UFSC-CTC-INE

INE5384 - Estruturas de Dados

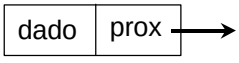
Listas Encadeadas

Prof. Ronaldo S. Mello

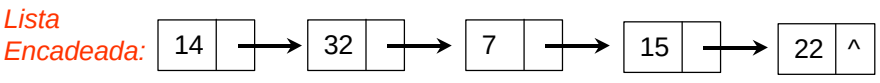
2002/2

Características

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



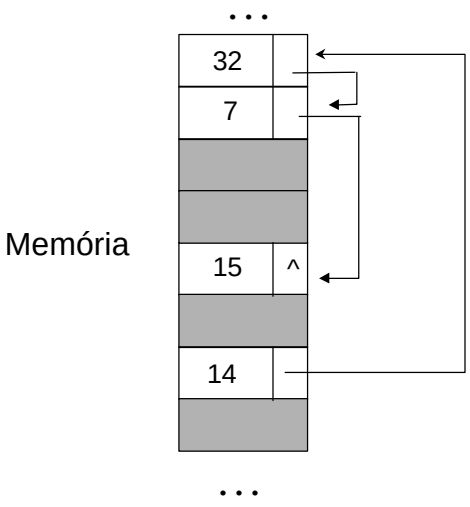
- Exemplo:



Características

- Não há limite máximo para o número de elementos na lista
  - O limite é a capacidade da memória!
- Elementos não estão necessariamente em posições contíguas da memória
  - Alocação de novos elementos em tempo de execução, conforme a lista cresce
  - Melhor aproveitamento do espaço livre da memória

Características

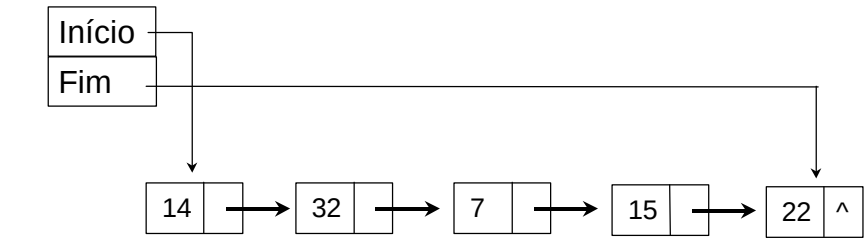


Operações sobre uma Lista Encadeada

- Supõe-se os seguintes atributos:
  - *Início* (referência a objeto da classe Elemento)
  - *Fim* (referência a objeto da classe Elemento)
- Classe Elemento
  - Atributos:
    - *Dado* (*object*)
    - *Prox* (referência a objeto da classe Elemento)
  - Métodos:
    - *ObtémDado()*
    - *ObtémProx()*

Exemplo de Objeto ListaEncadeada

Lista Encadeada:



Implementação

```
Classe ListaEncadeada
início
    início Elemento;
    fim Elemento;

    Classe Elemento
    início
        dado Object;
        prox  Elemento;

        Construtor Elemento (dado object, prox Elemento)
        início
            this.dado ← dado;
            this.prox ← prox;
        fim
    . . .
```

Implementação

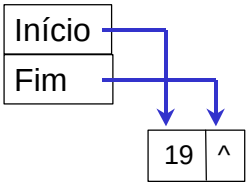
```
    . . .
        Método ObtémDado() retorna object;
        início
            retorna dado;
        fim;
        Método ObtémProx() retorna Elemento;
        início
            retorna prox;
        fim;
    . . .
    fim;
    Construtor ListaEncadeada ();
    início
        início ← null;     fim ← null;
    fim;
fim;
```

Inserção no Final

- Operação *Append*
  - Exemplo: inserir o número 19
    - Primeira ação: criar o novo elemento
    - Preciso testar lista cheia? Não
    - Preciso testar lista vazia? Sim
    - Se a lista **está** vazia, então:

Início ← NovoElemento;

Fim ← NovoElemento;

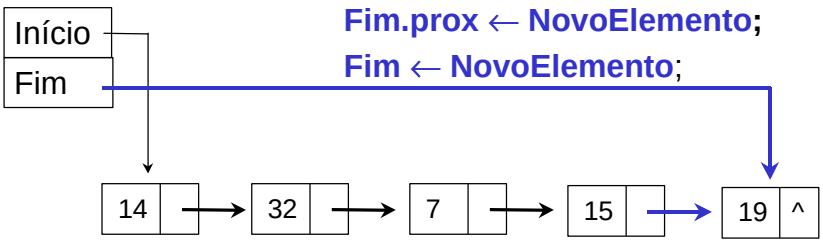


Inserção no Final

- Operação *Append*
  - Exemplo: inserir o número 19
    - Primeira ação: criar o novo elemento
    - Preciso testar lista cheia? Não
    - Preciso testar lista vazia? Sim
    - Se a lista **não está** vazia, então:

Fim.prox ← NovoElemento;

Fim ← NovoElemento;



Implementação

```
Classe ListaEncadeada
início
        Método insereNoFinal (objeto object);
        início
          elem Elemento;

          elem = NOVO Elemento (objeto, null);
          Se início = null então
          início
                início ← elem;
                fim ← elem;
           fim
          Senão início
                    fim.prox ← elem;
                    fim ← elem;
                fim;
fim;
```

Implementação

```
Classe ListaEncadeada
início
        Método insereNoFinal (objeto object);
        início
          elem Elemento;

          elem = NOVO Elemento (objeto, null);
          Se início = null então início ← elem
          Senão fim.prox ← elem;
          fim ← elem;
        fim;
```

Exercícios

- Implementar na classe *ListaEncadeada*:
  - *insereNoInicio(objeto Object)* (*prepend*)
  - *obtemPrimeiroElemento()* retorna *Object*
  - *obtemUltimoElemento()* retorna *Object*
- Implementar na classe *Elemento*:
  - *insereApos(objeto Object)* ***
  - *insereAntes(objeto Object)* ****

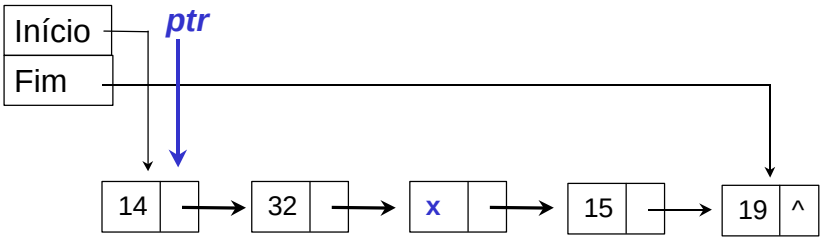
**** para entregar em aula***
***** para entregar na próxima aula***

Pesquisa na Lista por um Valor "x"

- Método que retorna verdadeiro ou falso se existe um objeto com valor "x" na lista

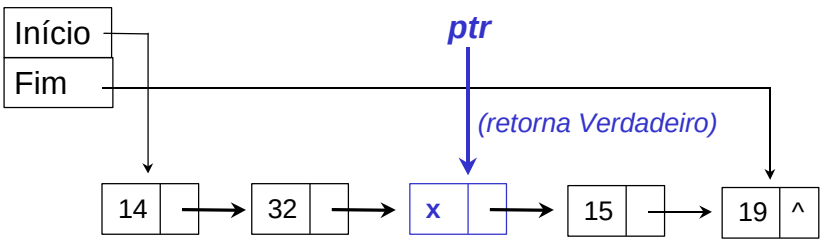
Pesquisa na Lista por um Valor "x"

- O que fazer?
  - Definir uma variável *ptr* (do tipo Elemento) para percorrer a lista (aponta inicialmente para *Início*)



Pesquisa na Lista por um Valor "x"

- Realizar a pesquisa: quais as condições?
  - Enquanto *ptr* ≠ NULL faça:
    - se *ptr*.dado = x retorna VERDADEIRO
    - *ptr* ← *ptr*.prox
  - Retorna FALSO



Implementação

```
Classe ListaEncadeada
início
   Método ExisteElemento (objeto object) retorna booleano;
   início
      ptr Elemento;

      ptr ← início;
      Enquanto ptr ≠ NULL faça
      início
         se ptr.dado = objeto então  retorna VERDADEIRO;
         ptr ← ptr.prox;
       fim;
       retorna FALSO;
   fim;
```

Exercícios

- Implementar na classe *ListaEncadeada*:
  - *buscaMaiorElemento()* retorna Objeto;*
  - *estahOrdenada()* retorna booleano; *

    (considerar ordenada uma lista vazia ou uma lista com apenas um elemento)

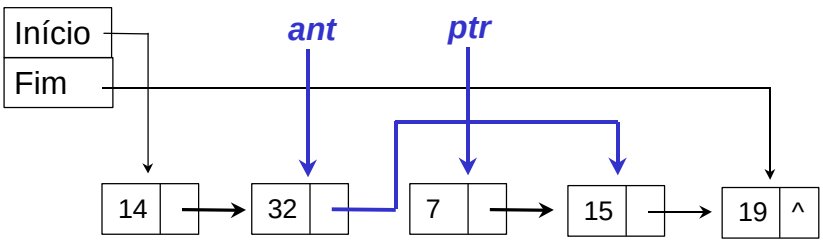
**** para entregar em aula***

Exclusão de um Elemento com Valor “x”

- Operação *Extract*
  - Exemplo: excluir o elemento 7
    - Primeira ação: **verificar se o elemento está na lista**
      - criar um apontador de Elemento ***(ptr)*** para percorrer a lista

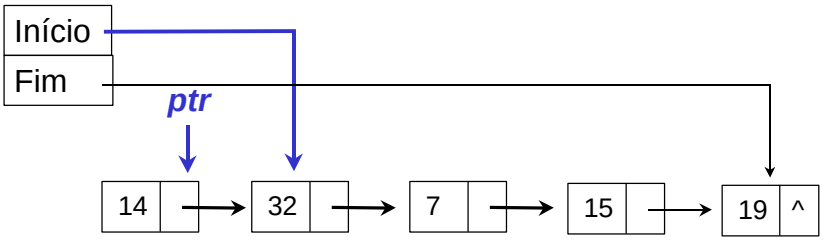
Exclusão de um Elemento com Valor “x”

- Operação *Extract*
  - Exemplo: excluir o elemento 7
    - Primeira ação: verificar se o elemento está na lista
    - Se não achou: exceção “Elemento inexistente”
    - Se achou, então:
      - **ant.prox ← ptr.prox**



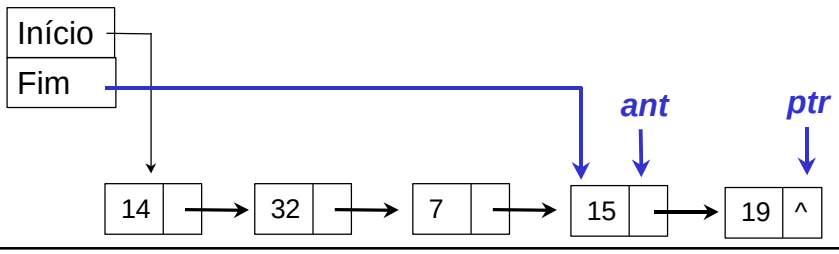
Exclusão de um Elemento com Valor “x”

- Operação *Extract*
  - Casos a considerar: se elemento é o primeiro?
    - **Início ← ptr.prox**



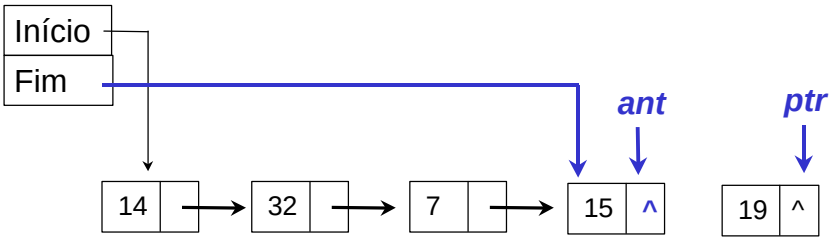
Exclusão de um Elemento com Valor “x”

- Operação *Extract*
  - Casos a considerar: se elemento é o último?
    - **fim ← ant**



Exclusão de um Elemento com Valor “x”

- Operação *Extract*
  - Casos a considerar: se elemento é o último?
    - **fim ← ant**
    - **ant.prox ← ptr.prox**



Implementação

```
Método exclui (objeto object);
início
   ptr, ant Elemento;

   ptr ← início;
   se ptr = NULL então Exceção EstruturaVazia();
   enquanto ptr ≠ NULL e ptr.dado ≠ objeto faça
   início
         ant ← ptr;
         ptr ← ptr.prox;
    fim;
    se ptr  = NULL então Exceção ObjetoInexistente();
    se ptr = início então início ← ptr.prox
    senão ant.prox ← ptr.prox;
    se ptr = fim então fim ← ant;
fim;
```

Exercícios

- Implementar na classe *ListaEncadeada*:
  - *ExcluiNoInício();* ***
  - *ExcluiNoFinal();* ***
  - *ExcluiMenor();* ****

**** para entregar em aula***

***** para entregar na próxima aula***