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

Matrizes:

Possui um nº 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 nº finito, porém variável de elementos, conjunto de dados ordenados e de nº variávelde 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;
x1,x2,x3,...,xn

que reflete as seguintes propriedades:

- Se n > 0, então x1 é o 1º nodo da lista e xn é o último
- Para 1 < k < n, Xk é precedido por Xk-1 e sucedido por Xk+1
- Para n <> 0 então é dito que a lista é **VAZIA.**

Utiliza a seqüencialidade da memória:

L(lista) :

x1	x2	x3	.....	xn	.....	
1	**2**	**3**		**n**		**m**

n: nº de elementos da lista (em 1 determinado momento);
m: nº 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 nº 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		**m**
		k		**n**		

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
```

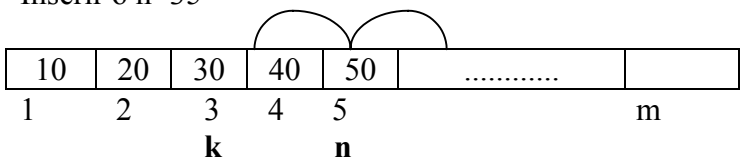
Obs.: b e c são iguais porque são implementados em cima de um vetor.

4) Procedimento Insere_Lista_Contigua_D (L,N,M,Valor,k)

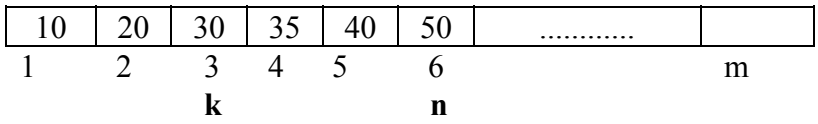
```
Se ( m = n )            {lista cheia}
Entao ERRO_1            {Overflow}
Senao  Se (k<1) ou (k>n)
          Entao ERRO_2          {posição inválida}
          Senao  Para CONT ← N Decrementa até k+1
                    Faça L[CONT+1] ← L[CONT]
                    L[k+1] ← Valor
                 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

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

Inserir o nº 35



Depois do algorítmo



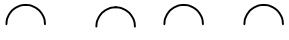
5) Procedimento Insere_Lista_Contigua_E (L,N,M,Valor)

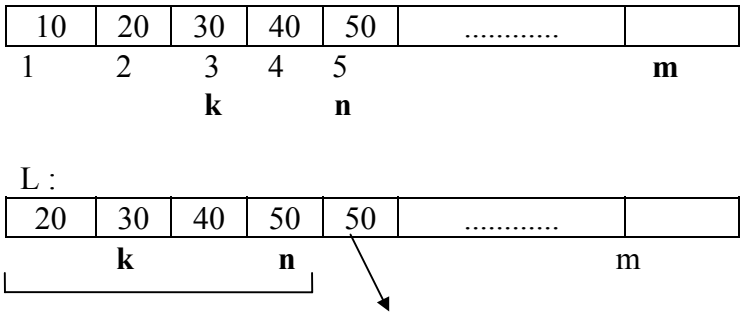
```
Se ( n = m )            {Lista cheia}
Entao ERRO_1            {Overflow}
Senao  L [N+1] ← Valor
          N ← N + 1
```

Remoção : Implica na diminuição do nº de elementos

Exercícios:

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

L : 



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
```

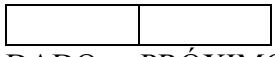
4) Procedimento Remove_Nodo_D (L,N,k)
```
Se ( N = 0)     {Lista Vazia}
Então ERRO   {Overflow}
Senão  Se (k<1) ou ( k>=N)
       Então ERRO_2        {posição Inválida}
       Senão  Para CONT ← k +1 até N - 1
              Faça L [ CONT ] ← L [ CONT +1 ]
              N ← N - 1
```

5) Procedimento Remove_Nodo_E (L,N)

```
Se ( N = 0)          {Lista Vazia}
Então ERRO_1         {Overflow}
Senão  N ← N - 1
```

B) Representação por encadeamento (Listas Encadeadas)

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



DADO PRÓXIMO

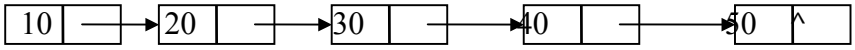
Dado: é o campo que contém as informações a serem processadas no nodo.
Próximo: é o campo que contém a referência (apontador) para o próximo nodo da lista.

Obs.:

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

EX:

Lista



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 ← NIL

- Procedimento Destroi_Lista (Lista)
  Enquanto (Lista <> NIL)
  Faça AUX ← Lista
  Lista ← Lista↑.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).

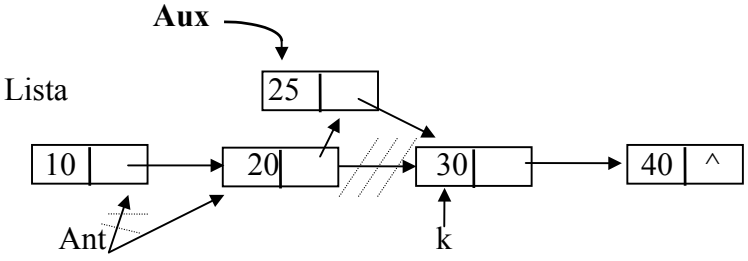
Prof.: José Luiz A. Duizith

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



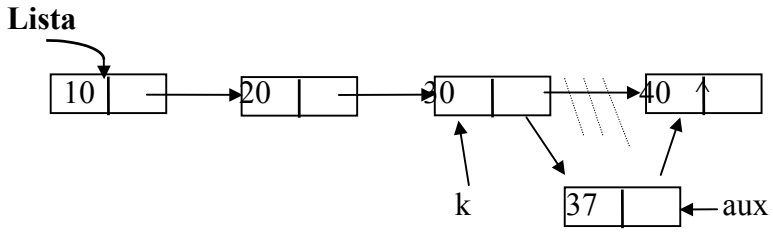
3) **Procedimento Insere_Lista_3**

{ Igual ao 2º }

4) **Procedimento Insere_Lista_4 (Lista, Valor, k)**

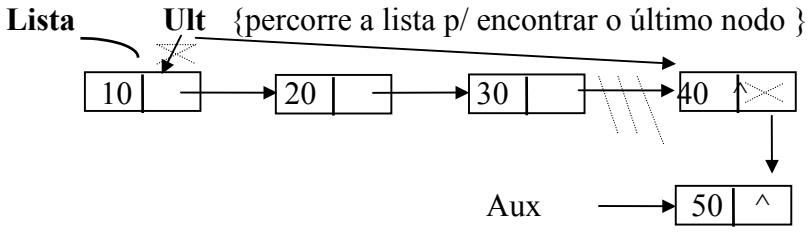
ALOQUE (Aux)

```
Se (Aux = Nil)
Então ERRO  {Overflow}
Senão Aux↑.Dado ← Valor
      Se (Lista = Nil )      { Lista Vazia}
      Então Lista ← Aux
            Aux↑.Prox ← Nil
      Senao 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
      Senao Ult  ← Lista
            Enquanto (Ult↑.Prox  <> Nil )
            Faça Ult  ← Ult↑.Prox
            Ult↑.Prox ← Aux
```



Remoção:

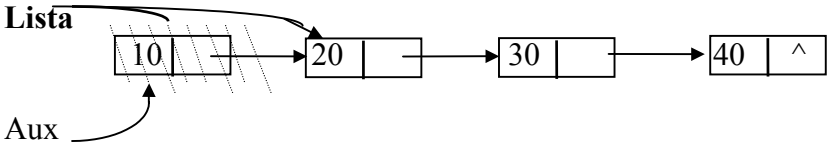
Exercício:

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}
Senao Se (K=Lista)
      Entao Erro_2
      Senao Aux ← Lista
            Se ( Aux↑.Prox = k )
            Entao Lista ← Lista↑.Prox
                  DESALOQUE (Aux)
            Senao 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
Senao Se (k = Lista)         {remove o 1º nodo}
      Entao Lista ← Lista↑.Prox
      Senao 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
Senao Se (k↑.Prox = Nil)
```

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

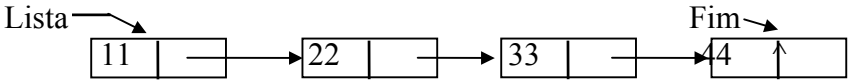
5) **Procedimento Remove_Lista_5 (Lista)**

```
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)
```

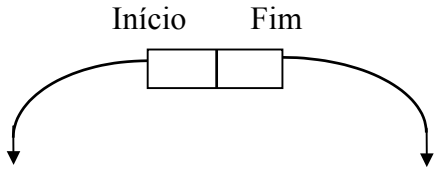
Lista com Descritor (listas com "HEADER")

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.

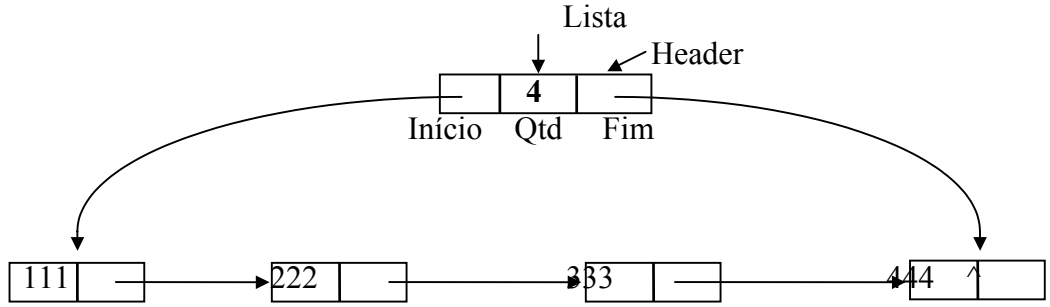


Entretanto, para facilitar a representação da lista acima podemos reunir as duas referências em uma única estrutura (registro).



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.



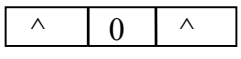
Ex:

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 :

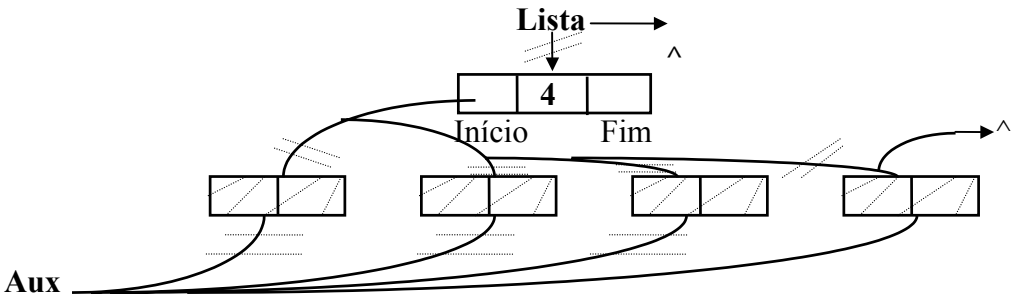
Procedimento Cria_Lista (Lista)

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

Destruição:

Procedimento Destroi_Lista (Lista)

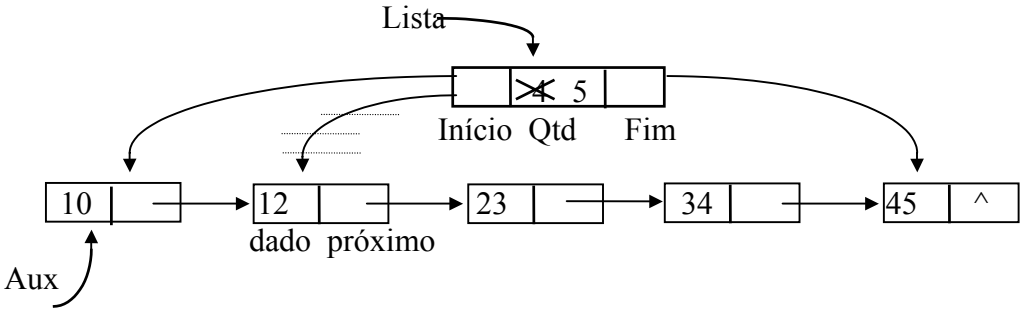
```
Enquanto (Lista↑.Inicio <> Nil)
Faça  | Aux ← Lista↑.Inicio
      | Lista↑.Inicio ← Aux↑.Prox
      | DESALOQUE (Aux)
DESALOQUE (Lista)
Lista ← Nil
```



Procedimento Insere_Esquerda (Lista,Valor)

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

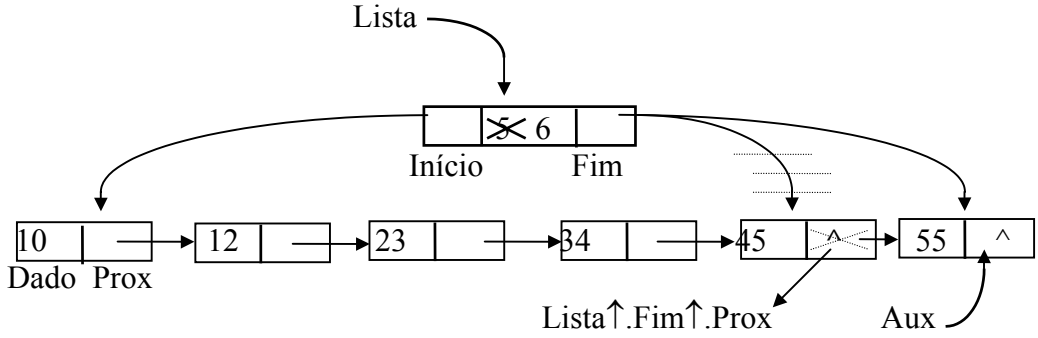
```
ALOQUE (Aux)
Se (Aux = Nil)
Entao ERRO   {Overflow}
Senao  Aux↑.Dado ← Valor
       Aux↑.Prox ← Lista↑.Inicio
       Lista↑.Inicio ← Aux
       Se (Lista↑.Qtd = 0 )          { Lista está vazia }
       Entao Lista↑.Fim ← Aux
       Lista↑.Qtd  ← Lista↑.Qtd + 1
```



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
```