

**Instituto Luterano de Ensino Superior de Ji-Paraná**  
**Curso Bacharelado em Informática**  
**Estrutura de Dados I**  
**Prof.: José Luiz A. Duizith**

**Matrizes:**

Possui um n° finito e fixo de elementos.

Operações:

- 1) Consulta
- 2) Alteração
- 3) Criação ( Procedimento Cria\_Matriz (M,índices) )
- 4) Destruição ( Procedimento Destroi\_Matriz (M) )

**Listas**

Possui um n° finito, porém variável de elementos, conjunto de dados ordenados e de n° variável de elementos.

**Listas Lineares ou Contíguas**

Cada elemento da lista ocupa posição sucessiva ao anterior.

**Listas Encadeadas**

A ordem dos elementos da lista é dado através de apontadores (elos).

**A) Representação por Contiguidade (Listas Contíguas)**

Uma Lista contígua é um conjunto de  $n > 0$  nodos;

$x_1, x_2, x_3, \dots, x_n$

que reflete as seguintes propriedades:

- Se  $n > 0$ , então  $x_1$  é o 1° nodo da lista e  $x_n$  é o último
- Para  $1 < k < n$ ,  $X_k$  é precedido por  $X_{k-1}$  e sucedido por  $X_{k+1}$
- Para  $n < 0$  então é dito que a lista é **VAZIA**.

Utiliza a seqüencialidade da memória:

**L(lista) :**

x1	x2	x3	.....	xn	.....	
<b>1</b>	<b>2</b>	<b>3</b>		<b>n</b>		<b>m</b>

**n:** n° de elementos da lista (em 1 determinado momento);

**m:** n° máximo suportável pela lista;

**Operações:**

- 1) Consulta a um determinado elemento
- 2) Inserção de um novo elemento (aumento da lista)
- 3) Remoção de um elemento (diminuição da lista)
- 4) Concatenação de 2 listas ( 2 listas separadas e unir em uma somente lista)
- 5) Contagem do n° de elementos
- 6) Separação de uma lista em várias outras

→ **Procedimento Cria\_Lista\_Contigua ( L,M,N )**

{ cria lista contigua de no máximo n elementos }  
 Cria\_Matriz (L,1,n)  
 N ← 0

→ **Procedimento Destroi\_Lista\_Contigua (L)**  
 Destroi\_Matriz (L)

→ **Funcao Consulta\_Lista\_Contigua (L,N,Pos): Valor**  
 { Pos = Posição }  
 Se (Pos < 1) ou (Pos > N)  
 Entao ERRO {posição inválida}  
 Senão Valor ← L [Pos]

**Inserção** : Implica no aumento do nº de elementos

**Exercício** :

- 1) Inserção de um elemento antes do 1º nodo
- 2) Inserção de um elemento antes de uma determinada posição (K-ésima)
- 3) Inserção de um elemento na k-ésima posição
- 4) Inserção de um elemento depois de uma determinada posição (K-ésima)
- 5) Inserção de um elemento depois do último nodo

L:

10	20	30	40	50		
1	2	3	4	5		<b>m</b>
		<b>k</b>		<b>n</b>		

**Obs.:** Quando  $m = n$  (lista cheia) não poderá haver inserção de elementos, caso contrário perderá as informações.

**Resultados:**

1) Procedimento Insere\_Lista\_Contigua\_A (L,N,M,Valor)  
 Se ( $m = n$ ) {Lista Cheia}  
 Entao ERRO {Overflow}  
 Senao Para CONT ← N Decrementa até 1  
 Faça L[CONT + 1] ← L [CONT]  
 L [1] ← Valor  
 N ← N + 1

**CONT** é uma variável local.

2) Procedimento Insere\_Lista\_Contigua\_B (L,N,M,Valor,k)  
 Se ( $m = n$ ) {lista cheia}  
 Entao ERRO {Overflow}  
 Senao Se ( $K < 1$ ) ou ( $K > N$ )  
 Entao ERRO\_2 {posição inválida}  
 Senao Para CONT ← N Decrementa até k  
 Faça L [ CONT + 1 ] ← L [ CONT ] {L[n+1] := Valor N }  
 L [k] ← Valor  
 N ← N + 1

3) Procedimento Insere\_Lista\_Contigua\_C ( L,N,M,Valor,k)

```

Se ( m = n )           {lista cheia}
Entao ERRO_1           {Overflow}
Senao Se ( k < 1 ) ou ( k > n )
    Entao ERRO_2       { posição inválida}
    Senao Para CONT ← N Decrementa até k
        Faça L [ CONT + 1 ] ← L [CONT]
        L [k] ← Valor
        N ← N + 1

```

**Obs.:** b e c são iguais porque são implementados em cima de um vetor.

4) Procedimento *Inserere\_Lista\_Contigua\_D* (L,N,M,Valor,k)

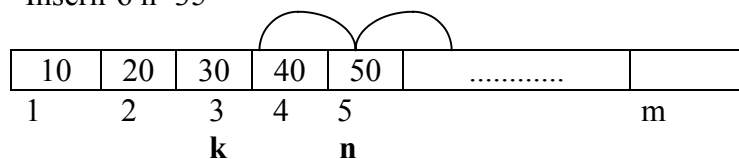
```

Se ( m = n )           {lista cheia}
Entao ERRO_1           {Overflow}
Senao Se ( k < 1 ) ou ( k > n )
    Entao ERRO_2       {posição inválida}
    Senao Para CONT ← N Decrementa até k+1
        Faça L[CONT+1] ← L[CONT]
        L[k+1] ← Valor
        N ← N + 1

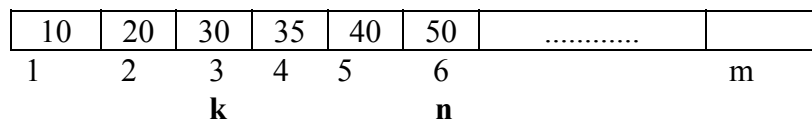
```

**Obs.:**  $K = N$  = pode inserir o elemento depois de N.

Inserir o n° 35



*Depois do algoritmo*



5) Procedimento *Inserere\_Lista\_Contigua\_E* (L,N,M,Valor)

```

Se ( n = m )           {Lista cheia}
Entao ERRO_1           {Overflow}
Senao L [N+1] ← Valor
    N ← N + 1

```

**Remoção :** Implica na diminuição do n° de elementos

**Exercícios:**

- 1) Remoção do 1° nodo
- 2) Remoção do nodo anterior a k-ésima posição
- 3) Remoção do nodo da k-ésima posição
- 4) Remoção do nodo posterior a k-ésima posição
- 5) Remoção do último nodo

L : ◡ ◡ ◡ ◡

10	20	30	40	50	.....	
1	2	3	4	5		<b>m</b>
		<b>k</b>		<b>n</b>		

L :

20	30	40	50	50	.....	
	<b>k</b>		<b>n</b>			<b>m</b>

Lista Nova

não faz parte da lista, pois é sujeira

### Resultados:

#### 1) Procedimento Remove\_Nodo\_A (L,N)

Se ( N = 0 ) {Lista Vazia}  
 Então ERRO {Overflow}  
 Senão Para CONT ← 1 até N - 1  
 Faça L [ CONT ] ← L [ CONT + 1 ]  
 N ← N - 1

#### 2) Procedimento Remove\_Nodo\_B (L,N,k)

Se ( N = 0 ) {Lista Vazia}  
 Então ERRO {Overflow}  
 Senão Se (k<=1) ou ( k>N)  
 Então ERRO\_2 {posição Inválida}  
 Senão Para CONT ← k - 1 até N - 1  
 Faça L [ CONT ] ← L [ CONT + 1 ]  
 N ← N - 1

#### 3) Procedimento Remove\_Nodo\_C (L,N,k)

Se ( N = 0 ) {Lista Vazia}  
 Então ERRO\_1 {Overflow}  
 Senão Se (k<1) ou ( k>N)  
 Então ERRO\_2 {posição Inválida}  
 Senão Para CONT ← k até N - 1  
 Faça L [ CONT ] ← L [ CONT + 1 ]  
 N ← N - 1

#### 4) Procedimento Remove\_Nodo\_D (L,N,k)

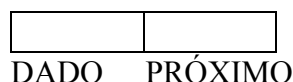
Se ( N = 0 ) {Lista Vazia}  
 Então ERRO {Overflow}  
 Senão Se (k<1) ou ( k>=N)  
 Então ERRO\_2 {posição Inválida}  
 Senão Para CONT ← k + 1 até N - 1  
 Faça L [ CONT ] ← L [ CONT + 1 ]  
 N ← N - 1

#### 5) Procedimento Remove\_Nodo\_E (L,N)

Se (  $N = 0$  )                    {Lista Vazia}  
 Então ERRO\_1                    {Overflow}  
 Senão  $N \leftarrow N - 1$

## B) Representação por encadeamento (Listas Encadeadas)

- Listas Simplesmente Encadeadas:  
 Cada nodo de uma lista simplesmente encadeada possui, no mínimo, 2 campos.



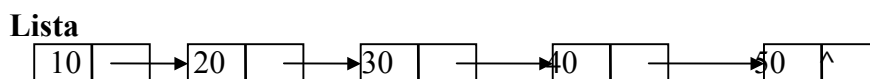
**Dado:** é o campo que contém as informações a serem processadas no nodo.

**Próximo:** é o campo que contém a referência (apontador) para o próximo nodo da lista.

**Obs.:**

- O último nodo de uma lista simplesmente encadeada é aquela que possui o campo próximo nulo (NIL)
- O primeiro nodo da lista deve ser referencial a qualquer instante, para tal será utilizada a variável apontadora LISTA que conterà a referência deste nodo.

**EX:**



**Comandos Especiais:**

**ALOQUE (X):** é o comando que pede um *nodo* livre. O sistema de gerência de memória coloca em X o endereço (referencia) do novo *nodo* alocado. Se o endereço em X for NIL é porque não foi possível alocar espaço para um novo nodo.

**DESALOQUE (X):** é o comando que devolve ao sistema de gerência de memória o espaço do nodo referenciado por X.

ALOQUE (X) → NEW (X)  
 DESALOQUE (X) → DISPOSE (X)  
 NIL → NIL

**Pascal:**

```

Type
  Aponta_Nodo = ^Nodo;
  Nodo = Record
    Dado : Informação;
    Próximo : Aponta_Nodo;
  end;
  
```

Var

```

  Lista, x : Aponta_Nodo;
  
```

**Operação:**

- Procedimento Cria\_Lista (Lista)  
Lista  $\leftarrow$  NIL
- Procedimento Destroi\_Lista (Lista)  
Enquanto (Lista  $\neq$  NIL)  
Faça AUX  $\leftarrow$  Lista  
Lista  $\leftarrow$  Lista $\uparrow$ .Prox  
Desaloque (Aux)

- **Suponha (Turbo Pascal)**

Program PI;

Var

x: ^String;

Begin

While (True)

Do Begin

New(x)

If (x=Nil)

Then Begin

Writeln('Faltou Memória')

Halt

*{termina o programa}*

end;

end;

end.

**→ Para evitar erro de execução (por falta de memória) redefinir a função HEAPFUNC:**

.....  
{F+}

Function HeapFunc (Size:Word): Integer;

{F-}

Begin

HeapFunc := 1

end;

**→ E inclua no programa principal:**

.....

HeapError := @HeapFunc;

**→ Desta forma, caso um comando NEW não conseguir alocar espaço o apontador do comando NEW fica nulo (NIL).**

<p><b>Instituto Luterano de Ensino Superior de Ji-Paraná</b>  <b>Curso Bacharelado em Informática</b>  <b>Estrutura de Dados I</b></p>
--

**LISTAS ENCADEADAS****Inserção:**

Exercícios:

- 1) Inserção de um novo elemento antes do 1º nodo
- 2) Inserção de um novo elemento antes da k-ésima posição
- 3) Inserção de um novo elemento na k-ésima posição
- 4) Inserção de um novo elemento depois da k-ésima posição
- 5) Inserção de um novo elemento depois da k-ésimo nodo

**1) Procedimento Insere\_lista\_1 (Lista,Valor)**

```

{ serve para uma lista que já existe ou para uma vazia }
ALOQUE(Aux)
Se (Aux = Nil) { não conseguiu alocar espaço para novo elemento }
Entao ERRO {Overflow - sem espaço }
Senao Aux↑.Dado ← Valor
      Aux↑.Prox ← Lista
      Lista ← Aux

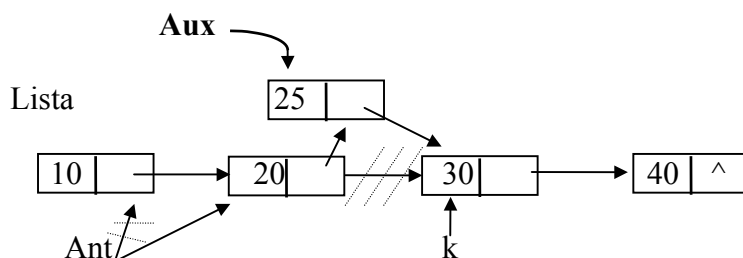
```

**2) Procedimento Insere\_lista\_2 (Lista,Valor,k)**

```

{ serve para uma lista que já existe ou para uma vazia }
ALOQUE(Aux)
Se (Aux = Nil) { não conseguiu alocar espaço para novo elemento }
Entao ERRO {Overflow - sem espaço }
Senao Aux↑.Dado ← Valor
      Aux↑.Prox ← k
      Se ( k = lista ) { insere antes da 1º posição }
      Então Lista ← Aux
      Senão Ant ← Lista
      Enquanto (Ant↑.Prox <> k)
      Faça Ant ← Aux↑.Prox
      Ant↑.Prox ← Aux

```

**3) Procedimento Insere\_Lista\_3**

```

{ Igual ao 2º }

```

**4) Procedimento Insere\_Lista\_4 (Lista, Valor, k)**

```

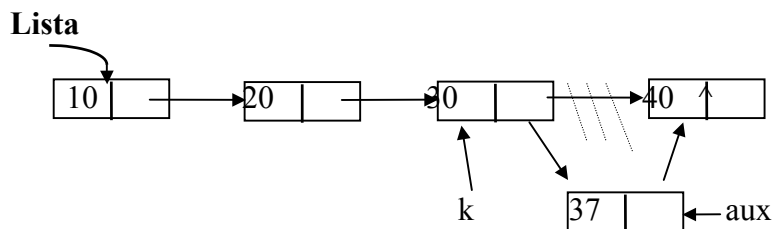
ALOQUE (Aux)

```

```

Se (Aux = Nil)
Então ERRO {Overflow}
Senão Aux↑.Dado ← Valor
      Se (Lista = Nil) { Lista Vazia}
      Então Lista ← Aux
      Aux↑.Prox ← Nil
      Senão Aux↑.Prox ← k↑.Prox
            k↑.Prox ← Aux

```

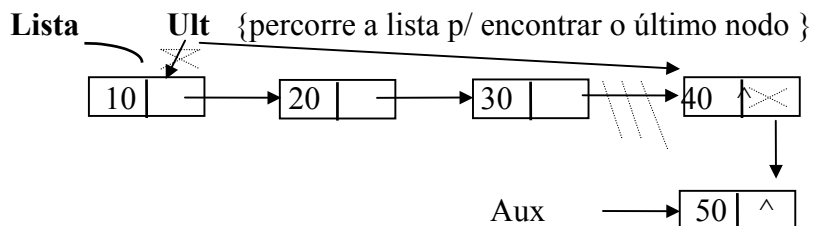


### 5) Procedimento Insere\_Lista\_5 (Lista, Valor)

```

ALOQUE (Aux)
Se (Aux = Nil)
Então ERRO {Overflow}
Senão Aux↑.Dado ← Valor
      Aux↑.Prox ← Nil
      Se (Lista = Nil) { Lista Vazia}
      Então Lista ← Aux
      Senão Ult ← Lista
            Enquanto (Ult↑.Prox ≠ Nil)
            Faça Ult ← Ult↑.Prox
            Ult↑.Prox ← Aux

```



### Remoção:

Exercício:

- 1) Remoção do 1º nodo
- 2) Remoção do nodo anterior a k-ésima posição
- 3) Remoção do nodo da k-ésima posição

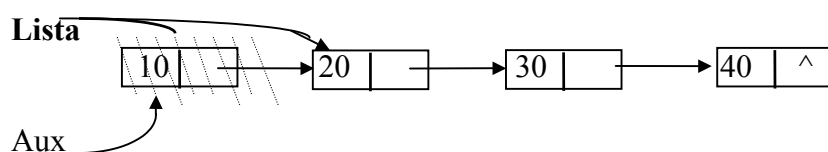
- 4) Remoção do nodo posterior a k-ésima posição
- 5) Remoção do último nodo

### 1) Procedimento Remove\_Lista\_1 (Lista)

```

Se (Lista = Nil)           { Lista Vazia }
Entao ERRO                 { Underflow }
Senão
  Aux ← Lista
  Lista ← Lista↑.Prox
  DESALOQUE (Aux)

```



### 2) Procedimento Remove\_Lista\_2 (Lista, k)

```

Se (Lista = Nil)           { Lista Vazia }
Entao ERRO_1              { Underflow }
Senão
  Se (K=Lista)
    Entao Erro_2
  Senão
    Aux ← Lista
    Se ( Aux↑.Prox = k )
      Entao
        Lista ← Lista↑.Prox
        DESALOQUE (Aux)
      Senão
        Ant ← Aux↑.Prox
        Enquanto (Ant↑.Prox <> k )
          Faça
            Ant ← Ant↑.Prox
            Aux ← Aux↑.Prox
          Aux↑.Prox ← k
        DESALOQUE (Ant)

```

### 3) Procedimento Remove\_Lista\_3 (Lista,k)

```

Se (Lista = Nil)
Entao ERRO
Senão
  Se (k = Lista)           {remove o 1º nodo}
    Entao Lista ← Lista↑.Prox
  Senão
    Ant ← Lista
    Enquanto (Ant↑.Prox <> k)
      Faça
        Ant ← Ant↑.Prox
      Ant↑.Prox ← k↑.Prox
    DESALOQUE (k)

```

### 4) Procedimento Remove\_Lista\_4 (Lista, k)

```

Se ( Lista = Nil )
Entao ERRO_1
Senão
  Se (k↑.Prox = Nil)

```

```

Entao ERRO_2
Senao Aux ← k↑.Prox
    k↑.Prox ← Aux↑.Prox
    DESALOQUE (Aux)

```

### 5) Procedimento Remove\_Lista\_5 (Lista)

```

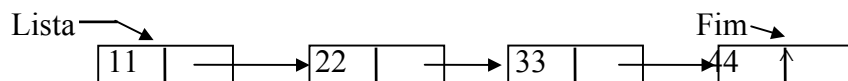
Se ( Lista = Nil )
Entao ERRO
Senao Aux ← Lista
    Se (Lista↑.Prox = Nil)    {Lista de 1 só nodo}
    Entao Lista ← Nil
    Senao Enquanto (Aux↑.Prox <> Nil)
        Faça
            Ant ← Aux
            Aux ← Aux↑.Prox
        Ant↑.Prox ← Nil
    DESALOQUE (Aux)

```

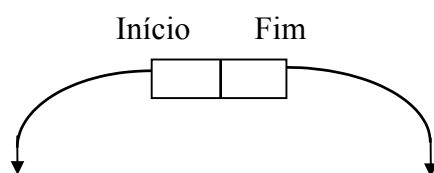
## Lista com Descritor (listas com “HEADER”)

Operação que envolve o último nodo de uma lista normalmente necessitam de um caminhamento sobre a lista (utilizando-se uma variável auxiliar) e finalmente encontrado o último nodo realiza-se a operação.

Para facilitar a operação sobre o último nodo podemos utilizar uma segunda variável (vamos chamá-la de FIM) que endereça o último nodo.

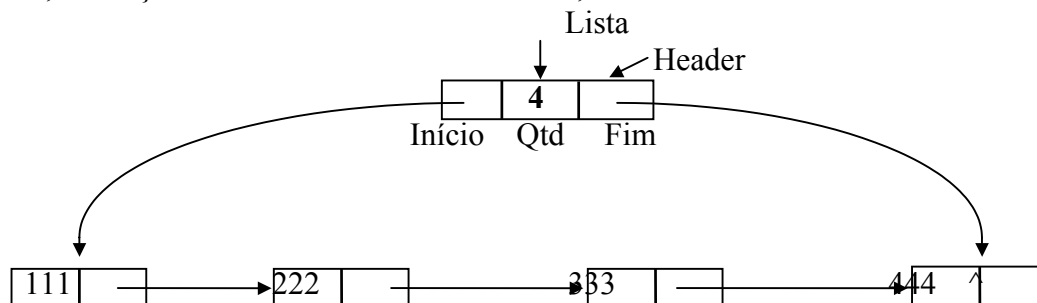


Entretanto, para facilitar a representação da lista acima podemos reunir as duas referências em uma única estrutura (registro).



A este tipo de estrutura chamamos de Nodo Descritor, líder ou “Header” da lista. O acesso aos elementos da lista será sempre realizada através de seu “header”.

O “header” pode conter outras informações adicionais como : quantidade de nodos da lista, descrição dos dados contados nos nodos, etc.



**Ex:**

**Pascal**

Type

Header = Record

Início : Aponta\_Nodo;                    {referência ao 1º nodo da lista}

Quant : Integer;                         {quantidade de nodos da lista}

Fim : Aponta\_Nodo;                    {referência ao último nodo da lista}

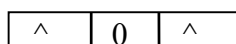
end;

Aponta\_Header = ^Header;

Var

Lista : Aponta\_Header;

**Lista Vazia:**



**Operação:****Criação :****Procedimento Cria\_Lista (Lista)**

```

ALOQUE (Lista)
Se (Lista = Nil )
Entao ERRO
Senao
  Lista↑.Inicio ← Nil
  Lista↑.Qtd ← 0
  Lista↑.Fim ← Nil

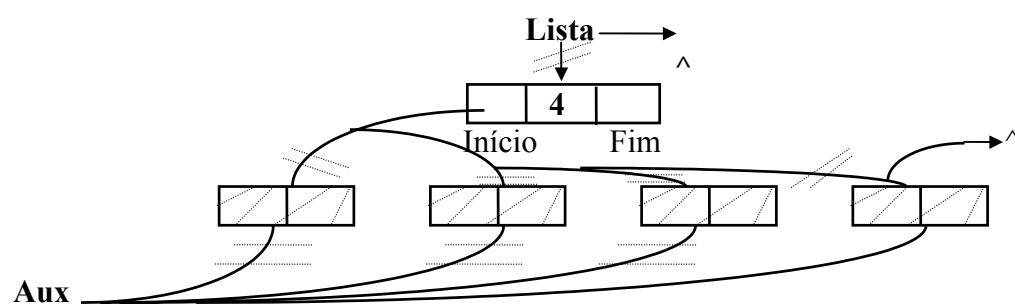
```

**Destruição:****Procedimento Destroi\_Lista (Lista)**

```

Enquanto (Lista↑.Inicio <> Nil)
Faça
  Aux ← Lista↑.Inicio
  Lista↑.Inicio ← Aux↑.Prox
  DESALOQUE (Aux)
DESALOQUE (Lista)
Lista ← Nil

```



**Procedimento Insere\_Esquerda (Lista,Valor)**

{ Insere elemento antes do 1º nodo da lista-antigo-Insere-Lista-1 }

ALOQUE (Aux)

Se (Aux = Nil)

Entao ERRO {Overflow}

Senao Aux↑.Dado ← Valor

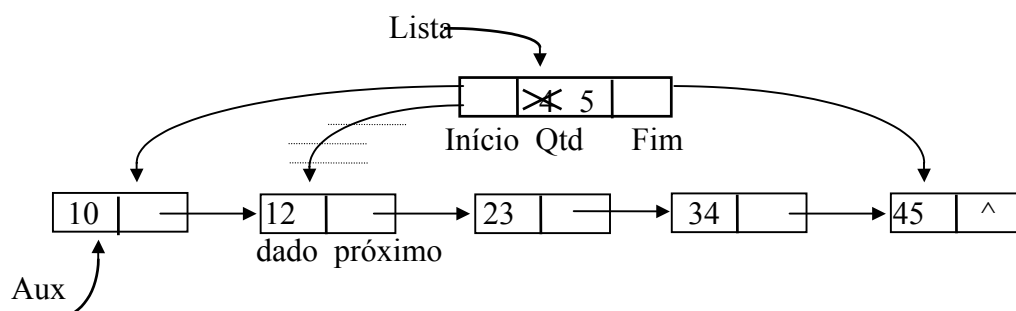
Aux↑.Prox ← Lista↑.Inicio

Lista↑.Inicio ← Aux

Se (Lista↑.Qtd = 0) { Lista está vazia }

Entao Lista↑.Fim ← Aux

Lista↑.Qtd ← Lista↑.Qtd + 1

**Procedimento Insere\_Direita (Lista,Valor)**

{ Insere elemento depois do último nodo da lista\_antiga Insere\_Lista\_5 }

ALOQUE (Aux)

Se (Aux = Nil)

Entao ERRO {Overflow}

Senao Aux↑.Dado ← Valor

Aux↑.Prox ← Nil

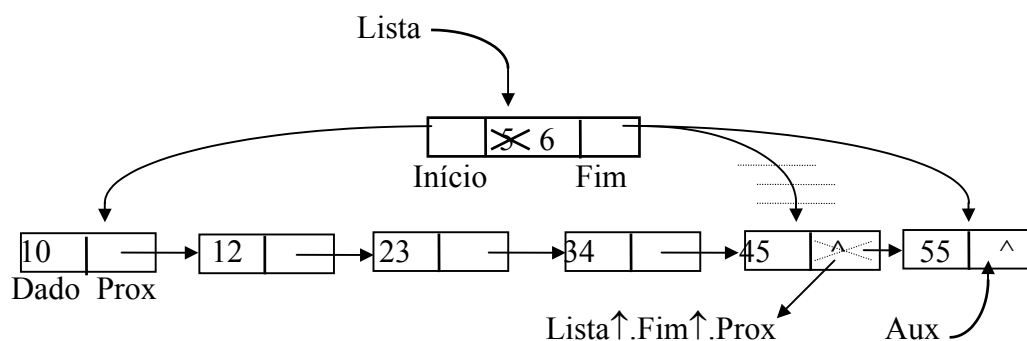
Se (Lista↑.Qtd = 0) {Lista Vazia}

Entao Lista↑.Inicio ← Aux

Senao Lista↑.Fim↑.Prox ← Aux

Lista↑.Fim ← Aux

Lista↑.Qtd ← Lista↑.Qtd + 1

**Procedimento Remove\_Esquerda (Lista)**

*{ Remove 1º nodo da lista }*

```

Se (Lista↑.Qtd = 0) {Lista Vazia}
Entao ERRO {Underflow}
Senao
  Aux ← Lista↑.Inicio
  Lista↑.Inicio ← Aux↑.Prox
  Se (Lista↑.Qtd = 1) {Lista possui apenas 1 nodo}
  Entao Lista↑.Fim ← Nil
  DESALOQUE (Aux)
  Lista↑.Qtd ← Lista↑.Qtd - 1

```

### Procedimento Remove\_Direita (Lista)

*{Remove o último nodo da lista}*

```

Se (Lista↑.Qtd = 0) {Lista Vazia}
Entao ERRO
Senao
  Aux ← Lista↑.Fim
  Se (Lista↑.Qtd = 1)
  Entao
    Lista↑.Inicio ← Nil
    Lista↑.Fim ← Nil
  Senao
    Ant ← Lista↑.Inicio
    Enquanto (Ant↑.Prox <> Aux)
    Faça Ant ← Ant↑.Prox
    Lista↑.Fim ← Ant
    Ant↑.Prox ← Nil {ou Lista↑.Fim↑.Prox ← Nil}
  DESALOQUE (Aux)
  Lista↑.Qtd ← Lista↑.Qtd - 1

```

