   : integer;
begin
  if T[i] = B[i+1]  {testa overflow na pilha i}
    then begin
         {procura pilha com folga à direita}
        achou:=false;
        j:=i+1;
```

```
while (not achou) and (j<=k) do
  begin
  if T[j]<B[j+1]
    then begin
        achou:=true;
        {desloca de B[i+1]+1 até T[j] uma pos. para a direita}

        For L:=T[j] downto B[i+1]+1 do
          No[L+1]:=No[L];
        {acerta bases e topos}

        For L:=i+1 to j do
          begin
          Inc(B[L]); Inc(T[L]);
          end;
        end;
  Inc( j );
  end;
j:=i-1;

while (not achou) and (j>=1) do
  begin
  if T[j]<B[j+1]
```

```
          then begin
              achou:=true;
              {desloca de B[j+1]+1 até T[i] uma pos. para a esquerda}

              For L:=B[j+1]+1 to T[i] do
                No[L-1]:=No[L];
              {acerta bases e topos}

              For L:=j+1 to i do
                begin
                Dec(B[L]); Dec(T[L]);
                end;
              end;
        Dec( j );
        end;


      if not achou
        then writeln('Não há  espaço na  área')
        else begin {insere}
            Inc(T[i]); No[T[i]]:=Info;
            end;
      end
  else begin {Insere}
```

```
        Inc(T[i]); No[T[i]]:=Info;
        end;
end;
```

USANDO FILAS

Filas nada mais são que estruturas lineares de informação que é acessado na ordem FIFO (first in first out - primeiro que entra é o primeiro a sair). O primeiro item colocado na Fila é o primeiro item a ser recuperado, e assim por diante. Uma característica especial da fila é não permitir o acesso randômico à seus dados.

As filas são usadas em muitas situações de programação, tal qual simulações, distribuição de eventos, armazenamento de entrada e saída etc. Considere o seguinte programa exemplo em Pascal, para representar o uso de filas, que utiliza as funções ***qstore***( ) – para realizar armazenamento de dados/elementos em uma certa fila, verificando também se a estrutura já está preenchida (cheia), e ***qretrieve***( ) – para realização de leituras de elementos/itens guardados em uma determinada fila.

OPERAÇÕES ASSOCIADAS

1.Criar (F) - criar uma fila F vazia
2.Inserir (x, F) - insere x no fim de F
3.Vazia (F) - testa se F está vazia
4.Primeiro (F) - acessa o elemento do início da fila
5.Elimina (F) - elimina o elemento do início da fila

DEFINIÇÃO DE UMA FILA

```
Type

        índice = 0..maxfila;
        fila   = array[1..maxfila] of TipoElem;

Var
   F: fila;
   Começo,
   Fim: índice;
```

OPERAÇÕES COM FILAS

1. Criar (F) - criar uma fila F vazia

```
Procedure CriaFila (Var Começo, Fim: indice);
Begin

        Começo := 0;
        Fim := 0;

End;
```

2. Inserir (x, F) - insere x no fim de F

```
Procedure Inserir (x: TipoElem; Var F: fila; Var Fim: indice);
Begin


   If Fim < maxfila Then


   Begin
      Fim := Fim + 1;
      F[Fim] := x;
   End


   Else
    { OVERFLOW }
End;
```

3. Vazia (F) - testa se F está vazia

```
Function Vazia (Começo, Fim: indice): Boolean;
  Begin
  If Começo = Fim Then
    {FILA VAZIA}
  End;
```

4. Primeiro (F) - acessa o elemento do início da fila

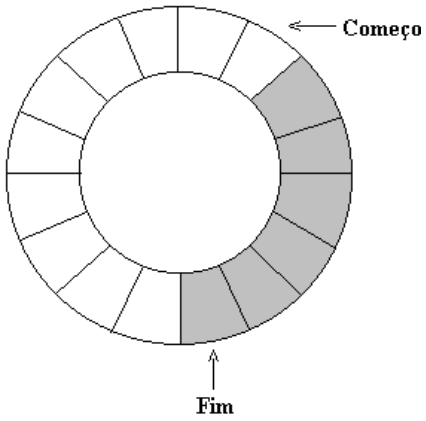
```
Function Primeiro (F: fila; Começo: indice): TipoElem;
Begin
   x := F[Começo + 1];
End;
```

5. Elimina (F) - elimina o elemento do início da fila

```
Procedure Eliminar (Var Começo: indice, Fim: indice);
Begin


    If (Começo = Fim) Then
          {FILA VAZIA}
          Else
                Começo := Começo + 1;
End;
```

FILA CIRCULAR

Na verdade, uma Fila Circular é implementada como se fosse um anel. Observe na explicação abaixo:



PARA INSERIR

```
Procedure Inserir (Var Fim: indice);
Begin
   If Fim = m-1 Then
       Fim := 0
       Else
           Fim := Fim + 1;
End;
```

PARA ELIMINAR UM ELEMENTO

```
Procedure Eliminar (Var Começo: indice);
Begin
    Começo := (Começo + 1) mod m;
End;
```

1) Seja a função esvazie( ) tal que, recebendo uma pilha como entrada, esvazie a pilha descartando todos os seus elementos. Escreva a função esvazie( ).

2) Escreva um programa que verifique que expressões aritméticas estão com a parentização correta. Armazene o resultado numa pilha também. Seu programa deve checar expressões para ver se cada "abre parênteses " tem um "fecha parênteses" correspondente.

3) Escreva um algoritmo que converta uma expressão escrita na notação parentizada no seu equivalente na notação polonesa reversa.

4) Uma palavra é uma palíndrome se a seqüência de letras que a forma é a mesma seja ela lida da esquerda para a direita ou vice-versa. Exemplos: arara, rairar, hanah. Escreva a função palíndrome que, dada uma palavra, retorne true caso a palavra seja uma palíndrome, e false caso contrário.

USANDO ÁRVORES

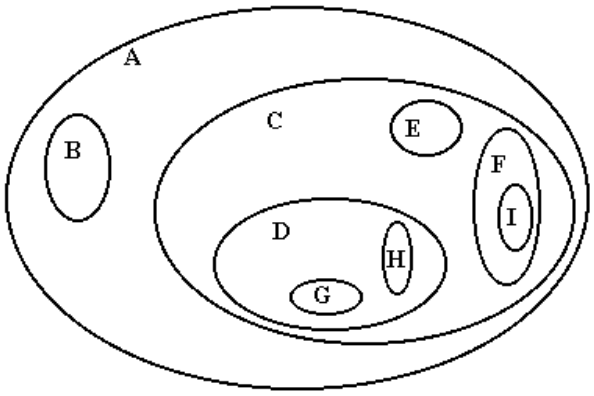
Embora existam muitos tipos de árvores, as árvores binárias são especiais porque, quando são ordenadas, elas se aplicam a buscas velozes, inserções e deleções. Cada item de uma árvore binária consiste em informação com um elo no ramo esquerdo e um no ramo direito.

A tecnologia especial necessária para discutir árvores é um caso clássico de metáforas misturadas. A raiz é o primeiro item da árvore. Cada item (dado) é chamado de nó (ou algumas vezes folhas) da árvore e qualquer parte da árvore de sub-árvore. A altura da árvore é igual ao número de camadas (de profundidade) que as raízes atingem.

As árvores binárias é uma forma especial de lista encadeada. Pode-se inserir, deletar e acessar itens em qualquer ordem.

A ordenação de uma árvore depende exclusivamente de como ela será referenciada. O procedimento de acesso a cada nó é chamado de ***árvore transversal***. Observe exemplo abaixo as três formas de representação de uma árvore:

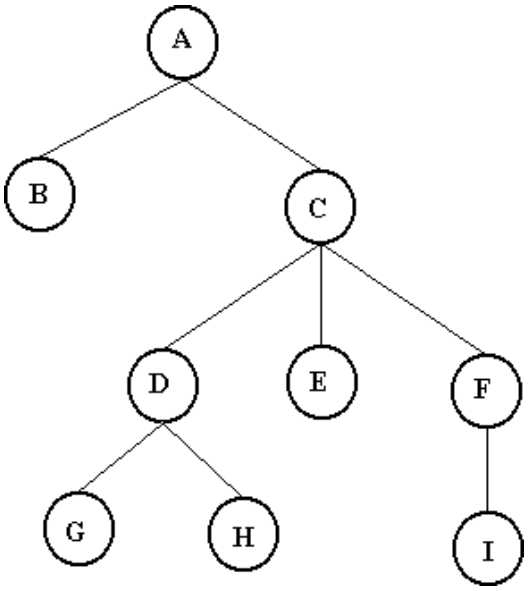
REPRESENTAÇÃO POR DIAGRAMA DE INCLUSÃO



REPRESENTAÇÃO POR PARÊNTESES ANINHADOS

(A (B) (C (D (G) (H)) (E) (F (I))))

REPRESENTAÇÃO HIERÁRQUICA



BUSCA EM ÁRVORE BINÁRIA

```
Procedure BuscaArvore(x, pt, f );
Begin
        If pt = LAMBIDA   { Constante definida no programa }
                Then
                        f := 0
                Else
                If x = pt↑.chave
                        Then
                        If pt↑.esq = LAMBIDA
```

```
                    Then
                        f := 2
                    Else
                    Begin
                        pt := pt↑.esq;
                        BuscaArvore(x, pt, f);
                    End
                Else
                If pt↑.dir = LAMBIDA
                    Then
                        f := 3
                    Else
                    Begin
                        pt := pt↑.dir;
                        BuscaArvore(x, pt, f);
                    End;
        pt := ptraiz;
        BuscaArvore(x, pt, f);
End;
```

A seguir vamos fazer algumas operações básicas com árvores binárias. Essas operações são consideradas por mim como essenciais para qualquer tratamento com árvores.

OPERAÇÕES COM ÁRVORES BINÁRIAS

DEFINE UMA ÁRVORE VAZIA

```
Procedure Define(var t: tree);
Begin
        T := nil;
 End;
```

CRIA UM NÓ RAIZ

```
Procedure Cria_Raiz(var t: tree; item: Telem);
Var
        No : tree;
Begin
        new(nó);
        no^.esq:=nil;
        no^.dir:=nil;
        no^.info:=item;
        t:=no;
End;
```

VERIFICA SE ÁRVORE VAZIA OU NÃO

```
Function Vazia (t:tree):boolean;
Begin
        Vazia := (t = nil);
End;
```

CRIAR UM FILHO À DIREITA DE UM DADO NÓ

```
Procedure AdicionarDireita(t: tree; item_pai, item: Telem)
Var
        pai, no: tree;
 Begin
         pai:=Localiza(t,item_pai);
         If(pai<>nil) Then
                 If (pai^.dir<>nil) Then
                          erro(´item já possui subárvore direita´)
                         Else
                         Begin
                                 New(no);
                                 no^.esq:=nil;
                                 no^.dir:=nil;
                                 no^.info:=item;
                                 pai^.dir:=no;
```

```
                End;
End;
```

CRIAR UM FILHO À ESQUERDA DE UM DADO NÓ

```
Procedure AdicionarEsquerda(t: tree; item_pai, item: Telem)
Var
        pai, no: tree;
Begin
        pai:=Localiza(t,item_pai);
        If(pai<>nil) Then
                If (pai^.esq<>nil) Then
                        erro(´item já possui subárvore esquerda´)
                        Else
                        Begin
                                New(no);
                                no^.esq:=nil;
                                no^.dir:=nil;
                                no^.info:=item;
                                pai^.esq:=no;
                         End;
End;
```

VERIFICAR QUAL O NÍVEL DE UM DADO NÓ

```
Function Nível (t: tree; item: Telem): integer;
Var
        p: integer;

 Procedure Travessia (ptr: tree; niv: integer);
 Begin
        If ptr<>nil Then
         Begin
                niv:=niv+1;
                If Igual(ptr^.info, item) Then
                                p:=niv;
                         Else
                        Begin
                                Travessia(ptr^.esq, niv);
                                If p=0 Then
                                        Travessia(ptr^.dir, niv);
                         End;
        End;
End;

Begin
        p:=0;
```

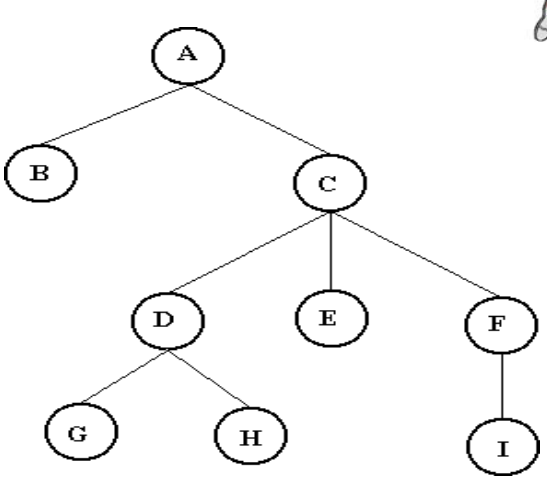
```
        Travessia(t,p);
        nivel:=p;
End;
```

RETORNAR O PAI DE UM DADO NÓ

```
Function Pai(t: tree; item:Telem):Telem;
Var
        achou: boolean;
        it : Telem;

Procedure Travessia(t: tree; var it: Telem);
Begin
    If not Vazia(t) Then
    Begin
        If t^.esq<>nil Then
            If Igual(item, t^.esq^.info) Then
            Begin
                achou:=true;
                it:=t^.info;
            End;
            If not achou Then
                If t^.dir<>nil Then
                    If Igual(item, t^.dir^.info) Then
                        Begin
                            achou:=true;
                            it:=t^.info;
                        End;
                    If not achou Then
                        Travessia(t^.esq, it);
                        If not achou Then
                            Travessia(t^.dir, it);
    End;
End;
```

```
Begin   { Definição do pai }


If t<>nil Then
Begin
      achou:=false;
      If Igual(item, t^.info) Then
            pai:=t^.info;   {pai do raiz é ele próprio}
            Else
            Begin
               Travessia(t,it);
                pai:=it;
            End;
    End;

End;
```

1. Reescreva Para a árvore abaixo:



a) Quantas subárvores ela contém?

b) Quais os nós folhas?

c) Qual o grau de cada nó?

d) Qual o grau da árvore?

e) Liste os ancestrais dos nós B, G e I.

f) Identifique todas as relações de parentesco entre os nós

g) Justifique: podemos dizer que uma árvore é um dígrafo conexo onde existe um único nó sem predecessor e todos os demais têm um único predecessor.

2. Dada uma árvore cujo grau da raiz é d, como transformá-la em uma floresta com d árvores?

3. Dada uma floresta com d árvores como transformá-la em uma árvore cujo nó raiz tem grau d ?

4. Para a árvore do exercício 1:

a) Liste os vértices de quem C é ancestral próprio

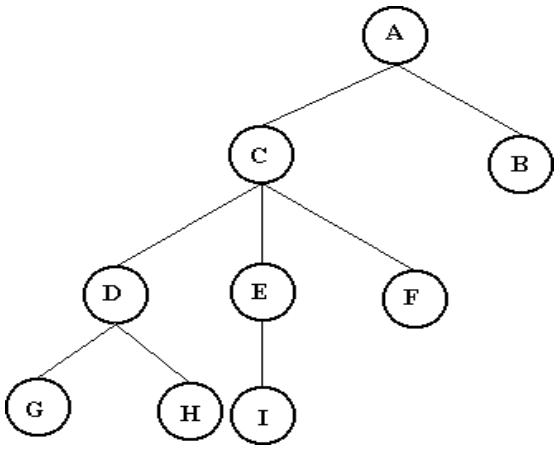
b) Liste os vértices de quem D é descendente próprio

c) Dê o nível e altura do vértice F.

d) Dê o nível e a altura do vértice A.

e) Qual a altura da árvore ?

5. Explique porque a árvore do primeiro exercício e a árvore abaixo são isomórfas.



6. Para uma árvore de grau d com número máximo de nós, diga:

    a) Qual o grau dos nós internos da árvore?

    b) Qual o grau dos nós folhas?

    c) Quantos nós tem a árvore se o grau d e a altura é h?

    d) Qual a altura da árvore se o grau é d e o número de nós é n?

7. Para uma árvore cheia de grau d, responda:

    a) Se um nó estiver armazenado na posição i de um array, em que posições estarão seus d filhos?

    b) Considerando esta alocação sequencial, quais as consequências de inserções e eliminações na árvore?

12
MANIPULAÇÃO COM ARQUIVOS

Um arquivo, na verdade representa uma tabela em disco (disquete, disco magnético, fita cassete, fita magnética, compact disc etc.). Com ele podemos executar algumas operações básicas, tais como: abertura, leitura ou escrita e fechamento de um arquivo, sendo que estas operações serão conseguidas na linguagem de programação Pascal com a utilização de algumas instruções apropriadas como por exemplo: **Assign**, **rewrite**, **reset**, **write** (**writeln**), **read** (**readln**) e **Close**, que poderão ser utilizadas em qualquer tipo de arquivo. Iniciemos então o nosso estudo propriamente dito:

INSTRUÇÃO ASSIGN

Esta instrução tem por finalidade associar um nome lógico de arquivo (nome interno do programa) ao arquivo físico (aquele que fica gravado em disco). Observe atentamente sua sintaxe de utilização:

Assign (<variável>,<arquivo>);

INSTRUÇÃO REWRITE

Esta instrução tem por finalidade criar um arquivo para gravação de dados, utilizando o nome associado ao parâmetro, variável. Caso o arquivo já exista, esta instrução o apagará recriando-o assim. Observe atentamente sua sintaxe de utilização:

Rewrite (<variável>);

INSTRUÇÃO RESET

Esta instrução tem por finalidade abrir um arquivo já existente para leitura de suas informações, disponibilizando-o também para escrita, utilizando o nome associado ao parâmetro <variável>. Observe atentamente sua sintaxe de utilização:

Reset (<variável>);

INSTRUÇÃO WRITE

Esta instrução tem por finalidade escrever (gravar) dados no arquivo indicado pelo parâmetro <variável>. Observe atentamente sua sintaxe de utilização:

Write (<variável>,<dado>);

INSTRUÇÃO READ

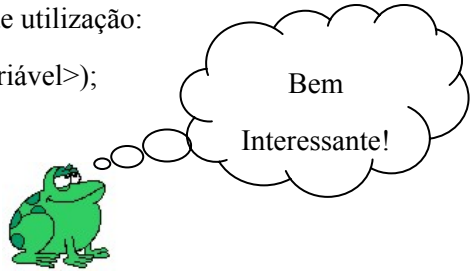
Esta instrução tem por finalidade ler a informação no arquivo indicado pelo parâmetro <variável> para a memória do computador. Observe atentamente sua sintaxe de utilização:

Read (<variável>,<dado>);

INSTRUÇÃO CLOSE

Esta instrução tem por finalidade fechar um arquivo que esteja aberto na memória do computador (em uso dentro pelo programa). Nenhum programa deve ser encerrado sem fechar os arquivos que estejam devidamente abertos na memória do computador. Observe atentamente sua sintaxe de utilização:

Close (<variável>);



FORMAS DE ACESSO A ARQUIVOS

Os arquivos criados em meios magnéticos poderão ser acessados para leitura e escrita na forma seqüencial, direta ou indexada. Vamos estudar cada uma delas detalhadamente a seguir.

ACESSO SEQÜENCIAL

O acesso seqüencial ocorre quando o processo de gravação e leitura é feito de forma continua, um após o outro. Desta forma, para se gravar um novo registro é necessário percorrer todo o arquivo a partir do primeiro registro, registro a registro, até localizar a primeira posição vazia após o último registro. O processo de leitura também ocorre de forma seqüencial. Se o registro a ser lido é o último, primeiro será necessário ler todos os registros que o antecedem até chegar a ele.

ACESSO DIRETO

O acesso direto ou também conhecido como acesso randômico, ocorre através de um campo chave previamente definido. Desta forma, passa-se a possuir um vinculo existente entre um dos campos do registro e sua posição de armazenamento, através da chave. Assim sendo, o acesso a um registro tanto para leitura como para escrita poderá ser feito de forma instantânea. Isto implica no fato de que os registros de um arquivo direto possuem um lugar (chave) previamente "reservado" para serem armazenados.

ACESSO INDEXADO

O acesso indexado ocorre quando se acessa de forma direta um arquivo seqüencial. Na sua maioria, todo o arquivo criado armazena os registros de forma seqüencial. A forma seqüencial de acesso se torna inconveniente, pois á medida que o arquivo aumenta de tamanho, aumenta também o tempo de acesso ao mesmo. Para trabalharmos com esta técnica, basta criar um arquivo direto que será o índice de consulta do arquivo seqüencial, passando este a ser o arquivo indexado. Assim sendo, existirão dois arquivos: o arquivo índice (arquivo direto) e o arquivo indexado (ar-

quivo seqüencial). Os acessos serão feitos como em um livro, primeiro se consulta o arquivo índice, o qual possui a chave de pesquisa, no caso seria o número da página, depois basta se posicionar de forma direta na página (no caso de um livro) identificada no índice, ou seja, no registro do arquivo indexado.

ARQUIVOS TEXTO

Iniciaremos o nosso estudo prático com o tipo de arquivo mais simples utilizado em Pascal, o tipo texto, que permite possuir armazenados registros com tamanhos diferentes (o que não ocorre com os outros tipos de arquivo). Se você direcionar o prompt do DOS para o diretório BP terá como exemplo de um arquivo texto o arquivo FILELIST.DOC que poderá ser visualizado com o comando DOS:

```
Type
        filelist.doc;
```

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

OPERAÇÕES COM ARQUIVOS

CRIANDO UM ARQUIVO

Antes de iniciar qualquer operação com arquivo, obviamente, é necessário criá-lo. Então, vamos fazer a criação do nosso primeiro arquivo em Pascal, digitando o programa a seguir e gravando-o com o nome de ARQTXT01.

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

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

Em seguida é utilizada a instrução **assign** (ARQUIVO_TXT, 'ARQTXT.XXX'); , que efetua a associação do nome do arquivo (ARQTXT.XXX) á variável de controle ARQUIVO_TXT a qual será utilizada em todas as operações do programa para fazer referência ao arquivo em uso. Depois é utilizada a instrução **rewrite** (ARQUIVO_TXT);, que cria o arquivo ARQTXT.XXX, mantendo-o aberto. Por fim é utilizada a instrução **close** (ARQUIVO_TXT); , que efetua o fechamento do arquivo criado.

GRAVANDO INFORMAÇÕES EM UM ARQUIVO

Tendo sido o arquivo criado este poderá ser agora utilizado para a gravação das informações que irá guardar. Digite o próximo programa e grave-o com o nome ARQTXT02.

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

O programa acima estabelece uma variável MENSAGEM do tipo **string** com a capacidade de armazenar até 50 caracteres. Para a variável ARQUIVO_TXT o identificador **text**, tem por finalidade definir para a variável o tipo de arquivo texto.

Em seguida é utilizada novamente a instrução **assign** (ARQUIVO_TXT 'ARQTXT.XXX');. Depois é utilizada a instrução **append** (ARQUIVO_TXT); , que abre o arquivo ARQTXT.XXX para a inclusão de um dado, posicionando o ponteiro de controle de registros sempre ao final do arquivo, ou seja, se o arquivo já possuir algum conteúdo, o próximo será sempre colocado ao final do último cadastrado anteriormente (acrescenta novos registros).

A instrução **readln** solicita a entrada de um dado, depois a instrução **writeln** (ARQUIVO_TXT MENSAGEM); faz a gravação no arquivo representado pela variável de controle ARQUIVO_TXT do conteúdo armazenado na variável MENSAGEM. Observe que a instrução **write** tinha sido usada até este momento

para escrever dados na tela, e agora está sendo utilizada para escrever uma informação no arquivo em uso. Ao final, o arquivo é fechado.

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

LENDO INFORMAÇÕES DE UM ARQUIVO

A seguir, é apresento um programa que irá efetuar a leitura de um arquivo texto, mostrando no vídeo a informação armazenada com o programa anterior. Digite o programa abaixo e grave-o com o nome ARQTXT03.

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

A instrução **reset**(ARQUIVO_TXT); faz apenas a abertura do arquivo. Não confundir com a instrução **append** que faz a abertura do arquivo posicionando o ponteiro de controle sempre após a última linha existente no arquivo. Depois, a instrução **readln** (ARQUIVO_TXT MENSAGEM); faz a leitura do registro no arquivo, transferindo a informação lida para a variável MENSAGEM que será apresentada no vídeo pela instrução **writeln** (MENSAGEM);.

UTILIZAÇÃO DE ARQUIVOS TEXTO

Neste tópico, você terá contato com um exemplo de utilização de arquivo texto em um pequeno programa de agenda. O programa de agenda proposto deverá solicitar apenas o nome e o telefone de uma pessoa. Este programa deverá ter um menu contendo quatro opções: criar arquivo, cadastrar registro, exibir registros cadastrados e finalizar o programa. Para tanto, observe o algoritmo a seguir:

Programa Principal

1. Definir as variáveis de controle e abertura do arquivo

2. Apresentar um menu de seleção com quatro opções:

a. Criar arquivo

b. Cadastrar registro

c. Exibir registros

d. Fim de Programa

3. Ao ser selecionado um valor, a rotina correspondente deverá ser executada;

4. Ao se escolher o valor 4, encerrar o programa.

Rotina 1 - Criar arquivo

1. Executar o comando de criação de um arquivo;

2. Fechar o arquivo;

3. Voltar ao programa principal.

Rotina 2 - Cadastrar registro

1. Abrir o arquivo para cadastramento, posicionando o ponteiro após último registro;

2. Ler o nome e o telefone;

3. Escrever os dados no arquivo;

4. Fechar o arquivo;

5. Voltar ao programa principal.

Rotina 3 - Exibir registro

1. Abrir o arquivo para leitura, posicionando o ponteiro no primeiro registro;

2. Apresentar os dados do arquivo enquanto não for encontrado o último registro;

3. Fechar o arquivo;

4. Voltar ao programa principal.

Apesar de simples, o programa dará uma boa visão, pois será necessário trabalhar com alguns novos recursos. Digite o programa a seguir, e grave-o com o nome AGENDTXT.

```
Program AGENDATXT;
uses
    Crt;
var
    ARQTXT : text;
```

```
        NOME : string [40];
        TELEFONE : string[8];
        TECLA : char;
        OPCAO: char;

(*** Rotinas de Visualização ***)

Procedure CENTER ( MENSAGEM : string);
var
        TAMANHO : integer;
begin
        TAMANHO := 40 + length (MENSAGEM) div 2;
        Writeln (MENSAGEM:TAMANHO);
end;

Procedure WRITEXY (X, Y : byte; MENSAGEM : string);
begin
        GOTOXY (X, Y); write (MENSAGEM);
end;

Procedure LINE;
var
        I : byte;
```

```
begin
        for I := 1 to 80 do
                write (#205);
end;

(*** Rotinas de Manipulação de Arquivos ***)
Procedure ARQUIVO;
begin
        clrscr;
        line;
        center ('Criação de Arquivo');
        line;
        rewrite(ARQTXT);
        gotoxy ( 1,12); Center ('Arquivo foi criado');
        writexy (25,24,'Tecle algo para voltar ao menu');
        TECLA := Readkey;
        close(ARQTXT);
end;

Procedure CADASTRA;
begin
        clrscr;
        line;
```

```
        center ('Cadastro de Registro');
        line;
        append (ARQTXT);
        writexy (10, 5, 'Entre com o Nome .....: ');
        readln (NOME);
        writexy (10, 6, 'Entre com o Telefone .....: ');
        readln (TELEFONE);
        writeln (ARQTXT, NOME);
        writeln (ARQTXT, TELEFONE);
        writexy (25,24,'Tecle algo para voltar ao menu');
        TECLA := Readkey;
        close (ARQTXT);
end;

Procedure EXIBIR;
var
        LINHA : byte;
begin
        clrscr;
        line;
        center ('Apresentação de Registros');
        line;
        LINHA := 5;
```

```
        Reset (ARQTXT);

        while not eof (ARQTXT) do
        begin
                readln (ARQTXT, NOME);

                readln (ARQTXT, TELEFONE);
                gotoxy ( 5,LINHA); write (NOME);
                gotoxy (50,LINHA); write (TELEFONE);
                LINHA := LINHA + 1;
        end;

        writexy (25,24,'Tecle algo para voltar ao menu' );
        TECLA := Readkey;
        Close (ARQTXT);
end;


(*** Programa Principal ***)
begin
        OPCAO := '0';
        Assign (ARQTXT, 'AGENDTXT.DAT');
```

```
while (OPCAO <> '4') do
begin
        clrscr;
        line;
        center ('menu Principal');
        line;
        gotoxy (28, 6);
        write ('l ..... criar arquivo');
        gotoxy (28, 8);
        write ('2 ..... cadastrar');
        gotoxy (28,10);
        write ('3 ..... Exibir registros');
        gotoxy (28,12);
        write ('4 ..... Fim de Programa');
        gotoxy (28,16);
        write ('Escolha uma opção .....: ');
        readln (OPCAO);

        if (OPCAO <> '4' ) then
                case OPCAO of
                '1' : Arquivo;
                '2' : Cadastra;
                '3' : Exibir;
```

```
                    else
                            gotoxy (27,24);
                            writeln ('Opção Invalida - Tecle algo');
                            opcao := Readkey ;
                    end;
            end;
end.
```

Ao fazer uso do programa, procure cadastrar poucos registros (ideal em torno de 15), pois não está sendo previsto exibir uma quantidade muito grande de dados na tela. Perceba que o programa principal associa á variável ARQTXT o arquivo AGENDTXT.DAT, que será usado nas rotinas de manipulação.

Na rotina EXIBIR, está sendo usada uma nova instrução, **eof( ),** sendo esta uma função que consegue identificar o final do arquivo (**eof - end of file**). Perceba que serão apresentados na tela os registros enquanto não for o fim de arquivo. Um arquivo, quando é fechado, grava na última posição um caractere de controle que pode ser identificado pela função eof( ). No restante, os demais recursos do programa já são conhecidos. Lembre-se ainda que a instrução **reset** ( ) faz a abertura do arquivo, colocando o ponteiro de registro posicionado na primeira linha de texto.

Depois de cadastrar alguns registros, se for solicitado efetuar a criação do arquivo novamente, este será recriado, destruindo os dados já cadastrados.

ARQUIVOS COM TIPO DEFINIDO

Os arquivos com tipo definido caracterizam-se por serem do tipo binário, diferente dos arquivos do tipo texto. Este arquivo permite armazenar específicos, podendo ser: integer, real, record, entre outros. Um arquivo com tipo definido executa as

operações de escrita e leitura mais rápido do que os arquivos textos. Os arquivos de tipo definido estão capacitados a armazenar dados na forma de registro.

CRIANDO UM ARQUIVO

Antes de mais nada, será criado um arquivo que esteja capacitado a receber dados numéricos inteiros. Para tanto, digite o programa abaixo e grave-o com o nome ARQINT01.

```
Program CRIA_ARQUIVO_INTEIRO;

Uses crt;

var

        ARQUIVO_INT : file of integer;

begin

        Assign (ARQUIVO_INT, 'ARQINT.XXX');

        Rewrite (ARQUIVO_INT);

        Close (ARQUIVO_INT);

end.
```

Após rodar o programa, o arquivo será criado. Caso queira verificar se o arquivo foi realmente criado, saia temporariamente para o sistema operacional e execute o comando DOS: *dir arqint.xxx.*

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

GRAVANDO INFORMAÇÕES EM UM ARQUIVO

Estando o arquivo criado, este poderá ser agora utilizado para a gravação dos dados de tipo inteiro. Digite o programa abaixo e grave-o com o nome ARQINT02.

```
Program CADASTRA_ARQUIVO_INTEIRO;
uses Crt;
var
        ARQUIVO_INT : file of integer;
        NUMERO : integer;
        RESF : char;
begin
        assign (ARQUIVO_INT, 'ARQINT.XXX');
        reset (ARQUIVO_INT);
        RESP :='S';

        while (RESP ='S') or (RESP ='s') do
        begin
                clrscr;
                writeln ('Gravação de Registros Inteiros');
                writeln ;
                seek (ARQUIVO_INT, filesize (ARQUIVO_INT));
                write ('Informe um numero inteiro: ');
                readln (NUMERO);
                write (ARQUIVO_INT, NUTERO);
```

```
            writeln ;
            write ('Deseja continuar? (S/N) ');
            readln (RESP);
        end;
        Close (ARQUIVO_INT);
end.
```

No programa acima é utilizada a instrução **assign** (ARQUIVO_INT 'ARQINT.XXX'); para associar o arquivo á variável de controle. Faz uso da instrução **reset** (ARQUIVO_INT); para abrir o arquivo. Depois, um pouco mais abaixo é apresentada uma linha com a instrução **seek** (ARQUIVO_INT **filesize** (ARQUIVO_INT)); que executa em um arquivo tipado o mesmo efeito que a instrução **append** executa em um arquivo texto, ou seja, se o arquivo já possuir algum registro, o próximo registro será sempre colocado logo após o último cadastrado.

A instrução **seek** (ARQUIVO_INT **Filesize** (ARQUIVO_INT)); é formada por duas funções distintas. Sendo **Filesize ( )** a função destinada a retornar o número de registros de um arquivo e a função **Seek ( )** utilizada para colocar o ponteiro de registro em uma determinada posição do arquivo.

A função **Filesize ( )** irá retomar o valor zero, caso o arquivo esteja vazio. Utilizada com arquivos não tipados, esta função considera que os registros deste arquivo possuem o tamanho de 128 caracteres. Se o arquivo em uso tiver um tamanho maior que 32 Kb, você deverá usar a função **Longfilesize ( ).** No caso da função **Seek ( ),** deverá ser utilizada a função **Longseek ( )** caso o arquivo seja maior que 32 Kb.

LENDO INFORMAÇÕES DE UM ARQUIVO

A seguir, é apresentado um programa que irá efetuar a leitura de um arquivo de números inteiros, mostrando no vídeo a informação armazenada com o programa anterior. Digite o programa a seguir e grave-o com o nome ARQINT03.

```
Program LE_ARQUIVO_INTEIRO;
uses Crt;
var
        ARQUIVO_INT : file of integer;
        NUMERO : integer;
begin
        clrscr;
        assign (ARQUIVO_INT, 'ARQINT.XXX');
        reset (ARQUIVO_INT);

        while not eof (ARQUIVO_INT) do
        begin
                read (ARQUIVO_INT, NUMERO);
                writeln (NUMERO);
        end;
        Close (ARQUIVO_INT);
        Readln;
end.
```

O programa acima é basicamente o mesmo utilizado para apresentar os dados de um arquivo do tipo texto.

Com os programas anteriores você teve um contato com o trabalho executado com arquivos de tipo definido. Apresentamos apenas como sendo do tipo inteiro, uma vez que para os demais tipos as operações serão idênticas.

UTLIZAÇÃO DE ARQUIVOS COM TIPO DEFINIDO

A seguir, serão apresentadas duas rotinas que exemplificarão o uso de arquivos com matrizes de uma dimensão. A primeira rotina deverá solicitar a entrada de dez valores inteiros, armazenar os valores em uma matriz para transferi-los a um arquivo. Depois deverá ser escrita uma segunda rotina que leia os dados do arquivo, transferindo-os para uma matriz, para então apresentá-los.

Digite o programa a seguir, e grave-o com o nome MATINTO1. Observe que será criado um arquivo chamado MATINT.DBP que conterá dez valores inteiros. Perceba que assim que os elementos são fornecidos para a matriz A, estes são em seguida gravados no arquivo com a instrução **write** (ARQ, A[I]).

```
Program MATRIZ_E_ARQUIVO;

Uses Crt;

var

        A : array [1..10] of integer;

        I : integer;

        ARQ : file of integer;

begin

        assign (ARQ. 'MATINT.DBP');

        rewrite (ARQ):

        clrscr;

        I := 1;


        while (I<=10) do

        begin

                write ('Digite o elemento - ' , I:2, ': ' );
```

```
            readln (A([I]);
            write (ARQ, A[I]);
            I := I + 1;
        end;
        Close (ARQ);
end.
```

Digite o programa a seguir, e grave-o com o nome MATINT02. Observe que o programa fará a abertura e a leitura do arquivo MATINT.DBP que contém dez valores inteiros. Perceba que pelo fato de saber a quantidade de registros armazenados no arquivo, a leitura está sendo feita com a instrução **while** (I<= 10) **do**, e não com a instrução **while not eof** (ARQ) **do**, apesar de também poder ser utilizada.

```
Program ARQUIVO_E_MATRIZ;
Uses Crt;
var
        A : array[1..10] of integer;
        I : integer;
        ARQ : file of integer;
begin
        assign (ARQ, 'MATINT.DBP');
        reset (ARQ);
        clrscr;
        I := 1;
```

```
        while (I <=10) do
        begin
                read(ARQ, A(1]);
                write ('O registro ', 1:2, ' contem o elemento: ');
                writeln (A[I]:2);
                I := I + 1;
        end;
        Close (ARQ);
end.
```

ARQUIVO COM TIPO DEFINIDO DE REGISTRO

Vejamos um exemplo mais profissional com a utilização de arquivos gerenciados pela linguagem Pascal. Para tanto, serão criadas rotinas básicas que deverão fazer o controle de um arquivo do tipo **record**.

ARQUIVO DE ACESSO DIRETO

Nos exemplos de arquivos anteriormente estudados, não se levou em consideração a forma de acesso aos arquivos (todos os exemplos foram criados baseando-se na forma de acesso seqüencial). Para o exemplo que será construído (gerenciamento de notas escolares) no final deste módulo, será utilizada a forma de acesso direta, pois este tipo de arquivo permite o acesso imediato a cada um dos seus registros, desde que se conheça previamente a posição em que estes registros estão gravados.

As operações de leitura e escrita em um arquivo de acesso seqüencial ocorrem através do apontamento de um indicador de registro, que é avançado a cada operação executada, do primeiro até ao último registro. Este indicador de registro é o ponteiro do arquivo (similar ao índice de uma tabela em memória - matriz). No caso de arquivos de acesso direto, o ponteiro pode ser movimentado para qualquer

posição do arquivo, através da instrução **seek**, já exemplificada. Desta forma, é possível se posicionar em qualquer registro do arquivo.

As operações que são executadas em um arquivo são semelhantes ás operações executadas em matrizes, podendo ser: alteração, remoção, classificação, pesquisa binária, pesquisa seqüencial, cadastramento, entre outras. A diferença entre uma matriz e um arquivo de acesso direto é que as operações do arquivo são executadas diretamente no arquivo.

UTILIZAÇÃO DE ARQUIVO DIRETO

Para este exemplo será considerado um arquivo para gerenciar o cadastro de 'n' alunos de uma escola, que deverá possuir os campos matricula, nome e notas, conforme a seguir:

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

Perceba que na estrutura acima, existe um campo denominado FLAG com a capacidade de guardar apenas um caractere. Este campo será utilizado especialmente na operação de remoção lógica de um determinado registro, o qual guardará o caractere '*' asterisco (este procedimento não é obrigatório, sendo apenas uma definição).

Então, dessa forma, será possível detectar se algum registro foi "removido". Estando um registro removido, poder-se-á gravar um outro registro sobre o mesmo através do controle do campo FLAG. Para o programa que será construído a seguir, o

campo FLAG poderá guardar um de dois valores, o asterisco "*" para remoções lógicas ou um espaço em branco " " para determinar que o registro é ativo.

Vale lembrar que o que limita o tamanho de um arquivo é o espaço de gravação do meio físico dele.

MANIPULAÇÃO DE ARQUIVO DIRETO

O programa de gerenciamento de registros de alunos deverá ao final executar as rotinas de: cadastramento, pesquisa, remoção e alteração em um arquivo.

A rotina de pesquisa a ser utilizada será seqüencial, uma vez que o arquivo não estará ordenado. O funcionamento do processo de pesquisa em um arquivo é semelhante ao funcionamento em uma matriz, pois será necessário percorrer todo o arquivo até localizar a informação desejada, se esta existir. Vale lembrar que aqui será necessário verificar o campo FLAG para saber se a informação pesquisada não foi previamente marcada para remoção, ou seja, contenha um asterisco. Estando o campo marcado, a informação pesquisada não deverá ser apresentada, pois não mais existe. Esta rotina será utilizada pelas rotinas de cadastramento, alteração, remoção e consulta.

A rotina de alteração executará em primeiro lugar uma pesquisa do registro, encontrando-o deverá apresentar os dados atuais e solicitar a informação dos novos dados. Após a informação dos novos dados, grava-se novamente o registro no arquivo exatamente na mesma posição em que estavam os dados "velhos". A rotina de remoção executará o processo chamado remoção lógica do registro. Este processo é mais rápido, pois uma vez marcado um registro no arquivo, outro poderá ser gravado sobre o mesmo. O programa-exemplo gravará no campo FLAG um espaço em branco para o registro ativo e um asterisco quando a rotina de remoção for executada. A rotina de cadastramento antes de inserir um novo registro, deverá pesquisar o arquivo para verificar em primeiro lugar se as "novas" informações já existem, evitando assim duplicidade de registros. Não existindo, deverá ser verificado se existe algum campo FLAG marcado com um asterisco. Existindo, deverá gravar o novo registro na posição marcada, pois o registro anterior não é mais válido. Assim sendo, estaremos aproveitando os mesmos espaços que foram inutilizados pela rotina de remoção.

A rotina de consulta fará uma pesquisa no arquivo a fim de localizar o registro solicitado; encontrando-o, esta rotina o apresenta para visualização. Assim como as

demais esta rotina, deverá estar preparada para desconsiderar todo o registro removido logicamente. Observe atentamente o programa exemplo a seguir:

```
Program GERENCIAMENTO_ESCOLAR;
USES CRT;
Type
        BIMESTRE = array[1..4] of real;
        REG_ALUNO = record
                                FLAG : char;
                                MATRICULA : longint;
                                NOME : String[30];
                                NOTAS : bimestre;
                        end;
Var
        ALUNO : reg_aluno;
        ARQALU : file of reg_aluno;
        NR_MATRIC : longint;
        SISTEMA : String;
        RESP, TECLA : char;
        I : longint;

Procedure CENTER(MENSAGEM : string);
var
        TAMANHO : integer;
```

```
begin
        TAMANHO := 40 + length (MENSAGEM) div 2;
        writeln (MENSAGEM: TAMANHO);
end;

Procedure WRITEXY(X, Y : byte; MENSAGEM : string);
begin
        gotoxy(X, Y); write(MENSAGEM);
end;

Procedure LINE;
var
        I : LONGINT;
begin
        for I := 1 to 80 do
                write (#205);
end;

Procedure ACESSA_ARQUIVO;
begin
        assign(ARQALU, 'CADALU2.DAT');
        {$I-}
        reset(ARQALU);
```

```
        {$I+}

        if (IORESULT= 2) then
        begin
                rewrite(ARQALU);
                write(ARQALU, ALUNO);
        end;
end;

Procedure TELA;
begin
        gotoxy(18, 10);
        clreol;
        gotoxy(18,11);
        clreol;
        gotoxy(18, 12);
        clreol;
        gotoxy(18,13);
        clreol;
        gotoxy(18, 14);
        clreol;
        gotoxy(18,15);
        clreol;
```

```
        writexy ( 1,10,'Matricula ...: ');
        writexy ( 1,11,'Nome .....: ');
        writexy ( 1,12,'1a. Nota ....: ');
        writexy ( 1,13,'2a. Nota ....: ');
        writexy ( 1,14,'3a. Nota ....: ');
        writexy ( 1,15,'4a. Nota ....: ');
end;

Function PESQUISA(NUMERO : integer) : boolean;
var
        ACHOU : boolean;
begin
        ACHOU := false;
        seek(ARQALU, 1);

        while (not eof (ARQALU)) and (not ACHOU) do
        begin
                read(ARQALU, ALUNO);
                ACHOU := (NUMERO = ALUNO.MATRICULA) and
                                (ALUNO.FLAG <> '*');
        end;
        seek (ARQALU,filepos(ARQALU)-1);
        PESQUISA := ACHOU;
```

```
end;

Procedure ROT_CADASTRAR;
begin
        clrscr;
        Line;
        Center(SISTEMA);
        Center('MODULO DE CADASTRAMENTO');
        Line;
        WriteXy( 1, 6,'Digite os dados abaixo:');
        repeat
                WriteXy ( 1,24,'Digite [0] Para encerrar o Modulo Cadastro');
                Tela;
                gotoxy (18,10);
                readln (NR_MATRIC);
                gotoxy ( 1,24);
                clreol;

                if (NR_MATRIC<> 0) then
                        begin
                        if (Pesquisa(NR_MATRIC) = true) then
                                begin
                                { Apresenta os dados caso exista no arquivo }
```

```
                    gotoxy (18,10);
                    writeln(NR_MATRIC);
                    gotoxy (18,11);
                    writeln(ALUNO.NOME);

                    for I := 1 to 4 do
                    begin
                            gotoxy(18,11 + I);
                            writeln (ALUNO.NOTAS[I]:8:2);
                    end;

            WriteXy(1,23,'Este registro ja esta cadastrado');
            gotoxy (1,24);
            write ('Pressione algo para continuar.');
            TECLA := readkey;
            gotoxy (1,23);
            clreol;
end
else
begin
{ Localiza posicao para gravar registro }
            seek(ARQALU, 0);
            repeat
```

```
                read(ARQALU,ALUNO);
        until (ALUNO.FLAG = '*') Or (eof(ARQALU));

if (ALUNO.FLAG = '*') then
        seek(ARQALU,filepos(ARQALU)-1)
        else
                seek(ARQALU,filesize(ARQALU) );
                { Grava registro }
        ALUNO.FLAG := ' ';
        gotoxy (18,11);
        readln (ALUNO.NOME);

        for I:=1 to 4 do
        begin
                gotoxy(18,11 + I);
                readln(ALUNO.NOTAS[I]);
        end;

        ALUNO.MATRICULA := NR_MATRIC;
        write(ARQALU,ALUNO);
        gotoxy (1,24);
        write('Pressione algo para continuar...');
        TECLA := readkey;
```

```
        end;
    end;
    until (NR_MATRIC = 0);
end;

Procedure ROT_REMOVER;
begin
    clrscr;
    Line;
    Center (SISTEMA);
    Center ('MODULO DE REMOCAO');
    Line;
    WriteXy (1, 6,'Digite o numero de matricula:');
    repeat
        WriteXy (1,24,'Digite (0) Para encerrar o modulo Remocao');
        Tela;
        gotoxy (18,10);
        readln (NR_MATRIC);
        gotoxy (1,24);
        clreol;

        if (NR_MATRIC<> 0) then
        begin
```

```
if (Pesquisa(NR_MATRIC) = true) tHEN
begin
        Tela;
        gotoxy(18,10);
        writeln (NR_MATRIC);
                gotoxy(18,11);
                writeln (ALUNO.NOME);
        for I := 1to 4 do
        begin
                gotoxy (18,11 + I);
                writeln(ALUNO.NOTAS[I]:8:2);
        end;

gotoxy(1,17);
write('Remover este registro? [S/N): ');
read (RESP);
gotoxy( 1,17);
clreol;

if (RESP = 'S') or (RESP = 's') then
begin
        ALUNO.FLAG := '*';
        write(ARQALU, ALUNO);
```

```
                                Writexy(1,23,'Registro removido do arquivo');
                                gotoxy( 1,24);
                                Write('Pressione algo para continuar...');
                                TECLA := Readkey;
                        end;
                        gotoxy(1,15);
                        clreol;
                        gotoxy(1,23);
                        clreol;
                end
                else
                begin
                        Writexy(1,23,'Este registro NAO esta cadastrado');
                        gotoxy(1,24);
                        writeln('Pressione algo para continuar...');
                        TECLA := readkey;
                        gotoxy(1,23);
                        clreol;
                end;
        end;
        until (NR_MATRIC = 0);
end;
```

```
Procedure ROT_ALTERAR;
var
        NOVO_NOME : string[30];
        NOVA_NOTAS : bimestre;
begin
        clrscr;
        Line;
        Center(SISTEMA);
        Center('MODULO DE ALTERACAO');
        Line;
        WriteXy(1, 6,'Digite o numero de matricula:');
        repeat
                WriteXY(1,24,'Digite [0] Para encerrar o modulo alteracao');
                Tela;
                gotoxy (18,10);
                readln(NR_MATRIC);
                gotoxy(1,24);
                clreol;

                if (NR_MATRIC <> 0) then
                begin
                        if (pesquisa(NR_MATRIC) = true) then
                        begin
```

```
        gotoxy(18,10);
        writeln(NR_MATRIC);
        gotoxy(18,11);
        writeln(ALUNO.NOME);
        for I := i to 4 do
        begin
                gotoxy(18,11 + I);
                writeln(ALUNO.NOTAS[I]:8:2);
        end;

        WriteXy (1,23,'Deseja efetuar alteracao? [S/N]: ');
        readln (RESP);
        gotoxy(1,23);
        clreol;
    if (RESP = 's') or (RESP = 's') then
    begin
        WriteXy (1,23,'Digite as novas informacoes');
        WriteXy (1,17,'Novo Nome ...: ');
        WriteXy (1,18,'1a. Nota ....: ');
        WriteXy (1,19,'2a. Nota ....: ');
        WriteXy (1,20,'3a. Nota ....: ');
        WriteXy (1,21,'4a. Nota ,.,.: ');
        gotoxy (18,17);
```

```
                readln (Novo_NOME);

                for I := 1 to 4 do
                begin
                        gotoxy (18,17 + I);
                        readln (NOVA_NOTAS[I]);
                end;

        gotoxy( 1,23);
        clreol;
        WriteXy( 1,23,'Altera? [S/N]: ');
        readln(RESP);

        if (RESP = 'S') or (RESP = 's') then
        begin
                ALUNO.NOME := NOVO_NOME;

        for I := 1 to 4 do
                ALUNO.NOTAS[I] := NOVA_NOTAS[I];
        write (ARQALU, ALUNO);
        writeXy ( 1,23,'Alteracoes executadas com sucesso');
end;
gotoxy(1,24);
```

```
            write('Pressione algo para continuar...');
            TECLA :=readkey;
            gotoxy(1,17);
            clreol;
            gotoxy(1,18);
            clreol;
            gotoxy(1,19);
            clreol;
            gotoxy(1,20);
            clreol;
            gotoxy(1,21);
            clreol;
            gotoxy(1,23);
            clreol;
    end;
end
else
begin
            WriteXy( 1,23,'Este registro nao esta cadastrado');
            gotoxy(1,24);
            write('Pressione algo para continuar...');
            TECLA := readkey;
            gotoxy( 1,23);
```

```
                clreol;
        end;
  end;
  until (NR_MATRIC = 0);
end;

Procedure ROT_PESQUISAR;
begin
        clrscr;
        Line;
        center(SISTEMA);
        center('MODULO DE PESQUISA');
        Line;
        WriteXy(1, 6,'Digite o numero de matricula:');
        repeat
                WriteXy(1,24,'Digite [0] Para encerrar o modulo Pesquisa');
                Tela;
                gotoxy(18,10);
                readln(NR_MATRIC);
                gotoxy(1,24);
                clreol;

                If (NR_MATRIC <> 0) then
```

```
begin
if (Pesquisa(NR_MATRIC) = true)then
begin
        gotoxy (18,10);
        writeln(NR_MATRIC);
        gotoxy (18,11);
        writeln(ALUNO.NOME);

        for I :=1 to 4 do
        begin
                gotoxy(18,11 + I);
                writeln(ALUNO.NOTAS[I]:8:2);
        end;
        gotoxy(1,24);
        write('Pressione algo Para continuar...');
        TECLA := Readkey ;
        end
        else
        begin
                WriteXy ( 1,23,'Este registro nao esta cadastrado');
                gotoxy( 1,24);
                write('Pressione algo Para continuar...');
                TECLA := readkey;
```

```
                                gotoxy(1,23);
                                clreol;
                        end;
        end;
        until (NR_MATRIC = 0);
end;

Procedure ROT_ENCERRAR;
begin
        clrscr;
        close(ARQALU);
        writeln ('Fim de execucao do Cadastro de Alunos');
        writeln;
        exit;
end;



{ Programa Principal }

var
        OPCAO : char;
begin
        SISTEMA := 'SISTEMA DE CONTROLE ESCOLAR - v1.0';
```

```
Acessa_Arquivo;
repeat
        clrscr;
        line;
        center(SISTEMA);
        center('MENU PRINCIPAL');
        line;
        writeXy(26, 9,'[I] .....: Incluir Alunos ');
        writeXy(26,10,'[R] .....: Remover ');
        writeXy(26,11,'[A] .....: Alterar ');
        writeXY(26,12,'[C] .....: Consultar ');
        writexy(26,13,'[S] .....: Sair do Sistema');
        writexy(26,17,'Entre com a opcao: --> ');
        readln(OPCAO);
        writeln;
        OPCAO := upcase(OPCAO);
        if (OPCAO in ['I','R','A','C','S']) then
        begin
        case OPCAO of
                'I' : Rot_Cadastrar;
                'R' : Rot_Remover;
                'A' : Rot_Alterar;
                'C' : Rot_Pesquisar;
```

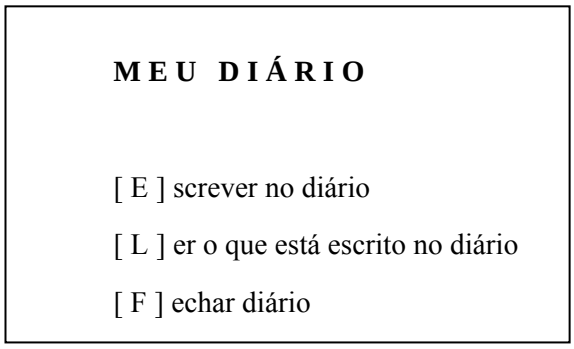
```
                        'S' : ROT_Encerrar;
                end;
        end
        else
        begin
        writeXy( 1,23,'Erro - Opcao invalida');
        gotoxy( 1,24);
        write('Pressione algo para continuar...');
        TECLA := readkey;
   end;
   until (OPCAO = 'F');
end.
```

1- Quais os tipos de arquivos existentes na linguagem Pascal?

2- Na criação de um arquivo texto, quais os comandos utilizados para gravar um caracter pro vez e uma string por vez?

3- Quanto a criação de um arquivo registro, qual o comando utilizado para gravar um registro?

4- Escreva na linguagem de programação Pascal um programa que permita ao usuário gerar e ler um arquivo texto para registrar os acontecimentos de sua vida pessoal como se fosse um diário. Utilize como base o seguinte menu de opções:



5- Diferencie acesso direto e acesso indexado.

6- O que caracteriza o acesso seqüencial a arquivos?

7- Qual o comando utilizado para fechar um determinado arquivo nomeado internamente como ARQFITA.DAT?

8- O que significa EOF? Para que é utilizado em um determinado programa escrito em Pascal?

13
TRABALHANDO COM JANELAS E EFEITOS GRÁFICOS

Inicialmente observe atentamente o programa exemplo abaixo que utiliza manipulações com janelas, cores entre outros recursos especiais da linguagem de programação Pascal.

```
PROGRAM AUDIENCIA_CANAIS;
USES
  CRT;
VAR
  CANAL,PES,TOTPES,TC2,TC4,TC6,TC7,TC9,TC13:INTEGER;
  PC2,PC4,PC6,PC7,PC9,PC13:REAL;

BEGIN
  CLRSCR;
  { -- ENTRADA DE DADOS SE O CANAL FOR DIFERENTE DE ZERO -- }

  REPEAT
```

```
{WINDOW LC-ESQ / LC-DIR}
WINDOW(10,10,20,50);
TEXTCOLOR(13);
GOTOXY(10,5);
WRITE('CANAIS VµLIDOS** 2 4 6 7 9 13 **> 0 PARA TERMINAR');
GOTOXY(10,7);WRITE('INFORME O CANAL:');
READLN(CANAL);

IF (CANAL<>0)
 THEN
  BEGIN
   GOTOXY(10,8);
   WRITE('TOTAL DE PESSOAS ASSISTINDO?');
   READLN(PES);
   TOTPES:=TOTPES+PES;

     CASE CANAL OF
     2: TC2:=TC2+PES;
     4: TC4:=TC4+PES;
     6: TC6:=TC6+PES;
     7: TC7:=TC7+PES;
     9: TC9:=TC9+PES;
     13: TC13:=TC13+PES;
```

```
      ELSE
       BEGIN
         WRITELN('C A N A L   I N V A L I D O !!');
         TOTPES:=TOTPES-PES;
       END;
      END;
    END;
  UNTIL (CANAL=0);
  PC2:=(TC2*100)/TOTPES;
  PC4:=(TC4*100)/TOTPES;
  PC6:=(TC6*100)/TOTPES;
  PC7:=(TC7*100)/TOTPES;
  PC9:=(TC9*100)/TOTPES;
  PC13:=(TC13*100)/TOTPES;

  CLRSCR;
  WINDOW(15,15,23,55);
  TEXTBACKGROUND(4);
  TEXTCOLOR(3);
  GOTOXY(12,6);WRITE('TOTAL DE PESSOAS PESQUISADAS:',TOTPES);
  GOTOXY(12,8);WRITE('AUDIENCIA DOS CANAIS:');
  GOTOXY(12,10);WRITE('CANAL2 - TOTAL DE PESSOAS:',PC2:0:2,'%');
  GOTOXY(12,11);WRITE('CANAL4 - TOTAL DE PESSOAS:',PC4:0:2,'%');
```

```
  GOTOXY(12,12);WRITE('CANAL6 - TOTAL DE PESSOAS:',PC6:0:2,'%');
  GOTOXY(12,13);WRITE('CANAL7 - TOTAL DE PESSOAS:',PC7:0:2,'%');
  GOTOXY(12,14);WRITE('CANAL9 - TOTAL DE PESSOAS:',PC9:0:2,'%');
  GOTOXY(12,15);WRITE('CANAL13- TOTAL DE PESSOAS:',PC13:0:2,'%');

  TEXTCOLOR(17);    { Efeito de Blink pisca-pisca }

  TEXTBACKGROUND(2);
  GOTOXY(12,18);
  WRITE('PRESSIONE <ENTER> ');
  READKEY;
END.
```

Neste programa utilizamos o comando de criação e delimitação de janelas **Window(x1, y1, x2, y2 : Integer);** que define uma janela de texto de canto superior esquerdo em (x1, y1) e canto inferior direito em (x2, y2). Os parâmetros padrões definidos pelo Pascal são 1, 1, 80, 25 nos modos de texto 80 colunas. Logo após o uso desse comando a posição do cursor passa a ser relativa a partir do canto superior esquerdo da janela.

Ainda, foram utilizados os comandos para definição de cores **TextBackGround (Cor : Integer);** e **TextColor (Cor : Integer);** que permite, respectivamente, selecionar uma nova cor para o fundo da tela e uma cor para as letras utilizadas.

Quanto a tabela de cores assumida pela linguagem de programação Pascal, você poderá ter acesso através do Apêndice A desse livro.

14
ESTUDOS DE CASOS RESOLVIDOS

ESTUDO DE CASO 1

Como você representaria uma lista de alunos cujo nó tem os seguintes campos:

Matrícula	**Nome**	**Idade**

Declare essa estrutura atraés da linguagem de programação Pascal.

ESTUDO DE CASO 2

Alocar *K* pilhas seqüenciais em um vetor NO de N posições. Cada pilha deve ter um ponteiro indicando sua base e um indicando seu topo. Fazer os seguintes procedmentos em Pascal:

a) Inicialização das pilhas;

b) Inserção na pilha ***i***;

c) Retirada da pilha ***i***.

A base da *i-ésima* pilha é dada pela fórmula

Trunc((I - 1) / K * N)

ESTUDO DE CASO 3

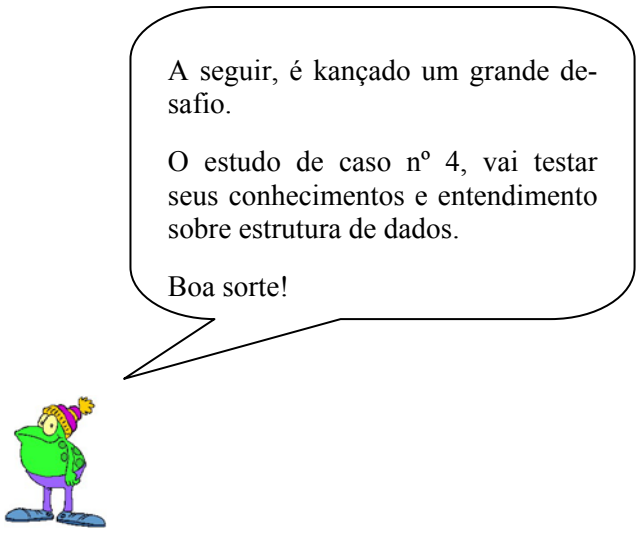
Fazer um programa em Pascal que inicialize uma lista simplesmente encadeada de números, apontada por **PtrIni**. O nó deve ter os campos **Info** e **Prox**.

Em seguida, leia diversos números inteiros, usando o zero como flag para determinar o fim dos dados. Para cada número lido, fazer o que segue:

a) Se for positivo, inseri-lo no final da lista;

b) Se for negativo, retirar o maior valor da lista (pode ser sua primeira ocorrência, se houver mais de um maior) e imprimi-lo; se a lista estiver vazia, exibir mensagem.

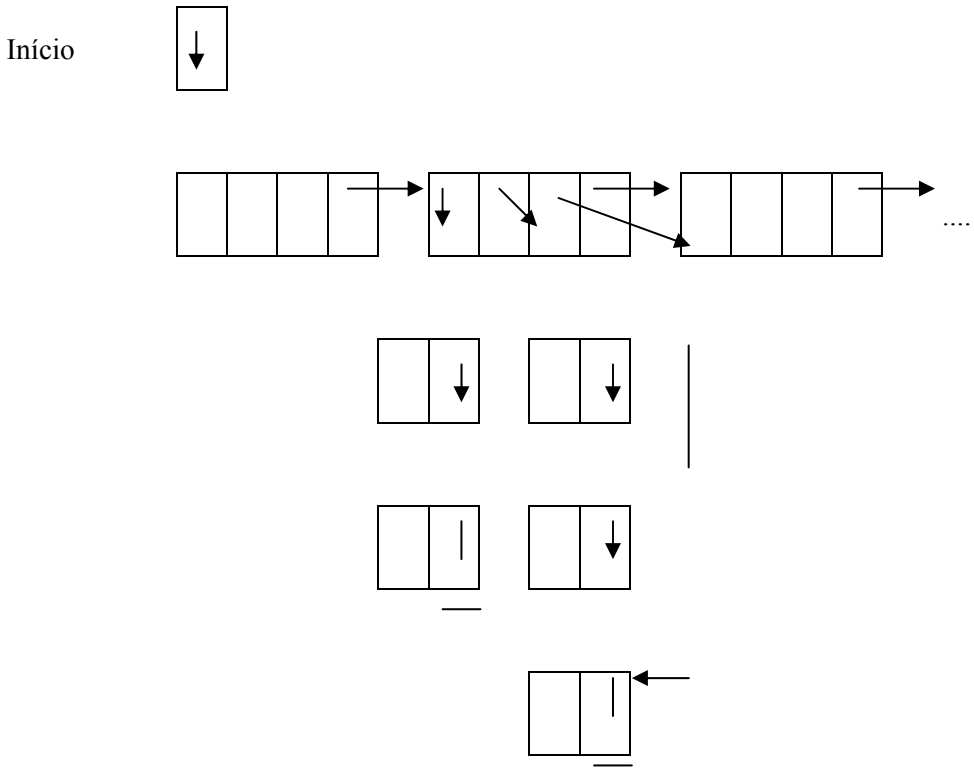
Após cada operação, exibir o estado atual da lista. Usar subprogramas adequados para tal solução.



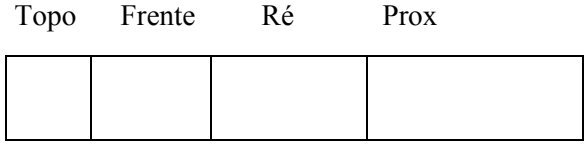
DESAFIO:

ESTUDO DE CASO 4

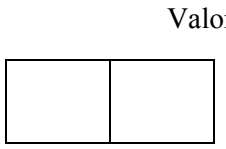
O desenho a seguir representa uma **lista geral simplesmente encadeada** apontada por **Início**, onde cada nó possui ponteiros para uma **pilha** de números e uma **fila** de números, ambas **simplesmente encadeadas.**



Os nós da lista têm quatro campos cada um, todos ponteiros, como segue:



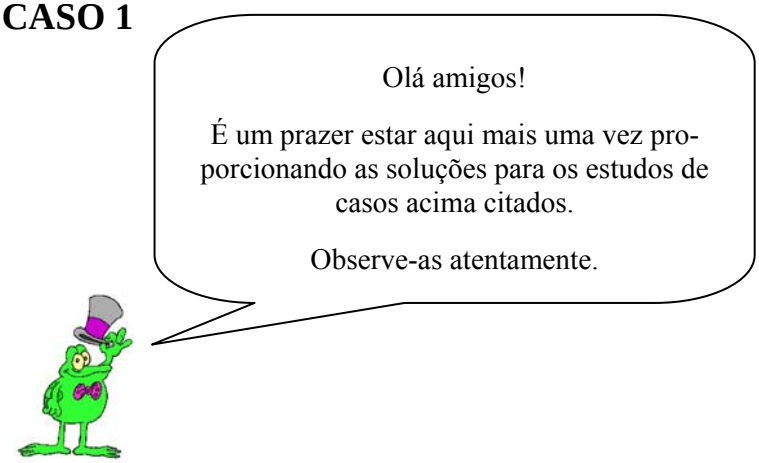
Os nós das pilhas e das filas, têm o seguinte layout:



a) Declarar os tipos desses nós e os tipos dos ponteiros respectivos (Type);

b) Escrever uma procedure em Pascal que recebe o ponteiro **Início** para a lista e transfere (move) um nó da pilha de maior tamanho para a fila que está vinculada ao primeiro nó da lista. Se a operação não for possível, por algum motivo, imprimir mensagem especificando o motivo. Se houver mais de uma pilha com o maior tamanho, transferir nó de qualquer uma delas. O fato de todas as pilhas terem o mesmo tamanho não impede a operação. Usar os tipos declarados em (a).

SOLUÇÃO DO CASO 1



Através de uma lista seqüencial geral apontada por Fim, onde cada aluno é representado por um registro contendo os três campos:

```
Type
    TipoRegAluno = record
        Matrícula, Idade: integer;
            Nome : string[30];
        end;

        TipoListaAlunos = record
            Aluno : array [1..Max] of TipoRegAluno;
            Fim : integer;
    end;
```

onde Max é uma constante. Repare que *empacotamos* a estrutura num registro que contém um *array* de registros (os nós da lista) e o ponteiro Fim.

SOLUÇÃO DO CASO 2

```
Procedure InicializaPilhas(No:Tno; var T,B:TApont);
var
  i:integer;
begin

  for i:=1 to k+1 do
     begin
     B[i]:=trunc((i-1)/k*N);
     T[i]:=trunc((i-1)/k*N);
     end;
end;

Procedure RetiraPilha(var No:Tno;var T,B:TApont; i:integer);
begin
   if T[i]=B[i]
      then writeln('Pilha ',i,' vazia')
      else Dec(T[i]);
end;

Procedure InserePilha(var No:Tno;var T,B:TApont;info,i:integer);
var
  achou : boolean;
```

```
 L,j   : integer;
begin
 if T[i] = B[i+1]  {testa overflow na pilha i}
   then begin
        {procura pilha com folga à direita}
       achou:=false;
       j:=i+1;

      while (not achou) and (j<=k) do
         begin
         if T[j]<B[j+1]
           then begin
               achou:=true;
               {desloca de B[i+1]+1 até T[j] uma pos. para a direita}

               For L:=T[j] downto B[i+1]+1 do
                 No[L+1]:=No[L];
               {acerta bases e topos}

               For L:=i+1 to j do
                 begin
                 Inc(B[L]); Inc(T[L]);
                 end;
```

```
            end;
      Inc( j );
      end;

{ procura pilha com folga … esquerda }

 j:=i-1;

 while (not achou) and (j>=1) do
   begin
   if T[j]<B[j+1]
     then begin
          achou:=true;
          {desloca de B[j+1]+1 até T[i] uma pos. para a esquerda}

          For L:=B[j+1]+1 to T[i] do
            No[L-1]:=No[L];
          {acerta bases e topos}

          For L:=j+1 to i do
            begin
            Dec(B[L]); Dec(T[L]);
            end;
```

```
            end;
        Dec(j);
        end;

      if not achou
        then writeln('Não há  espaço na  área')
        else begin {insere}
            Inc(T[i]); No[T[i]]:=Info;
            end;
      end
   else begin {Insere}
      Inc(T[i]); No[T[i]]:=Info;
      end;
end;
```

SOLUÇÃO DO CASO 3

```
Program Caso3;
uses crt;
Type TPtr = ^No;
   No = record
     Info : integer;
     Prox : TPtr;
   end;
```

```
Procedure ExibeLista(Ptrini: TPtr);
var
  PtrAux : TPtr;
begin
  write('Estado da lista: ');
  if PtrIni = nil
     then writeln('vazia')
     else begin
          PtrAux:= PtrIni;

          while (PtrAux <> nil) do
             begin
             write(PtrAux^.Info,'  ');
             PtrAux:=PtrAux^.Prox;
             end;
          writeln;
          end;
end;

Procedure InsereFinal(var PtrIni: TPtr; InfoIns: integer);
var
  PtrAux, PtrNovo : TPtr;
begin
```

```
  PtrAux := PtrIni;

  while (PtrAux <>nil) and (PtrAux^.Prox <> nil) do
     PtrAux:= PtrAux^.Prox;
  New(PtrNovo);
  PtrNovo^.Info := InfoIns;
  PtrNovo^.Prox := nil;
  if PtrAux = nil {está  vazia}
     then PtrIni:= PtrNovo
     else PtrAux^.Prox:= PtrNovo;
end;

Procedure RetiraMaior(var PtrIni:TPtr);
var
   Max : integer;
   PtrAux,PtrAnt,Pos,Ant : TPtr;
begin
   if PtrIni = nil
      then writeln('Retirada inviável - lista vazia')
      else begin
             Max := PtrIni^.Info;
             PtrAux := PtrIni; Pos:=PtrIni;
             PtrAnt := PtrIni; Ant:=PtrIni;
```

```
        while (PtrAux <> nil) do
          begin
           if PtrAux^.Info > Max
             then begin
                    Max := PtrAux^.Info;
                    Pos := PtrAux;
                    Ant := PtrAnt;
                    end;
          PtrAnt:=PtrAux;
          PtrAux:=PtrAux^.Prox;
          end;
        writeln('Retirado: ',Max);

        if Pos = Ant {o maior é o primeiro}
          then PtrIni := Pos^.Prox
          else Ant^.Prox := Pos^.Prox;
        Dispose(Pos);
        end;
end;

var
  PtrIni: TPtr;
  Num   : integer;
```

```
begin
  clrscr;
  PtrIni:= nil;
  readln(Num);

  while Num <> 0 do
    begin
    if Num > 0
      then InsereFinal(PtrIni,Num)
      else RetiraMaior(PtrIni);
    ExibeLista(PtrIni);
    Readln(Num);
    end;
  readln;
end.
```

SOLUÇÃO DO CASO 4

Consiste em percorrer a lista na procura do nó que contém a maior pilha. Uma vez encontrado, o nó do topo desta pilha deve ser "desligado" dela e "encadeado" no final da fila do início da lista. Como os nós das pilhas e das filas têm o mesmo tipo, não há necessidade de alocar (New) nem liberar (dispose) nós. O mesmo nó que está no topo da pilha transfere-se para o fim da fila, mediante, apenas, um acerto dos ponteiros.

Existem dois casos que impedem a operação de transferência: o *underflow* na lista (lista vazia) e o fato de todas as pilhas estarem vazias. Observe que o problema usa **dois** tipos de nós diferentes (somente dois). Portanto, não há necessidade de definir

três tipos de nós, embora isso não impeça o funcionamento do programa. Observe, também, o cálculo do maior**,** assunto de Programação I, que alguns alunos ainda erram.

Procedimentos:

a) Type

```
Tpt1 = ^Tno1;
Tno1 = record
    Valor : integer;
    Prox : Tpt1;
end;
Tpt2 = ^Tno2;
Tno2 = record
    Topo, Frente, Re : Tpt1;
    Prox : Tpt2;
end;
```

b) Function Tamanho(Topo: Tpt1):**integer**;

```
  var Aux:Tpt1;     Cont:integer;
begin
  Cont := 0; Aux:=Topo;

  while Aux<>nil do
    begin
```

```
        Inc(Cont); Aux:=Aux^.Prox;
        end;
     Tamanho:=Cont;
end;

Procedure  MoveNo (Inicio : Tpt2);
  var
        Maior, Tam : integer;
        Aux, Pmaior : Tpt2;
        P : Tpt1;
begin
     if Inicio = nil
        then writeln( 'Lista vazia')
        else begin {procura a maior pilha}
             Maior:=0; Pmaior:=nil;
             Aux:=Inicio;

             while Aux <> nil do
                 begin
                 Tam := Tamanho(Aux^.Topo);
                 if Tam > Maior
                     then begin
                 Maior:= Tam; Pmaior:=Aux;
```

```
                end;
        Aux:=Aux^.Prox;
end;
            if Maior = 0
                then writeln('Todas as pilhas estão vazias.')
                else begin
                     {retira da maior pilha}
                     P := Pmaior^.Topo;
                     Pmaior^.Topo:= Pmaior^.Topo^.Prox;
                     {insere na primeira fila}
                     P^.Prox:=nil;
                     if Inicio^.Re = nil  {se fila vazia}
                         then Inicio^.Frente := P
                         else Inicio^.Re^.Prox := P;
    Inicio^.Re := P;
    end;
             end;
end;
```

15
TABELA ASCII

As tabelas mostradas neste apêndice representam os 256 códigos usados nos computadores da família IBM. Esta tabela refere-se ao *American Standard Code for Information Interchange* (código padrão americano para troca de informações), que é um conjunto de números representando caracteres ou instruções de controle usados para troca de informações entre computadores entre si, entre periféricos (teclado, monitor, impressora) e outros dispositivos.

Estes códigos tem tamanho de 1 byte com valores de 00h a FFh (0 a 255 decimal). Podemos dividir estes códigos em três conjuntos: controle, padrão e estendido.

Os primeiros 32 códigos de 00h até 1Fh (0 a 31 decimal), formam o **conjunto de controle** ASCII. Estes códigos são usados para controlar dispositivos, por exemplo uma impressora ou o monitor de vídeo. O código 0Ch (*form feed*) recebido por ima impressora gera um avanço de uma página. O código 0Dh (*carriage return*) é enviado pelo teclado quando a tecla ENTER é pressionada.

Embora exista um padrão, alguns poucos dispositivos tratam diferentemente estes códigos e é necessário consultar o manual para saber exatamente como o equipamento lida com o código. Em alguns casos o código também pode representar um caracter imprimível. Por exemplo o código 01h representa o caracter ☺ (*happy face*).

Os 96 códigos seguintes de 20h a 7Fh (32 a 127 decimal) formam o **conjunto padrão** ASCII. Todos os computadores lidam da mesma forma com estes códigos. Eles representam os caracteres usados na manipulação de textos: códigos-fonte, documentos, mensagens de correio eletrônico, etc. São constituídos das letras do alfabeto latino (minúsculo e maiúsculo) e alguns símbolos usuais.

Os restantes 128 códigos de 80h até FFh (128 a 255 decimal) formam o **conjunto estendido** ASCII. Estes códigos também representam caracteres imprimíveis porem cada fabricante decide como e quais símbolos usar. Nesta parte do código estão definidas os caracteres especiais: é, ç, ã, ü ...

Dec.	Hex.	Controle
0	00h	NUL (*Null*)
1	01h	SOH (*Start of Heading*)
2	02h	STX (*Start of Text*)
3	03h	ETX (*End of Text*)
4	04h	EOT (*End of Transmision*)
5	05h	ENQ (*Enquiry*)
6	06h	ACK (*Acknowledge*)
7	07h	BEL (*Bell*)
8	08h	BS (*Backspace*)
9	09h	HT (*Horizontal Tab*)
10	0Ah	LF (*Line Feed*)
11	0Bh	VT (*Vertical Tab*)
12	0Ch	FF (*Form Feed*)

13	0Dh	CR (*Carriage Return*)
14	0Eh	SO (*Shift Out*)
15	0Fh	SI (*Shift In*)
16	10h	DLE (*Data Link Escape*)
17	11h	DC1 (*Device control 1*)
18	12h	DC2 (*Device control 2*)
19	13h	DC3 (*Device control 3*)
20	14h	DC4 (*Device control 4*)
21	15h	NAK (*Negative Acknowledge*)
22	16h	SYN (*Synchronous Idle*)
23	17h	ETB (*End Transmission Block*)
24	18h	CAN (*Cancel*)
25	19h	EM (*End of Media*)
26	1Ah	SUB (*Substitute*)
27	1Bh	ESC (*Escape*)
28	1Ch	FS (*File Separator*)

29	1Dh	GS (*Group Separator*)
30	1Eh	RS (*Record Separator*)
31	1Fh	US (*Unit Separator*)

Caracter	Dec.	Hex.	0	48	30h	A	65	41h
<espaço>	32	20h	1	49	31h	B	66	42h
!	33	21h	2	50	32h	C	67	43h
"	34	22h	3	51	33h	Caracter	Dec.	Hex.
#	35	23h	4	52	34h	D	68	44h
$	36	24h	5	53	35h	E	69	45h
%	37	25h	6	54	36h	F	70	46h
&	38	26h	7	55	37h	G	71	47h
'	39	27h	8	56	38h	H	72	48h
(	40	28h	9	57	39h	I	73	49h
)	41	29h	:	58	3Ah	J	74	4Ah
*	42	2Ah	;	59	3Bh	K	75	4Bh
+	43	2Bh	<	60	3Ch	L	76	4Ch
,	44	2Ch	=	61	3Dh	M	77	4Dh
-	45	2Dh	>	62	3Eh	N	78	4Eh
.	46	2Eh	?	63	3Fh	O	79	4Fh
/	47	2Fh	@	64	40h	P	80	50h

Q	81	51h	b	98	62h	r	114	72h
R	82	52h	c	99	63h	s	115	73h
S	83	53h	d	100	64h	t	116	74h
T	84	54h	e	101	65h	u	117	75h
U	85	55h	f	102	66h	v	118	76h
V	86	56h	g	103	67h	w	119	77h
W	87	57h	Caracter	Dec.	Hex.	x	120	78h
X	88	58h	h	104	68h	y	121	79h
Y	89	59h	i	105	69h	z	122	7Ah
Z	90	5Ah	j	106	6Ah	{	123	7Bh
[	91	5Bh	k	107	6Bh	|	124	7Ch
\	92	5Ch	l	108	6Ch	}	125	7Dh
]	93	5Dh	m	109	6Dh	~	126	7Eh
^	94	5Eh	n	110	6Eh	<delete>	127	7Fh
_	95	5Fh	o	111	6Fh	Ç	128	80h
`	96	60h	p	112	70h	ü	129	81h
a	97	61h	q	113	71h	é	130	82h

â	131	83h	ô	147	93h	ñ	164	A4h
ä	132	84h	ö	148	94h	Ñ	165	A5h
à	133	85h	ò	149	95h	ª	166	A6h
å	134	86h	û	150	96h	º	167	A7h
ç	135	87h	ù	151	97h	¿	168	A8h
ê	136	88h	ÿ	152	98h	⌐	169	A9h
ë	137	89h	Ö	153	99h	¬	170	AAh
è	138	8Ah	Ü	154	9Ah	½	171	ABh
ï	139	8Bh	¢	155	9Bh	¼	172	ACh
Caracter	Dec.	Hex.	£	156	9Ch	¡	173	ADh
î	140	8Ch	¥	157	9Dh	«	174	AEh
ì	141	8Dh	Pts	158	9Eh	»	175	AFh
Ä	142	8Eh	*f*	159	9Fh	▒	176	B0h
Å	143	8Fh	ááááá	160	A0h	Caracter	Dec.	Hex.
É	144	90h	í	161	A1h	▒	177	B1h
æ	145	91h	ó	162	A2h	▓	178	B2h
Æ	146	92h	ú	163	A3h	│	179	B3h

┤	180	B4h	┼	197	C5h	Caracter	Dec.	Hex.
╡	181	B5h	╞	198	C6h	╓	214	D6h
╢	182	B6h	╟	199	C7h	╫	215	D7h
╖	183	B7h	╚	200	C8h	╪	216	D8h
╕	184	B8h	╔	201	C9h	┘	217	D9h
╣	185	B9h	╩	202	CAh	┌	218	DAh
║	186	BAh	╦	203	CBh	█	219	DBh
╗	187	BBh	╠	204	CCh	▄	220	DCh
╝	188	BCh	═	205	CDh	▌	221	DDh
╜	189	BDh	╬	206	CEh	▐	222	DEh
╛	190	BEh	╧	207	CFh	▀	223	DFh
┐	191	BFh	╨	208	DOh	α	224	E0h
└	192	C0h	╤	209	D1h	ß	225	E1h
┴	193	C1h	╥	210	D2h	Γ	226	E2h
┬	194	C2h	╙	211	D3h	π	227	E3h
├	195	C3h	╘	212	D4h	Σ	228	E4h
─	196	C4h	╒	213	D5h	σ	229	E5h

μ	230	E6h
τ	231	E7h
Φ	232	E8h
Θ	233	E9h
Ω	234	EAh
δ	235	EBh
∞	236	ECh
φ	237	EDh
∈	238	EEh
∩	239	EFh
≡	240	F0h
±	241	F1h
≥	242	F2h
≤	243	F3h
⌠	244	F4h
⌡	245	F5h
÷	246	F6h
≈	247	F7h
°	248	F8h
∙	249	F9h
·	250	FAh
Caracter	Dec.	Hex.
√	251	FBh
^{n}	252	FCh
^{2}	253	FDh
·	254	FEh
	255	FFh

Entre os caracteres da tabela ASCII estendidos os mais úteis estão, talvez, os caracteres de desenho de quadro em linhas simples e duplas: os caracteres de B3h até DAh (179 a 218 decimal). Em programas mais sofisticados procuramos utilizar bordas e molduras, que são desenhadas com esses caracteres da tabela ASCII.

Segue um programa na linguagem Pascal que mostra na tela do computador uma tabela ASCII completa:

```
{ Programa Simples que mostra símbolos contidos da tabela ASCII }
Program TabelaASCII;
Uses  Crt;
Var
        i : Integer;
Begin
        Clrscr;
        For i := 1 to 255 do
        Begin
                Writeln(´ALT + ´, i, ´ = ´, Chr( i ));
                If ( (i = 23 ) or (i = 47) or (i = 71) or (i=95) or (i = 119) or
                        (i = 143) or (i = 167) or (i = 191) or (i = 215) or
```

```
                    (i = 239) or (i = 255))
        Then
            Begin
                Gotoxy(1,25);
                Write(´Pressione uma tecla p/ continuar...´);
                Readkey;
            End;
        End;
End.
```