

SCC 122 – Estruturas de Dados

Lista Estática Encadeada

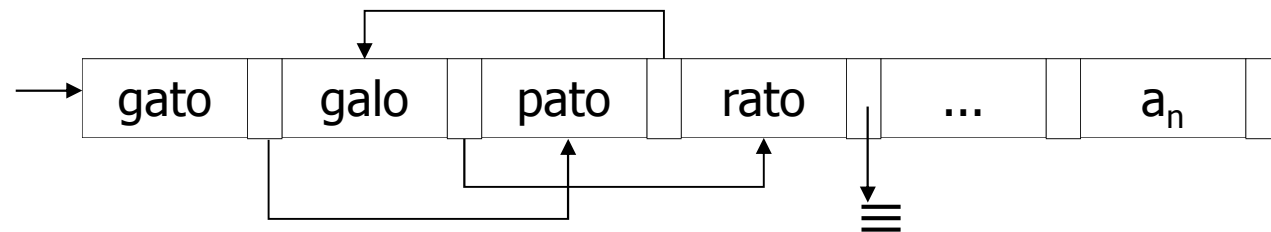
Lista Estática Encadeada

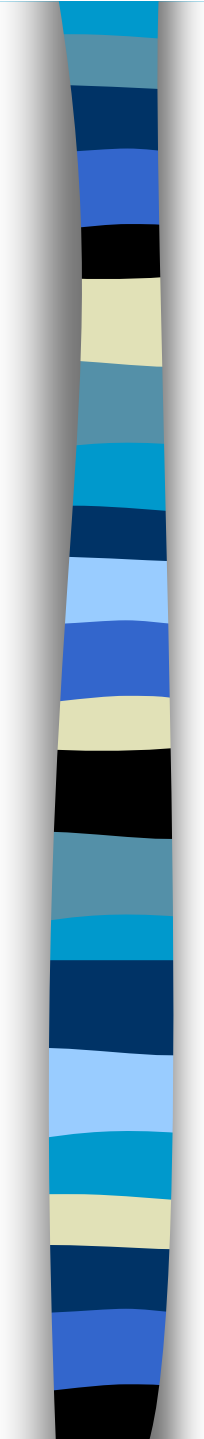
- Armazenada não seqüencialmente no array.

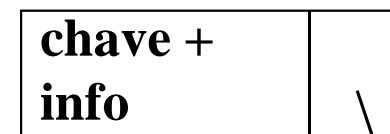
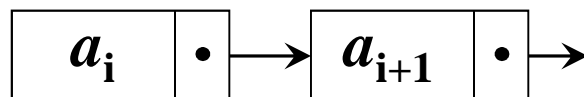
■ Seqüencial: A

a_1	a_2		a_n
1	2		n

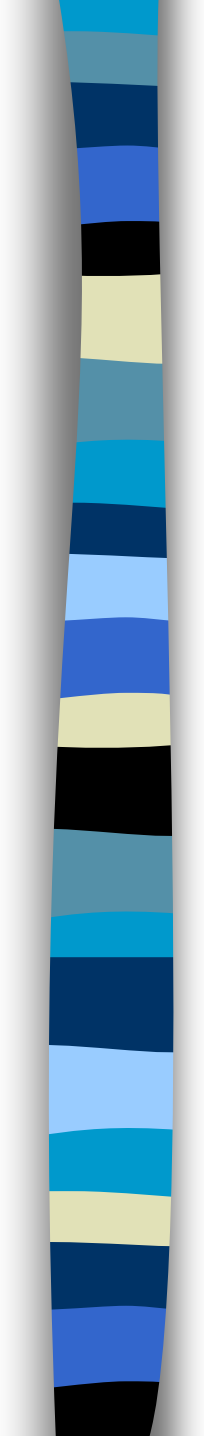
- Encadeada:



- 
- **Situação previa:** estimar tamanho máximo do array.
 - **Situação inicial:** todo o array está disponível para inserção de registro na lista.
 - **Requisito sobre os registros do array:** como seu endereço não indica sua ordem na lista, é preciso relacionar o elemento com seu sucessor na lista.



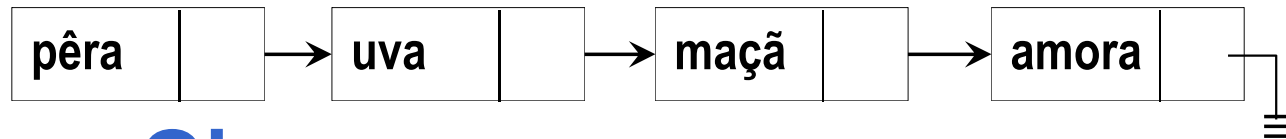
Campo de ligação com
o próximo elemento.

- 
- O conteúdo do campo **lig** é um índice do array, ou seja, o endereço do próximo elemento da lista (o índice da componente do vetor).
 - Deve haver um valor especial para o último.
 - Ex.: Num instante qualquer.
 - L = (pêra, uva, maçã, amora)

– Poderia estar armazenado como:

1	2	3	4	5	6					
amora	∅		maçã	1	pêra	6			Uva	3

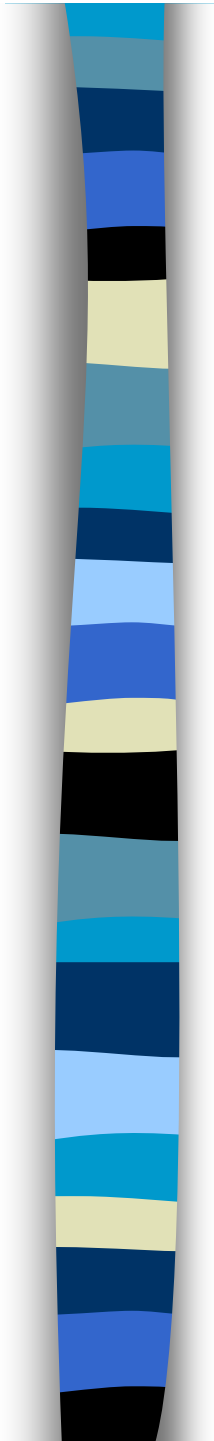
– Prim = 4



– Obs:

– valor especial no último elemento

– Variável especial indica o 1º Elemento.



- Já que teremos no mesmo **array** registros com elementos da lista e registros **vagos** disponíveis, então teremos de diferenciá-los a fim de usar os **disponíveis** para futuras **inserções**. Da mesma forma, nas **eliminações**, os válidos tornam-se disponíveis.
- A melhor maneira de fazer isso é juntar os registros disponíveis numa outra lista encadeada (pois estarão espalhados) no mesmo array!
- A variável DISPO indicará qual é a 1ª componente do vetor disponível.

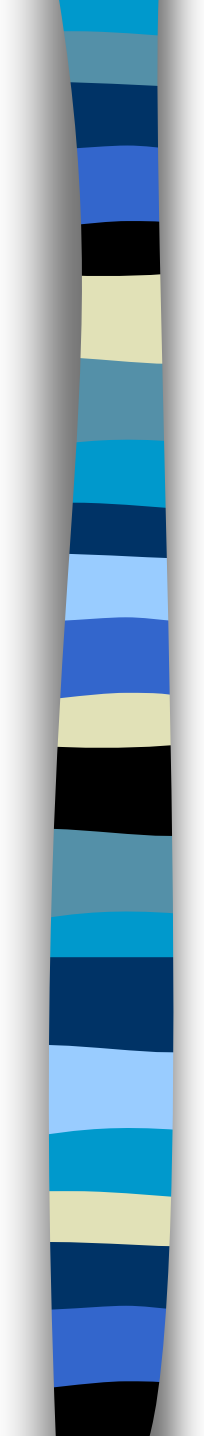


- **No exemplo anterior.**

- **PRIM = 4**
- **DISPO = 2**

1	2	3	4	5	6					
amora	∅		maçã	1	pêra	6			Uva	3

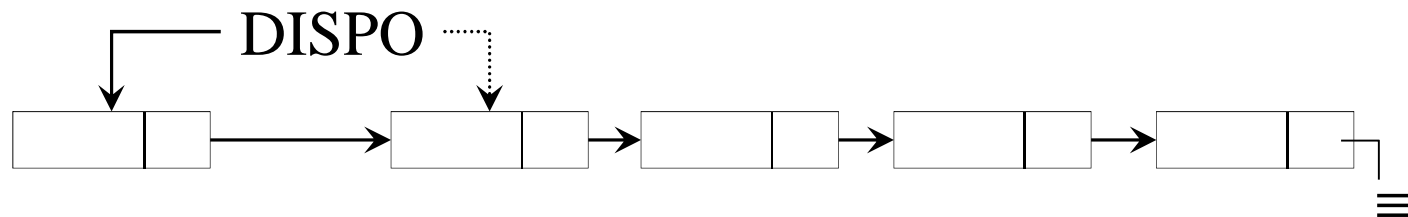
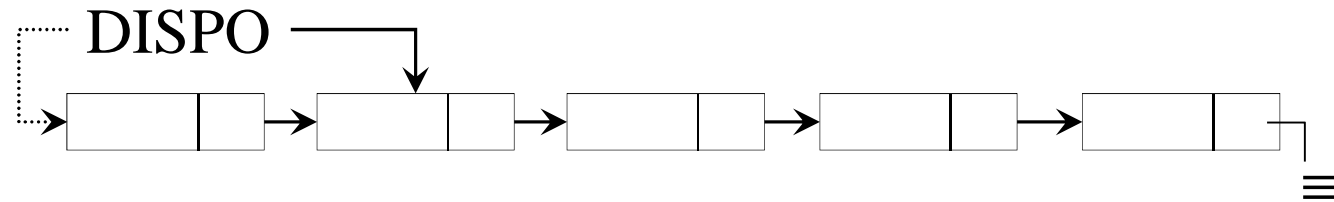
- **Lista: 4 → 6 → 3 → 1**
- **Lista dispo: 2 → 5**

- 
- Note que:
 - A inserção na Lista Principal requer a eliminação na lista DISPO.
 - A eliminação na Lista Principal resulta na inserção na lista DISPO.
 - Sempre que a DISPO está cheia a principal está VAZIA.
 - Uma característica importante na DISPO é que todos os seus registros são vazios.
 - Qualquer um pode ser eliminado da DISPO para ser inserido na principal.
 - Ao eliminar um novo registro “vazio” na DISPO, podemos fazê-lo em qualquer posição.
 - **Somos livres para escolher onde Inserir e de onde eliminar na DISPO.**
 - **De que forma seria mais eficiente/rápido?**



- Resposta:

- Eliminar o 1º registro da DISPO (apontado por DISPO)
- Inserir como 1º registro da DISPO.



- Com a inserção/eliminação ocorrerem numa única extremidade da lista DISPO tem comportamento de uma **PILHA**.

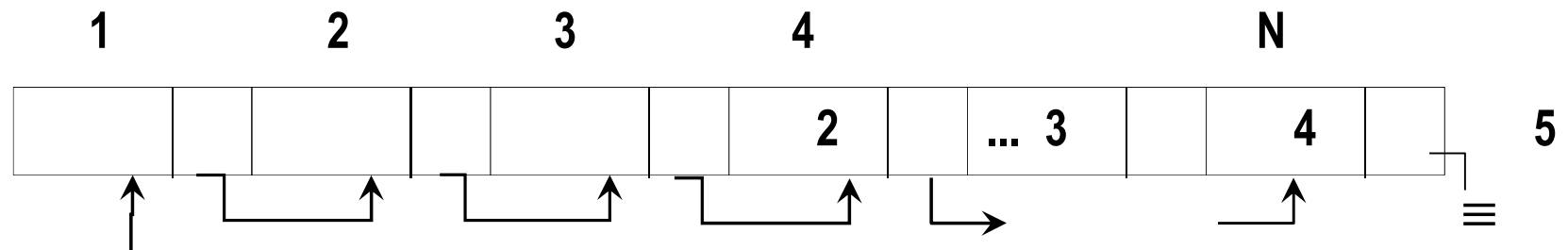


Lista Encadeada Estática

```
Const N=100;  
Type endereco=0..N;  
    registro= record  
        info: tipo_elem;  
        lig: endereco  
    end;  
Lista= record  
    A: array[1..N] of registro;  
    Prim, Dispo: endereco  
end;  
Var L:lista;
```

Inicialização da Lista Principal Vazia

- Todo o array pertence à DISPO



DISPO

```
For i := 1 to N-1 do  
    L.A[i].lig := i+1;  
L.A[N].lig := ∅;  
L.DISPO := 1;
```

- Lista Principal é vazia:

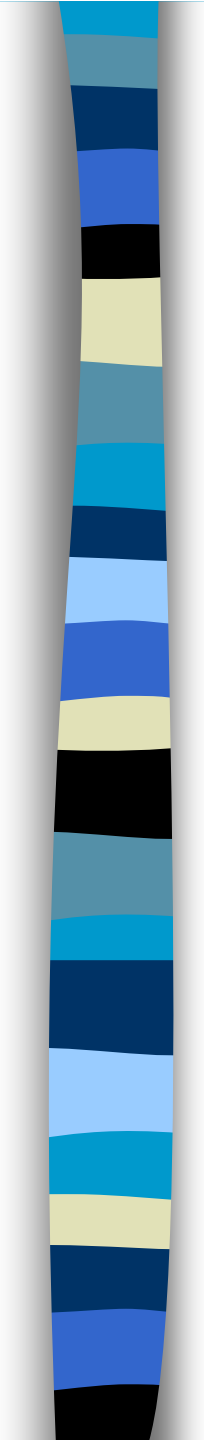
```
L.Prim := 0;
```

Operações do TAD Lista

```
function Lista_vazia(L : lista): boolean;  
{ Retorna true se lista vazia, false caso contrário}  
begin  
  Lista_vazia := (L.Prim = 0)  
end;
```

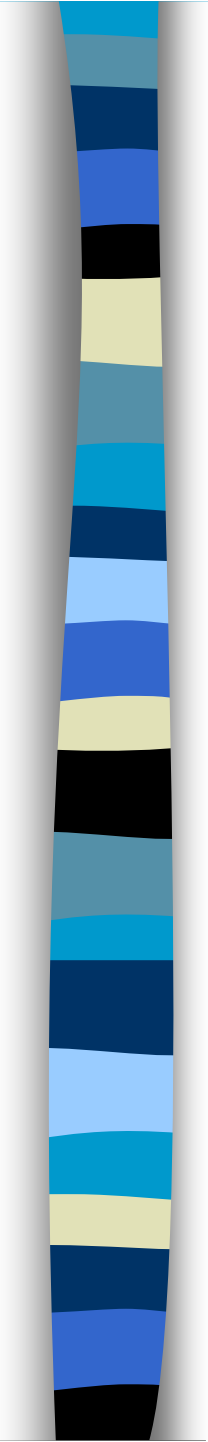
```
function Lista_cheia(L: lista): boolean;  
{ Retorna true se lista cheia, false caso contrário}  
begin  
  Lista_cheia := (L.Dispo = 0)  
end;
```

```
procedure Definir (var L: lista);  
{ Cria uma lista vazia. Chamado antes da execução de qualquer operação.}  
Begin  
  For i:= 1 to N-1 do  
    L.A[i].lig := i+1;  
  L.A[N].lig := Ø;  
  L.DISPO := 1;  
  L.Prim := 0  
end;
```



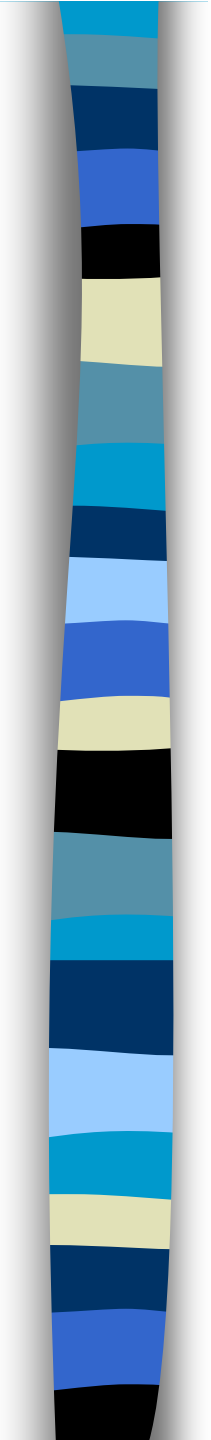
```
procedure Apagar (var L: lista);  
{ Apaga e reinicializa uma lista.}
```

```
var p: endereco;  
begin  
  if not Lista_vazia(L) then  
    begin  
      p := L.Prim;  
      while L.Prim <> 0 do  
        begin  
          L.Prim := L.A[L.Prim].lig;  
          devolver_no(L,p);  
          p := L.Prim  
        end;  
      end  
    end  
end;
```

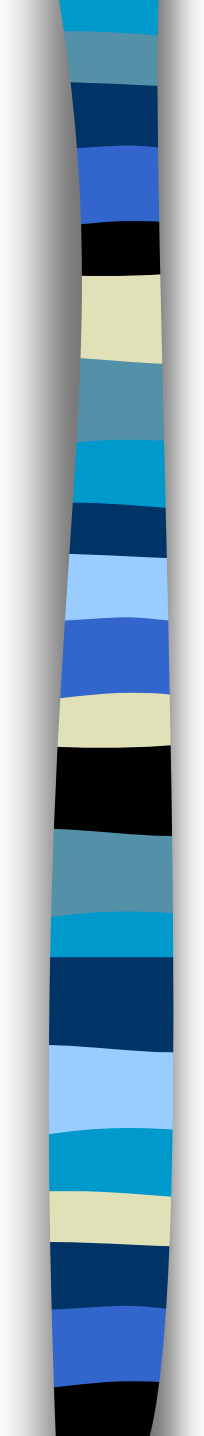


```
function Tamanho (L: lista): integer;  
{ Retorna o tamanho da Lista. Se L vazia retorna  
  0}
```

```
var p: endereco;  
    n : integer;  
begin  
  n := 0;  
  p := L.Prim;  
  while p <> 0 do  
    begin  
      n := n + 1;  
      p := L.A[p].lig  
    end;  
  Tamanho := n  
end;
```

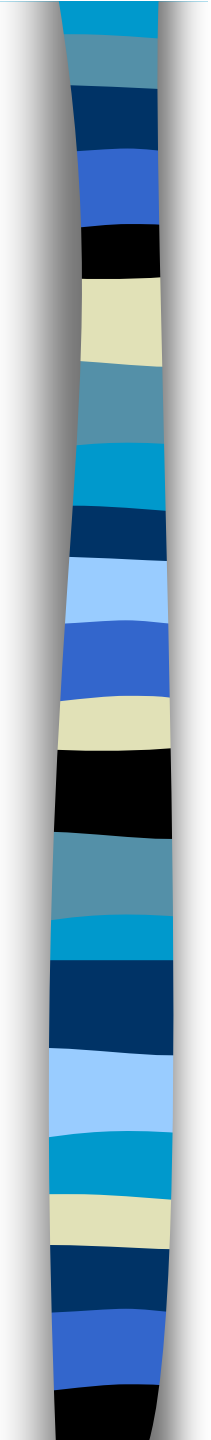


```
function Insere_apos(var L: lista; K: endereco;
    valor:tipo_elem): boolean;
{ Insere item após registro de endereço k. Retorna true
  se sucesso e false caso contrário (lista cheia)}
{K deve pertencer a PRINCIPAL e não a DISPO, sendo
  determinado previamente como no caso inserir_ord}
{K pode ser determinado por uma busca de conteúdo ou
  busca do k-esimo elemento da principal}
var j: endereco;
begin
  obter_no_da_dispo(L, j);
  if j <> 0 then {Existe espaço para inserir}
  begin
    L.A[j].info := valor;
    L.A[j].lig := L.A[k].lig;
    L.A[k].lig := j;
    Insere_apos := true
  end
  else Insere_apos := false
end;
```



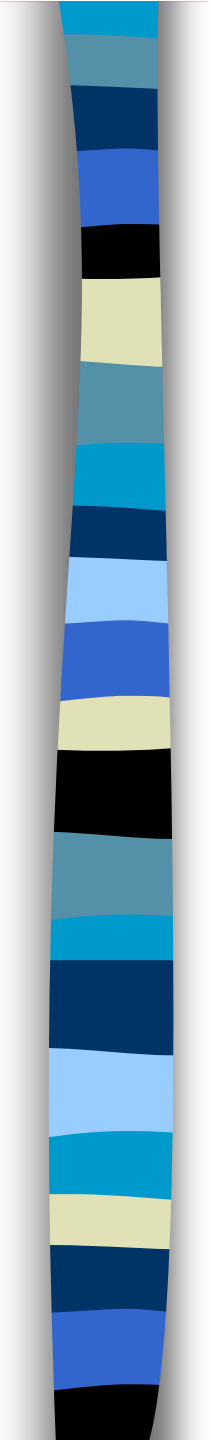
```
function Remove_apos(var L: lista; K: endereco):  
    boolean;  
{ Remove item após registro de endereço k.  
  Retorna true se sucesso e false caso contrário  
  (k é o último registro da lista)}
```

```
var j: endereco;  
begin  
  if L.A[k].lig <> 0 then {Existe registro para  
    remover}  
  begin  
    j := L.A[k].lig;  
    L.A[k].lig := L.A[j].lig  
    devolver_no_dispo(L, j);  
    Remove_apos := true  
  end  
  else Remove_apos := false  
end;
```

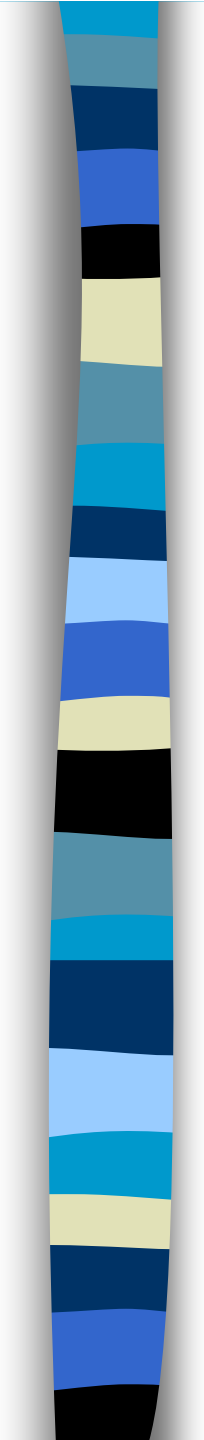
```
function Insere_frente(var L: lista; valor:
    tipo_elem): boolean;
{ Insere item na frente da lista. Retorna true se
    sucesso e false caso contrário (lista cheia)}
```

```
var j: indice;
begin
    obter_no_da_dispo(L, j);
    if j <> 0 then
        begin
            L.A[j].lig := L.Prim;
            L.A[j].info := valor;
            L.Prim := j;
            Insere_frente := true
        end
    else Insere_frente := false
    end;
end;
```



```
function Remove_frente(var L: lista; var
    valor:tipo_elem): boolean;
{ Remove o primeiro item da lista. Retorna true
  se sucesso e false caso contrário (lista
  vazia)}
```

```
var j: indice;
begin
  if L.Prim <> 0 then
  begin
    j := L.Prim;
    L.Prim := L.A[L.Prim].lig;
    devolver_no_dispo(L, j);
    Remove_frente := true
  end
  else Remove_frente := false
end;
```



```
function Inserir_ord(var L: lista; x:tipo_elem): boolean;
{ Insere item de forma a manter a Lista ordenada.
Devolve true se sucesso, false caso contrário (lista cheia).}
var achou : boolean;
    j, atual, ant: indice;
begin
    achou := false;
    atual := L.Prim;
    ant:= 0;
    while (atual <> 0) and not achou do
        if menor_que(x,L.A[atual].info) then achou := true
        else begin
            ant := atual;
            atual := L.A[atual].lig
        end;
    If ant = 0 then Inserir_ord:= Insere_frente(L,x)
    else Inserir_ord := Insere_apos(L, ant, x)
end;
```



```
function Localizar (L: lista; x:tipo_elem):  
    endereco;
```

```
{Retorna (busca) o endereço de x na Lista. Se x  
ocorre mais de uma vez, retorna o endereço da  
primeira ocorrência. Se x não aparece retorna  
0.}
```

```
var achou : boolean;  
    atual : indice;
```

```
begin
```

```
    achou := false;
```

```
    atual := L.Prim;
```

```
    while (atual <> 0) and not achou do
```

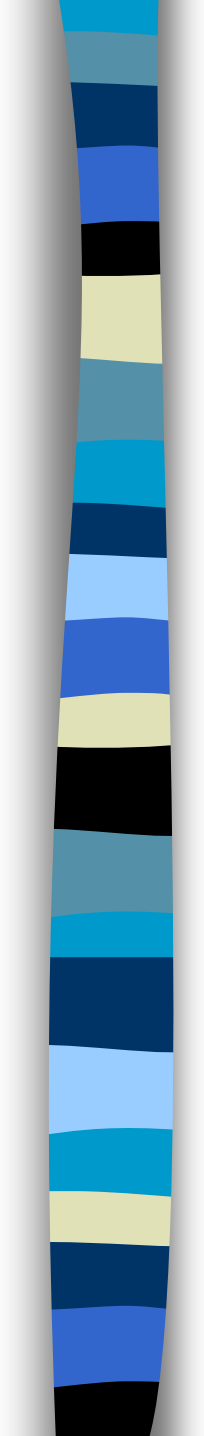
```
        if igual(x,L.A[atual].info) then achou := true
```

```
        else atual := L.A[atual].lig;
```

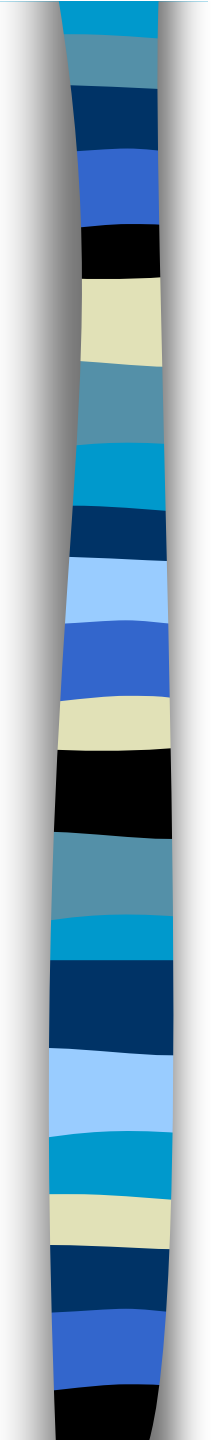
```
    If achou then Localizar := atual
```

```
    else Localizar := 0
```

```
end;
```

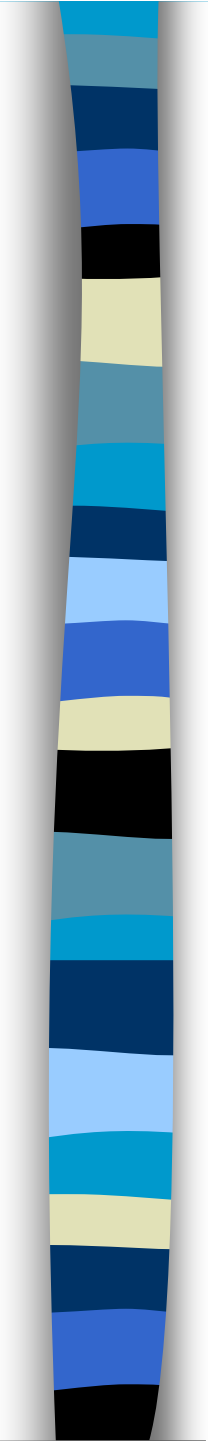


```
function Localizar_ord (L:lista; x:tipo_elem): endereco;
{Retorna (busca) o endereço de x numa Lista Ordenada. Se
  x não aparece retorna 0}
var achou : boolean;
    atual : indice;
begin
  If Lista_vazia(L) then Localizar_ord := 0
  else begin
    atual := L.Prim; acabou := false;
    while not acabou do
      if igual(x,L.A[atual]) then acabou := true
      else if menor_que(x,L.A[atual].info) then
        begin
          acabou := true; atual := 0
        end
      else begin
        atual := L.A[atual].lig; acabou :=
(atual = 0)
      end
    end;
    Localizar_ord := atual
  end;
end;
```



```
function Buscar(L: lista; p:
    endereco):tipo_elem;
{ Retorna (recupera) o item do endereço p
da Lista L se sucesso caso contrário (p
= nil) emite msg. No Pascal tipo_elem
deve ser um tipo pré-definido}
```

```
begin
    if p <> 0 then Buscar := L.A[p].info
    else writeln('Erro')
end;
```



```
function Remover (var L: lista; x: tipo_elem): boolean;
{ Remove da lista o item especificado. Se sucesso
  retorna true caso contrário (item não encontrado ou
  lista vazia) retorna false}
```

```
var achou : boolean;
    j, atual, ant: indice;
```

```
begin
```

```
  achou := false;
```

```
  atual := L.Prim;
```

```
  ant:= 0;
```

```
  while (atual <> 0) and not achou do
```

```
    if igual(x,L.A[atual].info) then achou := true
```

```
    else begin
```

```
      ant := atual;
```

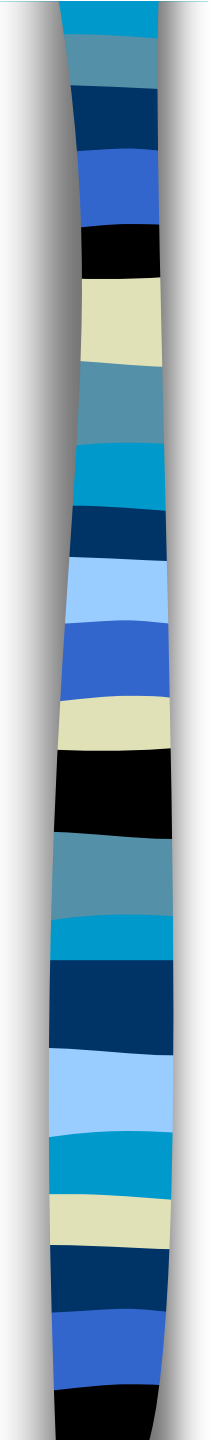
```
      atual := L.A[atual].lig
```

```
    end;
```

```
  If ant = 0 then Remover:= Remove_frente(L,x)
```

```
  else Remover := Remove_apos(L, ant, x)
```

```
end;
```



```
procedure Imprimir (L: lista);  
{ Imprime os elementos da lista.}
```

```
var p: indice;
```

```
begin
```

```
  p := L.Prim;
```

```
  while p <> 0 do
```

```
    begin
```

```
      imp_elem(L.A[p].info);
```

```
      p := L.A[p].lig
```

```
    end;
```

```
end;
```




Resumo: Listas Encad. Estáticas –Não Seq.

■ Vantagens:

- não há movimentos durante inserção e remoção de elementos da lista; apenas ponteiros são alterados.

■ Desvantagens:

- Acesso ao i -ésimo elemento deixa de ser direto; requer acesso aos $i-1$ elementos anteriores;
- Ainda exige previsão de espaço; (*)
- Requer gerenciamento da lista Dispo (*)

■ Alternativa: Lista Dinâmica: elimina desvantagens (*)