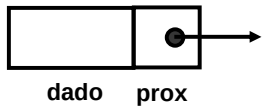
Listas Lineares Simplesmente Encadeadas

Renata de Matos Galante
INF 01203 – Estruturas de Dados

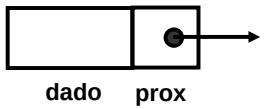
Características

- Cada elemento mantém um dado e uma referência (apontador) para o próximo elemento

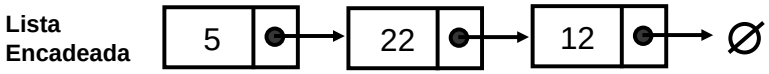


Características

- Cada elemento mantém um dado e uma referência (apontador) para o próximo elemento



- Exemplo de uma lista encadeada



Características

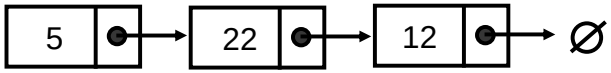
- Não há limite máximo para o número de elementos
  - ***O limite é a capacidade de memória***
- Elementos não estão em posições contíguas na memória
  - Alocação de novos elementos em tempo de execução
  - Melhor aproveitamento de espaço livre na memória

Estrutura de Dados - TAD

- **Dados**
  - Tipo de dados para armazenar os elementos da lista
  - Componentes para controle da estrutura lista
- **Operações**
  - **inicializa():** cria uma lista vazia
  - **insere():** insere um elemento em uma posição da lista
  - **remove():** remove um elemento na lista
  - **consulta():** consulta um elemento da lista
  - **lista():** exibe todos os elementos da lista
  - **destroi():** destrói a lista

Especificação dos Dados

- Dados de cada elemento da lista
  - ???
- Dados sobre a lista
  - ????

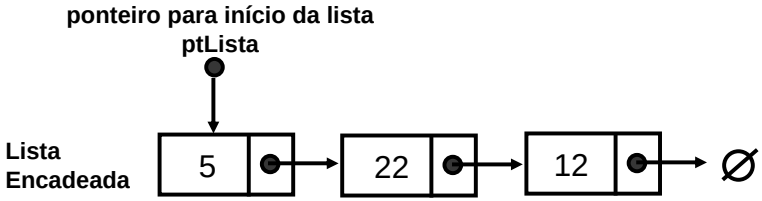


Especificação dos Dados

- Estrutura de cada elemento
  - Atributos
    - Dados
    - Ponteiro para o próximo
- Dados sobre a lista
  - Referência para o primeiro elemento da lista

Especificação dos Dados

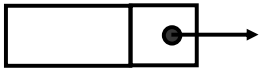
- Estrutura de cada elemento
  - Atributos
    - Dados
    - Ponteiro para o próximo
- Dados sobre a lista
  - Referência para o primeiro elemento da lista



Estrutura de Dados

- **Dados**

```
tipo pnodo = ↑nodo
     nodo = registro
              info : info
              elo  : pnodo
      fim
```



  - Ponteiro para o primeiro elemento da lista
- **Operações**
  - **inicializa():** cria uma lista vazia
  - **insere():** insere um elemento em uma posição da lista
  - **remove():** remove um elemento na lista
  - **consulta():** consulta um elemento da lista
  - **lista():** exibe todos os elementos da lista
  - **destroi():** destrói a lista

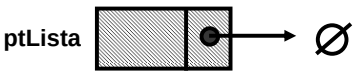
Operações

- **inicializa()**
- **insere()**
- **remove()**
- **consulta()**
- **listaTodos()**
- **lista()**
- **destroi()**

Algoritmos

Inicializa()

- Cria uma lista vazia



Inicializa() – Algoritmo

Inicializa lista()

Entrada Um ponteiro para um tipo lista encadeada

Saída um ponteiro para uma lista vazia

```
1  ptLista := θ
```

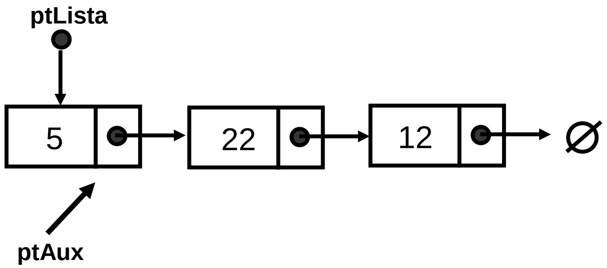
Inicializa() implementação em Pascal

```
type
    pnodo = ^nodo;

    nodo = record
              info  : tipoInfoNodo;
              prox : pnodo;
     end;

procedure inicializarLSE( var ptLista: pnodo);
```

```
procedure inicializaLSE( var ptLista: pnodo);
begin
   ptLista := nil;
end;
```

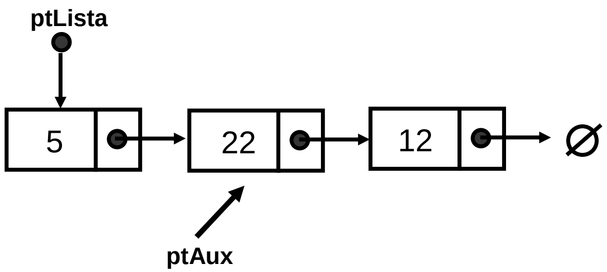

listaTodos()

- Exibe todos os elementos da lista
  - Definir uma variável ponteiro do tipo lista para percorrer todos os elementos da lista iniciando pelo **ptLista**



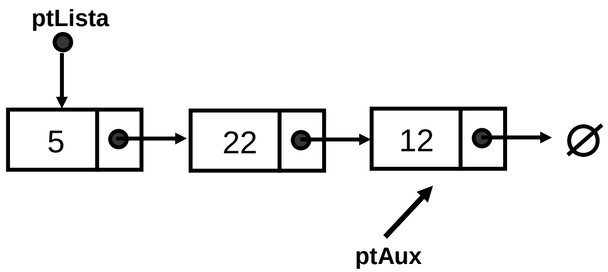
listaTodos()

- Exibe todos os elementos da lista
  - Definir uma variável ponteiro do tipo lista para percorrer todos os elementos da lista iniciando pelo **ptLista**



listaTodos()

- Exibe todos os elementos da lista
  - Definir uma variável ponteiro do tipo lista para percorrer todos os elementos da lista iniciando pelo **ptLista**



listaTodos() – Algoritmo

listaTodos()

Entrada Um ponteiro o primeiro elemento da lista

Saída Uma seqüência de todos os elementos da lista

```
1   ptAux := ptLista
2   enquanto ptAux ≠ θ
3      imprime (pont↑.info)
4      pont := pont ↑.prox
5   end-enquanto
```

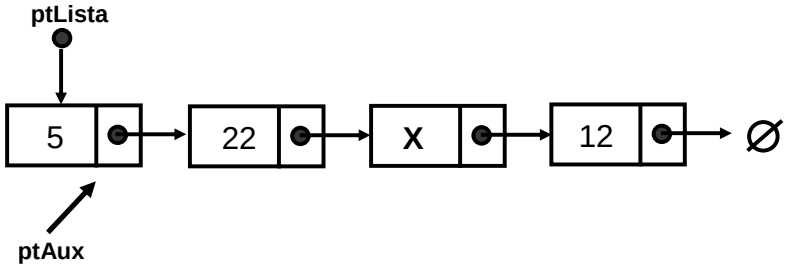

listaTodos() implementação em Pascal

procedure listaTodos(ptLista:pnodo);

```
procedure listaTodos(ptLista:pnodo);
var ptAux: pnodo;
begin
    if ptLista = nil then
      writeln('lista vazia')
    else
    begin
        ptAux:= PtLista;
        while PtAux <> nil do //imprimindo lista
        begin
           write('Codigo: '); writeln(ptAux^.info.cod);
           write('Nome: ');writeln(ptAux^.info.nome);
           ptAux := ptAux^.prox;
        end; //while
    end; //if
end; //begin
```

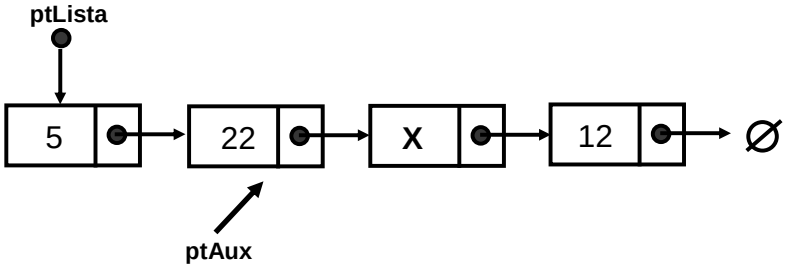

Consulta()

- Pesquisar na lista por um valor x
  - Definir uma variável ponteiro do tipo lista para percorrer todos os elementos da lista iniciando pelo **ptLista**



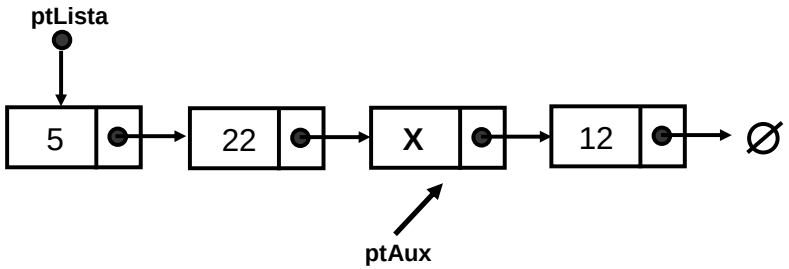
Consulta()

- Pesquisar na lista por um valor x
  - Definir uma variável ponteiro do tipo lista para percorrer todos os elementos da lista iniciando pelo **ptLista**



Consulta()

- Pesquisar na lista por um valor x
  - Definir uma variável ponteiro do tipo lista para percorrer todos os elementos da lista iniciando pelo **ptLista**



Consulta()

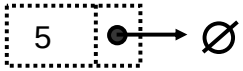
```
início
  se (K < 1) ou (PtLista = nulo)
  então PtK ← nulo
  senão início
        PtK ← PtLista
        enquanto (PtK ≠ nulo) e (K > 1)
        faça início
            K ← K-1
            PtK ← PtK↑.Elo
            fim
        se K > 1
        então PtK ← nulo
        fim
fim
```


Insere()

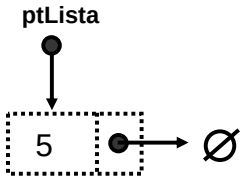
- Inserir primeiro elemento da lista
  - Primeira ação: criar o novo elemento
  - Preciso testar se a lista está vazia? Não
  - Preciso testar se a lista está cheia? Não
  - Se a lista está vazia
    - Criar o primeiro elemento
    - Apontar **ptLista** para o novo elemento

Insere()

Insere()



Insere()



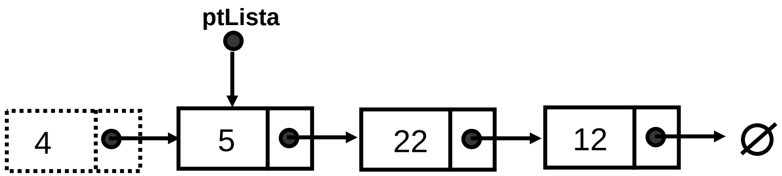
Insere()

```
Func InserirLLE_Prim (var PtLista: TipoPtNó;
                      Dados: TipoInfoNó) : lógico;
var Pt : TipoPtNó;

início
  aloca (Pt);
  se PtLista = nil
  então
        LerInfo (Dados);
        Pt ↑. Info := Dados;
        Pt ↑. Elo := nil ;
        PtLista := Pt;
        fim
fim InserirLLE_Prim;
```

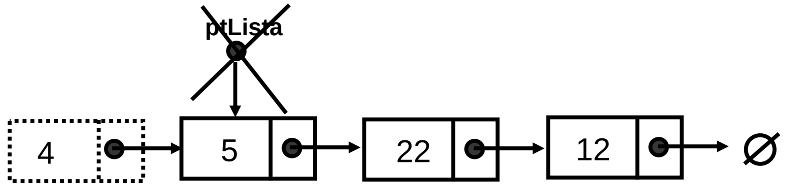

InsereInicio()

- Inserir <u>no início</u> da lista
  - Preciso testar se a lista está vazia? Sim
  - Preciso testar se a lista está cheia? Não
  - Se a lista não está vazia
    - Criar o novo elemento
    - Encadear o novo elemento com o primeiro elemento da lista
    - Apontar **ptLista** para o novo elemento

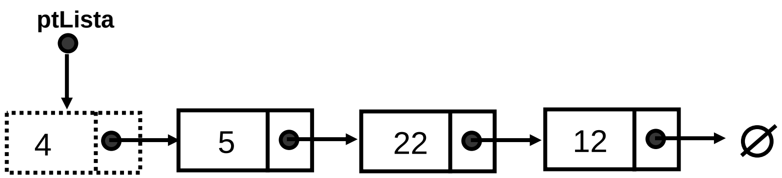
Insere()



Insere()



Insere()



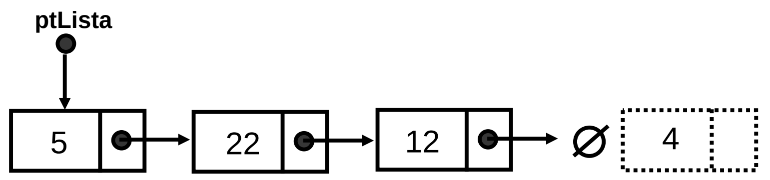
InsereInicio()

```
Func InserirLLE_Ini (var PtLista: TipoPtNó;
                      Dados: TipoInfoNó) : lógico;
var Pt : TipoPtNó;

início
  aloca (Pt);                          { aloca o novo nó }
  se Pt = nil                          { não foi possível alocar novo nó }
  então InsereLLE_Ini := falso
  senão início
          LerInfo (Dados);
          Pt ↑. Info := Dados;         { preenche com dados }
          Pt ↑. Elo := PtLista ;  { encadeia com o que era o primeiro }
          PtLista := Pt;               { passa a ser o primeiro nó }
          fim
fim InserirLLE_Ini;
```

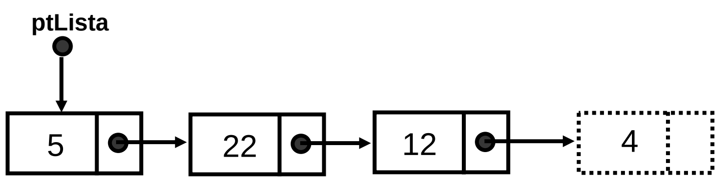
InsereFim()

- Inserir <u>no final</u> da lista
  - Preciso testar se a lista está vazia? Sim
  - Preciso testar se a lista está cheia? Não
  - Se a lista não está vazia
    - Criar o novo elemento
    - Apontar o elo do novo elemento para nulo
    - Encadear o último elemento da lista com o novo elemento

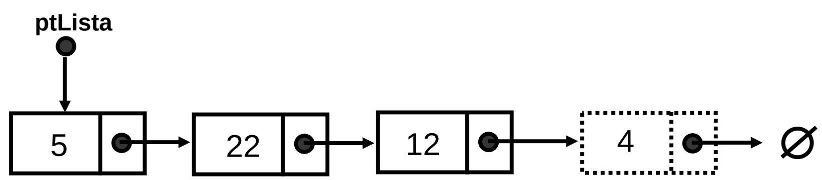
InsereFim()



InsereFim()



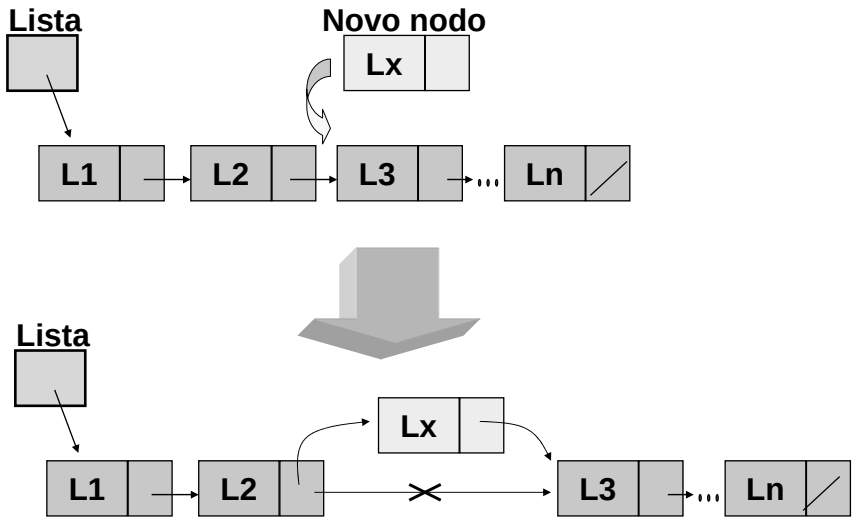
InsereFim()



InsereFim()

```
Proc InsereLLE_F (var ptista: pnodo);
var P1, P2 : pnodo;
    Dados : string;
início
aloca (P2);                          { aloca o novo nodo }
ler (Dados);
P2 ↑. Info := Dados;
P2 ↑ . Elo := nil;                   { vai ser o último }
se ptLista = nil
então  ptLista := P2;                { primeiro nodo }
senão início                         { mais um nodo }
        P1 := ptLista;               { P1 no início da lista }
        enquanto P1 ↑. Elo <> nil
        faça P1 := P1 ↑. Elo;  { P1 no final da lista }
        P1 ↑. Elo := P2;     { encadeia com o novo }
fim;    fim;
```


Inserir no meio



Inserir no meio

```
Proc InsereLLE_M (var Lista: pnodo; Lugar: inteiro);
{ O novo nodo, com valores lidos do teclado, será inserido na posição
Lugar da lista  - Lugar: <= # nodos da lista e >= 1 }
var PAnt, PNovo : pnodo;
    Dados : string;
    N : inteiro;
início
alocar (PNovo);                          { aloca o novo nodo }
ler (Dados);
PNovo ↑. Info := Dados;      { preenche dados do novo nodo }
se Lugar = 1                         { primeiro - atualizar Lista }
então início Lista := Pnovo; Pnovo ↑. Elo := nil   fim
senão início                                 { no meio mesmo }
  PAnt := Lista;                     { PAnt no início da lista }
  para N de 1 até Lugar - 2 faça
      PAnt := PAnt↑. Elo;                    { PAnt no anterior }
  PNovo↑. Elo := PAnt↑. Elo;{encadeia PNovo com próximo}
  PAnt↑. Elo := PNovo;               { encadeia PAnt com PNovo }
fim;    fim;
```


- O que faz o algoritmo a seguir:

Proc ????? (var Lista: pnodo);

```
var P1, P2 : pnodo;
    Dados : string;
    Resp : caracter;
    Primeiro : lógico;
início
  Primeiro := verdadeiro;
  escrever (´Novo nodo ? (S/N) ´);
  ler (Resp);
enquanto Resp <> ´N´
 faça início
       alocar (P2);     { aloca o novo nodo }
       escrever (´Dados para o nodo´: ´);
       ler (Dados);
       P2 ↑. Info := Dados;
       se Primeiro          { primeiro nodo }
       então início
              Lista := P2;  { atualiza Lista }
              P1 := P2;
              Primeiro := falso
              fim
       senão início              { outro nodo }
              P1 ↑. Elo := P2;
              P1 := P2
              fim;
       escrever (´Novo nodo ? (S/N) ´);
       ler (Resp);
       fim;
  P2 ↑.Elo := nil;     { encerra lista }
fim;
```


Proc ConstroiLLE_Fim (var Lista: pnodo);

```
var P1, P2 : pnodo;
    Dados : string;
    Resp : caracter;
    Primeiro : lógico;
início
  Primeiro := verdadeiro;
  escrever (´Novo nodo ? (S/N) ´);
  ler (Resp);
enquanto Resp <> ´N´
 faça início
       alocar (P2);     { aloca o novo nodo }
       escrever (´Dados para o nodo´: ´);
       ler (Dados);
       P2 ↑. Info := Dados;
       se Primeiro          { primeiro nodo }
       então início
              Lista := P2;  { atualiza Lista }
              P1 := P2;
              Primeiro := falso
              fim
       senão início              { outro nodo }
              P1 ↑. Elo := P2;
              P1 := P2
              fim;
       escrever (´Novo nodo ? (S/N) ´);
       ler (Resp);
       fim;
  P2 ↑.Elo := nil;     { encerra lista }
fim;
```


Proc ???? (var Lista: pnodo);

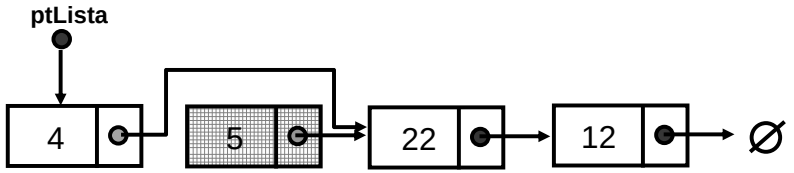
```
var P : pnodo;
    Dados : string;
    Resp : caracter;
início
  escrever (´Novo nodo ? (S/N) ´);
  ler (Resp);
  enquanto Resp <> ´N´
  faça início
       alocar (P);              { aloca o novo nodo }
       escrever (´Dados para o nodo´: ´);
       ler  (Dados);
       P ↑. Info := Dados;
       P ↑. Elo := Lista;       { encadeia com o que era o primeiro }
       Lista := P;              { este passa a  ser o primeiro }
       escrever (´Novo nodo ? (S/N) ´);
       ler (Resp);
       fim;
fim;
```


Proc ConstroiLLE_Reversa (var Lista: pnodo);

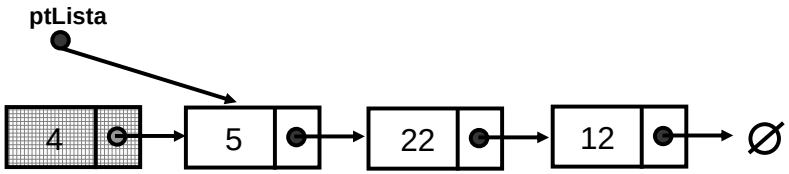
```
var P : pnodo;
    Dados : string;
    Resp : caracter;
início
  escrever (´Novo nodo ? (S/N) ´);
  ler (Resp);
  enquanto Resp <> ´N´
  faça início
       alocar (P);              { aloca o novo nodo }
       escrever (´Dados para o nodo´: ´);
       ler  (Dados);
       P ↑. Info := Dados;
       P ↑. Elo := Lista;       { encadeia com o que era o primeiro }
       Lista := P;              { este passa a  ser o primeiro }
       escrever (´Novo nodo ? (S/N) ´);
       ler (Resp);
       fim;
fim;
```

Remove()

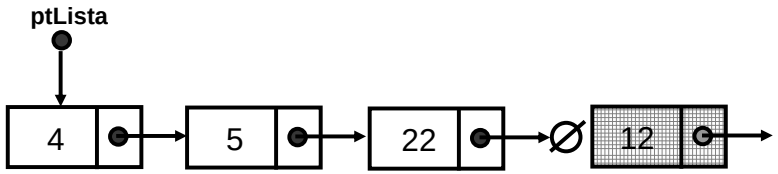
Remove()



Remove()



Remove()

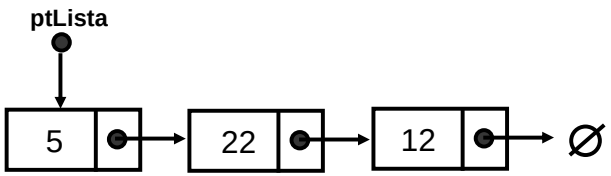


Remove()

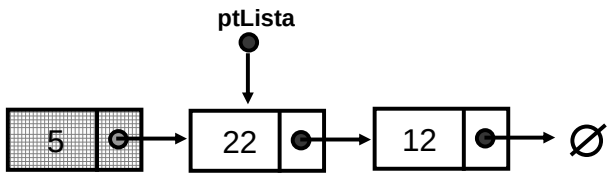
```
Func RemoverKLLE (var PtLista: TipoPtnodo;
                  K: inteiro) : lógico;
var PtAnt, PtK: TipoPtnodo;
início
   AcessarLLE (PtLista, K, PtAnt, PtK);
   se PtK ≠ nil
      então RemoverKLLE := falso { não encontrou }
   senão início
            se Ptk = PtLista
               então PtLista := PtLista ↑ . Elo
               senão  PtAnt ↑ . Elo := PtK  ↑ . Elo;
             liberar (PtK);    { libera o k-ésimo nodo }
             RemoverKLLE := true
          fim;
fim RemoverKLLE;
```


Destroi()

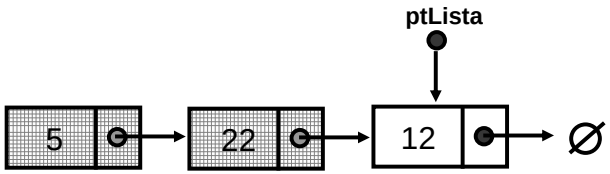
Destroi()



Destroi()



Destroi()



Destroi()

```
begin
  enquanto PtLista ≠ nulo
  faça início
      Pt ← PtLista
      PtLista ← PtLista↑.Elo
      liberar(Pt)
      fim
  liberar(PtLista)
fim
```