Universidade Católica de Pelotas
Escola de Informática

Estruturas de Dados

por
Dr. Paulo Roberto Gomes Luzzardi
luzzardi@atlas.ucpel.tche.br
http://gcg.ucpel.tche.br/luzzardi

Bibliografia

VELOSO, Paulo e SANTOS, Clésio - ***Estruturas de Dados*** - Editora Campus, 4 ed., Rio de Janeiro, 1986
WIRTH, Niklaus - ***Algoritmos e Estruturas de Dados***, PHB
PINTO, Wilson - ***Introdução ao Desenvolvimento de Algoritmos e Estrutura de Dados***, Editora Érica

Agosto, 2003

Conteúdo Programático

1. Tipos de Dados

1.1. Tipos Primitivos
1.2. Construção de Tipos (Estruturados)

2. Vetores e Matrizes

2.1. Conceitos Básicos

3. Listas Lineares

3.1. Listas Genéricas
3.2. Listas com disciplinas de Acesso (PILHA, FILA e DEQUE)
3.3. Representação por Contigüidade Física
3.4. Representação por Encadeamento

4. Pesquisa e Classificação de Dados

4.1. Pesquisa Seqüencial
4.2. Pesquisa Binária
4.3. Cálculo de Endereço
4.4. Classificaçao por Inserção
4.5. Classificação por Troca
4.6. Classificação por Seleção

5. Árvores

5.1. Conceitos Básicos
5.2. Operações Básicas sobre Árvores Binárias

Definições

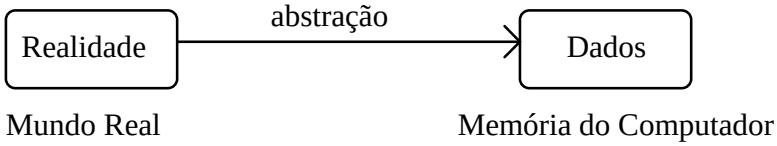
Estruturas de Dados

Estuda as principais técnicas de representação e manipulação de dados na ***memória principal*** (RAM)**.**

Organização de Arquivos

Estuda as principais técnicas de representação e manipulação de dados na ***memória secundária*** (DISCO).

Conceito



Dados

São as informações a serem representadas, armazenadas ou manipuladas.

Tipos de Dados

Refere-se ao conjunto de valores que uma constante, ou variável, ou expressão pode assumir, ou então a um conjunto de valores que possam ser gera-dos por uma função.

Na definição de uma variável, constante, expressão ou função deve-se definir o ***Tipo de Dado***, por algumas razões:

1) Representar um tipo abstrato de dado (***Realidade***);

2) Delimitar a faixa de abrangência (***Limites***);

3) Definir a quantidade de bytes para armazenamento;

4) E as operações que podem ser efetuadas.

Os tipos de dados podem ser: ***Primitivos*** ou ***Estruturados***, sendo que os estruturados, são chamados de Complexos.

Tipos Primitivos

Os tipos de dados dependem das características do sistema, do processador e do co-processador.

Integer (***Byte***, ***Word***, ***ShortInt***, ***LongInt***)
Real
Boolean
Char

Construção de Tipos (*Estruturados* ou *Complexos*)

Tipos obtidos através de tipos primitivos, podem ser:

String (Cadeia de Caracteres)
Array (Agregados Homogêneos)
Record (Agregados Heterogêneos)
Pointer (Ponteiros)
Enumeração (Conjunto Ordenado de Identificadores)
SubIntervalares (Subconjunto de Tipos Primitivos)

String (Cadeia de Caracteres)

Tipo de dado que permite que uma variável possua vários caracteres.

Exemplo:

```
Type PALAVRA = String[30];
Var nome: PALAVRA;
```

Array (Agregados Homogêneos)

Tipo de dado que permite que uma variável possua vários elementos, todos do mesmo tipo.

Exemplo:

```
Const MAX = 10;
Type VETOR = Array[1..MAX] Of Real;
Var n: VETOR;
```

Record (Registro)

Tipo de dado que permite que uma variável possua vários campos. Os campos podem ser de tipos de dados distintos.

Exemplos:

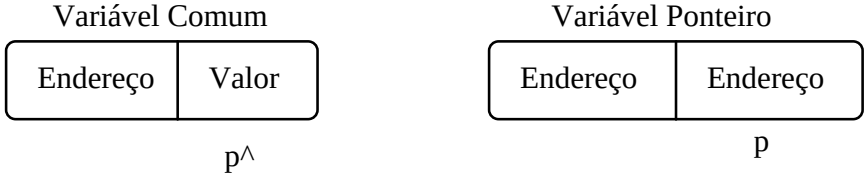
```
Type REGISTRO_ALUNO = Record
                                    Matrícula: Integer;
                                    Nome: String[30];
                                    Endereço: String[40];
                              End;
     TURMA = Array[1..50] Of REGISTRO_ALUNO;
Var dados: TURMA;

Type DATA = Record
                        Dia: Integer;
                        Mes: Integer;
                        Ano: Integer;
               End;
     NOTAS = Array[1..3] Of Real;
     ALUNO = Record
                        Matrícula: Integer;
                        Nome: String[30];
                        Data_Nascimento: DATA;
                        Nota: NOTAS;
                End;
```

Pointer (Ponteiros)

É um tipo de dado, onde a variável contém o endereço de outra variável, ou um endereço de memória. Permite ainda, alocação dinâmica de Memória, ou seja, alocação de memória em tempo de execução do programa.

Exemplo:

```
Var     p: ^Integer;
```



Exemplo:

```
Uses Crt;
Var     p: ^Integer;
        n: Integer;
Begin
        ClrScr;
        n := 65;
        p := @n;
        WriteLn('Conteúdo: ',p^);
End.
```

@n ..... Endereço da variável "n" na memória principal (RAM)

***Exemplo*:**

```
Uses Crt;
Var p: ^Integer;
Begin
        ClrScr;
        New(p);
        If p = NIL Then
                WriteLn('ERRO FATAL: Falta de Memória')
        Else
                Begin
                        p^ := 65;
                        WriteLn('Conteúdo: ', p^);
                        Dispose(p);
                End;
End.
```

Definições

New(p) ..... Aloca dinamicamente memória para o ponteiro "p"
Dispose(p) ..... Desaloca memória ocupada pela variável "p"
GetMem(p,n) ..... Aloca dinamicamente memória para vários elementos
FreeMem(p,n) ..... Desaloca memória ocupada pela variável "p"
NIL ..... Palavra reservada para ponteiro NULO
p ..... Endereço da memória RAM
p^ ..... Conteúdo do ponteiro

Enumeração

Tipo de Dado que permite que uma variável possua um conjunto ordenado de identificadores.

```
Type Meses = (Jan,Fev,Mar,Abr,Mai,Jun,Jul,Ago,Set,Out,Out,Nov,Dez);
Var mes: Meses;
```

Subintervalares

Tipo de dado que permite que uma variável seja um subconjunto de um tipo primitivo, sendo também chamado de sub-faixa.

```
Type dia = 1..31;
        maiúsculas  = 'A'..'Z';
        minúsculas = 'a'..'z';
Var     d: dia;
        a: maiúsculas;
        b: minúsculas;
```

Operadores(Aritméticos, Relacionais e Lógicos)

Aritméticos

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
Mod	Resto Inteiro da Divisão
Div	Quociente Inteiro da Divisão

Relacionais

>	Maior
<	Menor
>=	Maior ou Igual
<=	Menor ou Igual
=	Igual
<>	Diferente

Lógicos

AND	e
OR	ou
NOT	não
XOR	ou exclusivo

Vetores e Matrizes

Permitem armazenamento de vários dados na memória RAM ao mesmo instante de tempo e com contigüidade física, ou seja, uma variável com possui vários elementos, igualmente distanciados, ou seja, um ao lado do outro.

Vetor: (É uma matriz unidimensional)

```
Type VETOR = Array[1..10] Of Integer;
Var v: VETOR;
```

Matriz: (Possui mais de uma dimensão)

```
Type MATRIZ = Array[1..10,1..10] Of Real;
Var m: MATRIZ;
```

Índice: Constante númerica inteira que referencia cada elemento

Exemplos:

Dada a definição acima:

```
v[1].....primeiro elemento
v[10]..... último elemento

m[1,1].....primeiro elemento
m[10,10].....último elemento
```

Entrada de um Vetor

```
For i := 1 To 10 Do
    ReadLn(v[i]);
```

Entrada de uma Matriz Bidimensional

```
For i := 1 To 10 Do
    For j := 1 To 10 Do
        ReadLn(m[i,j]);
```

Exercícios:

1) Faça um programa em PASCAL que lê uma matriz A (6x6) e cria 2 vetores SL e SC de 6 elementos que contenham respectivamente a soma das linhas (SL) e a soma das colunas (SC). Imprimir os vetores SL e SC.

2) Faça um programa em PASCAL que lê uma matriz A (12x13) e divide todos os elementos de cada uma das 12 linhas de A pelo valor do maior elemento daquela linha. Imprimir a matriz A modificada.

Observação: Considere que a matriz armazena apenas elementos inteiros

Operações sobre os Dados

- Criação dos Dados
- Manutenção dos Dados
    - Inserção de um Componente
    - Remoção de um Componente
    - Alteração de um Componente
- Consulta aos Dados
- Destruição dos Dados
- Pesquisa e Classificação

Alocação de Memória (RAM - **R**andom **A**ccess **M**emory)

Alocação Estática de Memória

É a forma mais simples de alocação, na qual cada dado tem sua área reservada, não variando em tamanho ou localização ao longo da execução do programa.

```
Var r: Real;     (* a variável "r" ocupa 6 bytes durante toda a execução do programa *)
```

Alocação Dinâmica de Memória

Nesta forma de alocação, são feitas requisições e liberações de porções da memória ao longo da execução do programa. Para isto, são usadas variáveis do tipo ***Ponteiro***.

```
Var p: ^Integer;          (* a variável "p" poderá ocupar "n" bytes a qualquer momento *)
```

Célula, Nodo ou Nó

Espaço alocado na memória para uma variável (tipo primitivo ou complexo), ou seja, número de bytes gastos para o armazenamento de um dado.

Campo de um Nodo: É uma subdivisão de um nodo

Listas Lineares

Listas Genéricas

Conceito

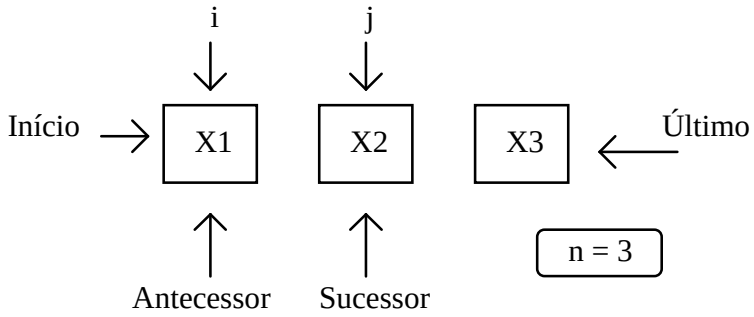
Conjunto de dados que mantém a relação de ordem ***Linear*** entre os componentes. É composta de elementos (componentes ou nós), os quais podem conter um dado ***primitivo*** ou ***estruturado***.

Lista Linear

É uma estrutura que permite representar um conjunto de dados de forma a preservar a relação de ordem entre eles.

Uma lista linear X é um conjunto de nodos (nós) X1, X2, ... Xn, Tais que:

1) Existem “n” nodos na lista (n >= 0)
2) X1 é o primeiro nodo da lista
3) Xn é o último nodo da lista
4) Para todo i,j entre 1 e n, se i<j, então o elemento Xi antecede o elemento Xj
5) Caso i = j-1, Xi é o antecessor de Xj e Xj é o sucessor de Xi



Observação: Quando n=0, dizemos que a ***Lista*** é ***Vazia***

Exemplos de Listas:

- Lista de clientes de um Banco
- Lista de Chamada
- Fichário

Operações sobre Listas:

1) *Percurso:*

Permite utilizar cada um dos elementos de uma lista, de tal forma que:

- 0 primeiro nodo utilizado é o primeiro da lista;
- Para utilizar o nodo Xj, todos os nodos de X1 até X(j-1) já foram utilizados;
- O último nodo utilizado é o último nodo da lista.

2) *Busca*:

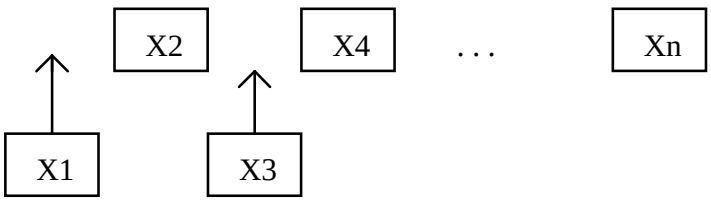
Procura um nodo específico da lista linear, de tal forma que:

- O nodo é identificado por sua posição na lista;
- O nodo é identificado pelo seu conteúdo.

3) *Inserção*:

Acrescenta um nodo X a uma lista linear, de tal forma que:

- O nodo X terá um sucessor e/ou um antecessor;
- Após inserir o nodo X na posição i (i >= 1 e i <= n+1), ele passará a ser i-ésimo nodo da lista;
- O número de elementos (n) é acrescido de uma unidade.



4) *Retirada*: (Exclusão)

Retira um nodo X da lista, de tal forma que:

- Se Xi é o elemento retirado, o seu sucessor passa a ser o sucessor de seu antecessor. X(i+1) passa a ser o sucessor de X(i-1). Se Xi é o primeiro nodo, o seu sucessor passa a ser o primeiro, se Xi é o último, o seu antecessor passa a ser o último;
- O número de elementos (n) é decrescido de uma unidade.

Operações Válidas sobre Listas:

- Acessar um elemento qualquer da lista;
- Inserir um novo elemento à lista;
- Concatenar duas listas;
- Determinar o número de elementos da lista;
- Localizar um elemento da lista com um determinado valor;
- Excluir um elemento da lista;
- Alterar um elemento da lista;
- Criar uma lista;
- Destruir a lista.

Representações:

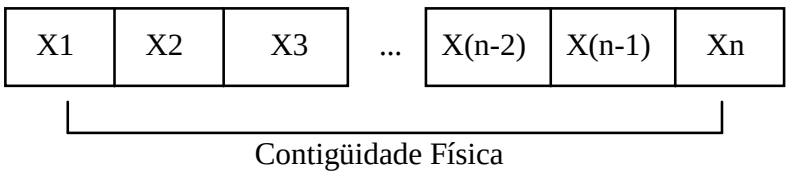
Por Contigüidade Física:

Os nodos são armazenados em endereços contíguos, ou igualmente distanciados um do outro.

Os elementos são armazenados na memória um ao lado do outro, levando-se em consideração o tipo de dado, ou seja, a quantidade de bytes.

Se o endereço do nodo Xi é conhecido, então o endereço do nodo X(i+1) pode ser determinado.

Esquema:

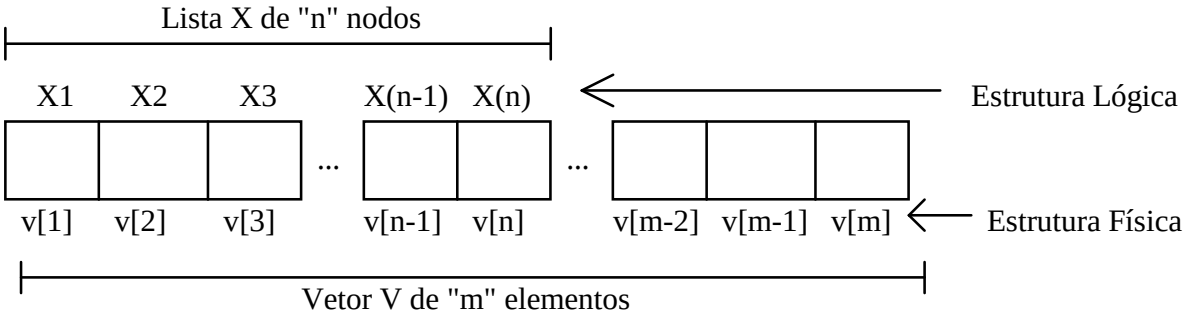


- Os relacionamentos são representados pela disposição física dos componentes na memória;
- A posição na estrutura lógica determina a posição na estrutura física.

Observação: Uma lista pode ser implementada através de um vetor de “m” elementos.

Atenção: Se “n’ = “m” a ***Lista*** é chamada ***Cheia***

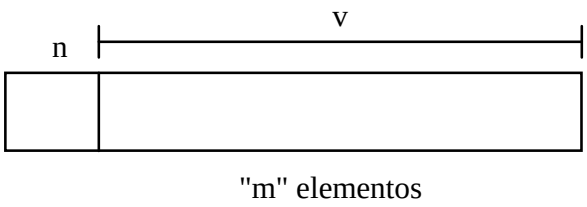
Observação: Como o número de nodos armazenados na lista pode ser modificado durante a execução do programa, deve-se representá-la como parte de um vetor de “m” elementos com “m” < “n”.



Representação: A lista X está representada por um vetor V de “m” elementos.

Componentes de uma Lista:

- Número de nodos da lista (n);
- Vetor de nodos (v);
- Tamanho total da lista (m).




```
Const m = 50;
Type   TIPO_DO_NODO = Real;
       LISTA = Record
                     n: Integer;
                     v: Array[1..m] of  TIPO_DO_NODO;
               End;
Var l: LISTA;
```

Observação: Considera-se que o primeiro nodo da lista será armazenado na primeira posição do vetor.

Exemplo:

```
Const  SUCESSO = 1;
       LISTA_CHEIA = 2;

(* ---------------------------------------- Cria lista vazia *)

Procedure CRIA_LISTA(Var x: LISTA);
Begin
```

```
        x.n := 0;
End;

(* ---------------------------------------- Inclui nodo no fim da lista *)

Function INCLUI_NO_FIM(Var x: LISTA; no: NODO): Integer;
Begin
        If x.n = m Then
                INCLUI_NO_FIM := LISTA_CHEIA
        Else
                Begin
                        x.n := x.n + 1;
                        x.v[x.n] := no;
                        INCLUI_NO_FIM := SUCESSO;
                End;
End;

(* ---------------------------------------- Inclui nodo no início da lista *)

Function INCLUI_NO_INÍCIO(Var x: LISTA; no: NODO): Integer;
Var i: Byte;
Begin
        If x.n = m Then
                 INCLUI_NO_INÍCIO := LISTA_CHEIA
        Else
                Begin
                        For i := x.n DOWNTO 1 Do
                                x.v[i+1] := x.v[i];
                        x.v[1] := no;
                        INCLUI_NO_INÍCIO := SUCESSO;
                End;
End;
```

Problema Proposto:

Incluir dados em uma lista de números inteiros, mantendo-a ordenada.

Solução do Problema Proposto:

```
Program Inserir_Lista_Mantendo_Ordenada;

Uses Crt;

Const  m = 50;
        SUCESSO = 1;
```

```
        LISTA_CHEIA = 2;
        LISTA_VAZIA = 3;

Type LISTA = Record
                        n: Integer;
                        v: Array[1..m] Of Integer;
                End;

Var     l: LISTA;
        valor: Integer;
        op: Integer;
        i: Word;

(* ------------------------------------ Cria_Lista *)

Procedure Cria_Lista(Var x: LISTA);
Begin
        x.n := 0;
End;

(* ------------------------------------ Inclui_no_Fim *)

Function Inclui_no_Fim(Var x: LISTA; no: Integer): Integer;
Begin
        If x.n = m Then
                Inclui_no_Fim := LISTA_CHEIA
        Else
                Begin
                        x.n := x.n + 1;
                        x.v[x.n] := no;
                        Inclui_no_Fim := SUCESSO;
                End;
End;

(* ------------------------------------ Inclui_no_Inicio *)

Function Inclui_no_Inicio(Var x: LISTA; no: Integer): Integer;
Var i: Word;
Begin
        If x.n = m Then
                Inclui_no_Inicio := LISTA_CHEIA
        Else
                Begin
                        For i := x.n DownTo 1 Do
                                x.v[i+1] := x.v[i];
```

```
                        x.v[1] := no;
                        x.n := x.n + 1;
                Inclui_no_Inicio := SUCESSO;
        End;
End;

(* ------------------------------------ Inclui_Dado *)

Function Inclui_Dado(Var x: LISTA;no,pos: Integer): Integer;
Var i: Word;
Begin
        If x.n = m Then
                inclui_Dado := LISTA_CHEIA
        Else
                Begin
                        For i := x.n DownTo 1 Do
                                x.v[i+1] := x.v[i];
                        x.v[pos] := no;
                        x.n := x.n + 1;
                        Inclui_Dado := SUCESSO;
                End;
End;

(* ------------------------------------ Verifica_Dado *)

Function Verifica_Dado(x: LISTA; no: Integer):Integer;
Var     i: Word;
        flag: Boolean;
Begin
        i := 0;
        flag := FALSE;
        Repeat
                i := i + 1;
                If no < x.v[i] Then
                        Begin
                                flag := TRUE;
                                Verifica_Dado := Inclui_Dado(l,no,i);
                        End;
        Until (i = x.n) Or (flag);
        If Not(flag) Then
                verifica_Dado := Inclui_no_Fim(l,no);
End;
```

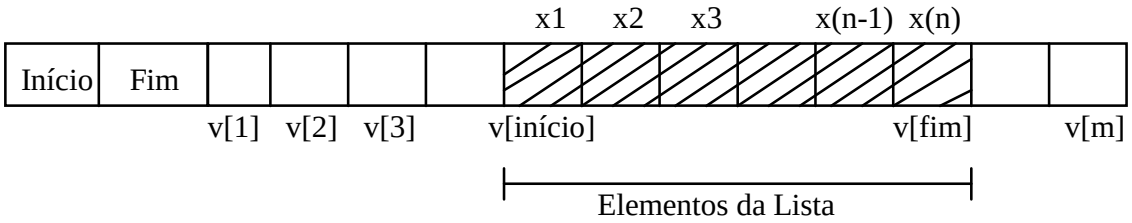
```
(* ------------------------------------ Erro *)

Procedure Erro(op: Integer);
Begin
        Case op
        Of
                LISTA_CHEIA: WriteLn('ERRO: Lista Cheia');
                LISTA_VAZIA: WriteLn('ERRO: Lista Vazia');
        End;
        WriteLn('Tecle [ENTER] para Continuar');
        ReadLn;
End;

(* ------------------------------------ Programa Principal *)

Begin
        Cria_Lista(l);
        ClrScr;
        Write('Valor: ');
        ReadLn(valor);
        If valor <> 0 Then
                Begin
                        op := Inclui_no_Fim(l,valor);
                        Erro(op);
                End;
        Repeat
                ClrScr;
                Write('Valor: ');
                ReadLn(valor);
                If valor <> 0 Then
                        Begin
                                op := Verifica_Dado(l,valor);
                                Erro(op);
                        End;
        Until (valor = 0) Or (op = LISTA_CHEIA);
        If l.n <> 0 Then
                For i := 1 to l.n Do
                        Begin
                                WriteLn('VALOR: ',l.v[i]);
                                ReadLn;
                        End
        Else
                Erro(LISTA_VAZIA);
End.
```

Problema Proposto:

Incluir dados em uma lista de números inteiros (máximo 50) sem repetição. O programa termina quando o dado lido for zero, então o programa deve imprimir a lista na tela sem repetição.

Contigüidade Física:

Uma alternativa para representação por contigüidade física é não iniciar no início do vetor, isto facilita as inserções.



Observação: As operações de inclusão e exclusão de nodos podem optar pela extremidade da lista que irá diminuir (no caso de exclusão) ou aumentar (no caso de inserção) de comprimento. “A escolha deverá considerar o caso que produz menor movimentação de elementos”.

```
Type LISTA = Record
                    início,fim: Integer;
                    v: Array[1..m] Of NODO;
               End;
Var x: LISTA;
```

Lista Vazia:

início = 0
fim = 0

Lista Cheia:

início = 1
fim = m

Problema Proposto:

O programa permitirá apenas inclusão de números inteiros no ***início*** e no ***fim*** da lista.

Solução do Problema Proposto (1):

```
Program Lista_Linear_1;

Uses Crt;

Const  m = 9;
       SUCESSO = 1;
       LISTA_CHEIA = 2;

Type LISTA = Record
                    inicio,fim: Integer;
                    v: Array[1..m] Of Integer;
               End;

Var     l: LISTA;
       valor: Integer;
       ch: Char;
       i,erro: Integer;

(* ---------------------------------------------- CRIA_LISTA *)

Procedure CRIA_LISTA(Var x: LISTA);
Begin
       x.inicio := 0;
       x.fim := 0;
End;
(* ---------------------------------------------- INCLUI_INÍCIO *)

Function INCLUI_INICIO(Var x: LISTA; valor: Integer): Integer;
Begin
       If x.inicio = 0 Then
              Begin
                     x.inicio := m DIV 2;
                     x.fim := x.inicio;
                     x.v[x.inicio] := valor;
                     INCLUI_INICIO := SUCESSO;
              End
       Else
              If x.inicio = 1 Then
                     INCLUI_INICIO := LISTA_CHEIA
              Else
                     Begin
                            x.inicio := x.inicio - 1;
```

```
                        x.v[x.inicio] := valor;
                        INCLUI_INICIO := SUCESSO;
                End;
End;

(* ---------------------------------------------- INCLUI_FIM *)

Function INCLUI_FIM(Var x: LISTA; valor: Integer): Integer;
Begin
        If x.fim = 0 Then
                Begin
                        x.inicio := m DIV 2;
                        x.fim := x.inicio;
                        x.v[x.fim] := valor;
                        INCLUI_FIM := SUCESSO;
                End
        Else
                If x.fim = m Then
                        INCLUI_FIM := LISTA_CHEIA
                Else
                        Begin
                                x.fim := x.fim + 1;
                                x.v[x.fim] := valor;
                                INCLUI_FIM := SUCESSO;
                        End;
End;

(* ---------------------------------------------- PROGRAMA PRINCIPAL *)

Begin
        CRIA_LISTA(l);
        Repeat
                ClrScr;
                Write('Valor: ');
                ReadLn(valor);
                If valor <> 0 Then
                        Begin
                                Write('[I]nício ou [F]im ?');
                                Repeat
                                        ch := UpCase(ReadKey);
                                Until ch IN ['I','F'];
                                Case ch
                                Of
                                        'I': erro := INCLUI_INICIO(l,valor);
                                        'F': erro := INCLUI_FIM(l,valor);
```

```
                        End;
                        If erro = LISTA_CHEIA Then
                                Begin
                                        WriteLn('ERRO: Lista Cheia');
                                        ReadLn;
                                End;
                End;
        Until valor = 0;
        ClrScr;
        If x.inicio = 0 Then
                WriteLn(‘ERRO: Lista Vazia’)
        Else
                For i := l.inicio to l.fim Do
                        WriteLn('Valor: ',l.v[i]);
        ReadLn;
End.
```

Problema Proposto:

O programa permitirá inclusão de números inteiros no ***início***, ***fim*** e na ***posição*** escolhida pelo usuário

Solução do Problema Proposto (2):

```
Program Lista_Linear_2;

Uses Crt;

Const  m = 9;
       SUCESSO = 1;
       LISTA_CHEIA = 2;

Type LISTA = Record
                        inicio,fim: Integer;
                        v: Array[1..m] Of Integer;
                End;

Var    l: LISTA;
       valor: Integer;
       ch: Char;
       i,erro: Integer;
       pos: Integer;
       inic,fim: Integer;
```

```
(* ------------------------------------ Cria_Lista *)

Procedure Cria_Lista(Var x: LISTA);
Begin
	x.inicio := 0;
	x.fim := 0;
End;

(* ------------------------------------ Inclui_Inicio *)

Function Inclui_Inicio(Var x: LISTA; valor: Integer): Integer;
Begin
	If x.inicio = 0 Then
		Begin
			x.inicio := m DIV 2;
			x.fim := x.inicio;
			x.v[x.inicio] := valor;
			Inclui_Inicio := SUCESSO;
		End
	Else
		If x.inicio = 1 then
			Inclui_Inicio := LISTA_CHEIA
		Else
			Begin
				x.inicio := x.inicio - 1;
				x.v[x.inicio] := valor;
				Inclui_Inicio := SUCESSO;
			End;
End;

(* ------------------------------------ Inclui_Fim *)

Function Inclui_Fim(Var x: LISTA; valor: Integer): Integer;
Begin
	If x.fim = 0 Then
		Begin
			x.fim := m DIV 2;
			x.inicio := x.fim;
			x.v[x.fim] := valor;
			Inclui_Fim := SUCESSO;
		End
	Else
		If x.fim = m then
			Inclui_Fim := LISTA_CHEIA
		Else
```

```
                        Begin
                                x.fim := x.fim + 1;
                                x.v[x.fim] := valor;
                                Inclui_Fim := SUCESSO;
                        End;
End;

(* ------------------------------------ Inclui_Posicao *)

Function Inclui_Posicao(Var x: LISTA;valor,pos: Integer): Integer;
Var erro: Integer;
Begin
        If x.inicio = 0 Then
                Begin
                        x.inicio := pos;
                        x.fim := pos;
                        x.v[x.inicio] := valor;
                        Inclui_Posicao := SUCESSO;
                End
        Else
        If (x.inicio = 1) And (x.fim = m) Then
                Inclui_Posicao := LISTA_CHEIA
        Else
                If pos = x.inicio-1 Then
                        Begin
                                erro := Inclui_Inicio(x,valor);
                                Inclui_Posicao := erro;
                        End
                Else
                        If pos = x.fim+1 Then
                                Begin
                                        erro := Inclui_Fim(x,valor);
                                        Inclui_Posicao := erro;
                                End
                        Else
                                Begin
                                        For i := x.fim Downto pos Do
                                                x.v[i+1] := x.v[i];
                                        x.v[pos] := valor;
                                        x.fim := x.fim + 1;
                                        Inclui_Posicao := SUCESSO;
                                End;
End;

(* ------------------------------------ Programa Principal *)
```

```
Begin
        Cria_Lista(l);
        Repeat
                ClrScr;
                Write('Valor: ');
                ReadLn(valor);
                If valor <> 0 Then
                        Begin
                                Write('[I]nício, [P]osição ou [F]im ?');
                                Repeat
                                        ch := UpCase(ReadKey);
                                Until ch IN ['I','F','P'];
                                Case ch
                                Of
                                        'I': erro := Inclui_Inicio(l,valor);
                                        'F': erro := Inclui_Fim(l,valor);
                                        'P': Begin
                                                        If l.inicio = 0 Then
                                                                inic := 1
                                                        Else
                                                                If l.inicio = 1 Then
                                                                        inic := 1
                                                                Else
                                                                        inic := l.inicio - 1;
                                                        If l.fim = 0 Then
                                                                fim := 1
                                                        Else
                                                                fim := l.fim + 1;
                                                        WriteLn;
                                                        Repeat
                                                                Write('Posição [',inic,'..',fim,']: ');
                                                                ReadLn(pos);
                                                        Until (pos >= inic) And (pos <= fim);
                                                        erro := Inclui_Posicao(l,valor,pos);
                                                End;
                                End;
                        If erro = LISTA_CHEIA Then
                                Begin
                                        WriteLn('ERRO: Lista Cheia');
                                        ReadLn;
                                End;
                        End;
        Until valor = 0;
        ClrScr;
```

```
        If l.inicio = 0 Then
                WriteLn('ERRO: Lista Vazia')
        Else
                For i := l.inicio To l.fim Do
                        WriteLn('VALOR: ',l.v[i]);
        ReadLn;
End.
```

Vantagens e Desvantagens da Representação por Contigüidade Física:

Vantagens:

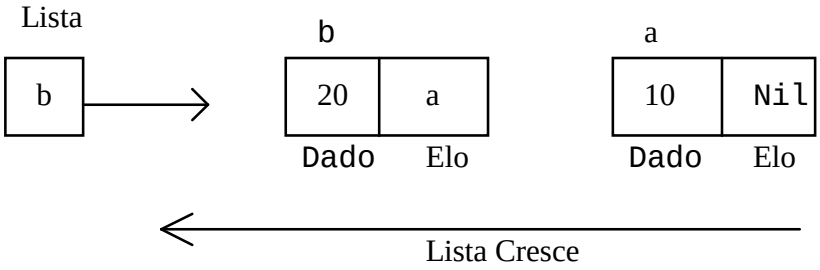
- A consulta pode ser calculada (acesso randômico);
- Facilita a transferência de dados (área de memória contígua);
- Adequada para o armazenamento de estruturas simples.

Desvantagens:

- O tamanho máximo da lista precisa ser conhecido e alocado antecipadamente (alocação estática de memória);
- Inserções e remoções podem exigir considerável movimentação de dados;
- Inadequada para o armazenamento de estruturas complexas;
- Mantém um espaço de memória ocioso (não ocupado);
- Como a lista é limitada, devem ser testados os limites.

Lista Encadeada:

Permite Alocação Dinâmica de Memória, ou seja, a lista cresce com a execução do programa. Operações como inserção e remoção são mais simples. Isto é feito através de variáveis do tipo ponteiro, ou seja, um elemento aponta (possui o endereço, posição de memória do próximo elemento) para o seguinte.




```
Const  SUCESSO = 1;
       FALTA_DE_MEMORIA = 2;
       LISTA_VAZIA = 3;

Type   ponteiro = ^elemento;
       elemento = Record
                        dado: Integer;
                        elo: ponteiro;
                  End;
Var    lista: ponteiro;
       valor: Integer;
```

Criar a Lista:

```
Procedure Cria_Lista (Var lista: Ponteiro);
Begin
       lista := NIL;
End;
```

Incluir na Lista:

```
Function INCLUI_LISTA (Var lista: Ponteiro; valor: Integer): Integer;
Var p: Ponteiro;
Begin
```

```
        New(p);
        If p = NIL Then
                INCLUI_LISTA := FALTA_DE_MEMORIA
        Else
                Begin
                        INCLUI_LISTA := SUCESSO;
                        If lista = NIL Then
                                Begin
                                        lista := p;
                                        p^.dado := valor;
                                        p^.elo := NIL;
                                End
                        Else
                                Begin
                                        p^.dado := valor;
                                        p^.elo := lista;
                                        lista := p;
                                End;
                End;
End;
```

Remover o Primeiro Elemento da Lista:

```
Function REMOVER_PRIMEIRO (Var lista: Ponteiro): Integer;
Var p: Ponteiro;
Begin
        If lista = Nil Then
                REMOVER_PRIMEIRO := LISTA_VAZIA
        Else
                Begin
                        REMOVER_PRIMEIRO := SUCESSO;
                        p := lista;
                        lista := p^.elo;
                        Dispose(p);
                End;
End;
```

Contar o Número de Elementos de uma Lista:

```
Function CONTAR_ELEMENTOS_LISTA (lista: Ponteiro): Integer;
Var     p: Ponteiro;
        n: Integer;
Begin
        If lista = NIL Then
```

```
            CONTAR_ELEMENTOS_LISTA := 0
        Else
            Begin
                n := 0;
                p := lista;
                While p^.elo <> NIL Do
                    Begin
                        n := n + 1;
                        p := p^.elo;
                    End;
                CONTAR_ELEMENTOS_LISTA := n;
            End;
End;
```

Remover o Último Elemento da Lista:

```
Function REMOVER_ÚLTIMO (Var lista: Ponteiro): Integer;
Var p,q: ponteiro;
Begin
        If lista = NIL Then
            REMOVER_ÚLTIMO := LISTA_VAZIA
        Else
            Begin
                REMOVER_ÚLTIMO := SUCESSO;
                q := lista;
                p := lista;
                While p^.elo <> NIL Do
                    Begin
                        q := p;
                        p := p^.elo;
                    End;
                If lista = q Then
                    lista := NIL
                Else
                    q^.elo := NIL;
                Dispose(p);
            End;
End;
```

Problema Proposto:

Escrever uma função que obtém o conteúdo do último elemento de uma lista encadeada.

Problemas das Listas Encadeadas:

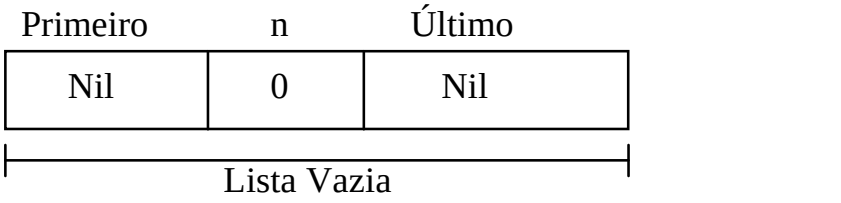
- Determinar o número de elementos da lista;
- Acessar o último elemento da lista.

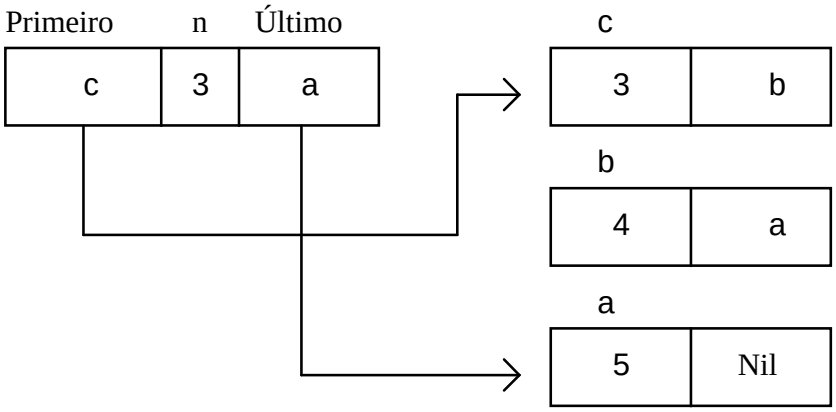
Vantagens das Listas Encadeadas:

- Lista cresce indeterminadamente (Alocação Dinâmica de Memória);
- O primeiro elemento da lista é conhecido.

Lista com Descritor

Como foi visto anteriormente, um dos problemas da lista encadeada, é descobrir o número de elementos e, o outro problema, é descobrir quem é o último elemento. Estes dois problemas podem ser resolvidos, usando-se um ***descritor*** para a lista, da seguinte maneira:






```
Program Lista_Encadeada_com_Descritor;

Const  SUCESSO = 1;
       FALTA_DE_MEMORIA = 2;
       LISTA_VAZIA = 3;

Type   ponteiro = ^elemento;
       descritor = Record
                        primeiro: ponteiro;
```

```
                    n: Integer;
                    último: ponteiro;
                End;
        elemento = Record
                    dado: Integer;
                    elo: ponteiro;
                End;
Var     d: descritor;
        valor: Integer;
```

Inicialização do Descritor:

```
Procedure INICIALIZA_DESCRITOR( Var d: Descritor);
Begin
        d.primeiro := NIL;
        d.n := 0;
        d.último := NIL;
End;
```

Inserir a Direita da Lista:

```
Function DIREITA(Var d: Descritor; valor: Integer): Integer;
Var p,q: ponteiro;
Begin
        New(p);
        If p = NIL Then
                DIREITA := FALTA_DE_MEMORIA
        Else
                Begin
                        DIREITA := SUCESSO;
                        p^.dado := valor;
                        p^.elo := NIL;
                        If d.n = 0 Then
                                Begin
                                        d.primeiro := p;
                                        d.último := p;
                                        d.n := 1;
                                End
                        Else
                                Begin
                                        q := d.último;
                                        d.último := p;
                                        q^.elo := p;
                                        d.n := d.n + 1;
```

```
                    End;
            End;
End;
```

Inserir a Esquerda da Lista:

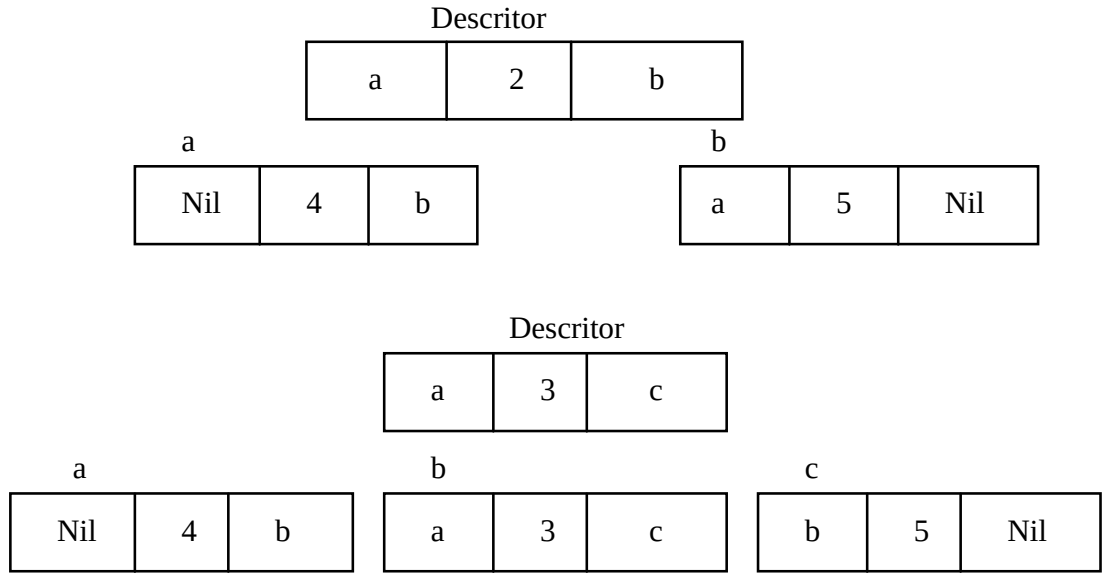
```
Function ESQUERDA(Var d: Descritor; valor: Integer): Integer;
Var p: ponteiro;
Begin
    New(p);
    If p = NIL Then
        ESQUERDA := FALTA_DE_MEMORIA
    Else
        Begin
            ESQUERDA ;= SUCESSO;
            p^.dado := valor;
            If d.n = 0 Then
                Begin
                    d.primeiro := p;
                    d.último := p;
                    d.n := 1;
                    p^.elo := NIL;
                End
            Else
                Begin
                    p^.elo := d.primeiro;
                    d.primeiro := p;
                    d.n := d.n + 1;
                End;
        End;
End;
```

Desvantagem da Lista Encadeada:

A lista encadeada, por possuir apenas um elo para o próximo elemento, só pode ser percorrida num sentido, para resolver este problema deve-se usar ***Listas Duplamente Encadeadas***.

Lista Duplamente Encadeada

Na lista duplamente encadeada, cada elemento possui um elo para o ***anterior*** e o ***posterior***, sendo que a lista pode ter ou não ***descritor***.



Definição de uma Lista Duplamente Encadeada com Descritor:

```
Type    ponteiro = ^elemento;
        elemento = Record
                                anterior: ponteiro;
                                dado: Integer;
                                posterior: ponteiro;
                        End;
        descritor = Record
                                primeiro: ponteiro;
                                n: Integer;
                                último: ponteiro;
                        End;
Var     d: descritor;
        valor: Integer;
```

Problema Proposto:

Escrever as seguintes funções:

Function **Esquerda**(Var d: Descritor; valor: Integer): Integer;
Function **Direita**(Var d: Descritor; valor: Integer): Integer;

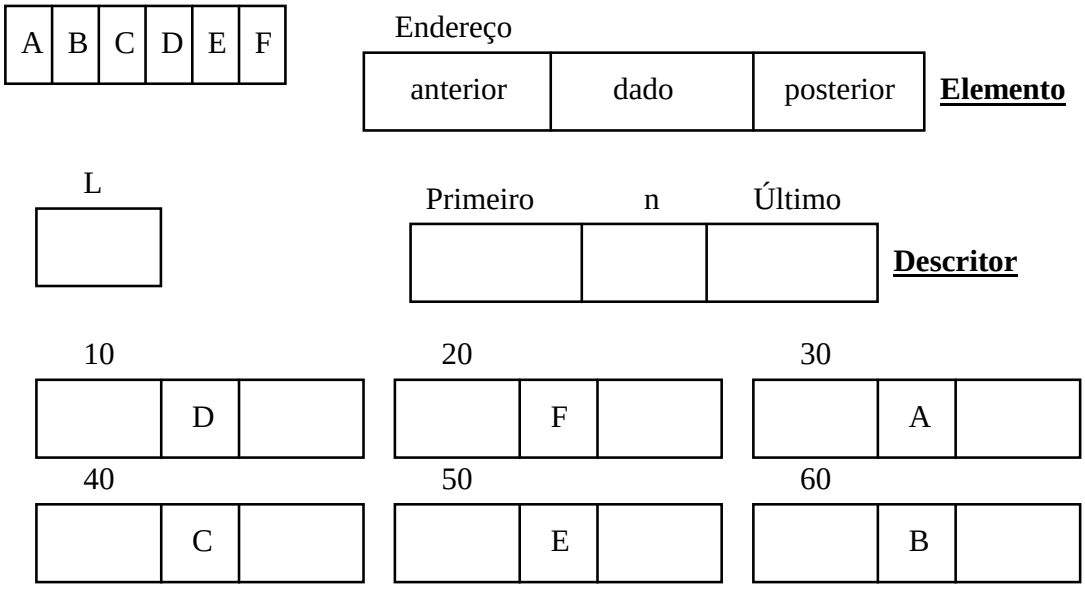
```
Function DIREITA (Var d: Descritor; valor: Integer): Integer;
Var p,q: Integer;
Begin
```

```
        New(p);
        If p = NIL Then
                DIREITA := FALTA_DE_MEMORIA
        Else
                Begin
                        DIREITA := SUCESSO;
                        p^.dado := valor;
                        p^.posterior := NIL;
                        If d.n = 0 Then
                                Begin
                                        d.primeiro := p;
                                        d.n := 1;
                                        d.ultimo := p;
                                        p^.anterior := NIL;
                                End
                        Else
                                Begin
                                        q := d.ultimo;
                                        d.ultimo := p;
                                        q^.posteior := p;
                                        p^.anterior := q;
                                End;
                End;
End;

Function ESQUERDA (Var d: Descritor; valor: Integer): Integer;
Var p,q: Integer;
Begin
        New(p);
        If p = NIL Then
                ESQUERDA := FALTA_DE_MEMORIA
        Else
                Begin
                        ESQUERDA := SUCESSO;
                        p^.dado := valor;
                        p^.anterior := NIL;
                        If d.n = 0 Then
                                Begin
                                        d.primeiro := p;
                                        d.n := 1;
                                        d.ultimo := p;
                                        p^.posterior := NIL;
                                End
                        Else
                                Begin
```

```
                    q := d.primeiro;
                    d.primeiro := p;
                    q^.anterior := p;
                    p^.posterior := q;
                End;
            End;
End;
```

Problema Proposto:

Preencha os campos de elo com os valores adequados para que seja representada a seqüência (A,B,C,D,E,F): (com ou sem descritor)



Vantagens e Desvantagens das Listas Duplamente Encadeadas:

Vantagens:

- Representação simples;
- Inserção e remoção de componentes sem movimentar os demais;
- Pode-se ter qualquer quantidade de elementos, pois é usado variáveis do tipo ponteiro.

Desvantagens:

- Gerência de memória mais onerosa, ***Alocação*** / ***Desalocação*** para cada elemento;
- Procedimentos mais complexos;

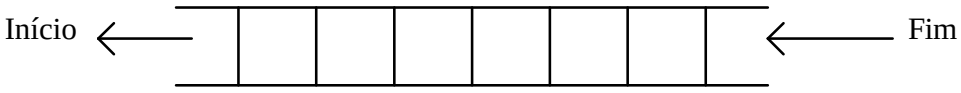
- Processamento serial (Acesso Seqüencial).

Listas Lineares com Disciplina de Acesso

Tipos especiais de ***Listas Lineares***, onde inserção, consulta e exclusão são feitas somente nos extremos. Estas listas lineares com disciplina de acesso são: ***Filas***, ***Pilhas*** e ***Deques***.

Filas

Lista Linear na qual as inserções são feitas no fim e as exclusões e consultas no início da fila.



***Critério de Utilização*:**

FIFO - "**F**irst **I**n **F**irst **O**ut" (primeiro a entrar é o primeiro a sair)

Operações sobre Filas:

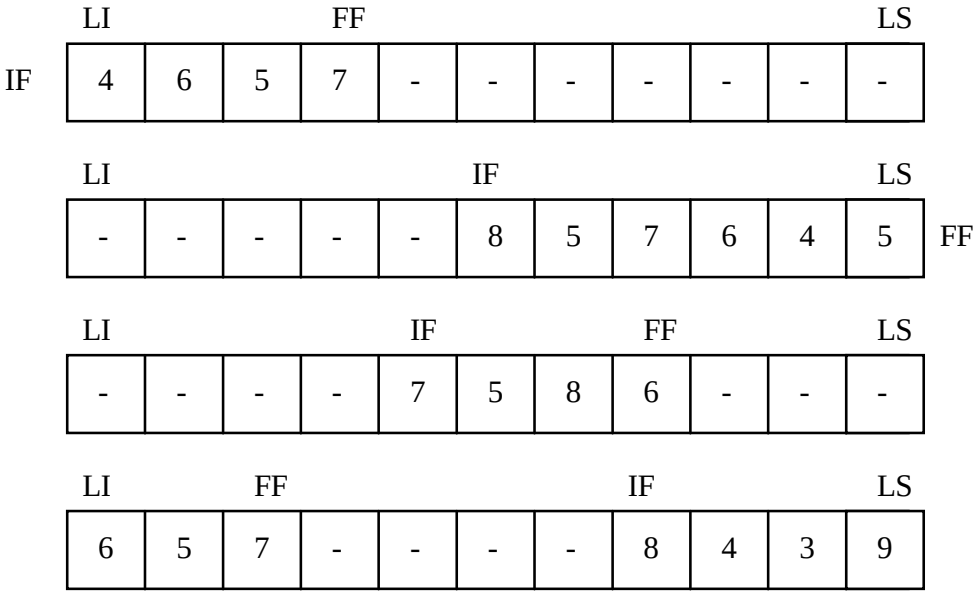
CRIA_FILA(f);	Cria fila Vazia
DESTROI_FILA(f);	Desfaz a fila
erro := INSERE_FILA(f,i);	Insere o dado "i" no fim da FILA
erro := EXCLUI_FILA(f);	Retira da FILA o primeiro elemento
erro := CONSULTA_FILA(f,j);	Copia em "j" o primeiro elemento

Erros nas Operações sobre Filas:

FILA CHEIA
FILA VAZIA

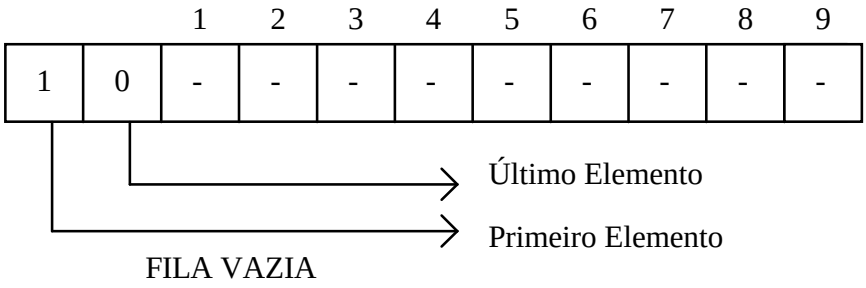
Identificação de uma Fila:

LI(f)	Limite Inferior da FILA
LS(f)	Limite Superior da FILA
IF(f)	Início da FILA
FF(f)	Final da FILA



Lista Circular

Uma fila ainda pode ser representada assim:



Fila com Vetor

```
Const m = 9;
Type    tipo_nodo = Integer;
        Fila = Record
                        primeiro: Integer;
                        último: Integer;
                        elem: Array[1..m] de tipo_nodo;
                End;
Var     f: fila;
        i,j: Integer;
```

Criar a Fila:

```
Procedure CRIA_FILA (Var f: fila);
Begin
        f.primeiro := 1;
        f.último := 0;
End;
```

Inserir na Fila:

```
Function INSERE_FILA(Var f: Fila; i: tipo_nodo): Integer
Begin
        If f.último = m Then
                INSERE_FILA := FILA_CHEIA
        Else
                Begin
                        f.último := f.último + 1;
                        f.elem[f.último] := i;
                        INSERE_FILA := SUCESSO;
                End;
End;
```

Excluir da Fila:

```
Function EXCLUI_FILA (Var f: Fila): Integer;
Begin
        If f.último = 0 Then
                EXCLUI_FILA := FILA_VAZIA
        Else
                Begin
                        f.primeiro := f.primeiro + 1;
                        EXCLUI_FILA := SUCESSO;
                End;
End;
```

Consultar o Primeiro Elemento da Fila:

```
Function CONSULTA_FILA(f: Fila; Var j: tipo_nodo): Integer;
Begin
        If f.último = 0 Then
                CONSULTA_FILA := FILA_VAZIA
        Else
                Begin
```

```
                    j := f.elem[f.primeiro];
                    CONSULTA_FILA := SUCESSO;
                End;
End;
```

Exemplo: FILA VAZIA

	1	2	3	4	5	6	7	8	9
1	0	-	-	-	-	-	-	-	-

Inclusão de: 3, 5, 4, 7, 8 e 6

IF	FF	1	2	3	4	5	6	7	8	9
1	6	3	5	4	7	8	6	-	-	-
		IF					FF			

Retirada dos Primeiros três Elementos:

IF	FF	1	2	3	4	5	6	7	8	9
4	6	-	-	-	7	8	6	-	-	-
					IF		FF			

Problema com as Filas:

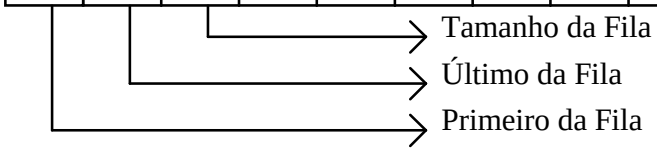
A ***reutilização*** da fila depois que alguns elementos foram extraídos.

Solução deste Problema:

Para reutilizar “as vagas” dos elementos já extraídos, deve-se usar uma ***Fila Circular:***

Fila Circular

	1	2	3	4	5	6	7	8	9
1	0	0	-	-	-	-	-	-	-	-

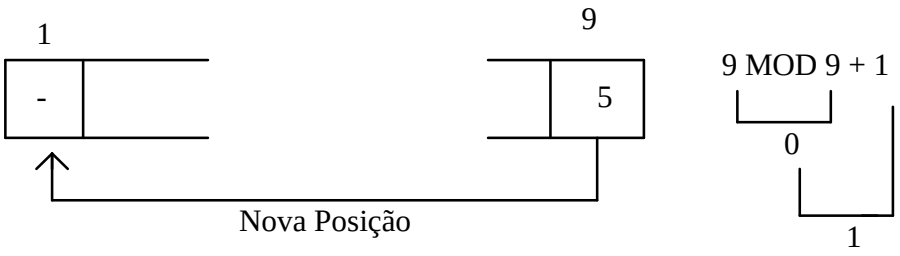


```
Program Fila_Estatica;

Uses Crt;

Const m = 9;
        SUCESSO = 1;
        FILA_CHEIA = 2;
        FILA_VAZIA = 3;

Type    tipo_nodo = Integer;
        fila = Record
                        primeiro: Integer;
                        ultimo: Integer;
                        tamanho: Integer;
                        elem: Array[1..m] Of tipo_nodo;
                End;

Var     f: fila;
        i,j: Tipo_Nodo;
        ch: Char;
        erro: Integer;
        valor: tipo_nodo;

(* ---------------------------------------------- CRIA_FILA_CIRCULAR *)

Procedure CRIA_FILA_CIRCULAR(Var f: fila);
Begin
        f.primeiro := 1;
        f.ultimo := 0;
        f.tamanho := 0;
End;

(* ---------------------------------------------- INSERE_FILA_CIRCULAR *)

Function INSERE_FILA_CIRCULAR(Var f: fila; i: tipo_nodo): Integer;
Begin
        If f.tamanho = m Then
                INSERE_FILA_CIRCULAR := FILA_CHEIA
        Else
                Begin
                        f.tamanho := f.tamanho + 1;
                        f.ultimo := f.ultimo MOD m + 1;
                        f.elem[f.ultimo] := i;
                        INSERE_FILA_CIRCULAR := SUCESSO;
```

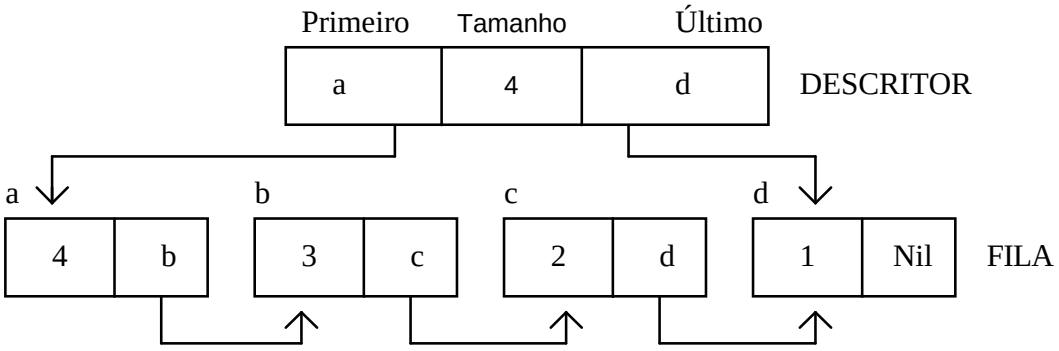
```
                End;
End;

(* ---------------------------------------------- EXCLUI_FILA_CIRCULAR *)

Function EXCLUI_FILA_CIRCULAR(Var f: fila): Integer;
Begin
        If f.tamanho = 0 Then
                EXCLUI_FILA_CIRCULAR := FILA_VAZIA
        Else
                Begin
                        f.tamanho := f.tamanho - 1;
                        f.primeiro := f.primeiro MOD m + 1;
                        EXCLUI_FILA_CIRCULAR := SUCESSO;
                End;
End;

(* ---------------------------------------------- CONSULTA_FILA_CIRCULAR *)

Function CONSULTA_FILA_CIRCULAR (f: fila; Var j: tipo_nodo): Integer;
Begin
        CONSULTA_FILA_CIRCULAR := SUCESSO;
        If f.tamanho = 0 Then
                CONSULTA_FILA_CIRCULAR := FILA_VAZIA
        Else
                j := f.elem[f.primeiro];
End;

(* ---------------------------------------------- PROGRAMA PRINCIPAL *)

Begin
        CRIA_FILA_CIRCULAR(f);
        Repeat
                ClrScr;
                Write('[I]ncluir, [E]xcluir, [C]onsultar ou [F]im?');
                Repeat
                        ch := UpCase(ReadKey);
                Until ch IN ['I','E','C','F'];
                If ch = 'I' Then
                Begin
                        ClrScr;
                        Write('Valor: ');
                        ReadLn(valor);
                End;
                Case ch
```

```
                Of
                        'I': erro := INSERE_FILA_CIRCULAR(f,valor);
                        'E': erro := EXCLUI_FILA_CIRCULAR(f);
                        'C': Begin
                                erro := CONSULTA_FILA_CIRCULAR(f,valor);
                                If erro = SUCESSO Then
                                        Begin
                                                ClrScr;
                                                WriteLn('VALOR: ',valor);
                                                ReadLn;
                                        End;
                          End;
                End;
                If (erro <> SUCESSO) And (ch <> ‘F’) Then
                Begin
                        ClrScr;
                        Case erro
                        Of
                                FILA_CHEIA: WriteLn('ERRO: Lista Cheia');
                                FILA_VAZIA: WriteLn('ERRO: Lista Vazia');
                        End;
                        ReadLn;
                End;
        Until ch = 'F';
End.
```

Como calcular a nova posição:



Fila com Alocação Dinâmica:




```
Program Fila_Dinamica;

Uses Crt;

Const SUCESSO = 1;
      FILA_VAZIA = 2;
      FALTA_DE_MEMORIA = 3;

Type  tipo_nodo = Integer;
      elemento = ^dado;
      dado = Record
                  info: tipo_nodo;
                  seg: elemento;
             End;
      fila = Record
                  primeiro: elemento;
                  tamanho: Integer;
                  ultimo: elemento;
             End;
Var   f: fila;
      t: elemento;
      i,j: tipo_nodo;
      ch: Char;
      valor: tipo_nodo;
      erro: Integer;

(* ---------------------------------------------- CRIA_FILA *)

Procedure CRIA_FILA(Var f: fila);
Begin
      f.primeiro := Nil;
      f.tamanho := 0;
      f.ultimo := Nil;
```

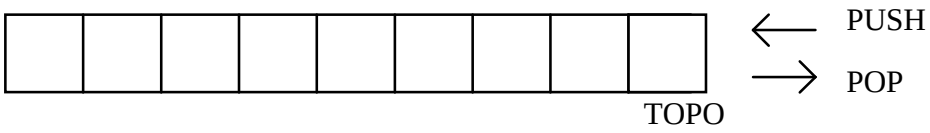
```
End;

(* ---------------------------------------------- INSERIR_FILA *)

Function INSERIR_FILA(Var f: fila; valor: tipo_nodo): Integer;
Begin
        New(t);
        If t = NIL Then
                INSERIR_FILA := FALTA_DE_MEMORIA
        Else
                Begin
                        t^.info := valor;
                        t^.seg := Nil;
                        If f.ultimo <> Nil Then
                                f.ultimo^.seg := t;
                        f.ultimo := t;
                        If f.primeiro = Nil Then
                                f.primeiro := t;                { FILA com 1 elemento }
                        INSERIR_FILA := SUCESSO;
                End;
End;

(* ---------------------------------------------- EXCLUIR_FILA *)

Function EXCLUIR_FILA (Var f: fila): Integer;
Begin
        EXCLUIR_FILA := SUCESSO;
        If f.primeiro <> Nil Then
                Begin
                        t := f.primeiro;
                        f.tamanho := f.tamanho - 1;
                        f.primeiro := t^.seg;
                        Dispose(t);
                        If f.primeiro = Nil Then
                                Begin
                                        f.ultimo := Nil; {FILA ficou VAZIA }
                                        EXCLUIR_FILA := FILA_VAZIA;
                                End;
                End;
End;

(* ---------------------------------------------- CONSULTAR_FILA *)

Function CONSULTAR_FILA (f: fila; Var j: tipo_nodo): Integer;
Begin
```

```
        CONSULTAR_FILA := SUCESSO;
        If f.primeiro = Nil Then
                CONSULTAR_FILA := FILA_VAZIA
        Else
                j := f.primeiro^.info;
End;

(* ---------------------------------------------- Imprime_Erro *)

Procedure Imprime_Erro(erro: Integer);
Begin
        Case erro
        Of
                FILA_VAZIA: WriteLn('ERRO: Fila Vazia');
                FALTA_DE_MEMORIA: WriteLn('ERRO: Falta de Memoria');
        End;
        ReadLn;
End;

(* ---------------------------------------------- Destroi_FIla *)

Procedure Destroi_Fila(Var f: Fila);
Begin
        While f.tamanho <> 0 Do
                erro := EXCLUIR_FILA(f);
End;

(* ---------------------------------------------- PROGRAMA PRINCIPAL *)

Begin
        CRIA_FILA(f);
        Repeat
                ClrScr;
                Write('[I]ncluir, [E]xcluir, [C]onsultar ou [F]im?');
                Repeat
                        ch := UpCase(ReadKey);
                Until ch IN ['I','E','C','F'];
                If ch = 'I' Then
                        Begin
                                ClrScr;
                                Write('Valor: ');
                                ReadLn(valor);
                        End;
                Case ch
                Of
```

```
                    'I': INSERIR_FILA(f,valor);
                    'E': Begin
                                erro := EXCLUIR_FILA(f);
                                If erro <> SUCESSO Then
                                        Imprime_Erro(erro);
                    'C': Begin
                                erro := CONSULTAR_FILA(f,valor);
                                If erro <> SUCESSO Then
                                        Imprime_Erro(erro);
                                Else
                                        Begin
                                                ClrScr;
                                                WriteLn('Valor: ',valor);
                                                ReadLn;
                                        End;
                                End;
                End;
        Until ch = 'F';
        Destroi_Fila(f);
End.
```

Pilhas

É uma ***Lista Linear*** na qual as inserções, exclusões e consultas são feitas em um mesmo extremo (TOPO).



***Critério de Utilização*:**

LIFO - "**L**ast **I**n **F**irst **O**ut" (último a entrar é o primeiro a sair)

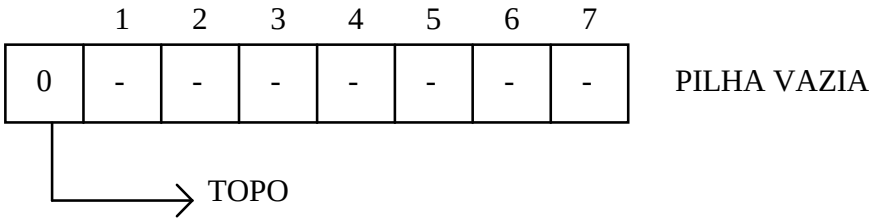
Operações sobre Pilhas:

CRIA_PILHA(p);	Cria pilha Vazia
DESTROI_PILHA(p);	Desfaz a pilha
erro := PUSH(p,i);	Empilha o dado "i" no fim da PILHA
erro := POP(p,i);	Desempilha o primeiro elemento
erro := CONSULTA_PILHA(p,j);	Copia em "j" o primeiro elemento

Identificação da Pilha:

B(p)	Base da PILHA
T(p)	Topo da PILHA
L(p)	Limite da PILHA

Pilha com Vetor




```
Program Pilha_com_Vetor;

Uses Crt;

Const m = 7;
      SUCESSO = 1;
      PILHA_CHEIA = 2;
      PILHA_VAZIA = 3;
```

```
Type    tipo_nodo = Integer;
        pilha = Record
                                topo: Byte;
                                elem: Array[1..m] Of tipo_nodo;
                End;

Var      p: pilha;
        i,j: tipo_ nodo;
        ch: Char;
        valor: tipo_nodo;
        erro: Integer;

(* ---------------------------------------------- CRIA_PILHA *)

Procedure CRIA_PILHA(Var p: pilha);
Begin
        p.topo := 0;
End;

(* ------------------------------------------------ PUSH *)

Function PUSH(Var p: pilha; i: tipo_nodo): Integer;
Begin
        If p.topo = m Then
                PUSH := PILHA_CHEIA
        Else
                Begin
                        p.topo := p.topo + 1;
                        p.elem[p.topo] := i;
                        PUSH := SUCESSO;
                End;
End;

(* ------------------------------------------------ POP *)

Function POP(Var p: pilha; Var i: tipo_nodo): Integer;
Begin
        If p.topo = 0 Then
                POP := PILHA_VAZIA
        Else
                Begin
                        i := p.elem[p.topo];
                        p.topo := p.topo - 1;
                        POP := SUCESSO;
```

```
                End;
End;

(* ---------------------------------------------- CONSULTA_PILHA *)

Function CONSULTA_PILHA(Var p: pilha; Var j: tipo_nodo): Integer;
Begin
        If p.topo = 0 Then
                CONSULTA_PILHA := PILHA_VAZIA
        Else
                Begin
                        j := p.elem[p.topo];
                        CONSULTA_PILHA := SUCESSO;
                End;
End;

(* ---------------------------------------------- Imprime_Erro *)

Prcedure Imprime_Erro(erro: Integer);
Begin
        Case erro
        Of
                PILHA_CHEIA: WriteLn('ERRO: Pilha Cheia');
                PILHA_VAZIA: WriteLn('ERRO: Pilha Vazia');
        End;
        Readln;
End;

(* ---------------------------------------------- PROGRAMA PRINCIPAL *)

Begin
        CRIA_PILHA(p);
        Repeat
                ClrScr;
                Write('[P]ush, p[O]p, [C]onsultar ou [F]im?');
                Repeat
                        ch := UpCase(ReadKey);
                Until ch IN ['P','O','C','F'];
                If ch = 'P' Then
                        Begin
                                ClrScr;
                                Write('Valor: ');
                                ReadLn(valor);
                        End;
                Case ch
```

```
            Of
                    'P': Begin
                                erro := PUSH(p,valor);
                                If erro <> SUCESSO Then
                                    Imprime_Erro(erro);
                        End;
                    'O': Begin
                                erro := POP(p,valor);
                                ClrScr;
                                If erro <> SUCESSO Then
                                    Imprime_Erro(erro)
                                Else
                                    Begin
                                        ClrScr;
                                        WriteLn('Valor: ',valor);
                                        ReadLn;
                                    End;
                            End;
                    'C': Begin
                                erro := CONSULTA_PILHA(p,valor);
                                If erro <> SUCESSO Then
                                    Imprime_Erro(erro)
                                Else
                                    Begin
                                        ClrScr;
                                        WriteLn('Valor: ',valor);
                                        ReadLn;
                                    End;
                        End;
            End;
    Until ch = 'F';
End.
```

Problema Proposto:

Torre de Hanoi

Solução do Problema:

```
Program Torre_de_Hanoi;

Uses Crt;

Const m = 3;
    SUCESSO = 1;
```

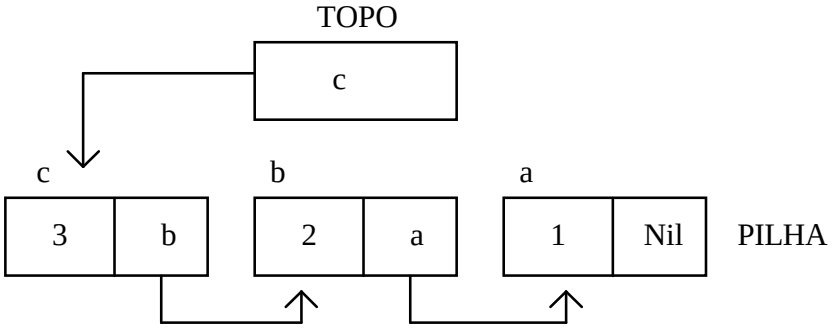
```
        PILHA_CHEIA = 2;
        PILHA_VAZIA = 3;

Type Tipo_Nodo = Integer;
        Pilha = Record
                        topo: Byte;
                        elem: Array[1..m] Of Tipo_Nodo;
                End;

Var     p1,p2,p3: Pilha;
        erro: Integer;
        valor: Tipo_Nodo;
        ch: Char;

(* ---------------------------------------- Cria_Pilha *)

Procedure Cria_Pilha(Var p: Pilha);
Begin
        p.topo := 0;
End;

(* ---------------------------------------- Push *)

Function Push(Var p: Pilha; i: Tipo_Nodo): Integer;
Begin
        If p.topo = m Then
                Push := PILHA_CHEIA
        Else
                Begin
                        p.topo := p.topo + 1;
                        p.elem[p.topo] := i;
                        Push := SUCESSO;
                End;
End;

(* ---------------------------------------- Pop *)

Function Pop(Var p: Pilha; Var j: Tipo_Nodo): Integer;
Begin
        If p.topo = 0 Then
                Pop := PILHA_VAZIA
        Else
                Begin
                        j := p.elem[p.topo];
                        p.topo := p.topo - 1;
```

```
                    Pop := SUCESSO;
                End;
End;

(* ---------------------------------------- Consulta_Pilha *)

Function Consulta_Pilha(Var p: Pilha; Var j: Tipo_Nodo): Integer;
Begin
        If p.topo = 0 Then
                Consulta_Pilha := PILHA_VAZIA
        Else
                Begin
                        j := p.elem[p.topo];
                        Consulta_Pilha := SUCESSO;
                End;
End;

(* ---------------------------------------- Hanoi *)

Procedure Hanoi(t: Byte);
Begin
        Case t
        Of
                2: Begin
                                erro := Pop(p1,valor);
                                erro := Push(p2,valor);
                                erro := Pop(p1,valor);
                                erro := Push(p3,valor);
                                erro := Pop(p2,valor);
                                erro := Push(p3,valor);
                                erro := Pop(p1,valor);
                                erro := Push(p2,valor);
                                erro := Pop(p3,valor);
                                erro := Push(p1,valor);
                                erro := Pop(p3,valor);
                                erro := Push(p2,valor);
                                erro := Pop(p1,valor);
                                erro := Push(p2,valor);
                  End;
                3: Begin
                                erro := Pop(p1,valor);
                                erro := Push(p3,valor);
                                erro := Pop(p1,valor);
                                erro := Push(p2,valor);
                                erro := Pop(p3,valor);
```

```
                                erro := Push(p2,valor);
                                erro := Pop(p1,valor);
                                erro := Push(p3,valor);
                                erro := Pop(p2,valor);
                                erro := Push(p1,valor);
                                erro := Pop(p2,valor);
                                erro := Push(p3,valor);
                                erro := Pop(p1,valor);
                                erro := Push(p3,valor);
                        End;
        End;
End;

(* ---------------------------------- Lista_Pilha *)

Procedure Lista_Pilha(p: Pilha);
Var i: Byte;
Begin
        If p.topo = 0 Then
                Write('Pilha Vazia')
        Else
                While (p.topo <> 0) Do
                        Begin
                                erro := Pop(p,valor);
                                Write(valor:2,' ');
                        End;
        WriteLn;
End;

(* ---------------------------------- Programa Principal *)

Begin
        ClrScr;
        Repeat
                Cria_Pilha(p1);
                Cria_Pilha(p2);
                Cria_Pilha(p3);
                erro := Push(p1,30);
                erro := Push(p1,20);
                erro := Push(p1,10);
                Write('Escolha: Pilha [1], Pilha[2], Pilha[3] ou [F]im ?');
                Repeat
                        ch := UpCase(ReadKey);
                Until ch IN ['1','2','3','F'];
                Case ch
```

```
                Of
                        '1': hanoi(1);
                        '2': hanoi(2);
                        '3': hanoi(3);
                End;
        If ch <> 'F' Then
                Begin
                        WriteLn;
                        Write('Pilha[1]: ');
                        Lista_Pilha(p1);
                        Write('Pilha[2]: ');
                        Lista_Pilha(p2);
                        Write('Pilha[3]: ');
                        Lista_Pilha(p3);
                End;
        Until ch = 'F';
End.
```

Pilha com Alocação Dinâmica:




```
Program Pilha_Encadeada;

Uses Crt;

Const SUCESSO = 1;
        FALTA_DE_MEMORIA = 2;
        PILHA_VAZIA = 3;

Type    tipo_nodo = Integer;
        pilha = ^dado;
        dado = Record
                        info: tipo_nodo;
                        seg: pilha;
                End;
```

```
Var     p,t: pilha;                     { t é o topo }
        i,j: tipo_nodo;
        ch: Char;
        erro: Integer;
        valor: tipo_nodo;

(* ---------------------------------------------- CRIA_PILHA *)

Procedure CRIA_PILHA;
Begin
        t := NIL;
End;

(* ---------------------------------------------- PUSH *)

Function PUSH(Var p: pilha; i: tipo_nodo): Integer;
Begin
        New(p);
        If p = NIL Then
                PUSH := FALTA_DE_MEMORIA
        Else
                Begin
                        PUSH := SUCESSO;
                        p^.info := i;
                        p^.seg := t;
                        t := p;
                End;
End;

(* ---------------------------------------------- POP *)

Function POP(Var p: pilha; Var i: tipo_nodo): Integer;
Begin
        If t = NIL Then
                POP := PILHA_VAZIA
        Else
                Begin
                        p := t;
                        i := p^.info;
                        t := p^.seg;
                        Dispose(p);
                        POP := SUCESSO;
                End;
End;
```

```
(* ---------------------------------------------- CONSULTA_PILHA *)

Function CONSULTA_PILHA(p: pilha; VAR j: tipo_nodo): Integer;
Begin
        If t = Nil Then
                CONSULTA_PILHA := PILHA_VAZIA
        Else
                Begin
                        j := t^.info;
                        CONSULTA_PILHA := SUCESSO;
                End;
End;

(* ---------------------------------------------- Imprime_Erro *)

Prcedure Imprime_Erro(erro: Integer);
Begin
        Case erro
        Of
                FALTA_DE_MEMORIA: WriteLn('ERRO: Falta de Memória');
                PILHA_VAZIA: WriteLn('ERRO: Pilha Vazia');
        End;
        Readln;

(* ---------------------------------------------- PROGRAMA PRINCIPAL *)

Begin
        CRIA_PILHA(p);
        Repeat
                ClrScr;
                Write('[P]ush, p[O]p, [C]onsultar ou [F]im?');
                Repeat
                        ch := UpCase(ReadKey);
                Until ch IN ['P','O','C','F'];
                If ch = 'P' Then
                        Begin
                                ClrScr;
                                Write('Valor: ');
                                ReadLn(valor);
                        End;
                Case ch
                Of
                        'P': Begin
                                        erro := PUSH(p,valor);
                                        If erro <> SUCESSO Then
```

```
                        Imprime_Erro(erro);
                End;
        'O': Begin
                        erro := POP(p,valor);
                        If erro <> SUCESSO Then
                                Imprime_Erro(erro)
                        Else
                                Begin
                                        ClrScr;
                                        WriteLn('VALOR: ',valor);
                                        ReadLn;
                                End;
                End;
        'C': Begin
                        erro := CONSULTA_PILHA(p,valor);
                        If erro <> SUCESSO Then
                                Imprime_Erro(erro)
                        Else
                                Begin
                                        ClrScr;
                                        WriteLn('VALOR: ',valor);
                                        ReadLn;
                                End;
                End;
        End;
Until ch = 'F';
End.
```

Trabalho Proposto:

Criar um Analisador de Expressões usando Pilhas.

Exemplo: (3 * (4 + 5))

String

0	1	2	3	4	5	6	7	8	9	
9	(	3	+	(	4	*	5	)	)	

e

DICAS:

1. Crie duas PILHAS (números e operandos)
2. "(" não faça nada
3. Número (PUSH) empilhe na pilha de números
4. Operando (PUSH) empilhe na pilha de operandos

5. ")" execute a operação
6. Até que i = e[0]

Valores Válidos:

Números: 0,1,2,3,4,5,6,7,8,9 (Números Inteiros: Positivos ou Negativos)
Operandos: +,-,*,/

	t	1	2	3	4	5	6	7	8	9	
P1	3	3	4	5	-	-	-	-	-	-	
P2	2	*	+	-	-	-	-	-	-	-	

	t	1	2	3	4	5	6	7	8	9	
P1	2	3	9	-	-	-	-	-	-	-	
P2	1	*	-	-	-	-	-	-	-	-	

	t	1	2	3	4	5	6	7	8	9	
P1	1	27	-	-	-	-	-	-	-	-	
P2	0	-	-	-	-	-	-	-	-	-	

Solução do Trabalho Proposto:

```
Program Analisador_Expressao;

Uses Crt;

Const N1 = 20;
      N2 = 10;
      PILHA_CHEIA = 1;
      PILHA_VAZIA = 2;
      SUCESSO = 3;

Type PILHA_VALOR = Record
                                topo: Integer;
                                valor: Array[1..N1] Of Integer;
                           End;
      PILHA_OPERADOR = Record
```

```
                                        topo: Integer;
                                        operador: Array[1..N2] Of Char;
                                End;

(* ------------------------------------------ Testa_Expressao *)

Function Testa_Expressao(s: String): Boolean;
Var     i,n: Byte;
        abre,fecha: Byte;
Begin
        abre := 0;
        fecha := 0;
        n := Length(s);
        For i := 1 to n Do
                If s[i] = '(' Then
                        Inc(abre)
                Else
                        If s[i] = ')' Then
                                Inc(fecha);
        If abre = fecha Then
                Testa_Expressao := TRUE
        Else
                Testa_Expressao := FALSE;
End;

(* ------------------------------------------ Cria_Pilha_Valor *)

Procedure Cria_Pilha_Valor(Var v: PILHA_VALOR);
Begin
        v.topo := 0;
End;

(* ------------------------------------------ Cria_Pilha_Operador *)

Procedure Cria_Pilha_Operador(Var op: PILHA_OPERADOR);
Begin
        op.topo := 0;
End;

(* ------------------------------------------ Empilha_Valor *)

Function Empilha_Valor(Var v: PILHA_VALOR; valor: Integer): Integer;
Begin
        If v.topo = N1 Then
                Empilha_Valor := PILHA_CHEIA
```

```
        Else
                Begin
                        v.topo := v.topo + 1;
                        v.valor[v.topo] := valor;
                        Empilha_Valor := SUCESSO;
                End;
End;

(* ---------------------------------------------- Empilha_Operador *)

Function Empilha_Operador(Var op: PILHA_OPERADOR; operador: Char): Integer;
Begin
        If op.topo = N2 Then
                Empilha_Operador := PILHA_CHEIA
        Else
                Begin
                        op.topo := op.topo + 1;
                        op.operador[op.topo] := operador;
                        Empilha_Operador := SUCESSO;
                End;
End;

(* ---------------------------------------------- Codifica *)

Function Codifica(ch: Char; Var valor: Integer; Var op: Char): Integer;
Begin
        Codifica := 4;
        If (ch >= '0') And (ch <= '9') Then
                Begin
                        Codifica := 1;
                        valor := Ord(ch) - 48;
                End;
        If ch IN ['+','-','*','/'] Then
                Begin
                        Codifica := 2;
                        op := ch;
                End;
        if ch = ')' Then
                Codifica := 3;
End;

(* ---------------------------------------------- Desempilha_Valor *)

Function Desempilha_Valor(Var v: PILHA_VALOR; Var valor: Integer): Integer;
Begin
```

```
	If v.topo = 0 Then
		Desempilha_Valor := PILHA_VAZIA
	Else
		Begin
			valor := v.valor[v.topo];
			v.topo := v.topo - 1;
			Desempilha_Valor := SUCESSO;
		End;
End;

(* ---------------------------------------------- Desempilha_Operador *)

Function Desempilha_Operador(Var op: PILHA_OPERADOR; Var operador: Char):
Integer;
Begin
	If op.topo = 0 Then
		Desempilha_Operador := PILHA_VAZIA
	Else
		Begin
			operador := op.operador[op.topo];
			op.topo := op.topo - 1;
			Desempilha_Operador := SUCESSO;
		End;
End;

(* ------------------------------------------- Calcula *)

Function Calcula(v1,v2: Integer; op: Char): Integer;
Begin
	Case op
	Of
		'+': Calcula := v1 + v2;
		'-': Calcula := v1 - v2;
		'*': Calcula := v1 * v2;
		'/': Calcula := v1 DIV v2;
	End;
End;

(* ------------------------------------------ Programa Principal *)

Var	v: PILHA_VALOR;
	op: PILHA_OPERADOR;
	s: String;
	n,i: Integer;
	tipo: Integer;
```

```
        valor: Integer;
        operador: Char;
        erro: Integer;
        v1,v2: Integer;
        resposta: Integer;
Begin
        ClrScr;
        Write('Expressão: ');
        ReadLn(s);
        If Testa_Expressao(s) Then
                Begin
                        Cria_Pilha_Valor(v);
                        Cria_Pilha_Operador(op);
                        n := Length(s);
                        For i := 1 to n Do
                                Begin
                                        tipo := Codifica(s[i],valor,operador);
                                        Case tipo
                                        Of
                                        1: erro := Empilha_Valor(v,valor);
                                        2: erro := Empilha_Operador(op,operador);
                                        3: Begin
                                                erro := Desempilha_Valor(v,v2);
                                                erro := Desempilha_Valor(v,v1);
                                                erro := Desempilha_Operador(op, operador);
                                                resposta := Calcula(v1,v2,operador);
                                                erro := Empilha_Valor(v,resposta);
                                           End;
                                        End;
                                End;
                        erro := Desempilha_Valor(v,resposta);
                        WriteLn('Resposta: ',resposta);
                End
        Else
                WriteLn('ERRO FATAL: Expressão Inválida');
End.
```

Deque (“**D**ouble-**E**nded **QUE**ue”)

É uma fila de duas extremidades. As inserções, consultas e retiradas são permitidas nas duas extremidades.

Deque de Entrada Restrita:

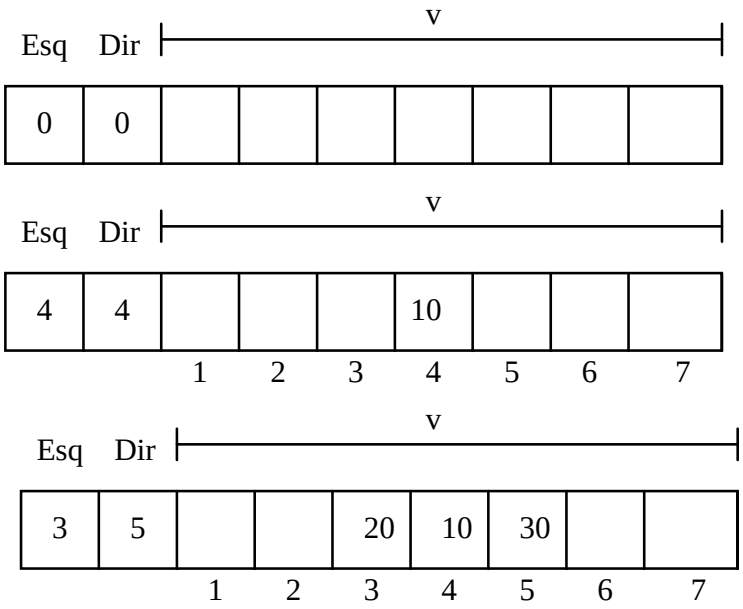
A inserção só pode ser efetuada ou no início ou no final da lista.

Deque de Saída Restrita:

A retirada só pode ser efetuada ou no início ou no final da lista.

Condição Inicial: Esquerda (Esq) e Direita (Dir) no meio do vetor

Esq = (m DIV 2 + 1
Dir = (m DIV 2) + 1



Deque Vazio:

Esq = 0
Dir = 0

Deque Cheio:

Esq = 1
Dir = m

Cálculo do Número de Elementos do Deque:

```
                número_de_elementos = Dir - Esq + 1;

Program Fila_Deque;

Uses Crt;

Const m = 9;
      SUCESSO = 1;
      DEQUE_ESQUERDO_CHEIO = 2;
      DEQUE_DIREITO_CHEIO = 3;
      DEQUE_VAZIO = 4;

Type DEQUE = Record
                        esq,dir: Integer;
                        v: Array[1..m] Of Integer;
                End;

Var     d: DEQUE;
      valor: Integer;
      ch,op: Char;
      erro: Integer;

(* ------------------------------------ Cria_Deque *)

Procedure Cria_Deque(Var d: DEQUE);
Begin
      d.esq := 0;
      d.dir := 0;
End;

(* ------------------------------------ Insere_Esquerda *)

Function Insere_Esquerda(Var d: DEQUE; valor: Integer): Integer;
Begin
      Insere_Esquerda := SUCESSO;
      If d.esq = 1 Then
            Insere_Esquerda := DEQUE_ESQUERDO_CHEIO
      Else
            Begin
                  If d.esq = 0 Then
                  Begin
                        d.esq := (m DIV 2) + 1;
                        d.dir := d.esq;
                  End
            Else
```

```
                d.esq := d.esq - 1;
            d.v[d.esq] := valor;
        End;
End;

(* ------------------------------------ Insere_Direita *)

Function Insere_Direita(Var d: DEQUE; valor: Integer): Integer;
Begin
        Insere_Direita := SUCESSO;
        If d.dir = m Then
            Insere_Direita := DEQUE_DIREITO_CHEIO
        Else
            Begin
                If d.dir = 0 Then
                    Begin
                        d.dir := (m DIV 2) + 1;
                        d.esq := d.dir;
                    End
                Else
                    d.dir := d.dir + 1;
                d.v[d.dir] := valor;
            End;
End;

(* ------------------------------------ Retira_Esquerda *)

Function Retira_Esquerda(Var d: DEQUE; valor: Integer): Integer;
Begin
        If d.esq = 0 Then
            Retira_Esquerda := DEQUE_VAZIO
        Else
            Begin
                valor := d.v[d.esq];
                d.esq := d.esq + 1;
                If d.esq > d.dir Then
                    Cria_Deque(d);
                Retira_Esquerda := SUCESSO;
            End;
End;

(* ------------------------------------ Retira_Direita *)

Function Retira_Direita(Var d: DEQUE; valor: Integer): Integer;
Begin
```

```
	If d.dir = 0 Then
		Retira_Direita := DEQUE_VAZIO
	Else
		Begin
			valor := d.v[d.dir];
			d.dir := d.dir - 1;
			If d.dir < d.esq Then
				Cria_Deque(d);
			Retira_Direita := SUCESSO;
		End;
End;

(* ------------------------------------ Consulta_Esquerda *)

Function Consulta_Esquerda(d: DEQUE; Var valor: Integer): Integer;
Begin
	If d.esq = 0 Then
		Consulta_Esquerda := DEQUE_VAZIO
	Else
		Begin
			valor := d.v[d.esq];
			Consulta_Esquerda := SUCESSO;
		End;
End;

(* ------------------------------------ Consulta_Direita *)

Function Consulta_Direita(d: DEQUE; Var valor: Integer): Integer;
Begin
	If d.dir = 0 Then
		Consulta_Direita := DEQUE_VAZIO
	Else
		Begin
			valor := d.v[d.dir];
			Consulta_Direita := SUCESSO;
		End;
End;

(* ------------------------------------ Lista_Deque *)

Procedure Lista_Deque(d: DEQUE);
Var i: Integer;
Begin
	If d.esq <> 0 Then
		Begin
```

```
                    Write('DEQUE: ');
                    For i := d.esq To d.dir Do
                        Write(d.v[i]:02,' ');
                End;
End;

(* ------------------------------------ Exibe_Erro *)

Procedure Exibe_Erro(erro: Integer);
Begin
    WriteLn;
    Case erro
    Of
      DEQUE_ESQUERDO_CHEIO: WriteLn('ERRO: Deque Cheio à Esquerda');
      DEQUE_DIREITO_CHEIO:  WriteLn('ERRO: Deque Cheio à Direita');
      DEQUE_VAZIO: WriteLn('ERRO: Deque Vazio');
    End;
End;

(* ------------------------------------ Programa Principal *)

Begin
    Cria_Deque(d);
    Repeat
        ClrScr;
        Write('[I]nsere, [R]etira, [C]onsulta ou [F]im: ');
        Repeat
            op := UpCase(ReadKey);
        Until op IN ['I','R','C','F'];
        If op IN ['I','R','C'] Then
            Begin
                WriteLn;
                Write('[E]squerda ou [D]ireita: ');
                Repeat
                    ch := UpCase(ReadKey);
                Until ch IN ['E','D'];
                WriteLn;
                Case op
                Of
                'I':  Begin
                            Write('Valor: ');
                            ReadLn(valor);
                            If valor <> 0 Then
                            Case ch
                            Of
```

```
                                        'E': erro := Insere_Esquerda(d,valor);
                                        'D': erro := Insere_Direita(d,valor);
                                        End;
                                End;
                        'R': Begin
                                        Case ch
                                        Of
                                        'E': erro := Retira_Esquerda(d,valor);
                                        'D': erro := Retira_Direita(d,valor);
                                        End;
                                        If erro = SUCESSO Then
                                                WriteLn('Valor Retirado: ',valor);
                                End;
                        'C': Begin
                                        Case ch
                                        Of
                                        'E': erro := Consulta_Esquerda(d,valor);
                                        'D': erro := Consulta_Direita(d,valor);
                                        End;
                                        If erro = SUCESSO Then
                                                WriteLn('Valor Consultado: ',valor);
                                End;
                        End;
                        If erro <> SUCESSO Then
                                Exibe_Erro(erro)
                        Else
                                Lista_Deque(d);
                        ReadLn;
                End;
        Until op = 'F';
End.
```

Pesquisa de Dados

Uma operação complexa e trabalhosa é a consulta em tabelas. Normalmente uma aplicação envolve grande quantidade de dados que são armazenadas em ***Tabelas***.

As tabelas são compostas de registros (normalmente possui uma chave), e os registros de campos.

TABELA

Chave	Nome	Altura	Peso
1	Francisco	1.75	80.4
2	Renato	1.80	76.9
3	Paulo	1.67	87.5
4	Ricardo	1.86	57.8

REGISTRO

CAMPOS

Pesquisa Seqüencial

Método mais simples de pesquisa em tabela, consiste em uma varredura seqüencial, sendo que cada campo é comparado com o valor que está sendo procurado. Esta pesquisa termina quando for achado o valor desejado ou quando chegar o final da tabela.

Número_Médio_de_Comparações = (n + 1) / 2

n -> Número de Elementos

```
Program Pesquisa_Sequencial;

Type   K = 10;
       palavra = String[10];
       reg = Record
                          chave: Integer;
                          nome: palavra;
                          altura: Real;
                          peso: Real;
             End;
       tabela: Array [1..K] Of reg;
Var    t: tabela;
       i,j,n: Integer;
       valor: Integer;
       ch: Char;

(* ---------------------------------------------- pesquisa_sequencial *)
```

```
Function pesquisa_sequencial(valor,n: Integer): Integer;
Var i,j: Integer;
Begin
        j := 0;
        i := 1;
        Repeat
                If t[i].chave = valor Then
                        j := i;
                i := i + 1;
        Until  (j <> 0) Or (i >= n);
        pesquisa_sequencial := j;
End;

(*---------------------------------------------- PROGRAMA PRINCIPAL *)

Begin
        i := 0;
        Repeat
                i := i + 1;
                Write('Chave: ');
                ReadLn(t[i].chave);
                Write('Nome: ');
                ReadLn(t[i].nome);
                Write('Peso: ');
                ReadLn(t[i].peso);
                Write('Altura: ');
                ReadLn(t[i].altura);
                Write('Continua [S/N] ?');
                Repeat
                        ch := ReadKey;
                Until ch IN ['S','N','s','n'];
        Until ch IN ['N','n'];
        Repeat
                Write('Valor: ');
                ReadLn(valor);
                j := pesquisa_sequencial(valor,i);
                Write('Chave: ',j);
                Write('Continua [S/N] ?');
                Repeat
                        ch := ReadKey;
                Until ch IN ['S','N','s','n'];
        Until ch IN [‘N','n'];
End.
```

Observação: A ***Pesquisa Sequencial*** apresenta desempenho melhor se a tabela estiver ordenada pela chave de acesso:

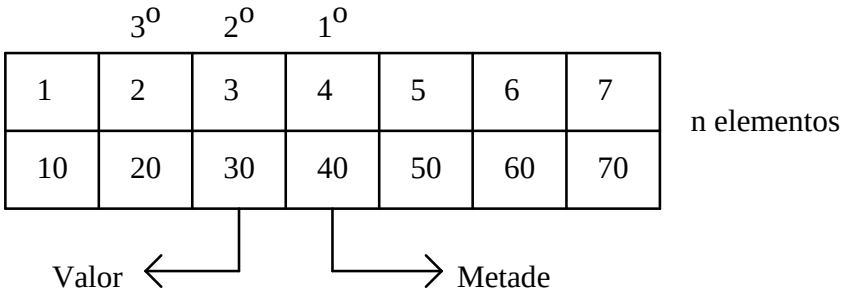
```
Program Pesquisa_Sequencial_com_Tabela_Ordenada;

Type    K = 10;
        palavra = String[10];
        reg = Record
                        chave: Integer;
                        nome: palavra;
                        altura: Real;
                        peso: Real;
                End;
        tabela: Array [1..K] Of reg;

Var      t: tabela;
        i,j,n: Integer;
        valor: Integer;
        ch: Char;

(* ---------------------------------------------- troca_real *)

Procedure troca_real(a,b: Real);
Var temp: Real;
Begin
        temp := a;
        a := b;
        b := temp;
End;

(* ---------------------------------------------- troca_Integer *)

Procedure troca_Integer(a,b: Integer);
Var temp: Integer;
Begin
        temp := a;
        a := b;
        b := temp;
End;

(* ---------------------------------------------- troca_string *)

Procedure troca_string(a,b: palavra);
Var temp: palavra;
Begin
```

```
        temp := a;
        a := b;
        b := temp;
End;

(* ---------------------------------------------- ordena_tabela *)

Procedure ordena_tabela(n: Integer);
Var i,j: Integer;
Begin
        For i := 1 To n-1 Do
                For j := i+1 To n Do
                        If t[i].chave > t[j].chave Then
                                Begin
                                        troca_Integer(t[i].chave,t[j].chave);
                                        troca_string(t[i].nome,t[j].nome);
                                        troca_real(t[i].altura,t[j].altura);
                                        troca_real(t[i].peso,t[j].peso]);
                                End;
End;

(* ---------------------------------------------- pesquisa_ordenada *)

Function pesquisa_ordenada(valor,n: Integer): Integer;
Var i,j: Integer;
Begin
        j := 0;
        i := 1;
        Repeat
                If t[i].chave >= valor Then
                        If t[i].chave = valor Then
                                j := i;
                i := i + 1;
        Until (j <> 0) Or (i >= n);
        pesquisa_ordenada := j;
End;

(* ---------------------------------------------- PROGRAMA PRINCIPAL *)

Begin
        i := 0;
        Repeat
                i := i + 1;
                Write('Chave: ');
                ReadLn(t[i].chave);
```

```
                Write('Nome: ');
                ReadLn(t[i].nome);
                Write('Peso: ');
                ReadLn(t[i].peso);
                Write('Altura: ');
                ReadLn(t[i].altura);
                Write('Continua [S/N] ?');
                Repeat
                        ch := ReadKey;
                Until ch IN ['S','N','s','n'];
        Until ch IN ['N','n'];
        ordena_tabela(i);
        Repeat
                Write('Valor: ');
                ReadLn(valor);
                j := pesquisa_ordenada(valor,i);
                Write('Chave: ',j);
                ReadLn('Continua [S/N] ?');
                Repeat
                        ch := ReadKey;
                Until ch IN [‘S','N','s','n'];
        Until ch IN ['N','n'];
End.
```

Pesquisa Binária

Método de Pesquisa que só pode ser aplicada em tabelas ordenadas.



O método consiste na comparação do "valor" com a chave localizada na metade da tabela, pode ocorrer:

valor = chave.....chave localizada
valor < chave.....chave está na primeira metade (esquerda)
valor > chave.....chave está na segunda metade (direita)

metade = (n DIV 2) + 1

A cada comparação, a área de pesquisa é reduzida a metade do número de elementos.

O número máximo de comparações será:

$nc = \log_2 n + 1$

```
Program Pesquisa_Binária;

Const k = 10;

Type tabela = Array[1..K] Of Integer;

Var      t: tabela;
         i,n: Integer;
         ch: CARACTER;
         valor: Integer;

(*---------------------------------------------- troca *)

Procedure troca (a,b: Integer);
Var t: Integer;
Begin
         t := a;
         a := b;
         b := t;
End;

(* ---------------------------------------------- ordena *)

Procedure ordena(n: Integer);
Var i,j: Integer;
Begin
         For i := 1 To n-1 Do
                  For j := i+1 To n Do
                           If t[i] > t[j] Then
                                    troca(t[i],t[j]);
End;

(* ---------------------------------------------- pesquisa *)
```

```
Function pesquisa(n,valor: Integer): Integer;
Var     i,j,índice: Integer;
        inic,fim,metade: Integer;
Begin
        inic := 1;
        fim := n;
        metade := (n DIV 2) + 1;
        indice := 0;
        Repeat
                If valor = t[metade] Then
                        índice := metade
                Else
                        If valor < t[metade] Then
                                Begin
                                        fim := metade - 1;
                                        metade := ((fim + inic) DIV 2);
                                End
                        Else
                                Begin
                                        inic := metade + 1;
                                        metade := ((fim + inic) DIV 2);
                                End;
        Until (índice <> 0) Or (inic = fim);
        pesquisa := índice;
End;

(* ---------------------------------------------- PROGRAMA PRINCIPAL *)

Begin
        n := 0;
        Repeat
                n := n + 1;
                Write('Número: ');
                ReadLn(t[n]);
                Write('Continua [S/N]? ');
                Repeat
                        ch := ReadKey;
                Until (ch = 'S') Or (ch = 'N');
        until (ch = 'N') Or (n >= k);
        ordena(n);
        Write('| ');
        For i := 1 To n Do
                Write(t[i],' | ');
        Write('Valor: ');
        ReadLn(valor);
```

```
        i := pesquisa(n,valor);
        If i = 0 Then
                WriteLn('ERRO: Valor não está na tabela')
        Else
                WriteLn('Indice: ',i);
End.
```

Cálculo de Endereço (Hashing)

Além de um método de pesquisa, este método é também um método de organização física de tabelas (***Classificação***). Onde cada dado de entrada é armazenado em um endereço previamente calculado (através de uma função). O processo de busca é igual ao processo de entrada, ou seja, eficiente.

Um dos problemas é definir bem o tipo de função a ser usada, pois normalmente as funções geram endereços repetidos. A eficiência deste método depende do tipo de função.

Numa tabela com "n" elementos com valores na faixa de [0..MAX] pode ser usada a seguinte a função:

endereço = (entrada MOD n) + 1

Exemplo:

n = 53
entrada: [0..1000]

Entrada	383	487	235	527	510	320	203	108	563	500	646	103
endereço	12	10	23	50	33	2	44	2	33	23	10	50

Nota-se que no cálculo dos endereços houve repetições, tais como: 2, 33 e 10, por causa disto, é necessário verificar se o endereço está ocupado ou não. Finalmente os endereços calculados são:

Entrada	383	487	235	527	510	320	203	108	563	500	646	103
endereço	12	10	23	50	33	2	44	3	34	24	11	51

No exemplo abaixo, a tabela terá um campo lógico da seguinte maneira:

situação = VERDADE ocupado
situação = FALSO desocupado

```
Program Hashing;

Const k = 53;

Type    Tab = Record
                    situação: Boolean;
                    valor: Integer;
              End;
        tabela = Array [1..k] Of Tab;

Var      t: tabela;
        n: Integer;
        entrada: Integer;
        endereço: Integer;
        ch: Char;

(* ---------------------------------------------- Inicializa_Tabela *)

Procedure Inicializa_Tabela;
Var i: Integer;
Begin
        For i := 0 To k Do
              Begin
                    t[i].situação := FALSE;
                    t[i].valor := 0;
              End;
End;

(* ---------------------------------------------- Insere_Tabela *)

Procedure Insere_Tabela(entrada: Integer);
Var endereco: Integer;
Begin
        endereco := (entrada MOD k) + 1;
        While t[end].situação Do
              endereco := (end MOD k) + 1;
        t[endereco].valor := entrada;
        t[endereco].situação := TRUE;
End;
```

```
(* ---------------------------------------------- Hashing *)

Function Hashing(entrada: Integer): Integer;
Var endereco,i: Integer;
Begin
        endereco := (entrada MOD k) + 1;
        While (t[end].valor <> entrada) And (end <> k) Do
                endereco := (end MOD k) + 1;
        If endereco <> K Then
                Hashing := endereco
        Else
                Hashing := 0;
End;

(* ---------------------------------------------- PROGRAMA PRINCIPAL *)

Begin
        Inicializa_Tabela;
        n := 0;
        Repeat
                n := n + 1;
                Write('Número: ');
                ReadLn(entrada);
                Insere_Tabela(entrada);
                Write('Continua [S/N]? ');
                Repeat
                        ch:= ReadKey;
                Until (ch = 'S') Or (ch = 'N');
        Until (n >= k) Or (ch = 'N');
        Repeat
                Write('Valor a ser CONSULTADO: ');
                ReadLn(entrada);
                endereço := Hashing(entrada);
                If endereço = 0 Then
                        WriteLn('ERRO: Valor Inválido')
                Else
                        WriteLn('Endereço: ',endereço);
                Write('Continua [S/N]? ');
                Repeat
                        ch := ReadKey;
                Until (ch = 'S') Or (ch = 'N');
        Until  (ch = 'N');
End.
```

Classificação de Dados (Ordenação)

É o processo pelo qual é determinada a ordem em que devem ser apresentados os elementos de uma tabela de modo a obedecer a seqüência de um ou mais campos (chaves de classificação).

Classificação Interna.....Memória Principal
Classificação Externa.....Memória Secundária

Contigüidade Física

1	Carla
2	Beatriz
3	Débora
4	Ana

Tabela Original

1	Ana
2	Beatriz
3	Carla
4	Débora

Tabela Ordenada

Logo, os elementos são fisicamente reorganizados.

Exemplo:

```
Program Ordenação;

Const QUANT = 10;

Var     v: Array[1..QUANT] Of Integer;
        i,j,n: Integer;
        ch: Char;

Begin
        n := 0;
        Repeat
                n := n + 1;
                Write('Valor: ');
                ReadLn(v[n]);
                Write('Continua [S/N] ?');
                Repeat
                        ch := ReadKey;
                Until ch IN [ 'N','n','S','s'];
```

```
        Until (ch = 'N') Or (n >= QUANT);
        For i := 1 To n-1 Do
                For j := i+1 To n Do
                        If v[i] > v[j] Then
                                Begin
                                        temp := v[i];
                                        v[i] := v[j];
                                        v[j] := temp;
                                End;
        For i := 1 To n Do
                WriteLn(v[i]);
End.
```

Vetor Indireto de Ordenação (Tabela de Índices)

Os elementos não são reorganizados fisicamente, apenas é criado um outro vetor (***tabela de índices***) que controla a ordem do primeiro.

Exemplo:

n

1	Carla
2	Beatriz
3	Débora
4	Ana

Tabela Original

vio

1	4
2	2
3	1
4	3

Tabela de Índices

Primeira Solução:

```
Program Vetor_Indireto_Ordenação;

Const QUANT = 10;

Var     n: Array[1..QUANT] Of String[20];
        ni: Array[1..QUANT] Of Integer;
        i,j,k,l,t: Integer;
        ch: Char;
        troca,ok: Boolean;

(* ---------------------------------------------- verifica *)
```

```
Function VERIFICA(i,k: Integer): Boolean;
Var      j: Integer;
         ok: Boolean;
Begin
         ok := TRUE;
         For j := 1 To k Do
                  If ni[j] = i Then
                  ok := FALSE;
         VERIFICA := ok;
End;

(* ---------------------------------------------- PROGRAMA PRINCIPAL *)

Begin
         t := 0;
         Repeat
                  t := t + 1;
                  Write('Nome: ');
                  ReadLn(n[t]);
                  Write('Continua [S/N] ?');
                  Repeat
                           ch := UpCase(ReadKey);
                  Until (ch = 'S') Or (ch = 'N');
         Until (ch = 'N') Or (t > QUANT);
         k := 0;
         j := 1;
         Repeat
                  troca := TRUE;
                  For i := 1 To t Do
                           Begin
                                    ok := VERIFICA(i,k);
                                    If ok Then
                                             If n[i] > n[j] Then
                                                      Begin
                                                               j := i;
                                                               troca := FALSE;
                                                      End;
                           End;
                  If Not(troca) Then
                           Begin
                                    k := k + 1;
                                    ni[k] := j;
                                    j := 1;
                           End;
```

```
                If troca Then
                        Begin
                                ok := VERIFICA(j,k);
                                If ok Then
                                Begin
                                        k := k + 1;
                                        ni[k] := j;
                                End;
                                If j < t Then
                                        j := j + 1
                                Else
                                        j := j - 1;
                        End;
        Until k = t;
        For i := 1 To t Do
                WriteLn(n[ni[i]]);
End.
```

Segunda Solução:

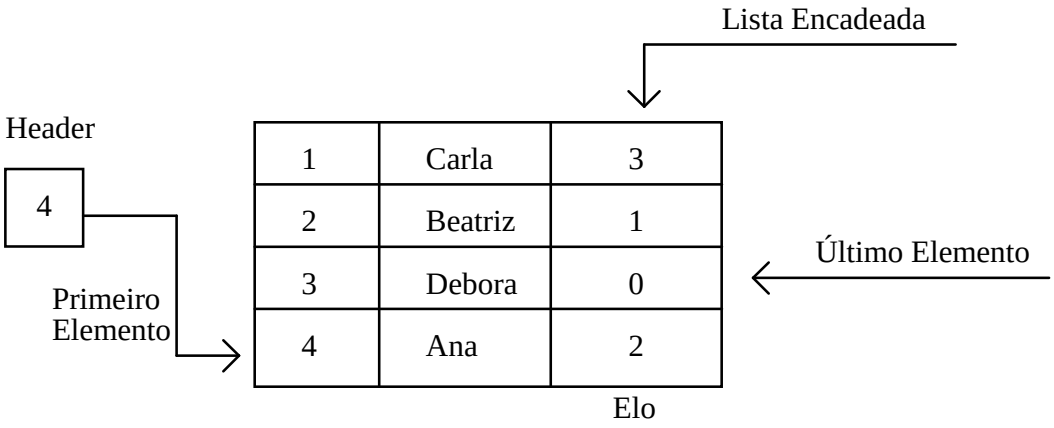
```
Program Vetor_Indireto_Ordenação;

Const QUANT = 10;

Var     v: Array [1..QUANT] Of String[30];
        vio: Array[1..QUANT] Of Integer;
        i,j,n: Integer;
        ch: Char;
        temp: Integer;
Begin
        n := 0;
        Repeat
                n := n + 1;
                Write('Valor: ');
                ReadLn(v[n]);
                Write('Continua [S/N] ?');
                Repeat
                        ch := UpCase(ReadKey);
                Until (ch = 'N') Or (ch = 'S');
        Until (ch = 'N') Or (n >= QUANT);
        For i := 1 To n Do
                vio[i] := i;
        For i := 1 To n-1 Do
                For j := i+1 To n Do
                        If v[vio[i]] > v[vio[j]] Then
```

```
                    Begin
                        temp := vio[i];
                        vio[i] := vio[j];
                        vio[j] := temp;
                    End;
    For i := 1 To n Do
        WriteLn(v[vio[i]]);
End.
```

Encadeamento

Os elementos pernamecem em seus lugares. É criado então uma lista encadeada ordenada. Esta lista possui um ***Header*** (Cabeça) o qual indica o primeiro elemento da lista.

Exemplo:




```
Program Encadeamento;

Const QUANT = 10;

Type    TABELA = Record
                        nome: String[20];
                        prox: Integer;
                 End;

Var     t,ta: Array[1..QUANT] Of TABELA;
        i,j,k,m,n: Integer;
        anterior: Integer;
        ch: Char;
        primeiro: Integer;
        sai: Boolean;

(* ---------------------------------------------- VERIFICA *)

Procedure VERIFICA (Var j: Integer);
Var     i: Integer;
        sai: Boolean;
Begin
        i := 1;
        Repeat
                sai := FALSE;
                If t[i].nome = ta[j].nome Then
```

```
                        Begin
                                j := i;
                                sai := TRUE;
                        End;
                i := i + 1;
        Until sai;
End;

(* ---------------------------------------------- COPIA *)

Procedure COPIA (Var m: Integer);
Begin
        m := 0;
        For i := 1 To n Do
                If t[i].prox = -1 Then
                        Begin
                                m := m + 1;
                                ta[m].nome := t[i].nome;
                                ta[m].prox := -1;
                        End;
End;

(* ---------------------------------------------- PROGRAMA PRINCIPAL *)

Begin
        n := 0;
        Repeat
                n := n + 1;
                Write('Nome: ');
                ReadLn(t[n].nome);
                Write('Continua [S/N]? ');
                Repeat
                        ch := UpCase(ReadKey);
                Until (ch = 'S') Or (ch = 'N');
        Until (ch = 'N') Or (n >= QUANT);
        For i := 1 To n Do
                t[i].prox := -1;
        primeiro := 1;
        For i := 2 To n Do
                If t[i].nome < t[primeiro].nome Then
                        primeiro := i;
        [primeiro].prox := 0;
        anterior := primeiro;
        Repeat
                COPIA(m);
```

```
        If m <> 0 Then
            Begin
                If m > 1 Then
                    Begin
                        i := 2;
                        j := 1;
                        Repeat
                            If ta[i].nome<ta[j].nome Then
                                j := i;
                            i := i + 1;
                        Until i > m;
                    End
                Else
                    j := 1;
                VERIFICA(j);
                t[anterior].prox := j;
                t[j].prox := 0;
                anterior := j;
                COPIA(m);
            End;
    Until m = 0;
    j := primeiro;
    For i := 1 To n Do
        Begin
            WriteLn(t[j].nome);
            j := t[j].prox;
        End;
End.
```

Métodos de Classificação Interna

Os métodos de Classificação Interna podem ser:

- Por ***Inserção***
- Por ***Troca***
- Por ***Seleção***

Observação: Para os seguintes métodos considere que as entradas são feitas no vetor "v", logo após é criado o vetor "c" (chaves) e o vetor "e" (endereços). A ordenação é feita no vetor "c" e o vetor "e" é o vetor indireto de ordenação, ou seja, será mantido o vetor de entrada intacto.

v

1	50
2	30
3	20
4	10
5	40

Vetor Original

c

1	50
2	30
3	20
4	10
5	40

Chaves

e

1	1
2	2
3	3
4	4
5	5

Vetor Indireto de Ordenação

Método por Inserção

Neste método ocorre a inserção de cada elemento em outro vetor ordenado.

v

1	50
2	30
3	20
4	10
5	40

Vetor Original

c

1	10
2	20
3	30
4	40
5	50

Chaves

e

1	4
2	3
3	2
4	5
5	1

Vetor Indireto de Ordenação

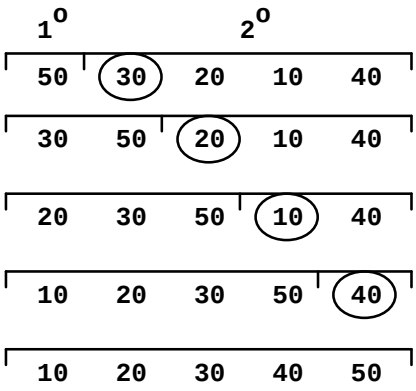
Método da Inserção Direta:

***Utilização*:** Pequena quantidade de dados, pois é pouco eficiente.

O vetor é dividido em dois segmentos. Inicialmente:

c[1] e c[2], c[3], ... c[n]

A classificação acontece por meio de interações, cada elemento do segundo segmento é inserido ordenadamente no primeiro até que o segundo segmento acabe. Por exemplo:




```
Program Inserção_Direta;

Const QUANT = 10;

Var     v,c,e: Array[1..QUANT] Of Integer;
        i,j,k,n: Integer;
        chave,endereço: Integer;
        ch: Char;

Begin
        n := 0;
        Repeat
                n := n + 1;
                Write('Número: ');
                ReadLn(v[n]);
                Write('Continua [S/N] ? ');
                Repeat
                        ch := UpCase(ReadKey);
                Until (ch = 'S') Or (ch = 'N');
        Until ch = 'N';
        For i := 1 To n Do
                Begin
                        c[i] := v[i];
                        e[i] := i;
                End;
        For i := 2 To n Do
                Begin
                        k := 1;
                        j := i - 1;
                        chave := c[i];
                        enderego := e[i];
                        While (j >= 1) e (k = 1) Do
                                If chave < c[j] Then
                                        Begin
```

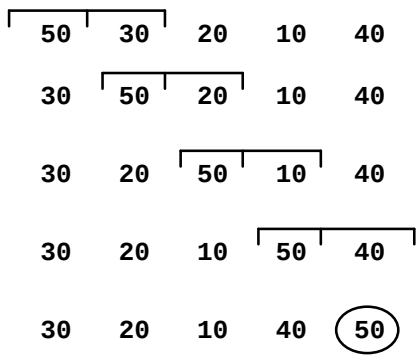
```
                                        c[j+1] := c[j];
                                        e[j+1] := e[j];
                                        j := j - 1;
                                End
                        Else
                                k := j + 1;
        c[k] := chave;
        e[k] := enderego;
End;
For i := 1 To n Do
        WriteLn(c[i]);
End.
```

Método por Troca

Neste método, compara-se pares de elementos, trocando-os de posição caso estejam desordenados.

Método da Bolha: (Bubble Sort)

Cada elemento do vetor é testado com o seguinte, se estiverem fora de ordem ocorre a troca, isto é repetido até não ocorrer mais trocas. Por exemplo:



Observação: Na primeira passagem completa o último elemento esta ordenado, logo na segunda passagem não é necessário ir até o fim.

```
Program Bolha;

Const QUANT = 10;

Var     v,c,e: Array[1..QUANT] Of Integer;
        i,j,n,m: Integer;
        chave,enderego: Integer;
        troca: Boolean;
```

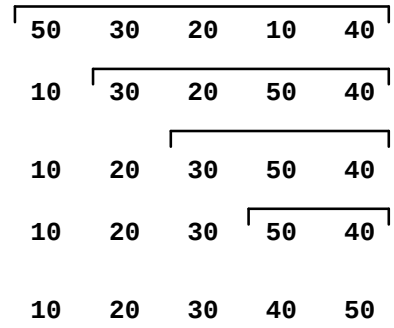
```
        ch: Char;

Begin
        n := 0;
        Repeat
                n := n + 1;
                Write('Número: ');
                ReadLn(v[n]);
                Write('Continua [S/N] ? ');
                Repeat
                        ch := UpCase(ReadKey);
                Until (ch = 'S') Or (ch = 'N');
        Until ch = 'N';
        For i := 1 To n Do
                Begin
                        c[i] := v[i];
                        e[i] := i;
                End;
        m := n - 1;
        Repeat
                troca := TRUE;
                For i := 1 To m Do
                        If c[i] > c[i+1] Then
                                Begin
                                        chave := c[i];
                                        c[i] := c[i+1];
                                        c[i+1] := chave;
                                        endereço := e[i];
                                        e[i] := e[i+1];
                                        e[i+1] := endereço;
                                        j := i;
                                        troca := FALSE;
                                End;
                m := j;
        Until troca;
        For i := 1 To n Do
                WriteLn(c[i]);
End.
```

Método de Seleção

Seleção sucessiva do menor valor da tabela. A cada passo o menor elemento é colocado em sua posição definitiva.

Método da Seleção Direta:

A cada passo é feita uma varredura do segmento que corresponde os elementos ainda não selecionados e determinado o menor elemento o qual é colocado na primeira posição do elemento por troca. Por exemplo:




```
Program Seleção_Direta;

Const QUANT = 10;

Var     v,c,e: Array[1..QUANT] Of Integer;
        i,j,n,min: Integer;
        chave,endereco: Integer;
        ch: Char;

Begin
        n := 0;
        Repeat
                n := n + 1;
                Write('Número: ');
                ReadLn(v[n]);
                Write('Continua [S/N] ? ');
                Repeat
                        ch := UpCase(ReadKey);
                Until (ch = 'S') ou (ch = 'N');
        Until ch = 'N';
        For i := 1 To n Do
                Begin
                        c[i] := v[i];
                        e[i] := i;
                End;
        For i :=1 To n-1 Do
                Begin
                        min := i;
                        For j :=i+1 To n Do
```

```
                    If c[j] < c[min] Then
                    min := j;
                chave := c[i];
                c[i] := c[min];
                c[min] := chave;
                endereco := e[i];
                e[i] := e[min];
                e[min] := endereco;
            End;
        For i := 1 To n Do
            WriteLn(c[i]);
End.
```

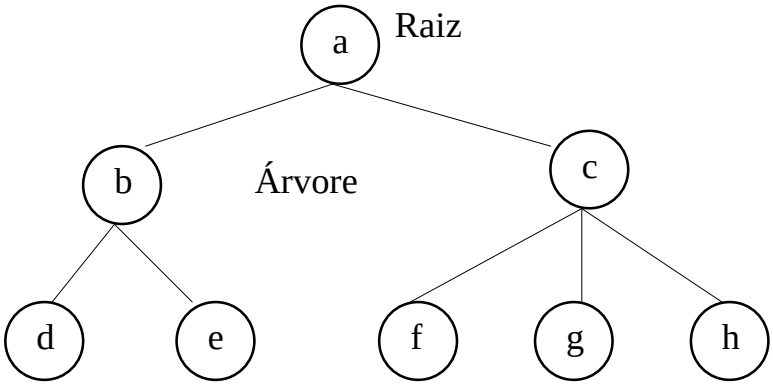
Árvores

São estruturas de dados que caracterizam uma relação entre os dados, a relação existente entre os dados é uma relação de hierarquia ou de composição (um conjunto é subordinado a outro).

Definição

É um conjunto finito T de um ou mais nós, tais que:

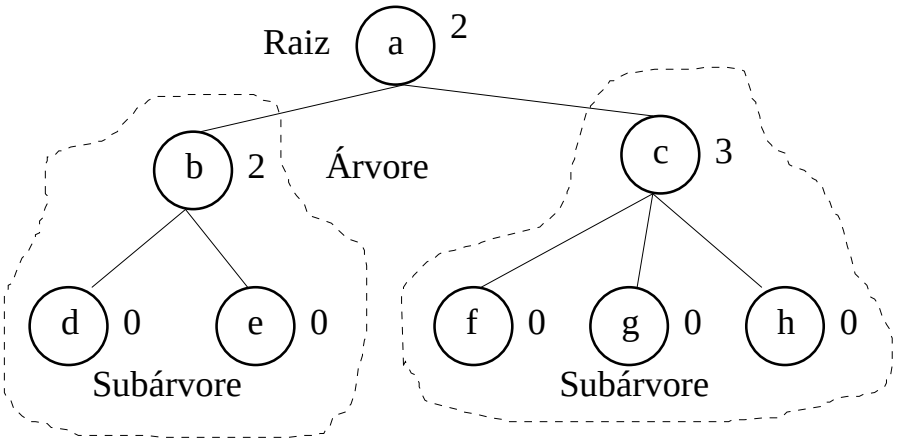
a) Existe um nó principal chamado **raiz** (root);
b) Os demais nós formam n >= 0 conjuntos disjuntos T1, T2, ... Tn, onde cada um destes subconjuntos é uma árvore. As árvores Ti (i >= 1 e i <= n) recebem a denominação de subárvores.



Terminologia

Grau

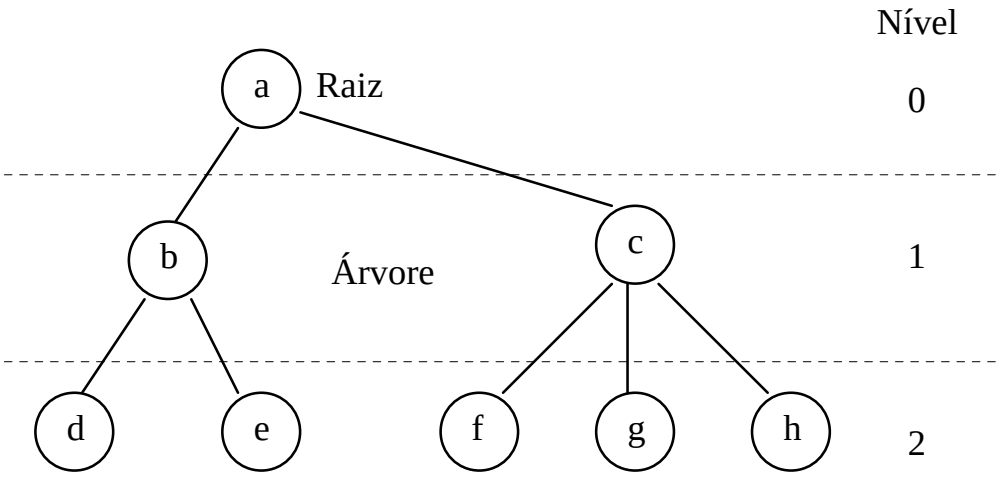
Indica o número de subárvores de um nó.



Observação: Se um nodo não possuir nenhuma subárvore é chamado de **Nó Terminal** ou **Folha**.

Nível

É o comprimento (número de linhas) do caminho (raiz até nó).



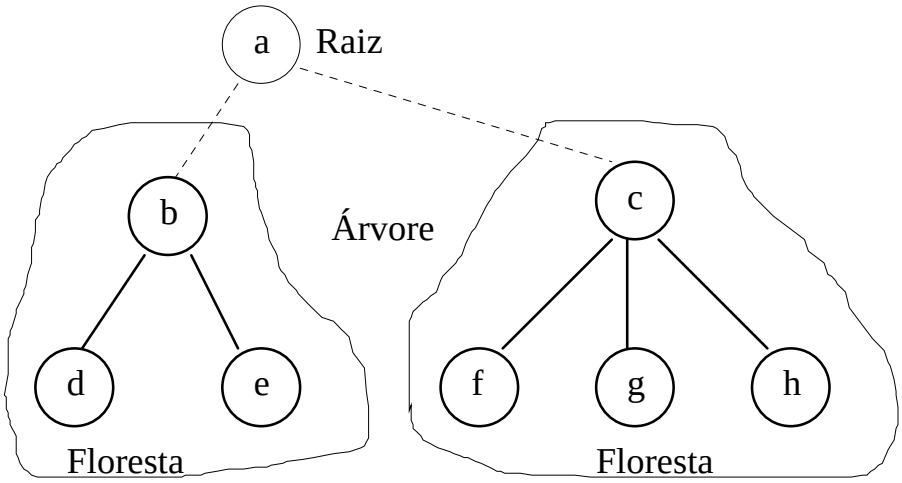
Observação: Raiz é nível zero (0)

Altura

É o nível mais alto da árvore. Na árvore acima, a altura é igual a 2.

Floresta

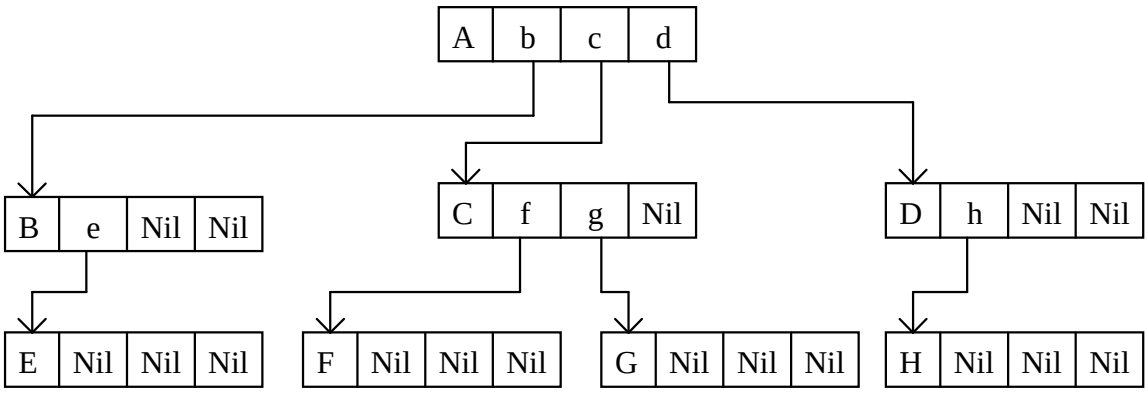
É um conjunto de zero ou mais árvores disjuntas, ou seja, se for eliminado o nó raiz da árvore, as subárvores que restarem chamam-se de florestas.



Formas de Representação de Árvores de Grau Arbitrário

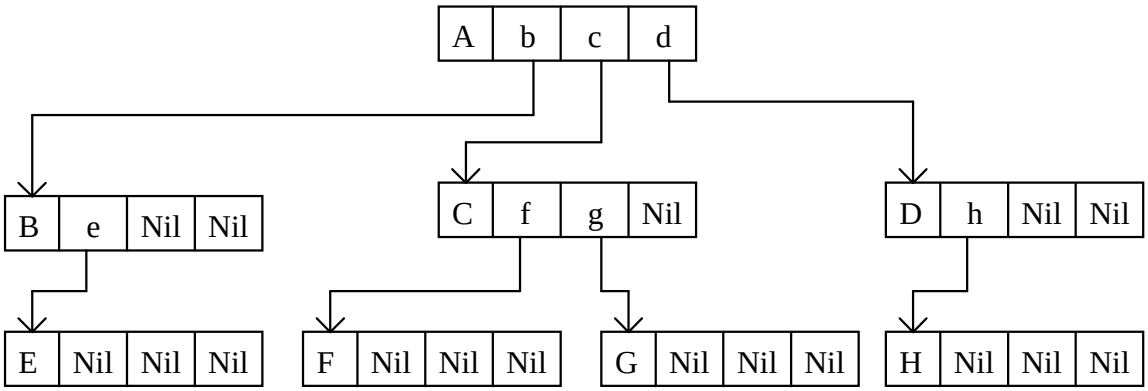
a) Registro Fixo

É usado se o grau da árvore for baixo e/ou a variabilidade do grau entre os nós for pouca.



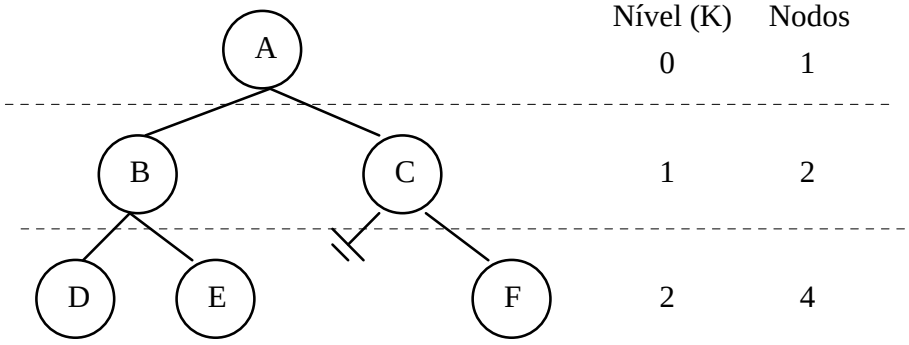
b) **Lista Ligada**

É usado quando o grau dos nós varia significativamente.



Árvores Binárias

São estruturas do tipo árvore onde o grau de cada nó é menor ou igual a dois.



Propriedades

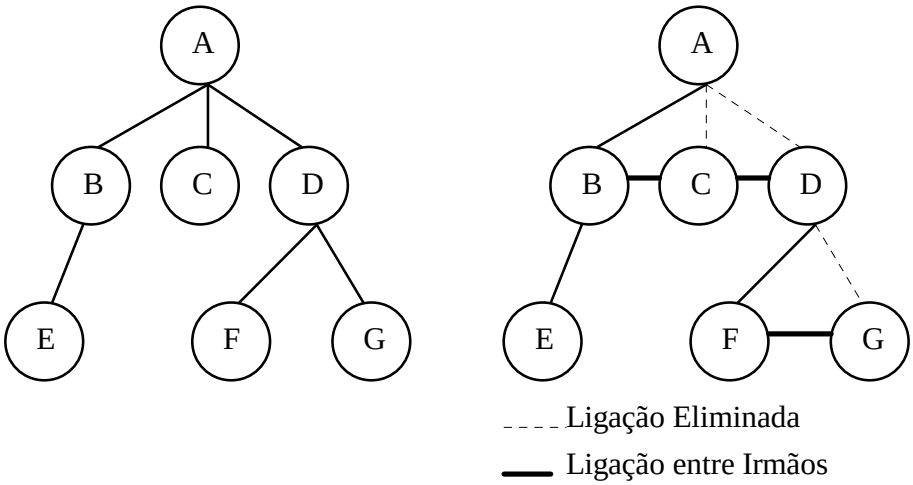
1) O número máximo de nodos no k-ésimo nível de uma árvore binária é 2^k;

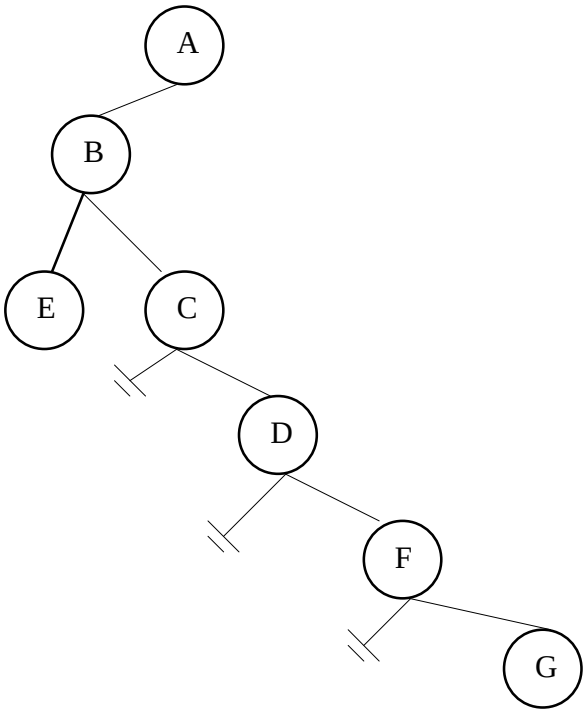
2) O número máximo de nodos em uma árvore binária com altura k é 2^{k+1}-1, para k >= 0;

3) Árvore completa: árvore de altura k com 2^{k+1}-1 nodos.

Conversão de Árvore Genérica em Árvore Binária

- Ligar os nós irmãos;
- Remover a ligação entre o nó pai e seus filhos, exceto as do primeiro filho;





Nota: O nó da subárvore à esquerda é ***filho*** e o nó da subárvore à direita é ***irmão***

Representação

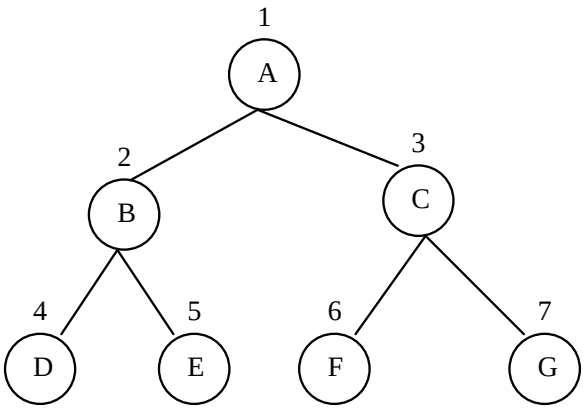
Por Contigüidade Física

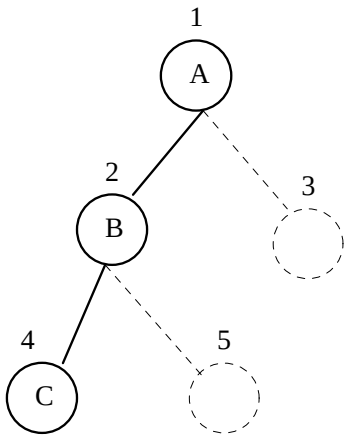
Os nodos são representados sequencialmente na memória. Devem ser tomados os seguintes cuidados:

- Ser alocado espaço suficiente para armazenar a estrutura completa;

- Os nodos devem ser armazenados em uma lista, onde cada nodo i da árvore ocupa o i-ésimo nodo da lista.

Exemplo:





1	2	3	4	5	6	7
A	B	C	D	E	F	G

1	2	3	4	5	6	7
A	B	-	C	-	-	-

Sendo "i" a posição de um nodo e "n" o número máximo de nodos da árvore.

Observações:

1) O pai de i está em i DIV 2 sendo i <= n. Se i = 1, i é a raiz e não possui pai.
2) O filho à esquerda de i está em 2i se 2i <= n. Se 2i > n, então tem filho à esquerda.
3) O filho à direita de i está em 2i+1. Se 2i+1 <= n. Se 2i+1 > n, então tem filho à direita.

Observação: Representação por Contigüidade Física não é um modo conveniente para representar árvores, na maioria dos casos.

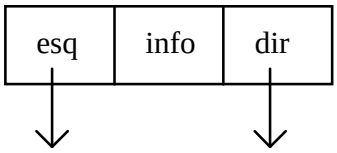
Vantagens

- Adequado para árvores binárias completas;
- Útil para o armazenamento em disco ou fita (seqüencial).

Desvantagem

- Dificuldade de manipulação da estrutura

Representação por Encadeamento



info: contém a informação do nodo
esq: endereço do nodo filho à esquerda
dir: endereço do nodo filho à direita

```
Type    PONTEIRO = ^NODO;
        NODO = Record
                                esq: PONTEIRO;
                                info: TIPO_INFO;
                                dir: PONTEIRO;
                    End;
        ÁRVORE = Record
                                raiz: PONTEIRO;
                        End;

Var     pt: PONTEIRO;
        a: ÁRVORE;

Const SUCESSO = 1;
        ÁRVORE_VAZIA = 2;
```

Caminhamento em Árvores

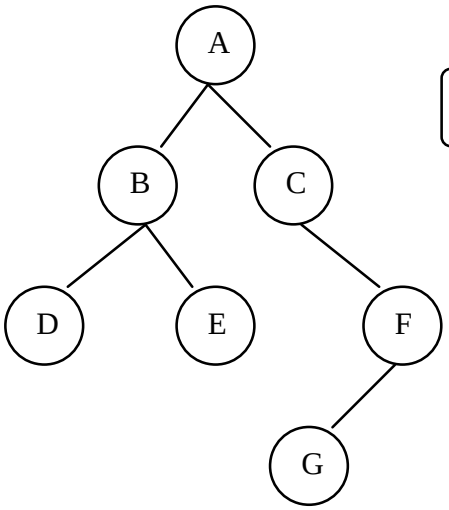
Consiste em processar de forma sistemática e ordenada cada nó da árvore apenas uma vez, obtendo assim uma seqüência linear de nós. Abaixo é mostrado os tipos de caminhamentos:

Caminhamento Pré-Fixado (Pré-Ordem)

1) Visitar a raiz
2) Caminhar na subárvore da esquerda
3) Caminhar na subárvore a direita

Observação: “Visitar” significa qualquer operação em relação a informação (info) do nodo.

Exemplo:



Caminhamento: ABDECFG

Caminhamento In-Fixado

1) Caminhar na subárvore da esquerda
2) Visitar a raiz
3) Caminhar na subárvore da direita

Exemplo: Conforme exemplo acima, o caminhamento In-Fixado é:

Caminhamento: DBEACGF

Caminhamento Pós-Fixado

1) Caminhar na subárvore da esquerda
2) Caminhar na subárvore da direita
3) Visitar a raiz

Exemplo: Conforme exemplo acima, o caminhamento Pós-Fixado é:

Caminhamento: DEBGFCA

Algorítmos para percorrer Árvores Binárias

```
Function Imprime_Árvore (a: ÁRVORE): Integer;
Begin
        pt := a.raiz;
```

```
        If pt = NIL
                Imprime_Árvore := ARVORE_VAZIA
        Else
                Imrprime_Árvore := Escreve_Fixado(pt);
End;

Function Escreve_Fixado (pt: PONTEIRO): Integer;
Begin
        WriteLn('Informação: ', pt^.info);
        Escreve_Fixado(pt^.esq);
        Escreve_Fixado(pt^.dir);
        Escreve_Fixado := SUCESSO;
End;
```

Observação: Algoritmo Recursivo

Caminhamento Pré-Fixado

```
Procedure Caminhamento_Pre_Fixado(a: ÁRVORE);
Var     paux: PONTEIRO;
        fim: Boolean;
Begin
        pt := a.raiz;
        fim := FALSE;
        Cria_Pilha(paux);
        While Not(fim) Do
                Begin
                        While pt <> NIL Do
                                Begin
                                        WriteLn(pt^.info);
                                        Empilha(paux,pt);
                                        pt := pt^.esq;
                                End;
                        If Desempilha(paux,pt) = PILHA_VAZIA Then
                                fim := TRUE
                        Else
                                pt := pt^.dir;
                End;
End;
```

Caminhamento In-Fixado

```
Procedure Caminhamento_In_Fixado(a: ÁRVORE);
Var     paux: PONTEIRO;
```

```
        fim: Boolean;
Begin
        pt := a.raiz;
        fim := FALSE;
        Cria_Pilha(paux);
        While Not(fim) Do
                Begin
                        While pt <> NIL Do
                                Begin
                                        Empilha(paux,pt);
                                        pt := pt^.esq;
                                End;
                        If Desempilha(paux,pt) = PILHA_VAZIA Then
                                fim := TRUE
                        Else
                                Begin
                                        WriteLn(pt^.info);
                                        pt := pt^.dir;
                                End;
                End;
End;
```

Caminhamento Pós-Fixado

```
Procedure Caminhamento_Pos_Fixado(a: ÁRVORE);
Var     paux: PONTEIRO;
        fim: Boolean;
        vez: Integer;
Begin
        pt := a.raiz;
        fim := FALSE;
        Cria_Pilha(paux);
        While Not(fim) Do
                Begin
                        While pt <> NIL Do
                                Begin
                                        Empilha(paux,pt,1);
                                        pt := pt^.esq;
                                End;
                        If Desempilha(paux,pt,vez) = PILHA_VAZIA Then
                                fim := TRUE
                        Else
                                If vez = 1 Then
                                        Begin
                                                Empilha(paux,pt,2);
```

```
                    pt := pt^.dir;
                End
            Else
                Begin
                    WriteLn(pt^.info);
                    pt := NIL;
                End;
        End;
End;
```