DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Estruturas de Dados

TADs: Listas, Filas, Pilhas

Professores: Anisio Lacerda
Wagner Meira Jr.
Washington Cunha

Módulo 2 - Sumário

- Introdução
  - Tipos Abstratos de Dados
- Listas Lineares
  - Implementação por arranjos (sequencial)
  - Implementação por apontadores (encadeada)
- Filas, Pilhas
  - Implementação por arranjos (sequencial)
  - Implementação por apontadores (encadeada)

Tipos Abstratos de Dados (TADs)

- Principais Vantagens:
  - O **Usuário** do TAD pode abstrair da implementação específica do TAD, focando na sua aplicação
  - O **Desenvolvedor** do TAD pode focar na sua funcionalidade e eficiência
  - Qualquer modificação interna na implementação do TAD fica restrita a ele, não influenciando o usuário
  - Modularização mais eficiente
  - Facilita a realização de testes

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Estruturas de Dados

Listas Lineares - Alocação Encadeada

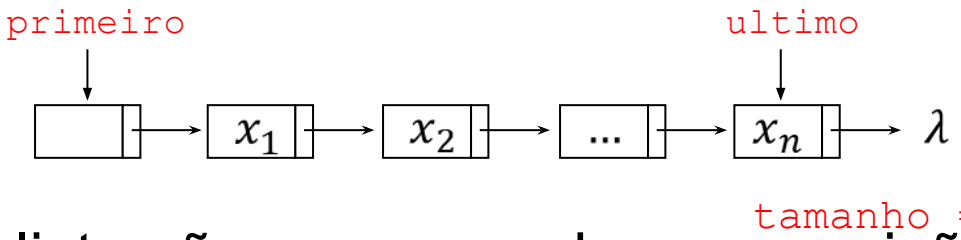
Professores: Anisio Lacerda
Wagner Meira Jr.
Washington Cunha

TAD: Lista

- Duas Implementações:
  - Sequencial (uso de arranjos, alocação estática)
  - Encadeada (uso de apontadores, alocação dinâmica)
- Operações:
  - Criar uma nova lista (construtor)
  - Métodos de Acesso (Get, Set)
  - Testar se é uma lista *vazia*
  - Inserção: no início, no final, em uma posição *p*
  - Remoção: do início, do final, de uma posição *p*
  - Pesquisar por uma chave
  - Imprimir a Lista
  - Limpar a Lista

Disclaimer:
os códigos que serão apresentados devem ser considerados como exemplos. Eles não são, necessariamente, os mais modulares ou eficientes…

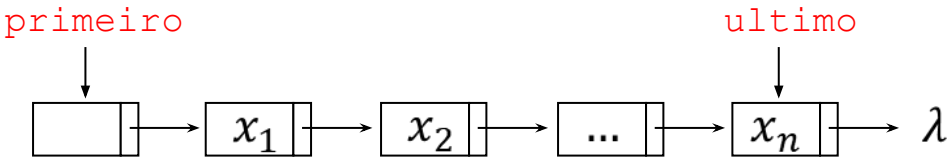
Alocação Encadeada



- Itens da lista são armazenados em posições não contíguas da memória
  - Utilização de **células** que são encadeadas usando **apontadores**
  - Alocação dinâmica, permitindo crescimento e redução de tamanho
  - **Não** permite acesso direto a qualquer item

- Inserção e Remoção
  - Não requer deslocamento de itens

Class Lista Encadeada



tamanho = n

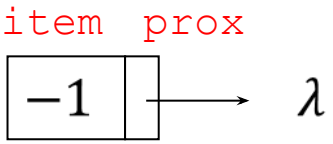
```
class ListaEncadeada : public Lista {
    public:
        ListaEncadeada();
        ~ListaEncadeada();

        TipoItem GetItem(int pos);
        void SetItem(TipoItem item, int pos);
        void InsereInicio(TipoItem item);
        void InsereFinal(TipoItem item);
        void InserePosicao(TipoItem item, int pos);
        TipoItem RemoveInicio();
        TipoItem RemoveFinal();
        TipoItem RemovePosicao(int pos);
        TipoItem Pesquisa(TipoChave c);
        void Imprime();
        void Limpa();
    private:
        TipoCelula* primeiro;
        TipoCelula* ultimo;
        TipoCelula* Posiciona(int pos, bool antes);
};
```

Class TipoCélula

- Classe para representar as células da lista
  - Campo **TipoItem item**: armazena o item
  - Campo **TipoCelula *prox**: apontador para a próxima célula
  - Possui método para inicialização (constructor)
  - Permite o acesso de atributos privados pela classe ListaEncadeada
    - *Friend class*

```
class TipoCelula
{
    public:
        TipoCelula();

    private:
        TipoItem item;
        TipoCelula *prox;

    friend class ListaEncadeada;
};
```

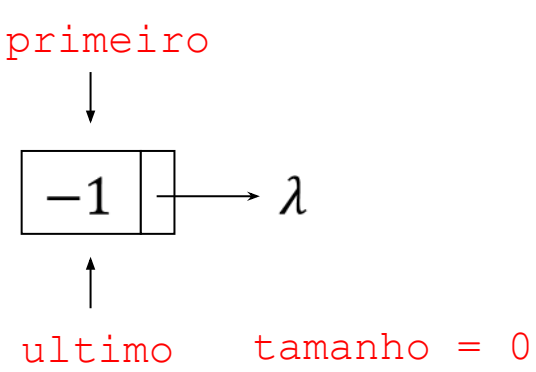
```
TipoCelula::TipoCelula()
{
    item.SetChave(-1);
    prox = NULL;
}
```



Class Lista Encadeada - Construtor

- Construtor
  - Chama o construtor da classe pai, que inicializa o atributo *tamanho* com o valor 0, e inicializa os apontadores *primeiro* e *ultimo*.
  - Uso de uma **célula cabeça**
    - Facilita as operações de inserção e remoção no início da lista
    - Primeiro elemento da lista vai estar na posição **`primeiro->prox`**

```
ListaEncadeada::ListaEncadeada() : Lista() {
    primeiro = new TipoCelula();
    ultimo = primeiro;
}
```

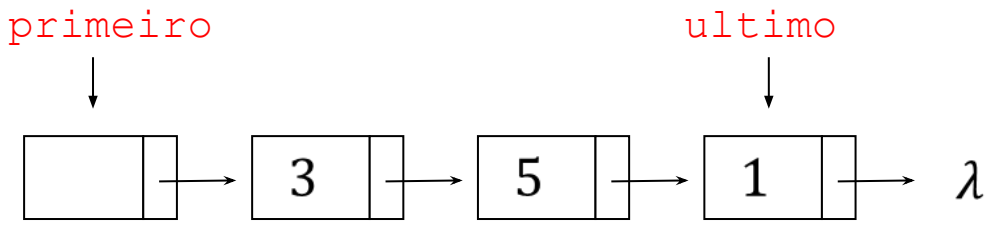



```
ListaEncadeada L;
...
```

Class Lista Encadeada - Destrutor

- Destrutor
  - Como utilizamos a alocação dinâmica, é importante implementar um destrutor para desalocar a memória adequadamente
  - Chama o método *Limpa*, que remove todas as células da lista e depois remove a célula cabeça

```
ListaEncadeada::~ListaEncadeada()
{
     Limpa();
     delete primeiro;
}
```



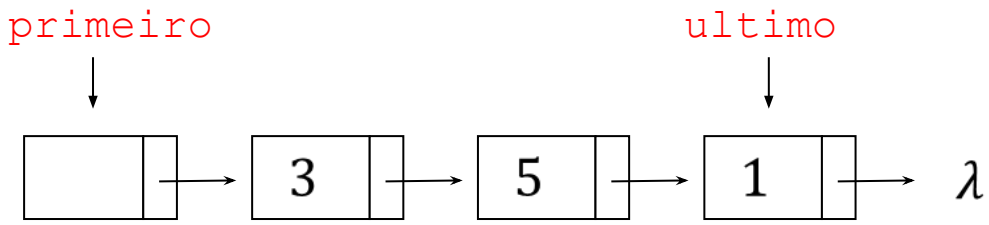
tamanho = 3

```
ListaEncadeada *L;
…
delete L;
```

Class Lista Encadeada - Destrutor

- Destrutor
  - Como utilizamos a alocação dinâmica, é importante implementar um destrutor para desalocar a memória adequadamente
  - Chama o método *Limpa*, que remove todos as células da lista e depois remove a célula cabeça

```
ListaEncadeada::~ListaEncadeada()
{
     Limpa();
     delete primeiro;
}
```



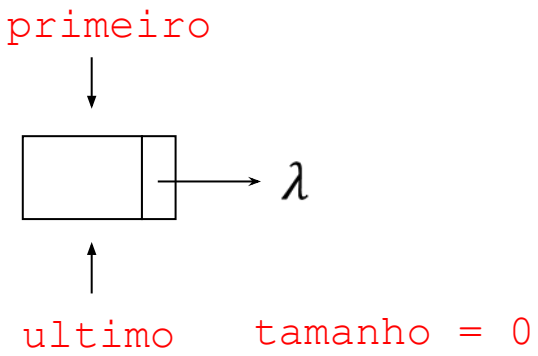
tamanho = 3

```
ListaEncadeada *L;
…
delete L;
```

Class Lista Encadeada - Destrutor

- Destrutor
  - Como utilizamos a alocação dinâmica, é importante implementar um destrutor para desalocar a memória adequadamente
  - Chama o método *Limpa*, que remove todos as células da lista e depois remove a célula cabeça

```
ListaEncadeada::~ListaEncadeada()
{
     Limpa();
     delete primeiro;
}
```



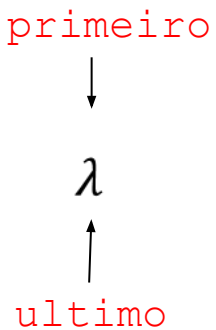

```
ListaEncadeada *L;
…
delete L;
```

Class Lista Encadeada - Destrutor

- Destrutor
  - Como utilizamos a alocação dinâmica, é importante implementar um destrutor para desalocar a memória adequadamente
  - Chama o método *Limpa*, que remove todos as células da lista e depois remove a célula cabeça

```
ListaEncadeada::~ListaEncadeada()
{
     Limpa();
     delete primeiro;
}
```

$O(n)$

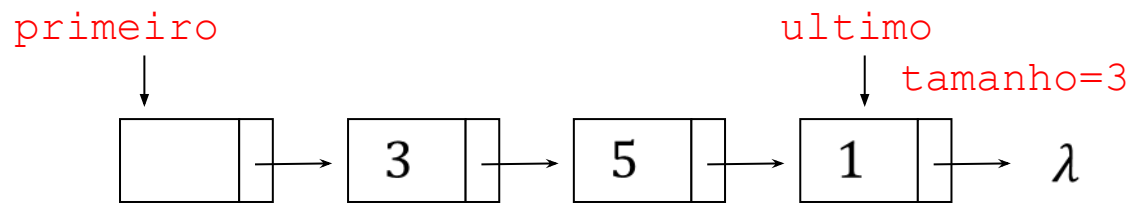



```
ListaEncadeada *L;
…
delete L;
```

Class Lista Encadeada - Posiciona

- **Função auxiliar** para posionar um apontador em uma determinada posição (célula) da lista
  - Opção de parar na célula anterior (útil para inserção e remoção)

```
TipoCelula* ListaEncadeada::Posiciona(int pos, bool antes=false){
    TipoCelula *p; int i;

    if ( (pos > tamanho) || (pos <= 0) )
        throw "ERRO: Posicao Invalida!";

    // Posiciona na célula anterior a desejada
    p = primeiro;
    for(i=1; i<pos; i++){
        p = p->prox;
    }
    // vai para a próxima
    // se antes for false
    if(!antes)
        p = p->prox;

    return p;
}
```

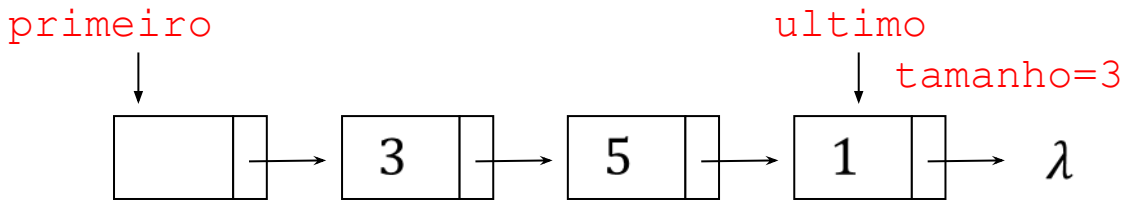



```
TipoCelula* q;
q = Posiciona(3,true);
```

Class Lista Encadeada - Posiciona

- **Função auxiliar** para posionar um apontador em em uma determinada posição (célula) da lista
  - Opção de parar na célula anterior (útil para inserção e remoção)

```
TipoCelula* ListaEncadeada::Posiciona(int pos, bool antes=false){
    TipoCelula *p; int i;

    if ( (pos > tamanho) || (pos <= 0) )
        throw "ERRO: Posicao Invalida!";

    // Posiciona na célula anterior a desejada
    p = primeiro;
    for(i=1; i<pos; i++){
        p = p->prox;
    }
    // vai para a próxima
    // se antes for false
    if(!antes)
        p = p->prox;

    return p;
}
```

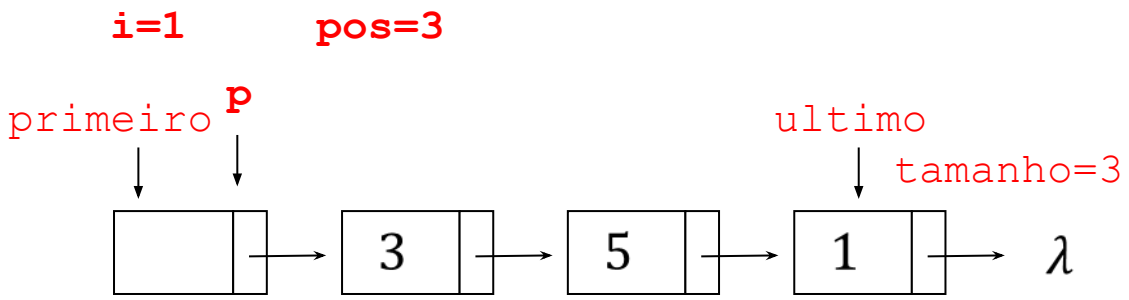



```
TipoCelula* q;
q = Posiciona(3,true);
```

Class Lista Encadeada - Posiciona

- **Função auxiliar** para posionar um apontador em em uma determinada posição (célula) da lista
  - Opção de parar na célula anterior (útil para inserção e remoção)

```
TipoCelula* ListaEncadeada::Posiciona(int pos, bool antes=false){
    TipoCelula *p; int i;

    if ( (pos > tamanho) || (pos <= 0) )
        throw "ERRO: Posicao Invalida!";

    // Posiciona na célula anterior a desejada
    p = primeiro;
    for(i=1; i<pos; i++){
        p = p->prox;
    }
    // vai para a próxima
    // se antes for false
    if(!antes)
        p = p->prox;

    return p;
}
```

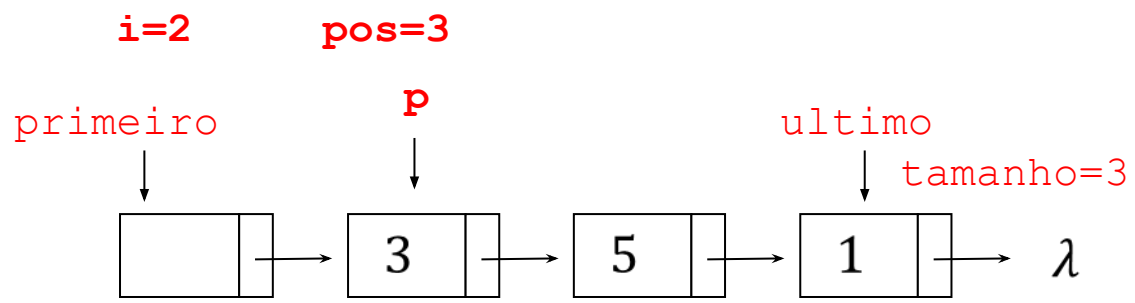



```
TipoCelula* q;
q = Posiciona(3,true);
```

Class Lista Encadeada - Posiciona

- **Função auxiliar** para posionar um apontador em em uma determinada posição (célula) da lista
  - Opção de parar na célula anterior (útil para inserção e remoção)

```
TipoCelula* ListaEncadeada::Posiciona(int pos, bool antes=false){
    TipoCelula *p; int i;

    if ( (pos > tamanho) || (pos <= 0) )
        throw "ERRO: Posicao Invalida!";

    // Posiciona na célula anterior a desejada
    p = primeiro;
    for(i=1; i<pos; i++){
        p = p->prox;
    }
    // vai para a próxima
    // se antes for false
    if(!antes)
        p = p->prox;

    return p;
}
```

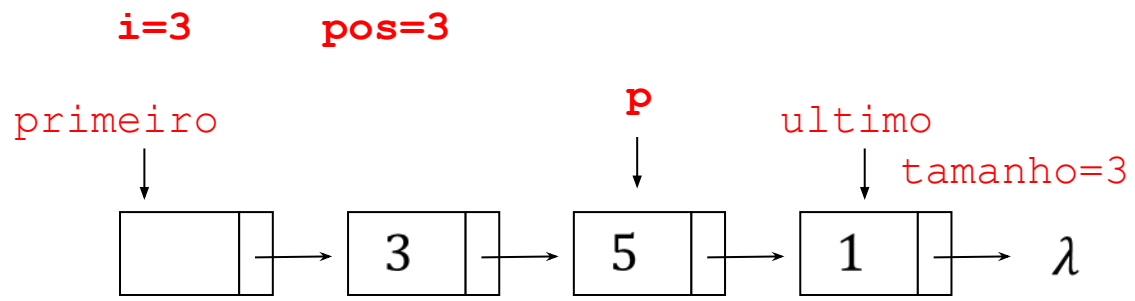



```
TipoCelula* q;
q = Posiciona(3,true);
```

Class Lista Encadeada - Posiciona

- **Função auxiliar** para posionar um apontador em em uma determinada posição (célula) da lista
  - Opção de parar na célula anterior (útil para inserção e remoção)

```
TipoCelula* ListaEncadeada::Posiciona(int pos, bool antes=false){
    TipoCelula *p; int i;

    if ( (pos > tamanho) || (pos <= 0) )
        throw "ERRO: Posicao Invalida!";

    // Posiciona na célula anterior a desejada
    p = primeiro;
    for(i=1; i<pos; i++){
        p = p->prox;
    }
    // vai para a próxima
    // se antes for false
    if(!antes)
        p = p->prox;

    return p;
}
```

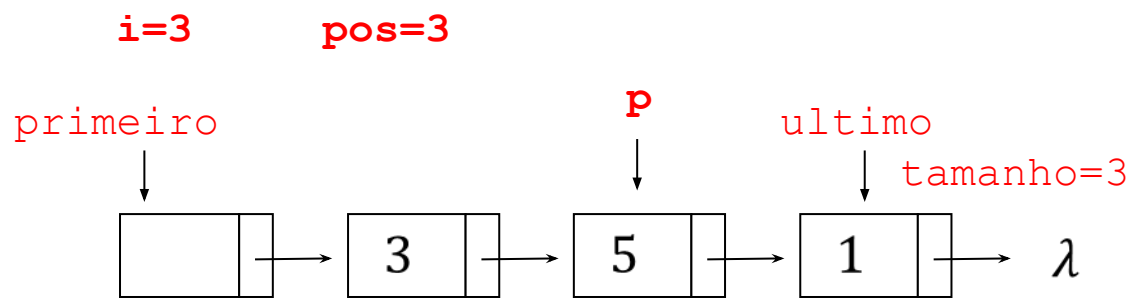



```
TipoCelula* q;
q = Posiciona(3,true);
```

Class Lista Encadeada - Posiciona

- **Função auxiliar** para posionar um apontador em em uma determinada posição (célula) da lista
  - Opção de parar na célula anterior (útil para inserção e remoção)

```
TipoCelula* ListaEncadeada::Posiciona(int pos, bool antes=false){
    TipoCelula *p; int i;

    if ( (pos > tamanho) || (pos <= 0) )
        throw "ERRO: Posicao Invalida!";

    // Posiciona na célula anterior a desejada
    p = primeiro;
    for(i=1; i<pos; i++){
        p = p->prox;
    }
    // vai para a próxima
    // se antes for false
    if(!antes)
        p = p->prox;

    return p;
}
```

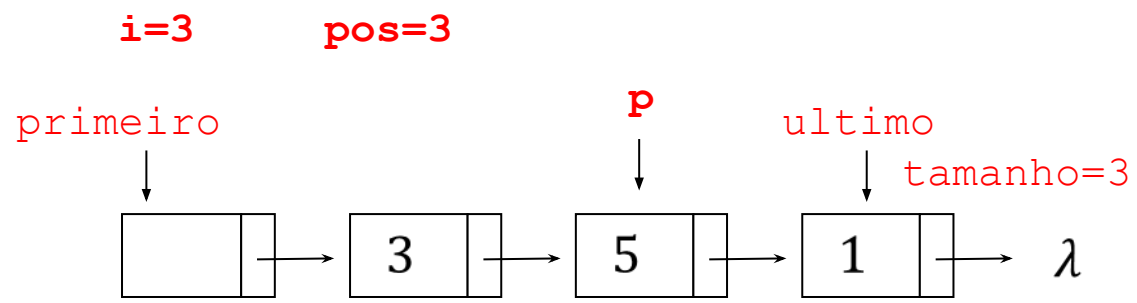



```
TipoCelula* q;
q = Posiciona(3,true);
```

Class Lista Encadeada - Posiciona

- **Função auxiliar** para posionar um apontador em em uma determinada posição (célula) da lista
  - Opção de parar na célula anterior (útil para inserção e remoção)

```
TipoCelula* ListaEncadeada::Posiciona(int pos, bool antes=false){
    TipoCelula *p; int i;

    if ( (pos > tamanho) || (pos <= 0) )
        throw "ERRO: Posicao Invalida!";

    // Posiciona na célula anterior a desejada
    p = primeiro;
    for(i=1; i<pos; i++){
        p = p->prox;
    }
    // vai para a próxima
    // se antes for false
    if(!antes)
        p = p->prox;

    return p;
}
```




```
TipoCelula* q;
q = Posiciona(3,true);
```

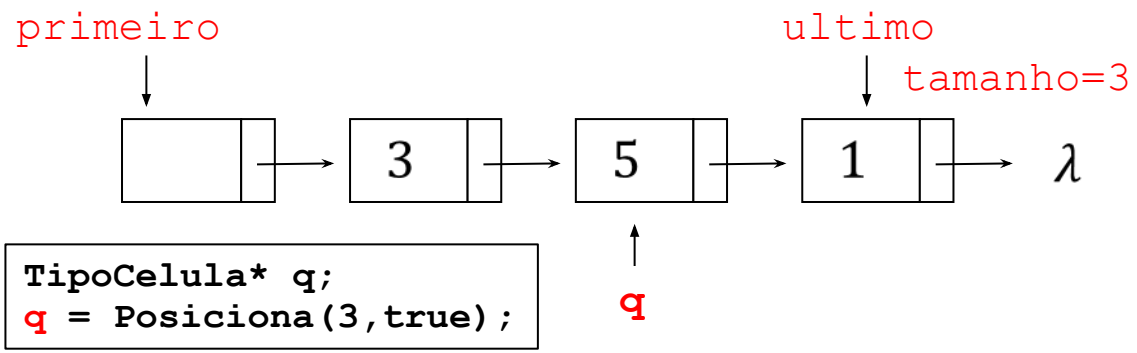
Class Lista Encadeada - Posiciona

- **Função auxiliar** para posionar um apontador em em uma determinada posição (célula) da lista
  - Opção de parar na célula anterior (útil para inserção e remoção)

```
TipoCelula* ListaEncadeada::Posiciona(int pos, bool antes=false){
    TipoCelula *p; int i;

    if ( (pos > tamanho) || (pos <= 0) )
        throw "ERRO: Posicao Invalida!";

    // Posiciona na célula anterior a desejada
    p = primeiro;
    for(i=1; i<pos; i++){
        p = p->prox;
    }
    // vai para a próxima
    // se antes for false
    if(!antes)
        p = p->prox;

    return p;
}
```

Melhor Caso $O(1)$

Pior Caso $O(n)$

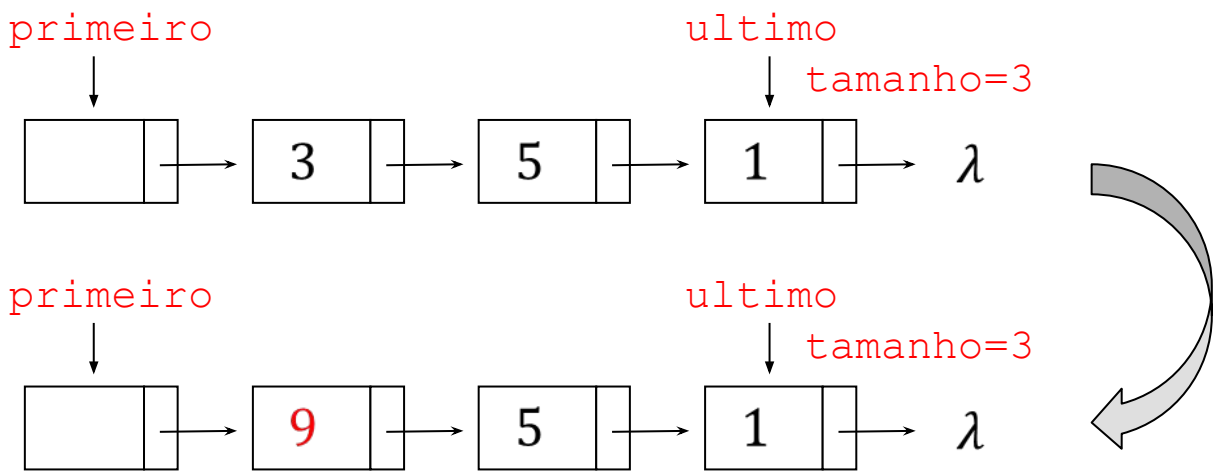


Class Lista Encadeada – Get & Set

```
TipoItem ListaEncadeada::GetItem(int pos){
    TipoCelula *p;

    p = Posiciona(pos);
    return p->item;
}

void ListaEncadeada::SetItem(TipoItem item, int pos){
    TipoCelula *p;

    p = Posiciona(pos);
    p->item = item;
}
```

Melhor Caso $O(1)$

Pior Caso $O(n)$




```
ListaEncadeada L;
TipoItem x;
…
x.SetChave(9)
L.SetItem(x,1);
x = L.GetItem(3)
x.Imprime();
```

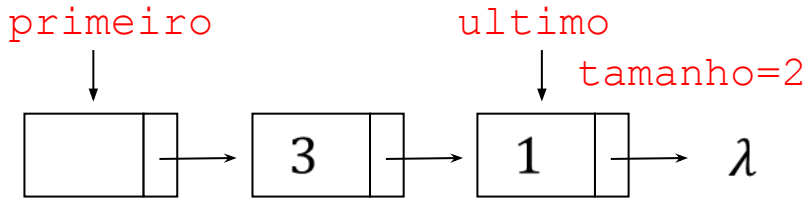
1

Class Lista Encadeada - Inserção

- Inserção pode ser feita no início, no final, ou em uma posição *p* qualquer
- Deve-se posicionar um apontador auxiliar **antes** da posição a ser inserida
- Uma nova célula é alocada dinamicamente e ligada a lista através da manipulação de apontadores.
- Se estiver inserindo na última posição, deve-se atualizar o apontador *ultimo*

Class Lista Encadeada - Inserção

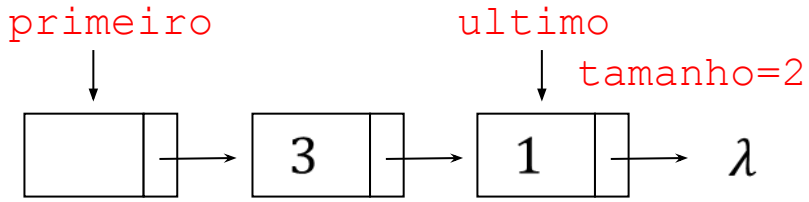
```
void ListaEncadeada::InsereInicio(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = primeiro->prox;
    primeiro->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereInicio(x)
```

Class Lista Encadeada - Inserção

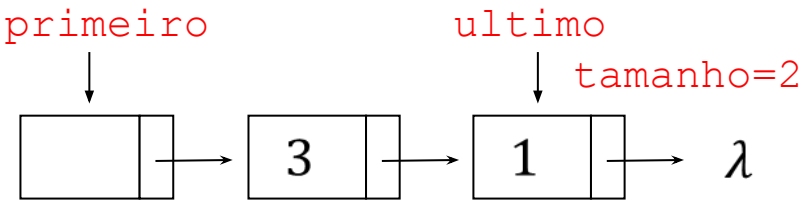
```
void ListaEncadeada::InsereInicio(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = primeiro->prox;
    primeiro->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```



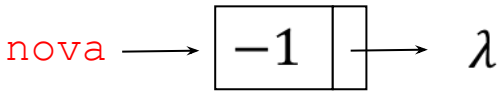

```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereInicio(x)
```

Class Lista Encadeada - Inserção

```
void ListaEncadeada::InsereInicio(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = primeiro->prox;
    primeiro->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```

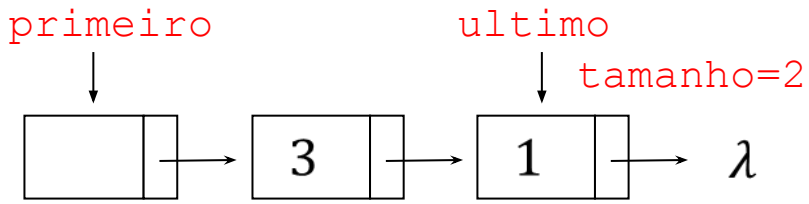



```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereInicio(x)
```



Class Lista Encadeada - Inserção

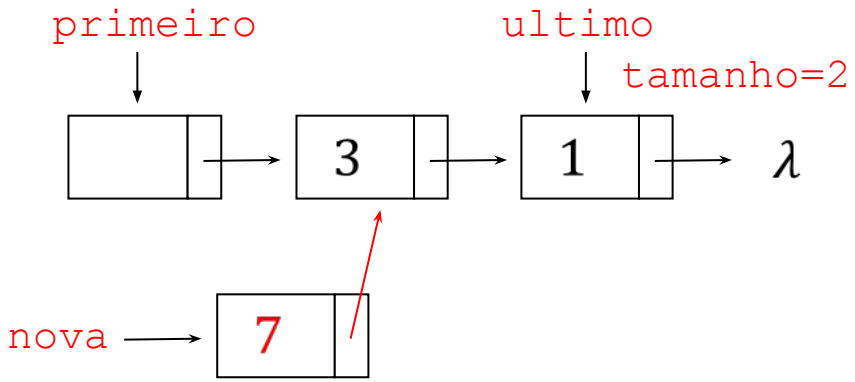
```
void ListaEncadeada::InsereInicio(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = primeiro->prox;
    primeiro->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereInicio(x)
```

Class Lista Encadeada - Inserção

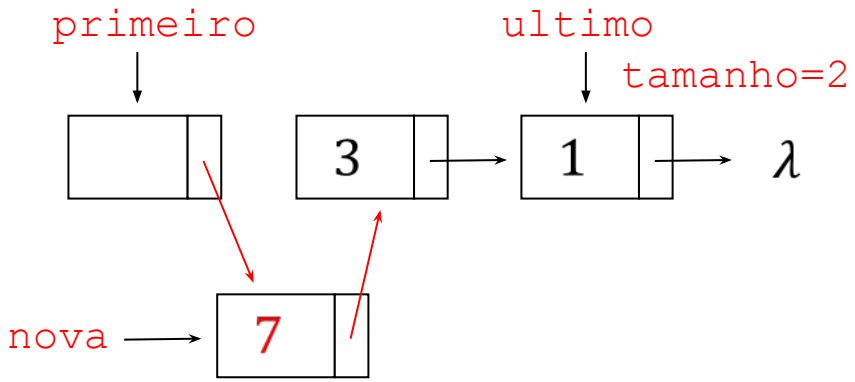
```
void ListaEncadeada::InsereInicio(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = primeiro->prox;
    primeiro->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereInicio(x)
```

Class Lista Encadeada - Inserção

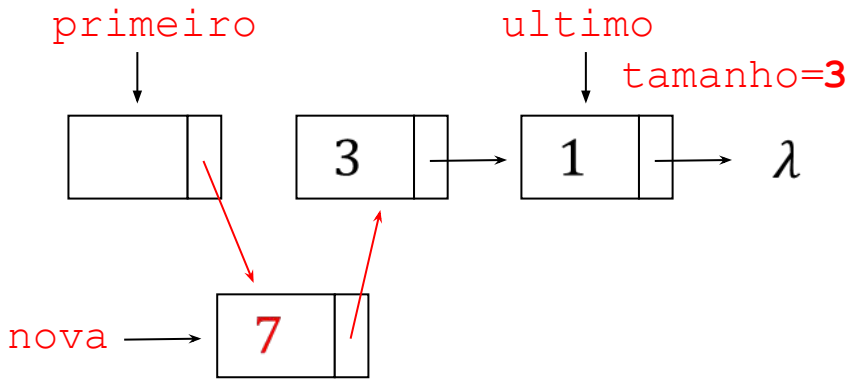
```
void ListaEncadeada::InsereInicio(TipoItem item)
{
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     nova->prox = primeiro->prox;
     primeiro->prox = nova;
     tamanho++;

     if(nova->prox == NULL)
         ultimo = nova;
};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereInicio(x)
```

Class Lista Encadeada - Inserção

```
void ListaEncadeada::InsereInicio(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = primeiro->prox;
    primeiro->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```

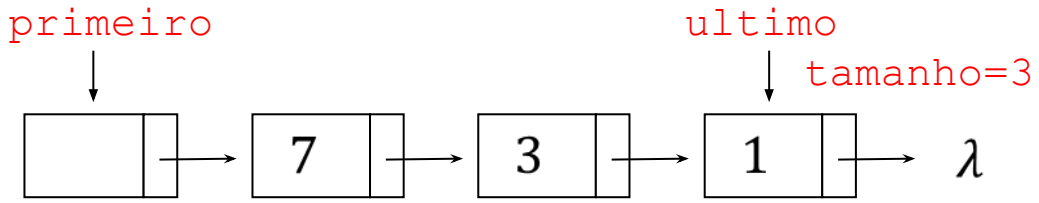



```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereInicio(x)
```

Class Lista Encadeada - Inserção

```
void ListaEncadeada::InsereInicio(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = primeiro->prox;
    primeiro->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```

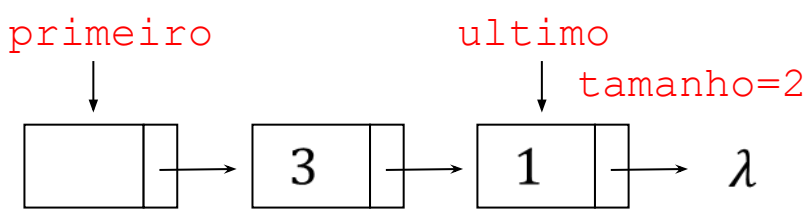
$O(1)$




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereInicio(x)
```

Class Lista Encadeada - Inserção

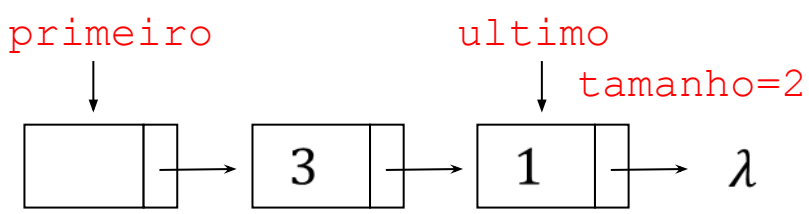
```
void ListaEncadeada::InsereFinal(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    ultimo->prox = nova;
    ultimo = nova;
    tamanho++;

};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereFinal(x)
```

Class Lista Encadeada - Inserção

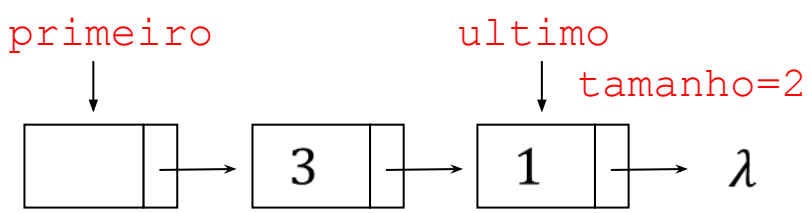
```
void ListaEncadeada::InsereFinal(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    ultimo->prox = nova;
    ultimo = nova;
    tamanho++;

};
```



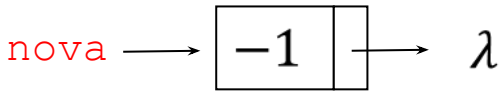

```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereFinal(x)
```

Class Lista Encadeada - Inserção

```
void ListaEncadeada::InsereFinal(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    ultimo->prox = nova;
    ultimo = nova;
    tamanho++;

};
```

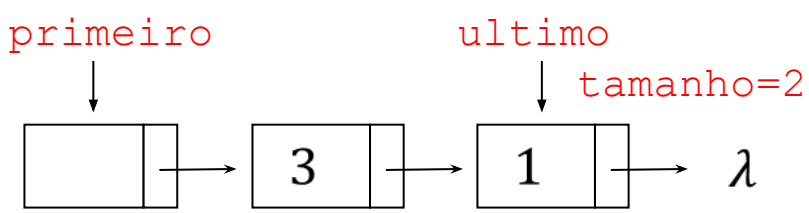



```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereFinal(x)
```



Class Lista Encadeada - Inserção

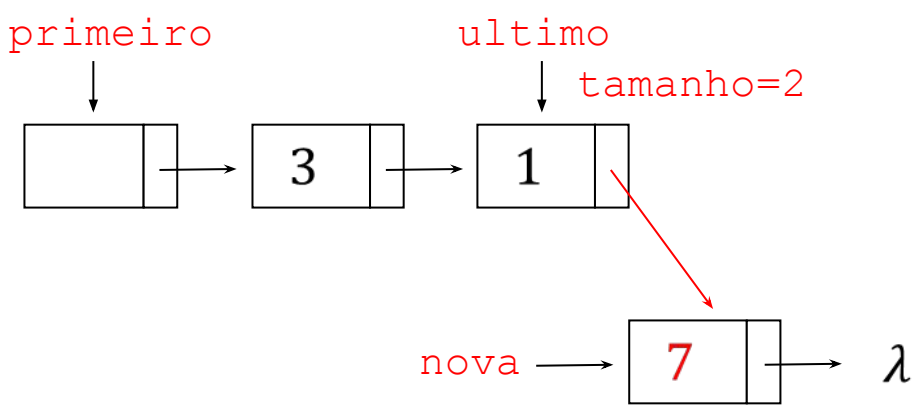
```
void ListaEncadeada::InsereFinal(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    ultimo->prox = nova;
    ultimo = nova;
    tamanho++;

};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereFinal(x)
```

Class Lista Encadeada - Inserção

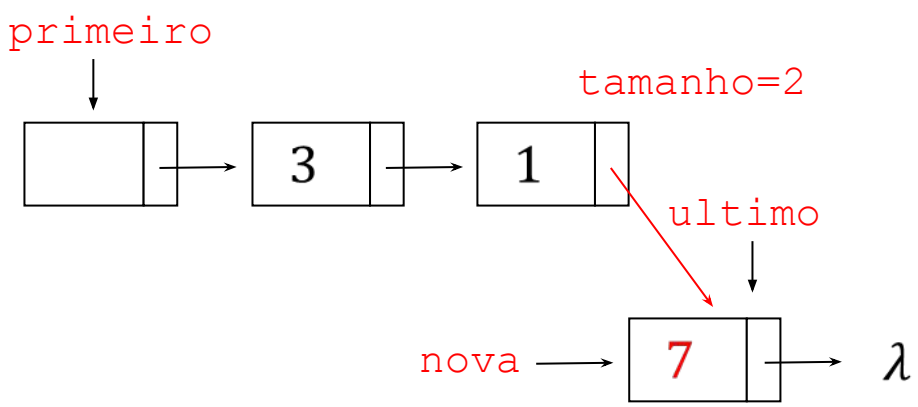
```
void ListaEncadeada::InsereFinal(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    ultimo->prox = nova;
    ultimo = nova;
    tamanho++;

};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereFinal(x)
```

Class Lista Encadeada - Inserção

```
void ListaEncadeada::InsereFinal(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    ultimo->prox = nova;
    ultimo = nova;
    tamanho++;

};
```

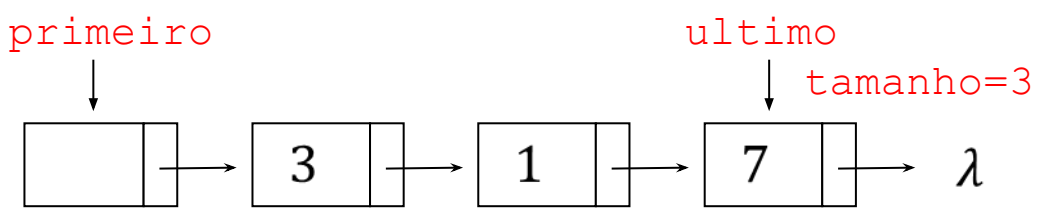



```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereFinal(x)
```

Class Lista Encadeada - Inserção

```
void ListaEncadeada::InsereFinal(TipoItem item)
{
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    ultimo->prox = nova;
    ultimo = nova;
    tamanho++;

};
```

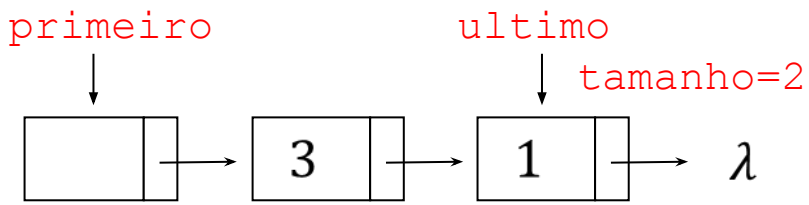
$O(1)$




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InsereFinal(x)
```

Class Lista Encadeada - Inserção

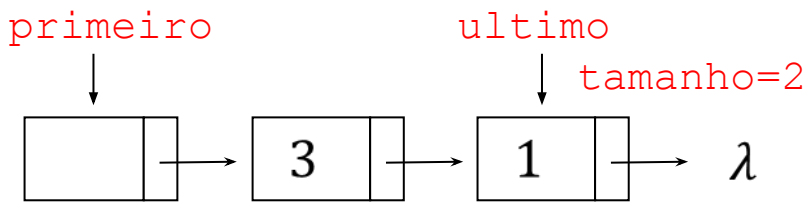
```
void ListaEncadeada::InserePosicao(TipoItem item, int pos) {
    TipoCelula *p, *nova;

    p = Posiciona(pos,true); // posiciona na célula anterior

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = p->prox;
    p->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InserePosicao(x,2)
```

Class Lista Encadeada - Inserção

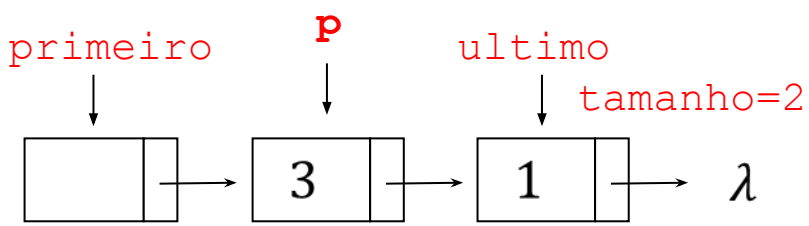
```
void ListaEncadeada::InserePosicao(TipoItem item, int pos) {
    TipoCelula *p, *nova;

    p = Posiciona(pos,true); // posiciona na célula anterior

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = p->prox;
    p->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InserePosicao(x,2)
```

Class Lista Encadeada - Inserção

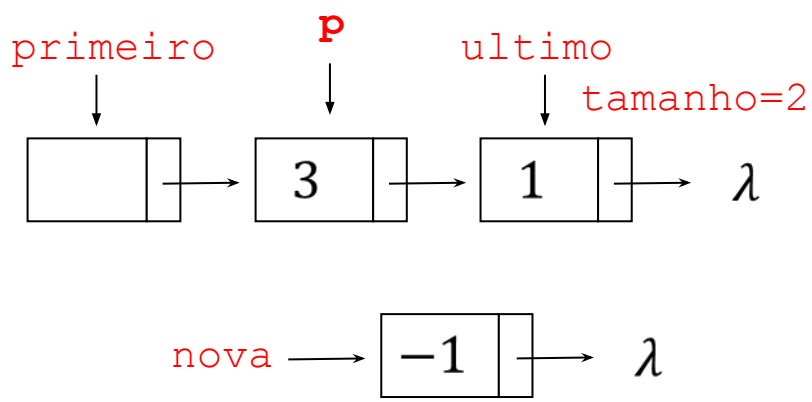
```
void ListaEncadeada::InserePosicao(TipoItem item, int pos) {
    TipoCelula *p, *nova;

    p = Posiciona(pos,true); // posiciona na célula anterior

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = p->prox;
    p->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InserePosicao(x,2)
```

Class Lista Encadeada - Inserção

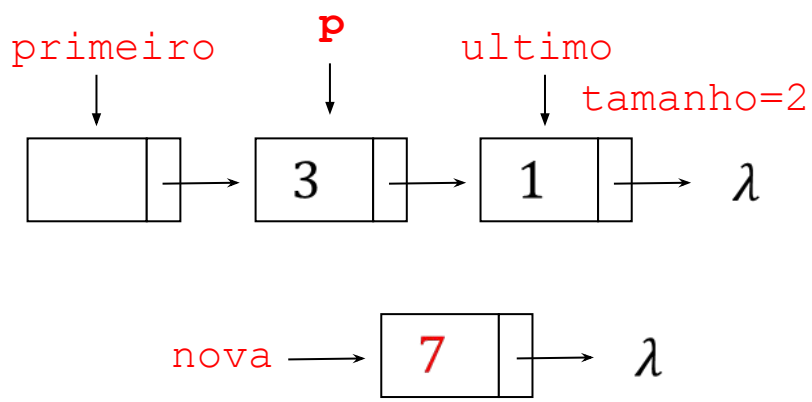
```
void ListaEncadeada::InserePosicao(TipoItem item, int pos) {
    TipoCelula *p, *nova;

    p = Posiciona(pos,true); // posiciona na célula anterior

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = p->prox;
    p->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InserePosicao(x,2)
```

Class Lista Encadeada - Inserção

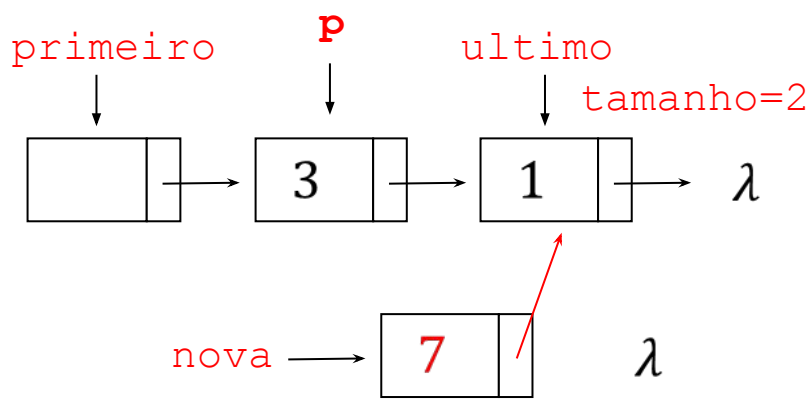
```
void ListaEncadeada::InserePosicao(TipoItem item, int pos) {
    TipoCelula *p, *nova;

    p = Posiciona(pos,true); // posiciona na célula anterior

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = p->prox;
    p->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InserePosicao(x,2)
```

Class Lista Encadeada - Inserção

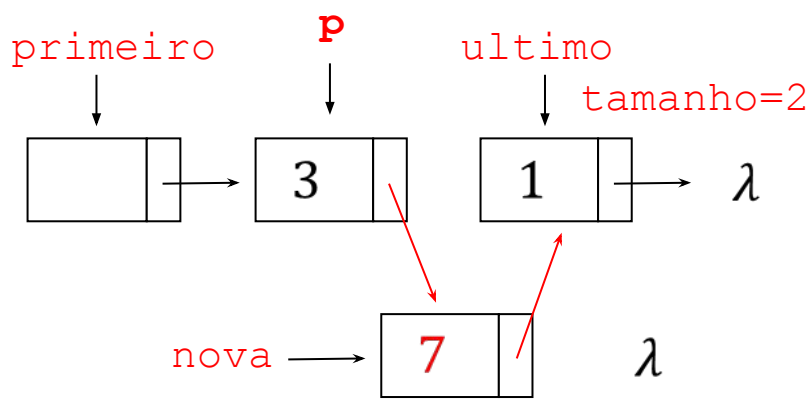
```
void ListaEncadeada::InserePosicao(TipoItem item, int pos) {
    TipoCelula *p, *nova;

    p = Posiciona(pos,true); // posiciona na célula anterior

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = p->prox;
    p->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InserePosicao(x,2)
```

Class Lista Encadeada - Inserção

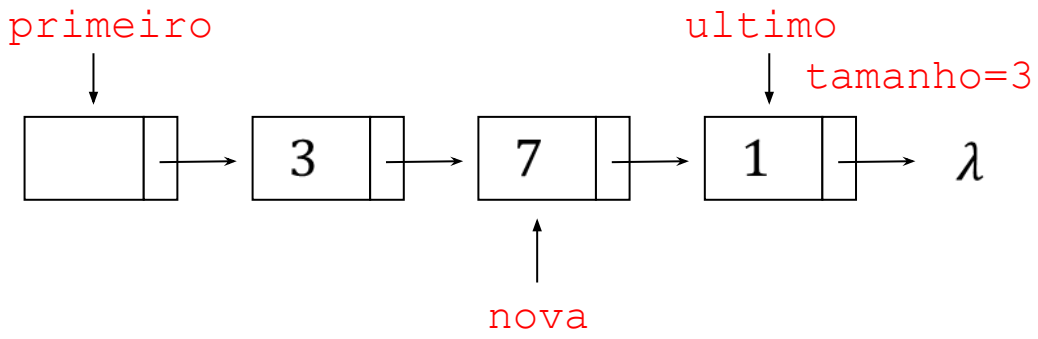
```
void ListaEncadeada::InserePosicao(TipoItem item, int pos) {
    TipoCelula *p, *nova;

    p = Posiciona(pos,true); // posiciona na célula anterior

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = p->prox;
    p->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InserePosicao(x,2)
```

Class Lista Encadeada - Inserção

```
void ListaEncadeada::InserePosicao(TipoItem item, int pos) {
    TipoCelula *p, *nova;

    p = Posiciona(pos,true); // posiciona na célula anterior

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = p->prox;
    p->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```




```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InserePosicao(x,2)
```

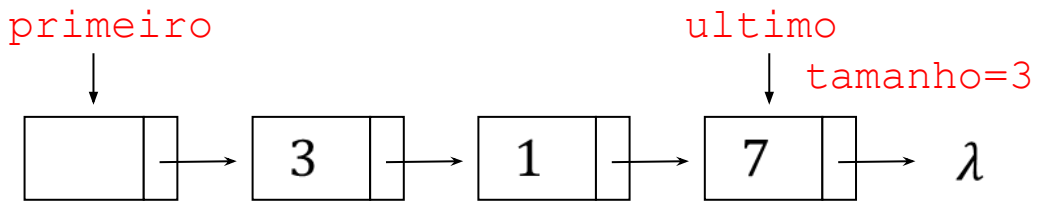
Class Lista Encadeada - Inserção

```
void ListaEncadeada::InserePosicao(TipoItem item, int pos) {
    TipoCelula *p, *nova;

    p = Posiciona(pos,true); // posiciona na célula anterior

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = p->prox;
    p->prox = nova;
    tamanho++;

    if(nova->prox == NULL)
        ultimo = nova;
};
```

Melhor Caso $O(1)$

Pior Caso $O(n)$

```
ListaEncadeada L;
TipoItem x;

x.SetChave(7)
L.InserePosicao(x,2)
```

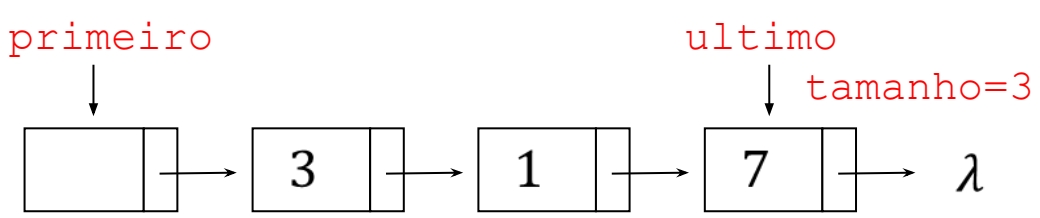


Class Lista Encadeada - Remoção

- Da mesma forma, a remoção pode ser feita no início, no final, ou em uma posição *p* qualquer
- Deve-se posicionar um apontador auxiliar **antes** da posição a ser removida
- Se estiver removendo na última posição, deve-se atualizar o apontador *ultimo*
- Deve-se verificar se há elementos e se a posição de remoção é válida
  - Gera uma exceção que pode ser tratada por quem chamou o método.
- O elemento removido é retornado pelo método e a célula desalocada da memória

Class Lista Encadeada - Remoção

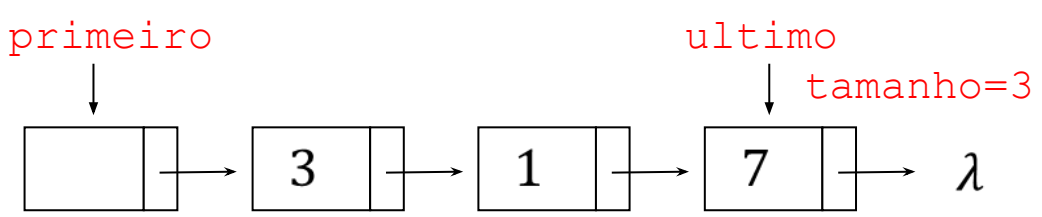
```
TipoItem ListaEncadeada::RemoveInicio() {;
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    primeiro->prox = p->prox;
    tamanho--;
    if(primeiro->prox == NULL)
        ultimo = primeiro;
    aux = p->item;
    delete p;

    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveInicio();
x.Imprime();
```

Class Lista Encadeada - Remoção

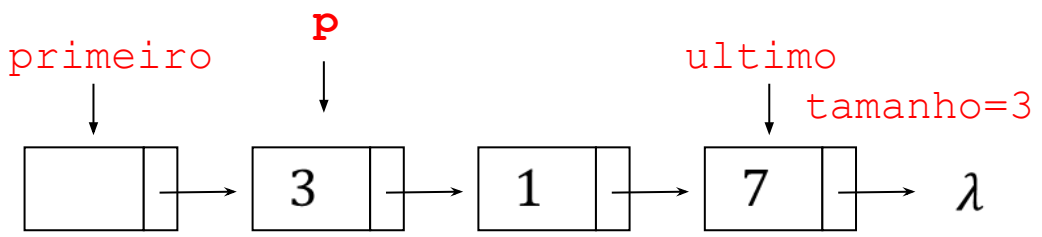
```
TipoItem ListaEncadeada::RemoveInicio() {;
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    primeiro->prox = p->prox;
    tamanho--;
    if(primeiro->prox == NULL)
        ultimo = primeiro;
    aux = p->item;
    delete p;

    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveInicio();
x.Imprime();
```

Class Lista Encadeada - Remoção

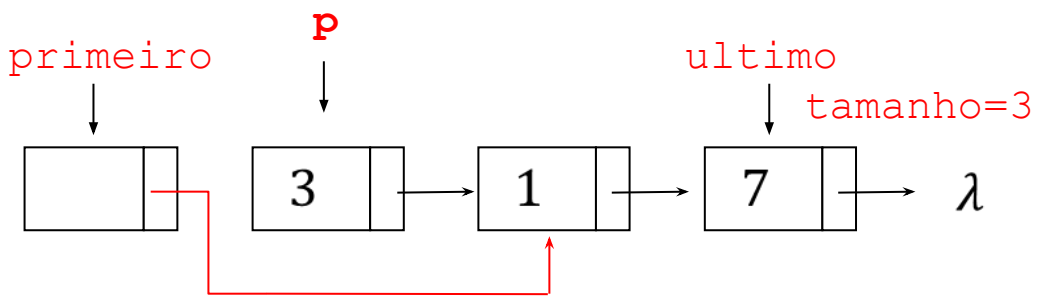
```
TipoItem ListaEncadeada::RemoveInicio() {;
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    primeiro->prox = p->prox;
    tamanho--;
    if(primeiro->prox == NULL)
        ultimo = primeiro;
    aux = p->item;
    delete p;

    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveInicio();
x.Imprime();
```

Class Lista Encadeada - Remoção

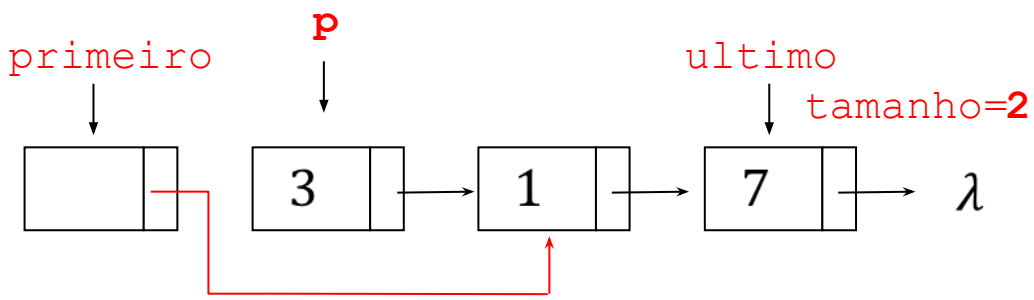
```
TipoItem ListaEncadeada::RemoveInicio() {;
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    primeiro->prox = p->prox;
    tamanho--;
    if(primeiro->prox == NULL)
        ultimo = primeiro;
    aux = p->item;
    delete p;

    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveInicio();
x.Imprime();
```

Class Lista Encadeada - Remoção

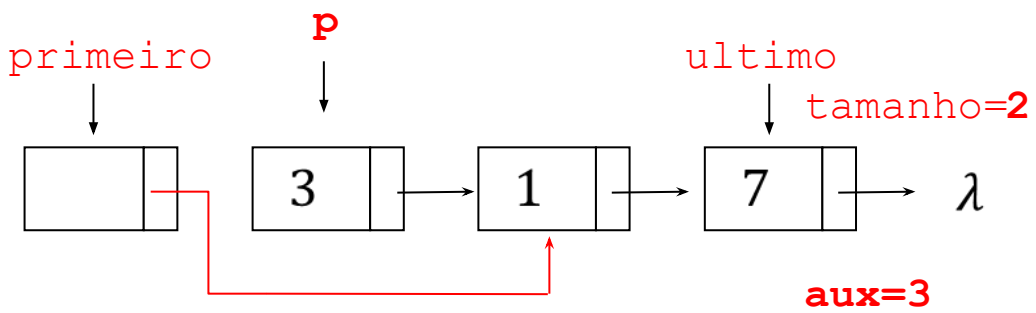
```
TipoItem ListaEncadeada::RemoveInicio() {;
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    primeiro->prox = p->prox;
    tamanho--;
    if(primeiro->prox == NULL)
        ultimo = primeiro;
    aux = p->item;
    delete p;

    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveInicio();
x.Imprime();
```

Class Lista Encadeada - Remoção

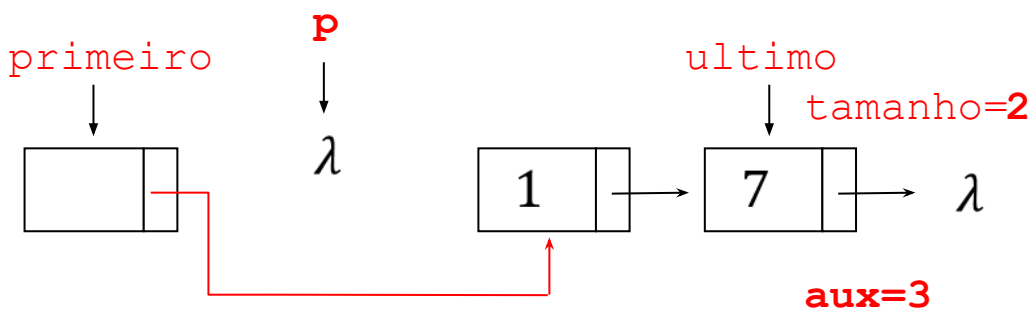
```
TipoItem ListaEncadeada::RemoveInicio() {;
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    primeiro->prox = p->prox;
    tamanho--;
    if(primeiro->prox == NULL)
        ultimo = primeiro;
    aux = p->item;
    delete p;

    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveInicio();
x.Imprime();
```

Class Lista Encadeada - Remoção

```
TipoItem ListaEncadeada::RemoveInicio() {;
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    primeiro->prox = p->prox;
    tamanho--;
    if(primeiro->prox == NULL)
        ultimo = primeiro;
    aux = p->item;
    delete p;

    return aux;
}
```

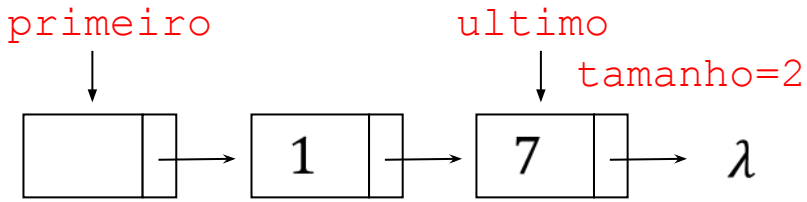



```
ListaEncadeada L;
TipoItem x;

x=L.RemoveInicio();
x.Imprime();
```

Class Lista Encadeada - Remoção

```
TipoItem ListaEncadeada::RemoveInicio() {;
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    primeiro->prox = p->prox;
    tamanho--;
    if(primeiro->prox == NULL)
        ultimo = primeiro;
    aux = p->item;
    delete p;

    return aux;
}
```

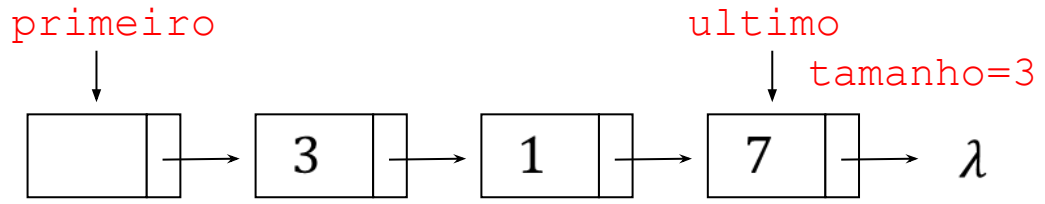
$O(1)$




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveInicio();
x.Imprime();
```

Class Lista Encadeada - Remoção

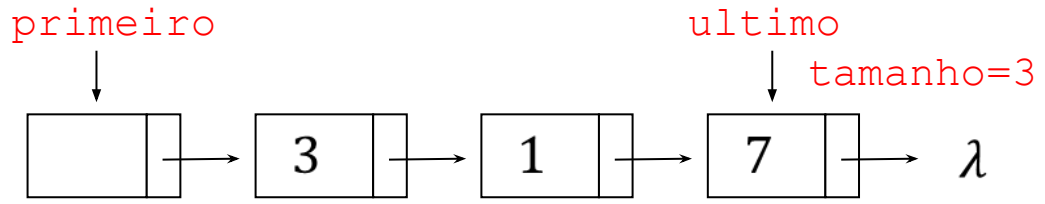
```
TipoItem ListaEncadeada::RemoveFinal() {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à última
    p = Posiciona(tamanho, true);

    p->prox = NULL;
    tamanho--;
    aux = ultimo->item;
    delete ultimo;
    ultimo = p;

    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveFinal();
x.Imprime();
```

Class Lista Encadeada - Remoção

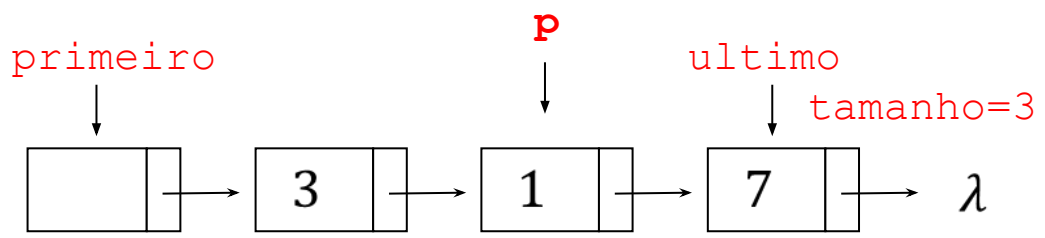
```
TipoItem ListaEncadeada::RemoveFinal() {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à última
    p = Posiciona(tamanho, true);

    p->prox = NULL;
    tamanho--;
    aux = ultimo->item;
    delete ultimo;
    ultimo = p;

    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveFinal();
x.Imprime();
```

Class Lista Encadeada - Remoção

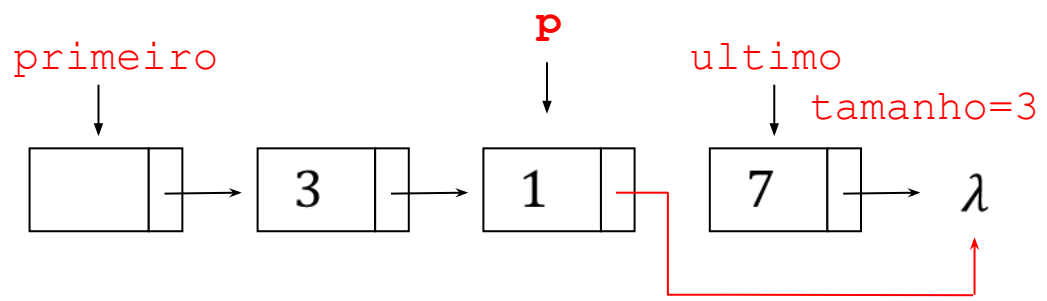
```
TipoItem ListaEncadeada::RemoveFinal() {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à última
    p = Posiciona(tamanho, true);

    p->prox = NULL;
    tamanho--;
    aux = ultimo->item;
    delete ultimo;
    ultimo = p;

    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveFinal();
x.Imprime();
```

Class Lista Encadeada - Remoção

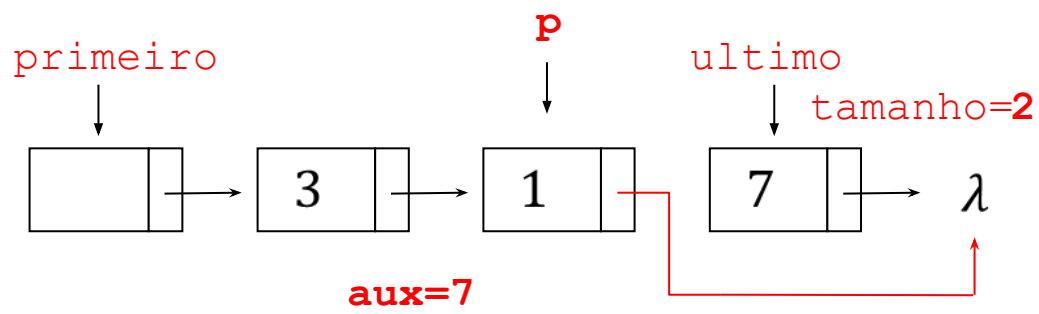
```
TipoItem ListaEncadeada::RemoveFinal() {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à última
    p = Posiciona(tamanho, true);

    p->prox = NULL;
    tamanho--;
    aux = ultimo->item;
    delete ultimo;
    ultimo = p;

    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveFinal();
x.Imprime();
```

Class Lista Encadeada - Remoção

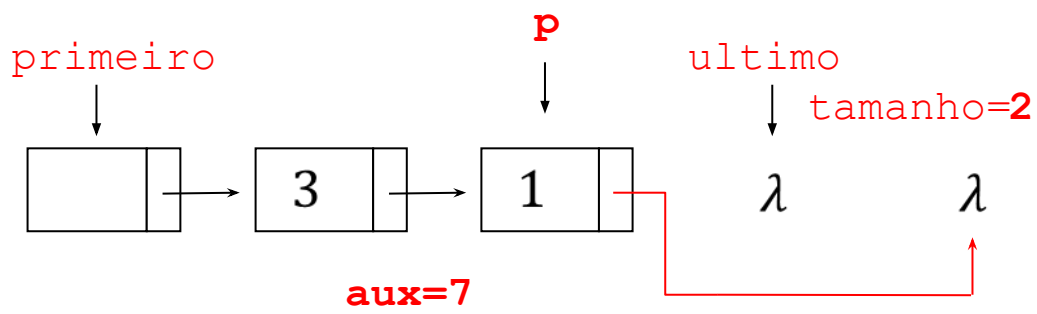
```
TipoItem ListaEncadeada::RemoveFinal() {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à última
    p = Posiciona(tamanho, true);

    p->prox = NULL;
    tamanho--;
    aux = ultimo->item;
    delete ultimo;
    ultimo = p;

    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveFinal();
x.Imprime();
```

Class Lista Encadeada - Remoção

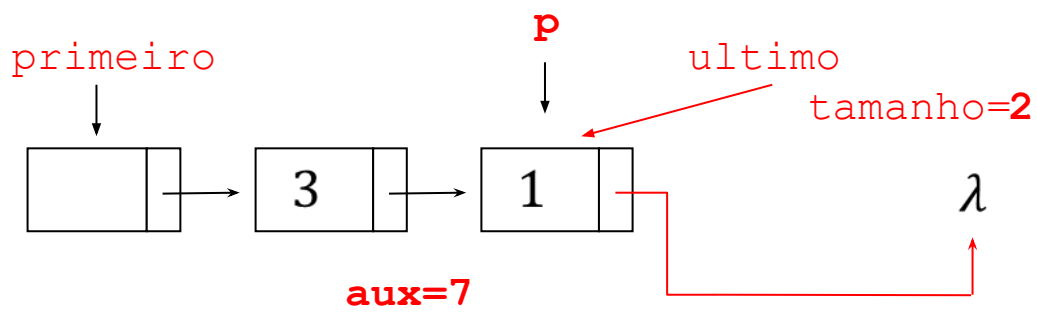
```
TipoItem ListaEncadeada::RemoveFinal() {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à última
    p = Posiciona(tamanho, true);

    p->prox = NULL;
    tamanho--;
    aux = ultimo->item;
    delete ultimo;
    ultimo = p;

    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveFinal();
x.Imprime();
```

Class Lista Encadeada - Remoção

```
TipoItem ListaEncadeada::RemoveFinal() {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à última
    p = Posiciona(tamanho, true);

    p->prox = NULL;
    tamanho--;
    aux = ultimo->item;
    delete ultimo;
    ultimo = p;

    return aux;
}
```

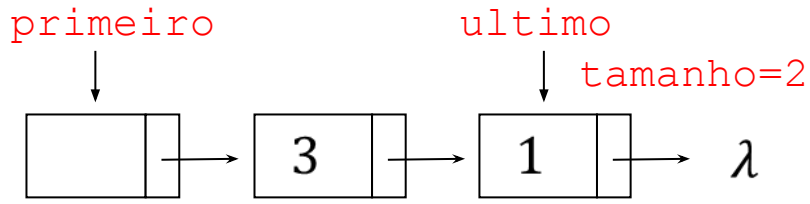



```
ListaEncadeada L;
TipoItem x;

x=L.RemoveFinal();
x.Imprime();
```

Class Lista Encadeada - Remoção

```
TipoItem ListaEncadeada::RemoveFinal() {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à última
    p = Posiciona(tamanho, true);

    p->prox = NULL;
    tamanho--;
    aux = ultimo->item;
    delete ultimo;
    ultimo = p;

    return aux;
}
```

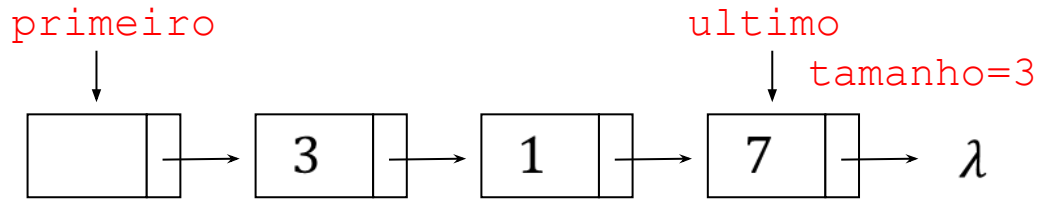
$O(n)$




```
ListaEncadeada L;
TipoItem x;

x=L.RemoveFinal();
x.Imprime();
```

Class Lista Encadeada - Remoção

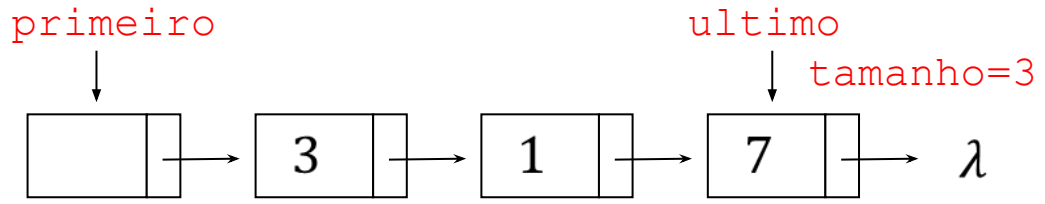
```
TipoItem ListaEncadeada::RemovePosicao(int pos) {;
    TipoItem aux;
    TipoCelula *p, *q;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à pos
    p = Posiciona(pos, true);
    q = p->prox;
    p->prox = q->prox;
    tamanho--;
    aux = q->item;
    delete q;
    if(p->prox == NULL)
        ultimo = p;
    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemovePosicao(2);
x.Imprime();
```

Class Lista Encadeada - Remoção

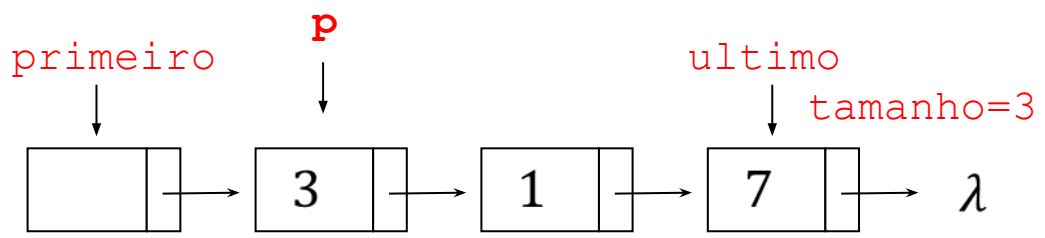
```
TipoItem ListaEncadeada::RemovePosicao(int pos) {;
    TipoItem aux;
    TipoCelula *p, *q;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à pos
    p = Posiciona(pos, true);
    q = p->prox;
    p->prox = q->prox;
    tamanho--;
    aux = q->item;
    delete q;
    if(p->prox == NULL)
        ultimo = p;
    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemovePosicao(2);
x.Imprime();
```

Class Lista Encadeada - Remoção

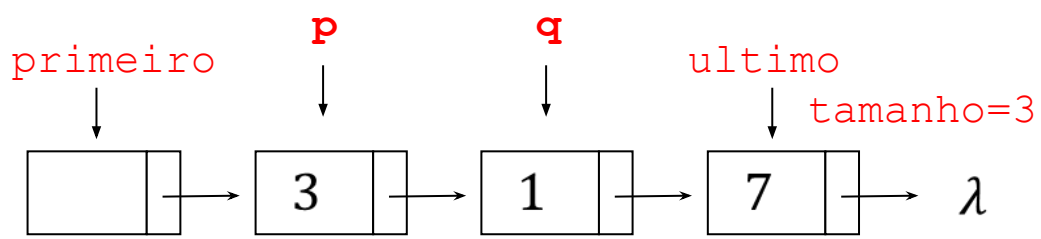
```
TipoItem ListaEncadeada::RemovePosicao(int pos) {;
    TipoItem aux;
    TipoCelula *p, *q;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à pos
    p = Posiciona(pos, true);
    q = p->prox;
    p->prox = q->prox;
    tamanho--;
    aux = q->item;
    delete q;
    if(p->prox == NULL)
        ultimo = p;
    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemovePosicao(2);
x.Imprime();
```

Class Lista Encadeada - Remoção

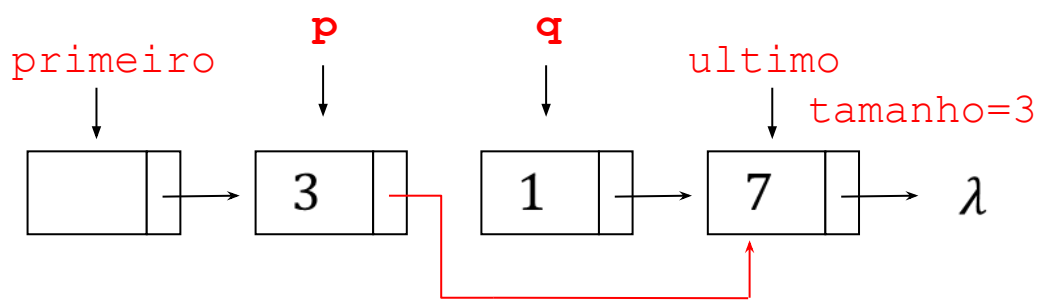
```
TipoItem ListaEncadeada::RemovePosicao(int pos) {;
    TipoItem aux;
    TipoCelula *p, *q;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à pos
    p = Posiciona(pos, true);
    q = p->prox;
    p->prox = q->prox;
    tamanho--;
    aux = q->item;
    delete q;
    if(p->prox == NULL)
        ultimo = p;
    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemovePosicao(2);
x.Imprime();
```

Class Lista Encadeada - Remoção

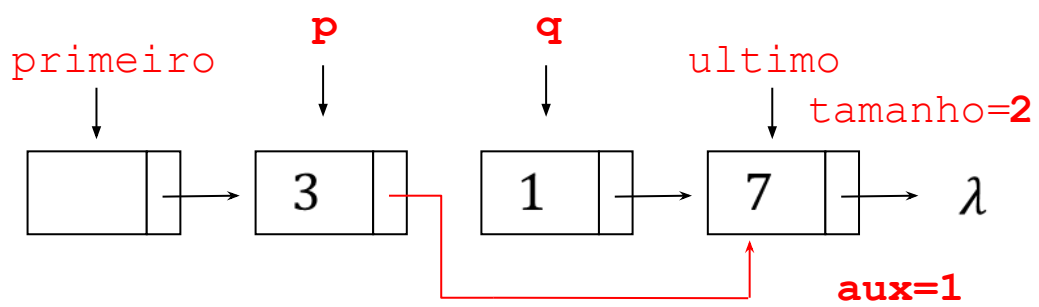
```
TipoItem ListaEncadeada::RemovePosicao(int pos) {;
    TipoItem aux;
    TipoCelula *p, *q;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à pos
    p = Posiciona(pos, true);
    q = p->prox;
    p->prox = q->prox;
    tamanho--;
    aux = q->item;
    delete q;
    if(p->prox == NULL)
        ultimo = p;
    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemovePosicao(2);
x.Imprime();
```

Class Lista Encadeada - Remoção

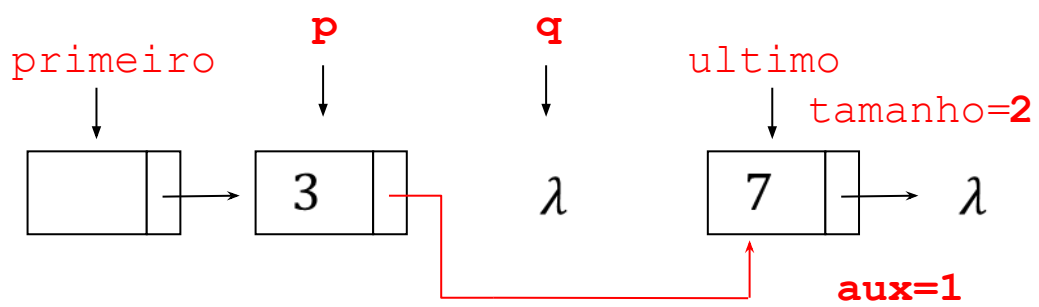
```
TipoItem ListaEncadeada::RemovePosicao(int pos) {;
    TipoItem aux;
    TipoCelula *p, *q;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à pos
    p = Posiciona(pos, true);
    q = p->prox;
    p->prox = q->prox;
    tamanho--;
    aux = q->item;
    delete q;
    if(p->prox == NULL)
        ultimo = p;
    return aux;
}
```




```
ListaEncadeada L;
TipoItem x;

x=L.RemovePosicao(2);
x.Imprime();
```

Class Lista Encadeada - Remoção

```
TipoItem ListaEncadeada::RemovePosicao(int pos) {;
    TipoItem aux;
    TipoCelula *p, *q;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à pos
    p = Posiciona(pos, true);
    q = p->prox;
    p->prox = q->prox;
    tamanho--;
    aux = q->item;
    delete q;
    if(p->prox == NULL)
        ultimo = p;
    return aux;
}
```



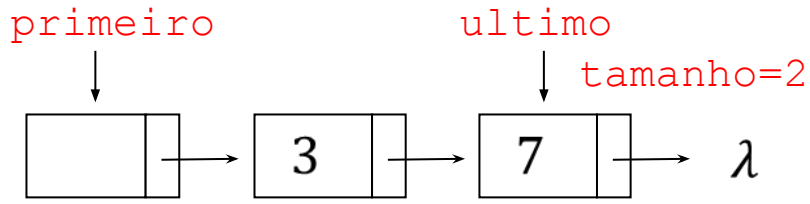

```
ListaEncadeada L;
TipoItem x;

x=L.RemovePosicao(2);
x.Imprime();
```

Class Lista Encadeada - Remoção

```
TipoItem ListaEncadeada::RemovePosicao(int pos) {;
    TipoItem aux;
    TipoCelula *p, *q;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    // posiciona p na celula anterior à pos
    p = Posiciona(pos, true);
    q = p->prox;
    p->prox = q->prox;
    tamanho--;
    aux = q->item;
    delete q;
    if(p->prox == NULL)
        ultimo = p;
    return aux;
}
```

Melhor Caso $O(1)$

Pior Caso $O(n)$




```
ListaEncadeada L;
TipoItem x;

x=L.RemovePosicao(2);
x.Imprime();
```

1

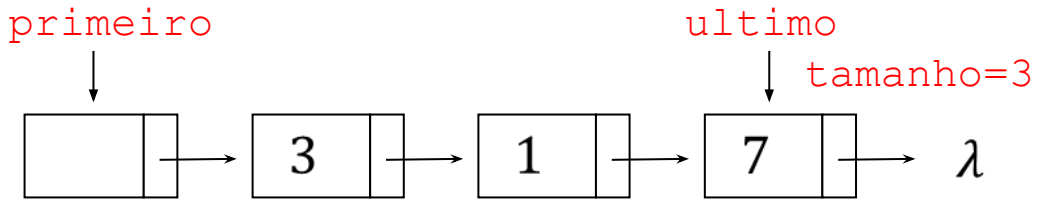
Class Lista Encadeada - Pesquisa

- Pesquisa por um item com uma determinada chave
  - Retorna o item encontrado ou um *flag* (-1)

```
TipoItem  ListaEncadeada::Pesquisa(TipoChave c) {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    aux.SetChave(-1);
    while (p!=NULL) {
        if (p->item.GetChave() == c) {
            aux = p->item;
            break;
        }
        p = p->prox;
    }

    return aux;
};
```

Class Lista Encadeada - Pesquisa

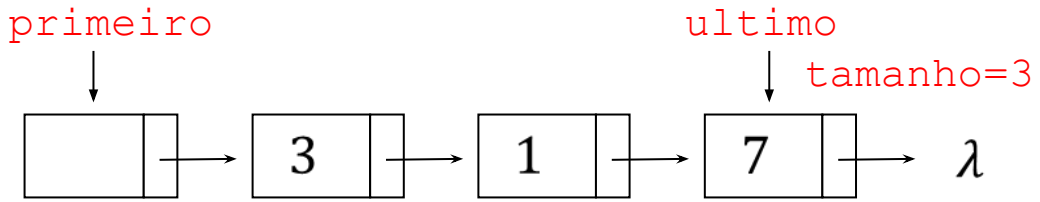
```
TipoItem  ListaEncadeada::Pesquisa(TipoChave c) {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    aux.SetChave(-1);
    while (p!=NULL) {
        if (p->item.GetChave() == c) {
            aux = p->item;
            break;
        }
        p = p->prox;
    }

    return aux;
};
```




```
ListaEncadeada L;
TipoItem x;

x = L.Pesquisa(1);
x.Imprime()
```

Class Lista Encadeada - Pesquisa

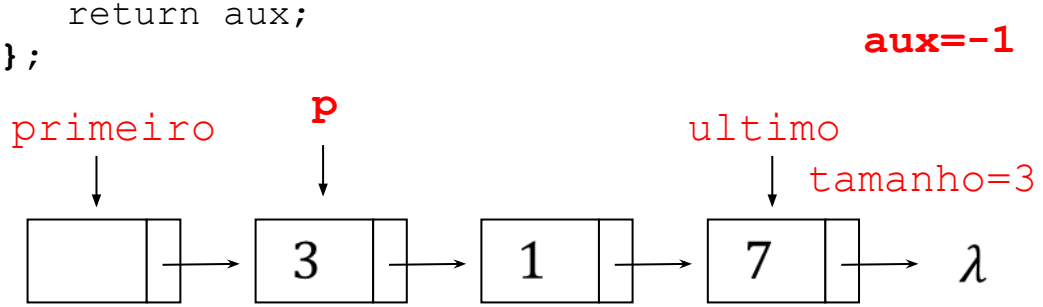
```
TipoItem  ListaEncadeada::Pesquisa(TipoChave c) {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    aux.SetChave(-1);
    while (p!=NULL) {
        if (p->item.GetChave() == c) {
            aux = p->item;
            break;
        }
        p = p->prox;
    }

    return aux;
};
```




```
ListaEncadeada L;
TipoItem x;

x = L.Pesquisa(1);
x.Imprime()
```

Class Lista Encadeada - Pesquisa

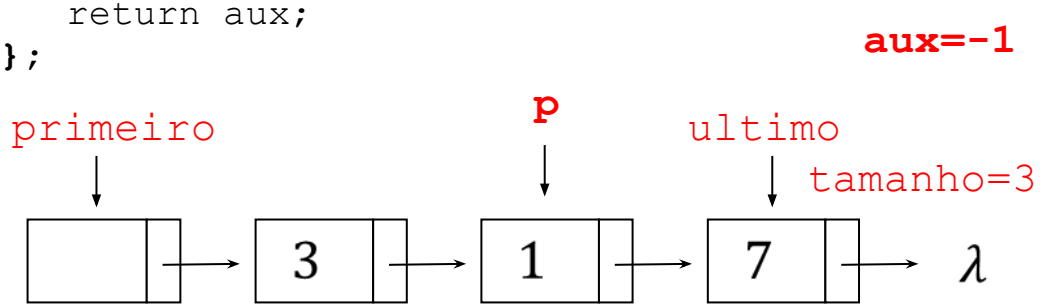
```
TipoItem  ListaEncadeada::Pesquisa(TipoChave c) {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    aux.SetChave(-1);
    while (p!=NULL) {
        if (p->item.GetChave() == c) {
            aux = p->item;
            break;
        }
        p = p->prox;
    }

    return aux;
};
```




```
ListaEncadeada L;
TipoItem x;

x = L.Pesquisa(1);
x.Imprime()
```

Class Lista Encadeada - Pesquisa

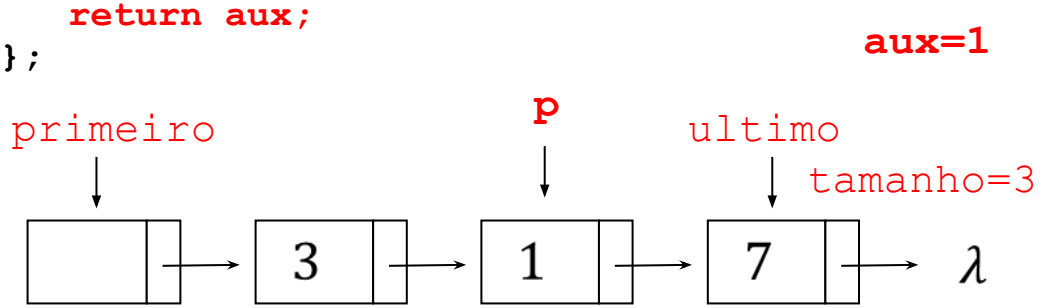
```
TipoItem  ListaEncadeada::Pesquisa(TipoChave c) {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    aux.SetChave(-1);
    while (p!=NULL) {
        if (p->item.GetChave() == c) {
            aux = p->item;
            break;
        }
        p = p->prox;
    }

    return aux;
};
```




```
ListaEncadeada L;
TipoItem x;

x = L.Pesquisa(1);
x.Imprime()
```

Class Lista Encadeada - Pesquisa

```
TipoItem  ListaEncadeada::Pesquisa(TipoChave c) {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    aux.SetChave(-1);
    while (p!=NULL) {
        if (p->item.GetChave() == c) {
            aux = p->item;
            break;
        }
        p = p->prox;
    }

    return aux;
};
```



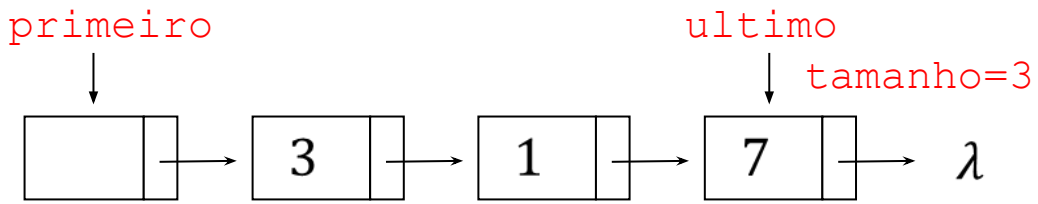

```
ListaEncadeada L;
TipoItem x;

x = L.Pesquisa(1);
x.Imprime()
```

Class Lista Encadeada - Pesquisa

```
TipoItem  ListaEncadeada::Pesquisa(TipoChave c) {
    TipoItem aux;
    TipoCelula *p;

    if (tamanho == 0)
        throw "ERRO: Lista vazia!";

    p = primeiro->prox;
    aux.SetChave(-1);
    while (p!=NULL) {
        if (p->item.GetChave() == c) {
            aux = p->item;
            break;
        }
        p = p->prox;
    }

    return aux;
};
```

Melhor Caso $O(1)$

Pior Caso $O(n)$




```
ListaEncadeada L;
TipoItem x;

x = L.Pesquisa(1);
x.Imprime()
```

1

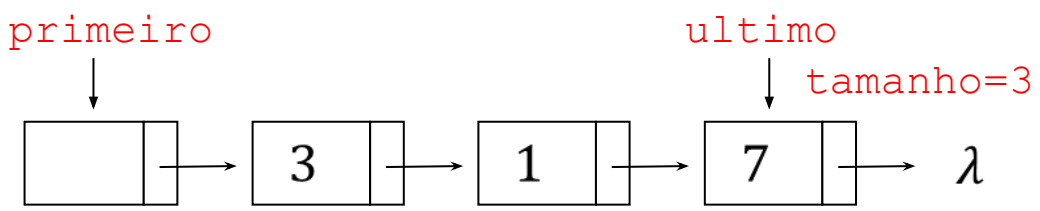
Class Lista Encadeada - Imprime

- Imprime todos os elementos

```
void ListaEncadeada::Imprime() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        p->item.Imprime();
        p = p->prox;
    }

    printf("\n");
};
```

$O(n)$

```
ListaEncadeada L;

L.Imprime();
```

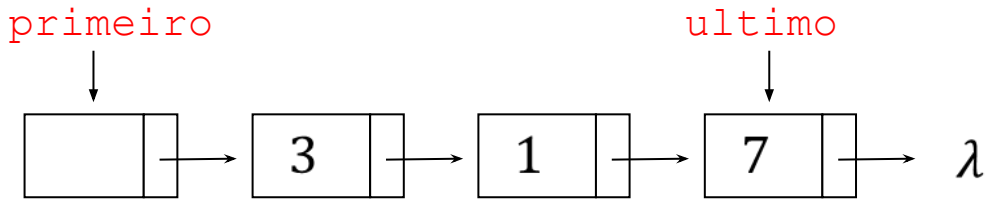


3 1 7

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```



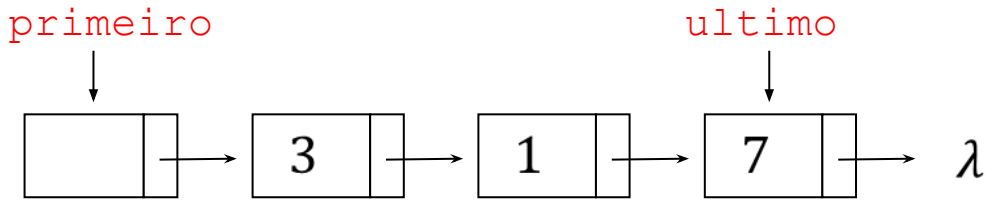
tamanho=3

```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```



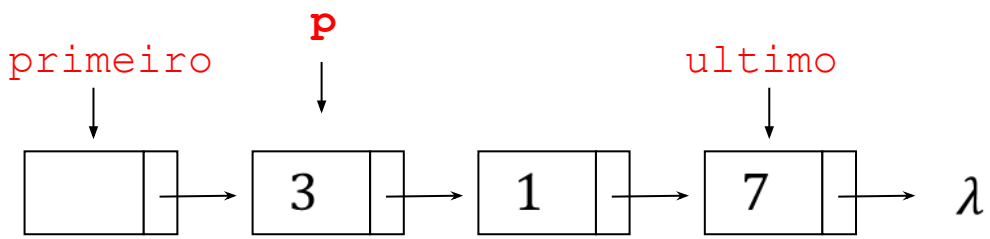
tamanho=3

```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```



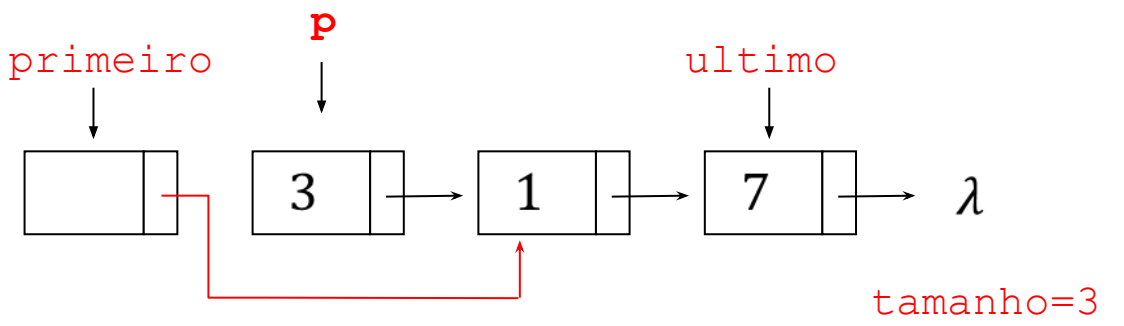
tamanho=3

```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```

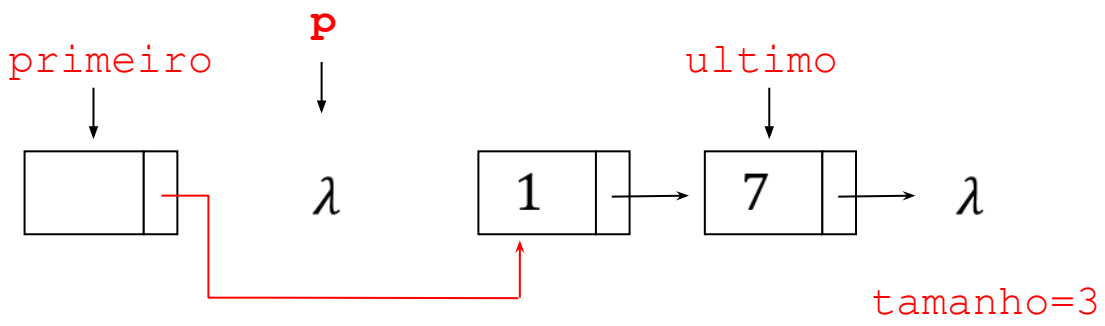



```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```

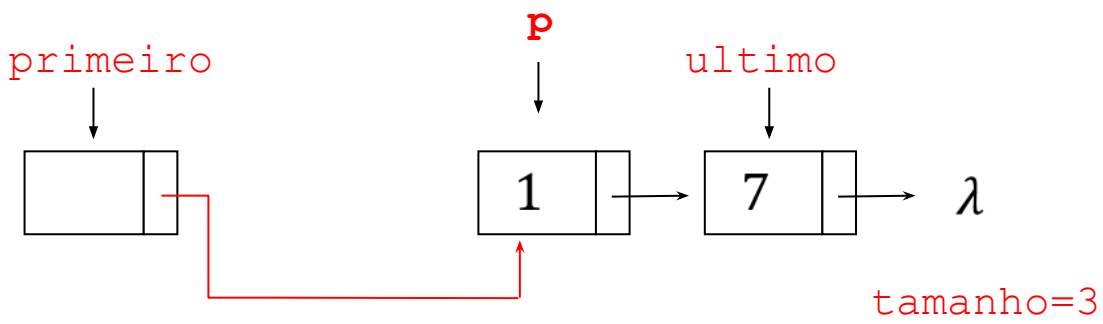



```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```

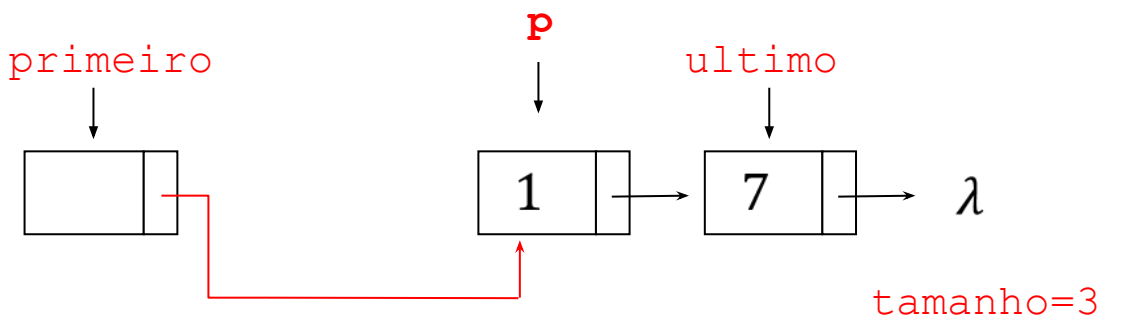



```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```

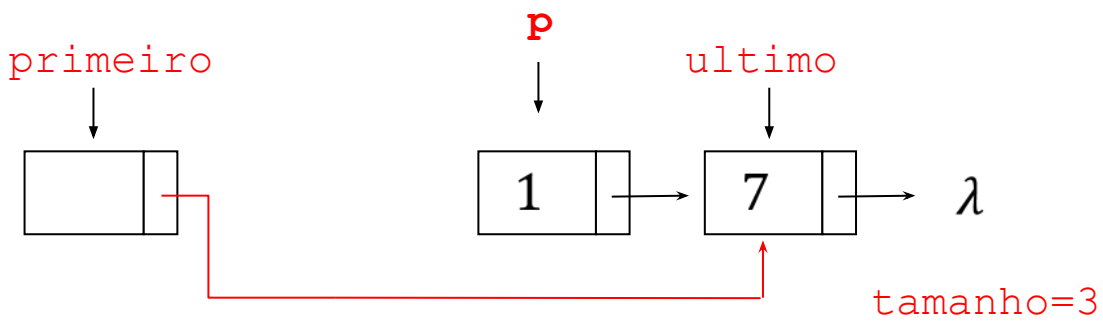



```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```

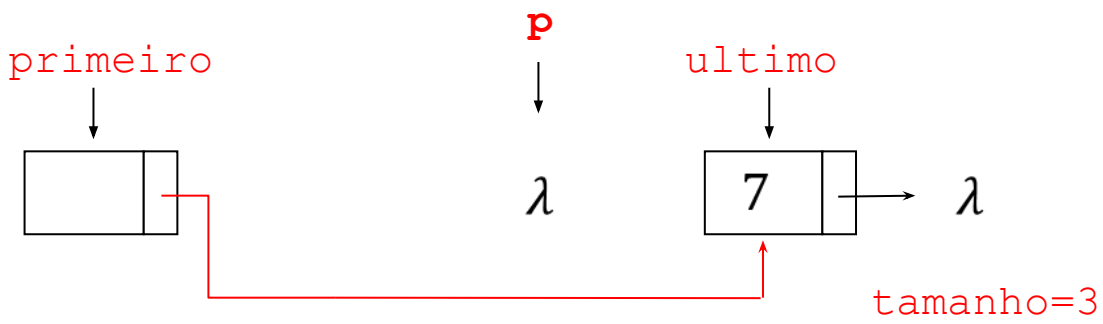



```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```

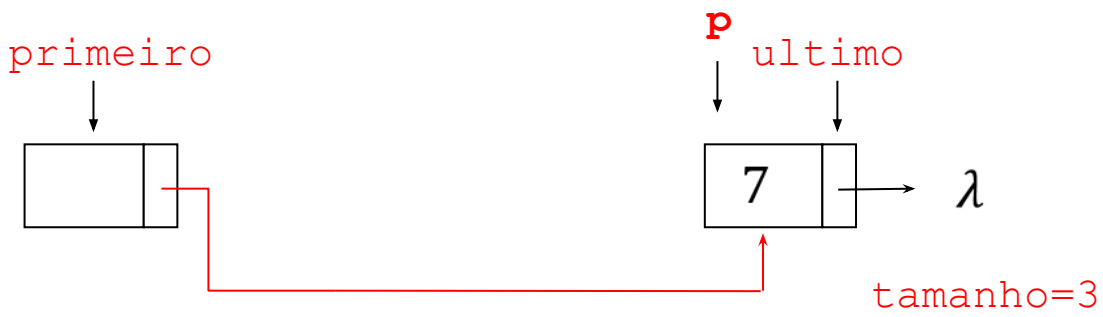



```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```

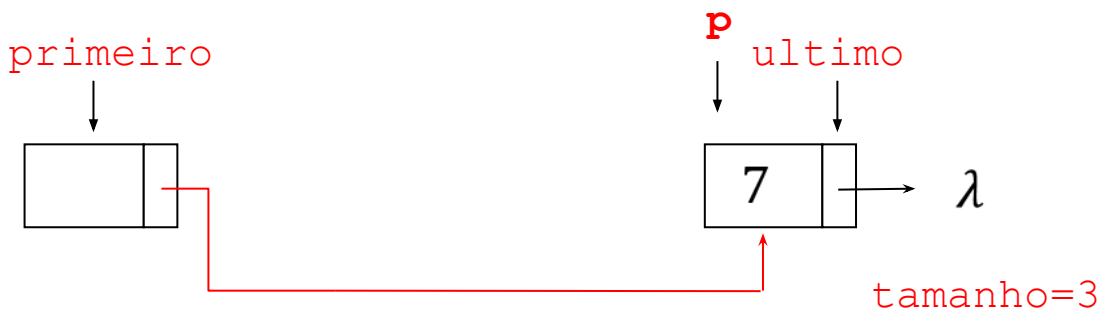



```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```

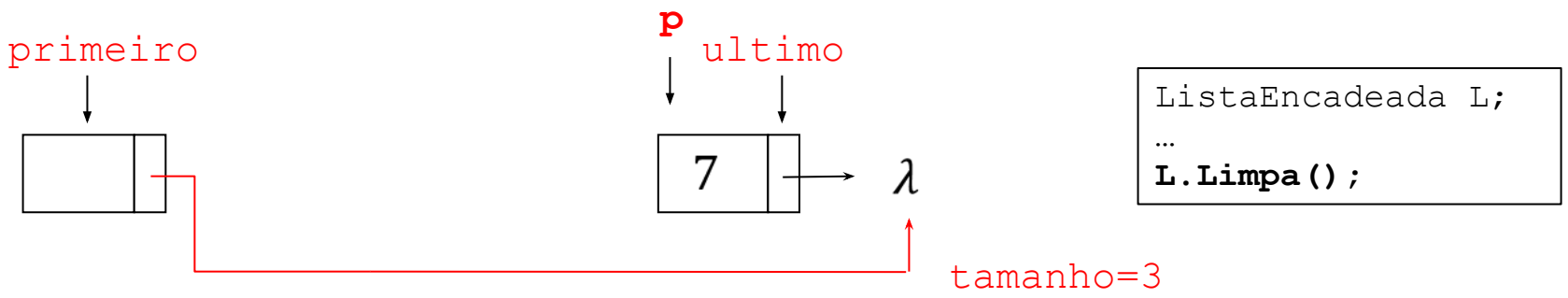



```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

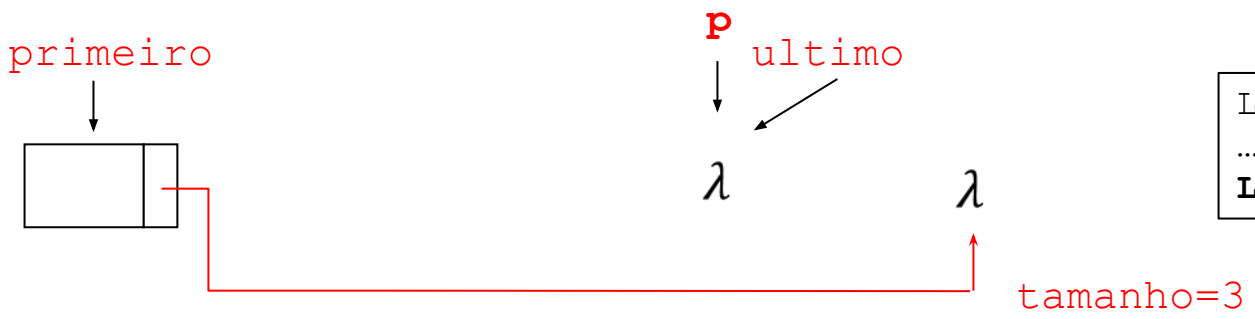
```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```



Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```

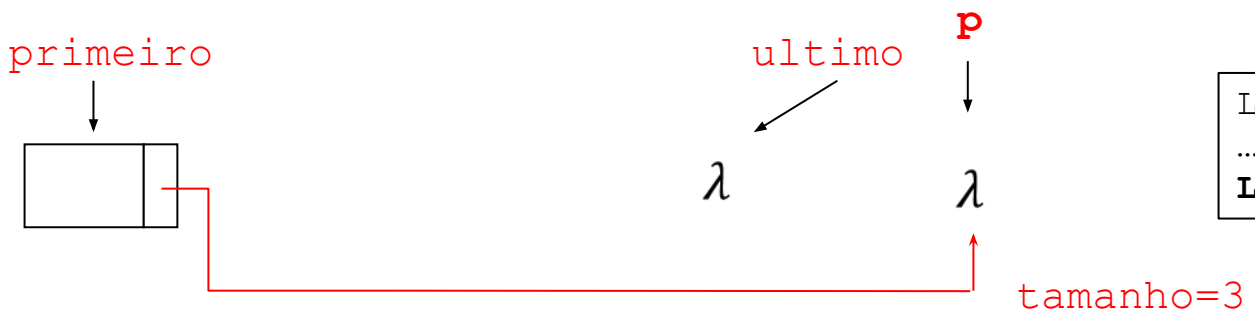



```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```

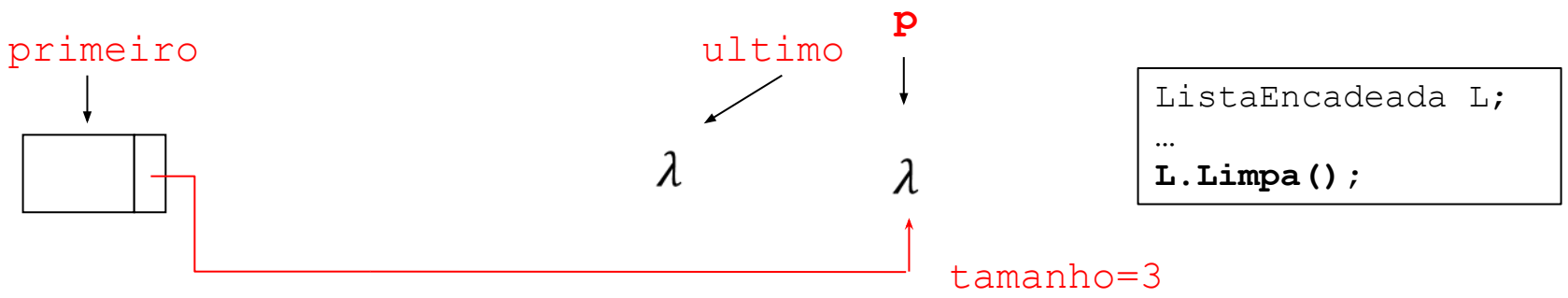



```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

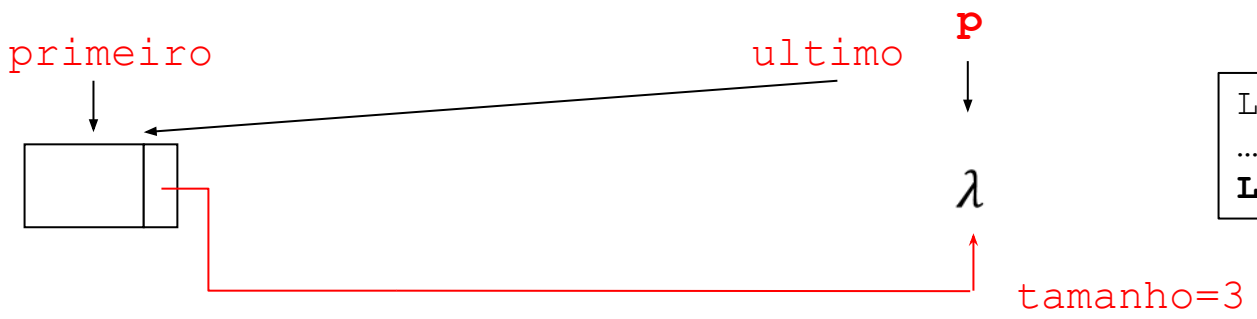
```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```



Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```



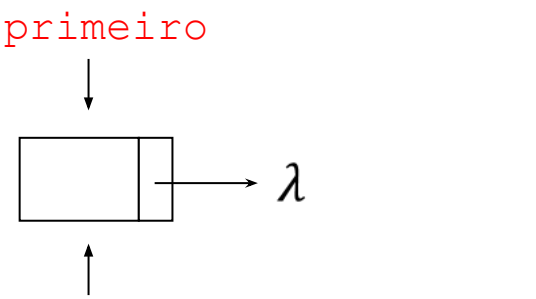

```
ListaEncadeada L;
…
L.Limpa();
```

Class Lista Encadeada - Limpa

- "Limpa" a Lista
  - Percorre a lista desalocando memória

```
void ListaEncadeada::Limpa() {
    TipoCelula *p;

    p = primeiro->prox;
    while (p!=NULL) {
        primeiro->prox = p->prox;
        delete p;
        p = primeiro->prox;
    }
    ultimo = primeiro;
    tamanho = 0;
};
```

$O(n)$




```
ListaEncadeada L;
…
L.Limpa();
```

Alocação Encadeada

- Vantagens:
  - Tamanho da lista em memória é dinâmico
    - Bom para aplicações onde a previsão do tamanho não pode ser feita a priori
  - Inserção e Remoção não requer o deslocamento de itens
- Desvantagens
  - Acesso a itens requer caminhamento na lista
  - Memória extra para armazenar os apontadores
  - Código mais complexo

Alocação Sequencial x Encadeada

	Sequencial	Encadeada
Acesso a um Item	**O(1)**	O(n)
Inserção / Remoção	O(n)	**O(1)***
Tamanho	Fixo	**Dinâmico**
Memória Extra	**Não**	Sim
Implementação Simples	**Sim**	Não

* Pode ser necessário posicionar um apontador auxiliar antes

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Estrutura de Dados

Pilhas e Filas

Professores: Anisio, Wagner e Washington

TAD Pilhas

- Tipo Abstrato de dados com a seguinte característica:

O último elemento a ser inserido é o primeiro a ser retirado *(LIFO – Last In First Out)*

- Analogia: pilha de pratos, pilha de livros, etc
- Usos: chamada de subprogramas, avaliação de expressões aritméticas, caminhamento em árvores, etc...

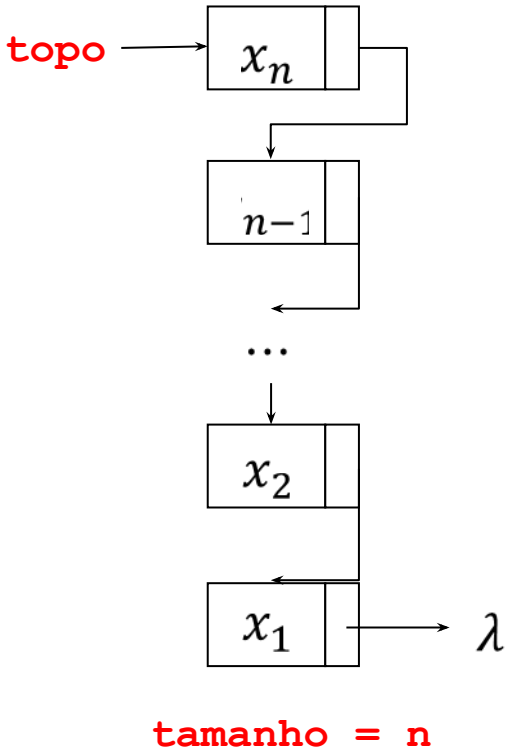
TAD: Pilha

- Duas Implementações:
  - Sequencial (uso de arranjos, alocação estática)
  - Encadeada (uso de apontadores, alocação dinâmica)
- Operações:
  - Criar uma nova pilha (construtor)
  - Testar se a pilha está *vazia*
  - **Empilhar** um item
  - **Desempilhar** um item
  - Limpar a pilha

Disclaimer: os códigos que serão apresentados devem ser considerados como exemplos. Eles não são, necessariamente, os mais modulares ou eficientes…

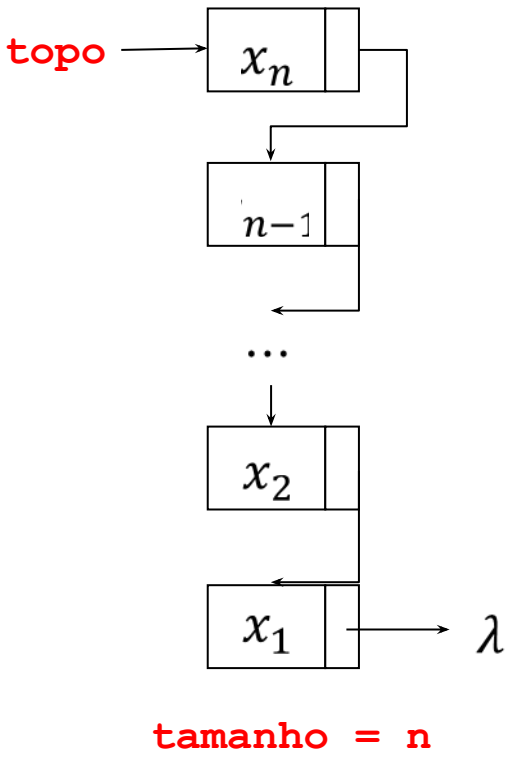
Alocação Encadeada

- Itens da pilha são armazenados em células
  - Alocação Dinâmica, tamanho variável
  - **topo** é um apontador para a célula que está no topo da pilha
  - **não vamos usar célula cabeça**
- Inserções e Retiradas em apenas um extremo do vetor
  - **Empilha**: cria uma nova célula e a liga no topo da pilha
  - **Desempliha**: retira o elemento do topo e apaga a célula



Class Pilha Encadeada

```
class PilhaEncadeada : public Pilha
{
    public:
        PilhaEncadeada();
        virtual ~PilhaEncadeada();

        void Empilha(TipoItem item);
        TipoItem Desempilha();
        void Limpa();

    private:
        TipoCelula* topo;

};
```

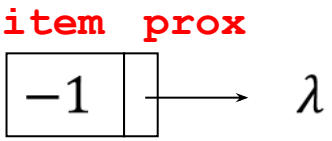


Class TipoCélula

- Classe para representar as células da Pilha
  - Campo **TipoItem item**: armazena o item
  - Campo **TipoCelula *prox**: apontador para a próxima célula
  - Possui método para inicialização (constructor)
  - Permite o acesso de atributos privados pela classe PilhaEncadeada
    - *Friend class*

```
class TipoCelula
{
    public:
        TipoCelula();

    private:
        TipoItem item;
        TipoCelula *prox;

    friend class PilhaEncadeada;
};
```

```
TipoCelula::TipoCelula()
{
    item.SetChave(-1);
    prox = NULL;
}
```



Class Pilha Encadeada
Construtor e Destrutor

- Construtor
  - Chama o construtor da classe pai, que inicializa o atributo *tamanho* com o valor 0 e inicializa o apontador topo com null.

```
PilhaEncadeada::PilhaEncadeada() : Pilha()
{
    topo = NULL;
}
```

topo ⟶ λ

tamanho = 0

- Destrutor
  - Chama o método Limpa que remove todas as células da lista

```
PilhaEncadeada::~PilhaEncadeada()
{
    Limpa();
}
```

Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

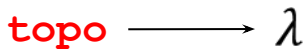
```
void PilhaEncadeada::Empilha(TipoItem item){
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = topo;
    topo = nova;
    tamanho++;
};
```

Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

```
void PilhaEncadeada::Empilha(TipoItem item){
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = topo;
    topo = nova;
    tamanho++;
};
```

```
PilhaEncadeada p
TipoItem x;

x.SetChave(5);
p.Empilha(x)
...
```



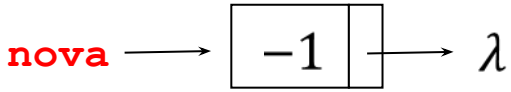
tamanho = 0

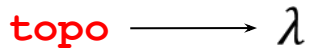
Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

```
void PilhaEncadeada::Empilha(TipoItem item){
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = topo;
    topo = nova;
    tamanho++;
};
```

```
PilhaEncadeada p
TipoItem x;

x.SetChave(5);
p.Empilha(x)
...
```





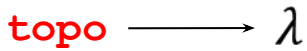
tamanho = 0

Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

```
void PilhaEncadeada::Empilha(TipoItem item){
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = topo;
    topo = nova;
    tamanho++;
};
```

```
PilhaEncadeada p
TipoItem x;

x.SetChave(5);
p.Empilha(x)
...
```



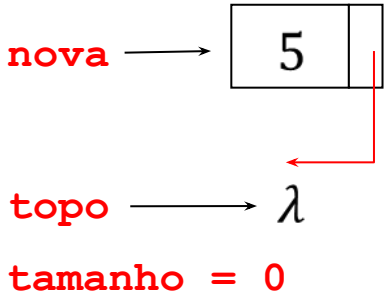
tamanho = 0

Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

```
void PilhaEncadeada::Empilha(TipoItem item){
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = topo;
    topo = nova;
    tamanho++;
};
```

```
PilhaEncadeada p
TipoItem x;

x.SetChave(5);
p.Empilha(x)
...
```

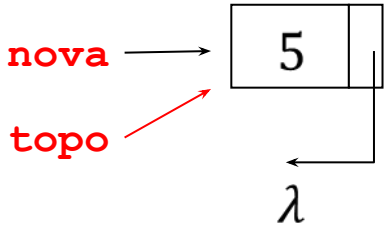


Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

```
void PilhaEncadeada::Empilha(TipoItem item){
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     nova->prox = topo;
     topo = nova;
     tamanho++;
};
```

```
PilhaEncadeada p
TipoItem x;

x.SetChave(5);
p.Empilha(x)
...
```

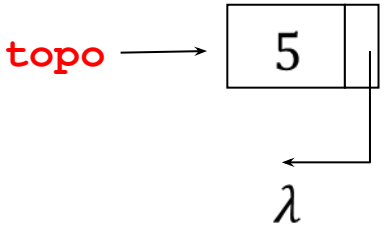


Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

```
void PilhaEncadeada::Empilha(TipoItem item){
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = topo;
    topo = nova;
    tamanho++;
};
```

```
PilhaEncadeada p
TipoItem x;

x.SetChave(5);
p.Empilha(x)
...
```

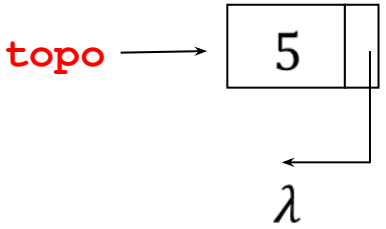


Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

```
void PilhaEncadeada::Empilha(TipoItem item){
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     nova->prox = topo;
     topo = nova;
     tamanho++;
};
```

```
PilhaEncadeada p
TipoItem x;

...
x.SetChave(8);
p.Empilha(x)
```

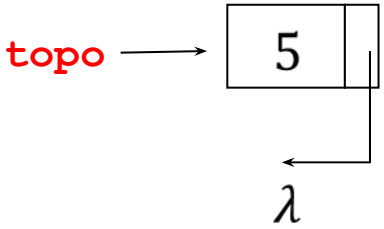


Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

```
void PilhaEncadeada::Empilha(TipoItem item){
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = topo;
    topo = nova;
    tamanho++;
};
```

```
PilhaEncadeada p
TipoItem x;

...
x.SetChave(8);
p.Empilha(x)
```

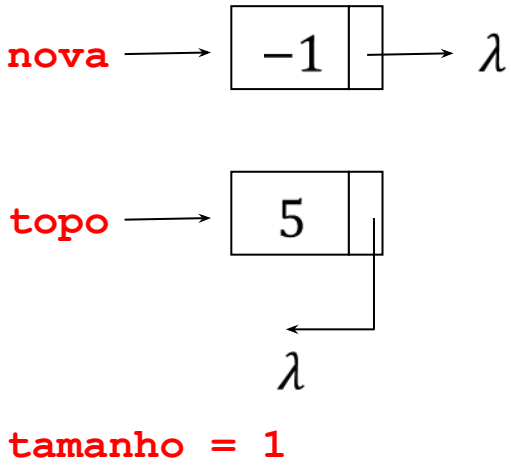


Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

```
void PilhaEncadeada::Empilha(TipoItem item){
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     nova->prox = topo;
     topo = nova;
     tamanho++;
};
```

```
PilhaEncadeada p
TipoItem x;

...
x.SetChave(8);
p.Empilha(x)
```

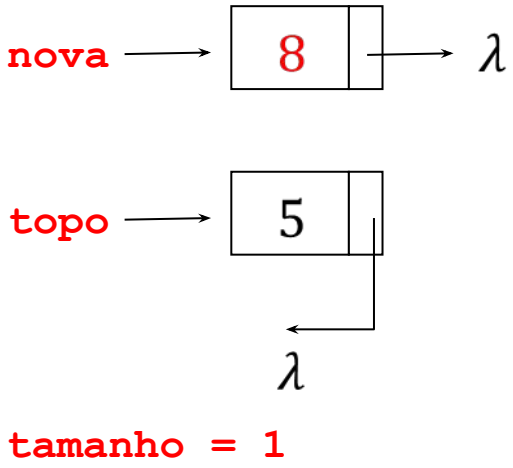


Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

```
void PilhaEncadeada::Empilha(TipoItem item){
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     nova->prox = topo;
     topo = nova;
     tamanho++;
};
```

```
PilhaEncadeada p
TipoItem x;

...
x.SetChave(8);
p.Empilha(x)
```

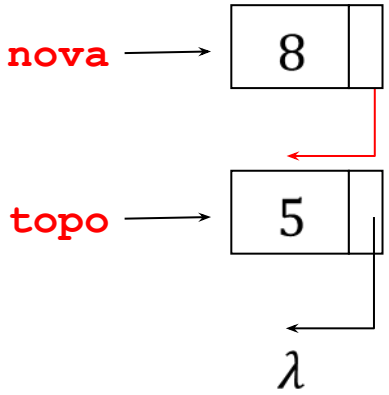


Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

```
void PilhaEncadeada::Empilha(TipoItem item){
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     nova->prox = topo;
     topo = nova;
     tamanho++;
};
```

```
PilhaEncadeada p
TipoItem x;

...
x.SetChave(8);
p.Empilha(x)
```

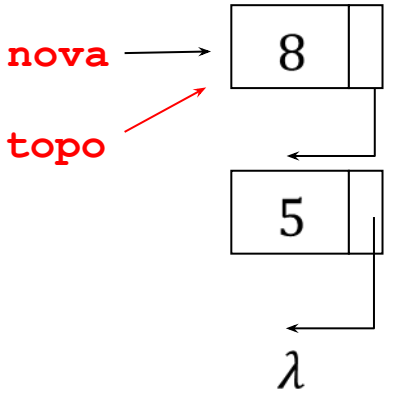


Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

```
void PilhaEncadeada::Empilha(TipoItem item){
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     nova->prox = topo;
     topo = nova;
     tamanho++;
};
```

```
PilhaEncadeada p
TipoItem x;

...
x.SetChave(8);
p.Empilha(x)
```



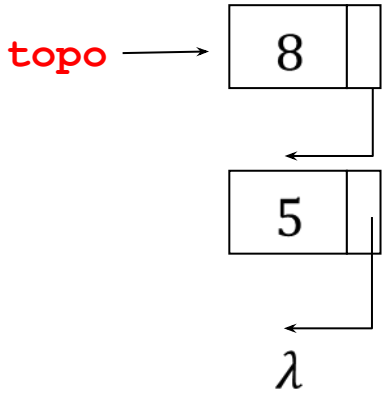
tamanho = 1

Class Pilha Encadeada - Empilha

- Empilha
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na lista
  - Incrementa o tamanho

```
void PilhaEncadeada::Empilha(TipoItem item){
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    nova->prox = topo;
    topo = nova;
    tamanho++;
};
```

```
PilhaEncadeada p
TipoItem x;

...
x.SetChave(8);
p.Empilha(x)
```



tamanho = 2

Class Pilha Encadeada - Desempilha

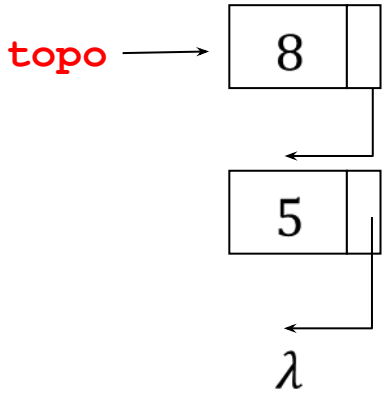
- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```

Class Pilha Encadeada - Desempilha

- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```



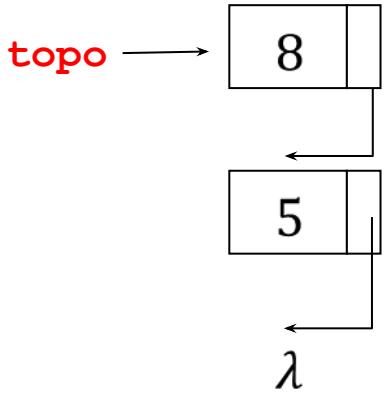

```
PilhaEncadeada p;
TipoItem x;

...
x = p.Desempilha();
x.Imprime();
```

tamanho = 2

Class Pilha Encadeada - Desempilha

- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```



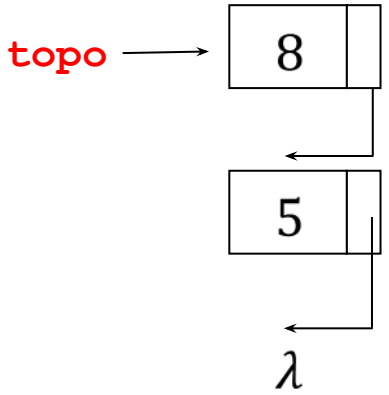

```
PilhaEncadeada p;
TipoItem x;

...
x = p.Desempilha();
x.Imprime();
```

tamanho = 2

Class Pilha Encadeada - Desempilha

- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```




```
PilhaEncadeada p;
TipoItem x;

...
x = p.Desempilha();
x.Imprime();
```

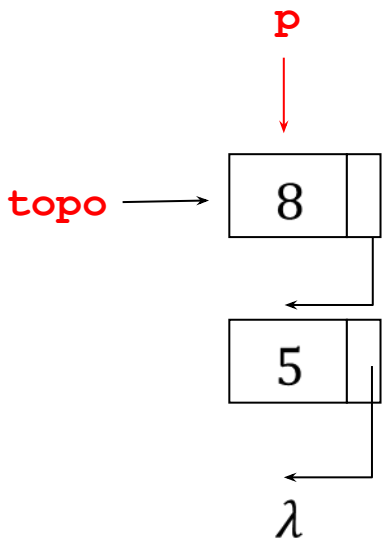
tamanho = 2

aux = 8

Class Pilha Encadeada - Desempilha

- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```




```
PilhaEncadeada p;
TipoItem x;

...
x = p.Desempilha();
x.Imprime();
```

tamanho = 2

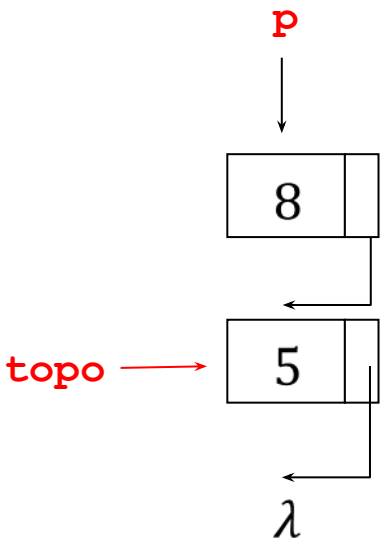
aux = 8

Class Pilha Encadeada - Desempilha

- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```

```
PilhaEncadeada p;
TipoItem x;

...
x = p.Desempilha();
x.Imprime();
```



tamanho = 2

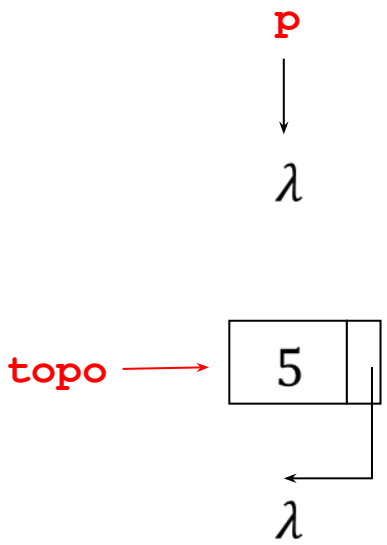
aux = 8

Class Pilha Encadeada - Desempilha

- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```

```
PilhaEncadeada p;
TipoItem x;

...
x = p.Desempilha();
x.Imprime();
```

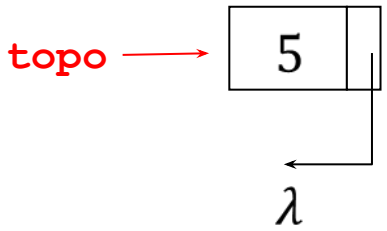


Class Pilha Encadeada - Desempilha

- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```

```
PilhaEncadeada p;
TipoItem x;

...
x = p.Desempilha();
x.Imprime();
```



tamanho = 1

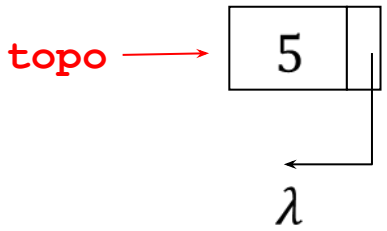
8

Class Pilha Encadeada - Desempilha

- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```

```
PilhaEncadeada p;
TipoItem x;

...
x = p.Desempilha();
x.Imprime();
```



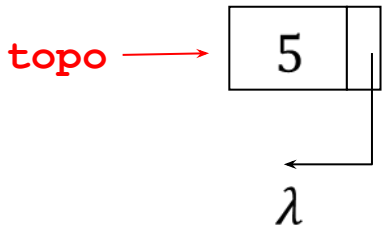
tamanho = 1

Class Pilha Encadeada - Desempilha

- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```

```
PilhaEncadeada p;
TipoItem x;

...
x = p.Desempilha();
x.Imprime();
```



tamanho = 1

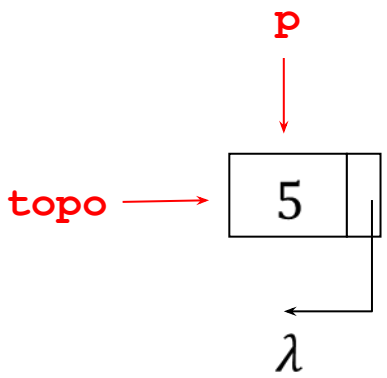
aux = 5

Class Pilha Encadeada - Desempilha

- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```

```
PilhaEncadeada p;
TipoItem x;

...
x = p.Desempilha();
x.Imprime();
```



tamanho = 1

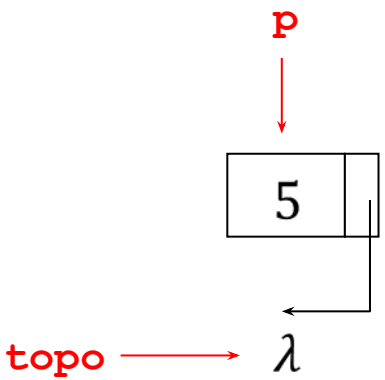
aux = 5

Class Pilha Encadeada - Desempilha

- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```

```
PilhaEncadeada p;
TipoItem x;

...
x = p.Desempilha();
x.Imprime();
```

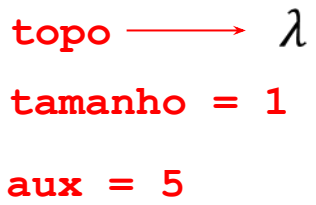


Class Pilha Encadeada - Desempilha

- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```


```
PilhaEncadeada p;
TipoItem x;

...
x = p.Desempilha();
x.Imprime();
```

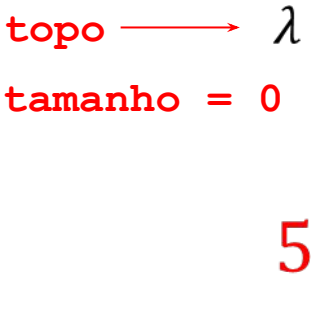


Class Pilha Encadeada - Desempilha

- Desempilha
  - Testa se a pilha está vazia, gerando uma exceção
  - Pega o valor que está no topo
  - Faz o topo apontar para a célula seguinte
  - Apaga a célula removida e retorna o item

```
TipoItem PilhaEncadeada::Desempilha(){
    TipoItem aux; TipoCelula *p;

    if(tamanho == 0)
        throw "A pilha está vazia!";

    aux = topo->item;
    p = topo;
    topo = topo->prox;
    delete p;
    tamanho--;

    return aux;
};
```

```
PilhaEncadeada p;
TipoItem x;

...
x = p.Desempilha();
x.Imprime();
```

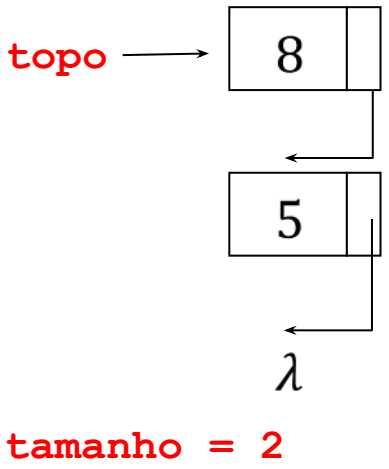


Class Pilha Encadeada - Limpa

- O método limpa chama repetidamente o Desempilha até que a pilha fique vazia

```
void PilhaEncadeada::Limpa(){
    while(!Vazia())
        Desempilha();
}
```

```
PilhaEncadeada p

...
p.Limpa(x)
```

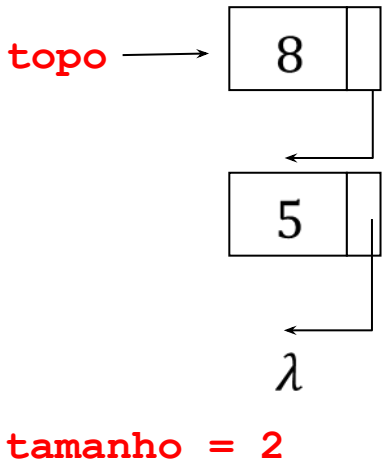


Class Pilha Encadeada - Limpa

- O método limpa chama repetidamente o Desempilha até que a pilha fique vazia

```
void PilhaEncadeada::Limpa(){
    while(!Vazia())
        Desempilha();
}
```

```
PilhaEncadeada p

...
p.Limpa(x)
```

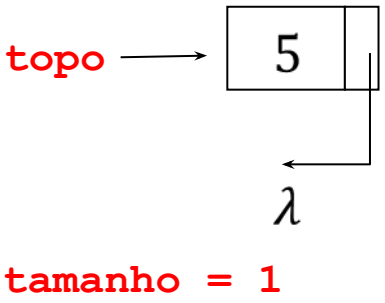


Class Pilha Encadeada - Limpa

- O método limpa chama repetidamente o Desempilha até que a pilha fique vazia

```
void PilhaEncadeada::Limpa(){
    while(!Vazia())
        Desempilha();
}
```

```
PilhaEncadeada p

...
p.Limpa(x)
```

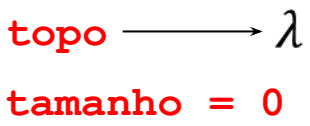


Class Pilha Encadeada - Limpa

- O método limpa chama repetidamente o Desempilha até que a pilha fique vazia

```
void PilhaEncadeada::Limpa(){
    while(!Vazia())
        Desempilha();
}
```

```
PilhaEncadeada p

...
p.Limpa(x)
```



Pilha Arranjo x Pilha Encadeada

	Pilha Arranjo	Pilha Encadeada
Construtor	**O(1)**	**O(1)**
Destrutor	**-**	O(n)
Empilha	**O(1)**	**O(1)**
Desempilha	**O(1)**	**O(1)**
Limpa	**O(1)**	O(n)
Tamanho	Fixo	**Dinâmico**
Memória Extra	**Não**	Sim
Implementação Simples	**Sim**	Não

Em geral, por sua simplicidade, a implementação por arranjo é mais indicada, a menos que a questão de tamanho dinâmico seja de fundamental importância

TAD Filas

- Tipo Abstrato de dados com a seguinte característica:

O primeiro elemento a ser inserido é o primeiro a ser retirado *(FIFO – First In First Out)*

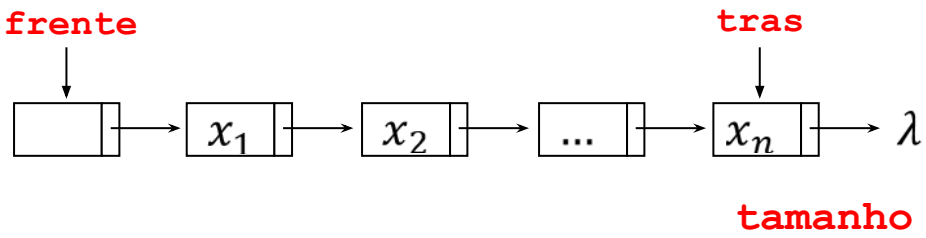
- Analogia: fila bancária, fila do cinema
- Usos: Sistemas operacionais: fila de impressão, processamento; Simulação

TAD: Fila

- Duas Implementações:
  - Sequencial (uso de arranjos, alocação estática)
  - Encadeada (uso de apontadores, alocação dinâmica)
- Operações:
  - Criar uma nova fila (construtor)
  - Testar se a fila está *vazia*
  - **Enfileirar** um item: colocar um item no final da fila
  - **Desenfileirar** um item: retirar um item do início da fila
  - Limpar a fila

Disclaimer: os códigos que serão apresentados devem ser considerados como exemplos. Eles não são, necessariamente, os mais modulares ou eficientes…

Class FilaEncadeada



tamanho = n

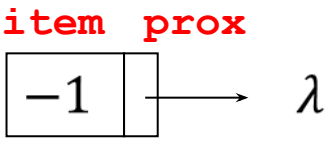
```
class FilaEncadeada : public Fila
{
    public:
        FilaEncadeada();
        virtual ~FilaEncadeada();

        void Enfileira(TipoItem item);
        TipoItem Desenfileira();
        void Limpa();

    private:
        TipoCelula* frente;
        TipoCelula* tras;
};
```

Class TipoCélula

- Classe para representar as células da lista
  - Campo **TipoItem item**: armazena o item
  - Campo **TipoCelula *prox**: apontador para a próxima célula
  - Possui método para inicialização (constructor)
  - Permite o acesso de atributos privados pela classe FilaEncadeada
    - *Friend class*

```
class TipoCelula
{
    public:
        TipoCelula();

    private:
        TipoItem item;
        TipoCelula *prox;

    friend class FilaEncadeada;
};
```

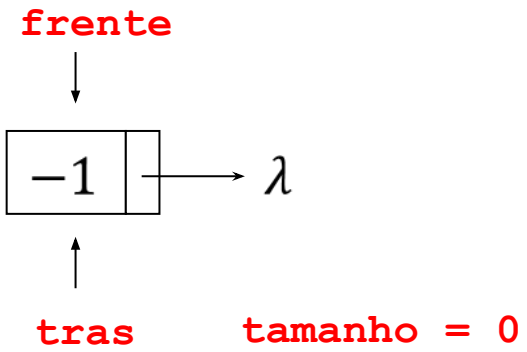
```
TipoCelula::TipoCelula()
{
    item.SetChave(-1);
    prox = NULL;
}
```



Class FilaEncadeada - Construtor

- Construtor
  - Chama o construtor da classe pai, que inicializa o atributo *tamanho* com o valor 0, e inicializa os apontadores *frente e tras*.
  - Uso de uma **célula cabeça**
    - Simplifica a operação enfileira quando a fila está vazia
    - Primeiro elemento da fila vai estar na posição **`frente->prox`**

```
FilaEncadeada::FilaEncadeada() : Fila()
{
    frente = new TipoCelula; // Célula cabeça;
    tras = frente;
}
```

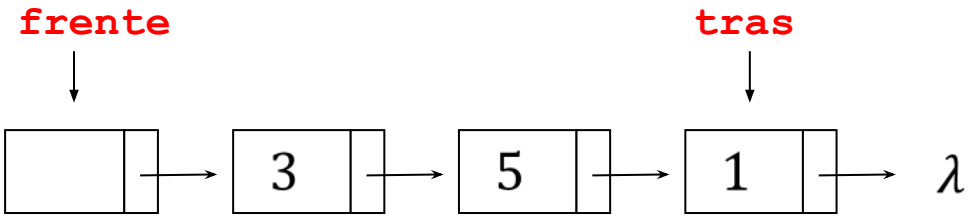



```
FilaEncadeada f;
...
```

Class FilaEncadeada - Destrutor

- Destrutor
  - Como utilizamos a alocação dinâmica, é importante implementar um destrutor para desalocar a memória adequadamente
  - Chama o método *Limpa*, que remove todos as células da fila e depois remove a célula cabeça

```
FilaEncadeada::~FilaEncadeada()
{
    Limpa();
    delete frente;
}
```



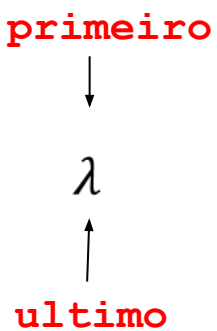

```
FilaEncadeada *f;
…
delete f;
```

tamanho = 3

Class FilaEncadeada - Destrutor

- Destrutor
  - Como utilizamos a alocação dinâmica, é importante implementar um destrutor para desalocar a memória adequadamente
  - Chama o método *Limpa*, que remove todos as células da fila e depois remove a célula cabeça

```
FilaEncadeada::~FilaEncadeada()
{
    Limpa();
    delete frente;
}
```



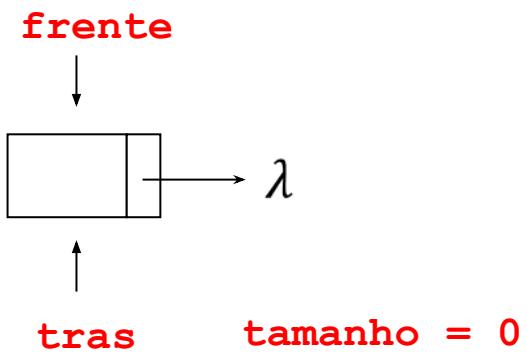

```
FilaEncadeada *f;
…
delete f;
```

Class FilaEncadeada - Enfileira

- Enfileira
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento no final da fila
  - Incrementa o tamanho

```
FilaEncadeada F;
TipoItem x;

x.SetChave(7);
F.Enfileira(x);
```

```
void FilaEncadeada::Enfileira(TipoItem item) {
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     tras->prox = nova;
     tras = nova;
     tamanho++;
}
```

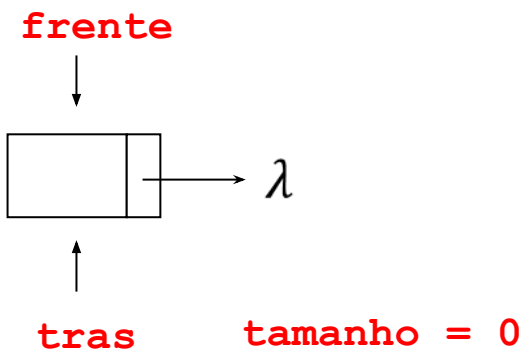


Class FilaEncadeada - Enfileira

- Enfileira
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento no final da fila
  - Incrementa o tamanho

```
FilaEncadeada F;
TipoItem x;

x.SetChave(7);
F.Enfileira(x);
```

```
void FilaEncadeada::Enfileira(TipoItem item) {
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     tras->prox = nova;
     tras = nova;
     tamanho++;
}
```

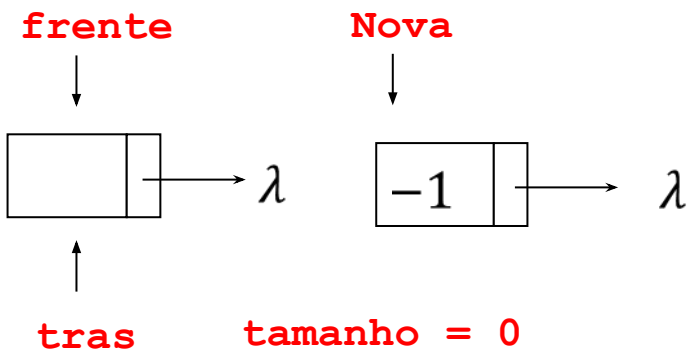


Class FilaEncadeada - Enfileira

- Enfileira
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento no final da fila
  - Incrementa o tamanho

```
FilaEncadeada F;
TipoItem x;

x.SetChave(7);
F.Enfileira(x);
```

```
void FilaEncadeada::Enfileira(TipoItem item) {
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     tras->prox = nova;
     tras = nova;
     tamanho++;
}
```

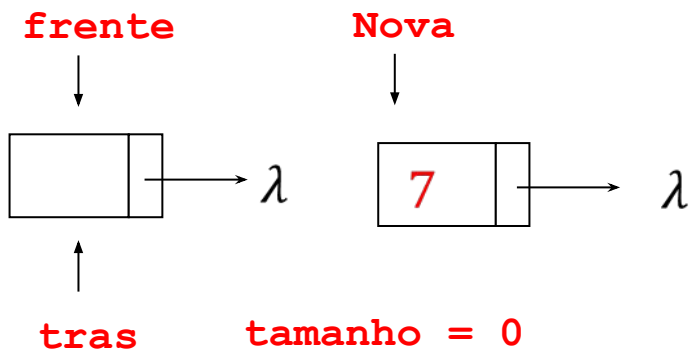


Class FilaEncadeada - Enfileira

- Enfileira
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento no final da fila
  - Incrementa o tamanho

```
FilaEncadeada F;
TipoItem x;

x.SetChave(7);
F.Enfileira(x);
```

```
void FilaEncadeada::Enfileira(TipoItem item) {
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     tras->prox = nova;
     tras = nova;
     tamanho++;
}
```

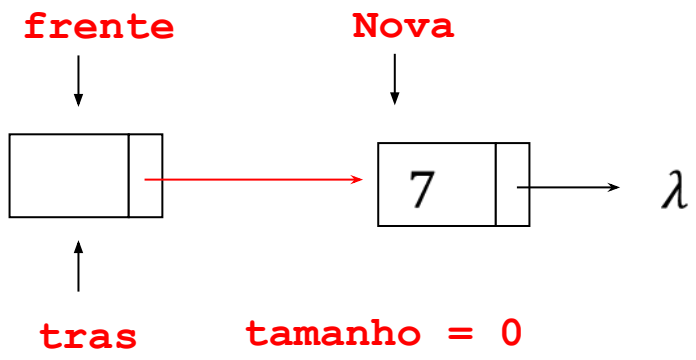


Class FilaEncadeada - Enfileira

- Enfileira
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento na fila
  - Incrementa o tamanho

```
FilaEncadeada F;
TipoItem x;

x.SetChave(7);
F.Enfileira(x);
```

```
void FilaEncadeada::Enfileira(TipoItem item) {
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     tras->prox = nova;
     tras = nova;
     tamanho++;
}
```

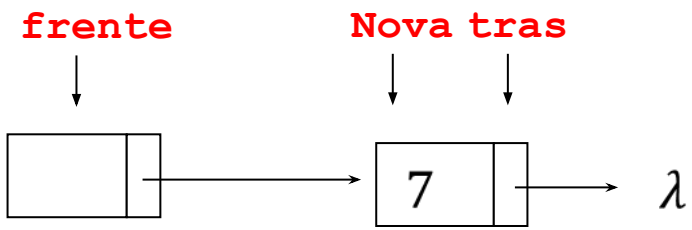


Class FilaEncadeada - Enfileira

- Enfileira
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento no final da fila
  - Incrementa o tamanho

```
FilaEncadeada F;
TipoItem x;

x.SetChave(7);
F.Enfileira(x);
```

```
void FilaEncadeada::Enfileira(TipoItem item) {
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     tras->prox = nova;
     tras = nova;
     tamanho++;
}
```



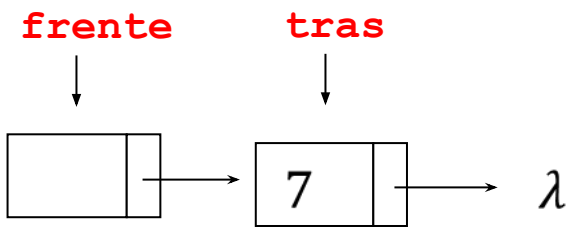
tamanho = 0

Class FilaEncadeada - Enfileira

- Enfileira
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento no final da fila
  - Incrementa o tamanho

```
FilaEncadeada F;
TipoItem x;

x.SetChave(7);
F.Enfileira(x);
```

```
void FilaEncadeada::Enfileira(TipoItem item) {
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    tras->prox = nova;
    tras = nova;
    tamanho++;
}
```



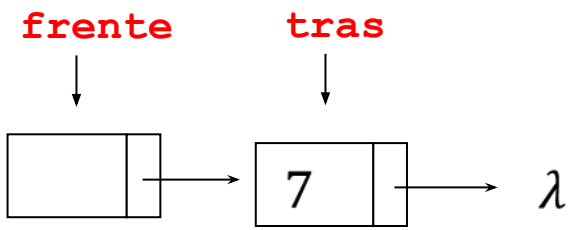
tamanho = 1

Class FilaEncadeada - Enfileira

- Enfileira
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento no final da fila
  - Incrementa o tamanho

```
FilaEncadeada F;
TipoItem x;

x.SetChave(1);
F.Enfileira(x);
```

```
void FilaEncadeada::Enfileira(TipoItem item) {
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    tras->prox = nova;
    tras = nova;
    tamanho++;
}
```



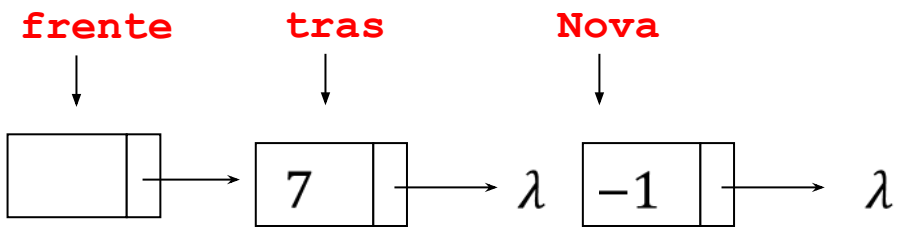
tamanho = 1

Class FilaEncadeada - Enfileira

- Enfileira
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento no final da fila
  - Incrementa o tamanho

```
FilaEncadeada F;
TipoItem x;

x.SetChave(1);
F.Enfileira(x);
```

```
void FilaEncadeada::Enfileira(TipoItem item) {
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    tras->prox = nova;
    tras = nova;
    tamanho++;
}
```



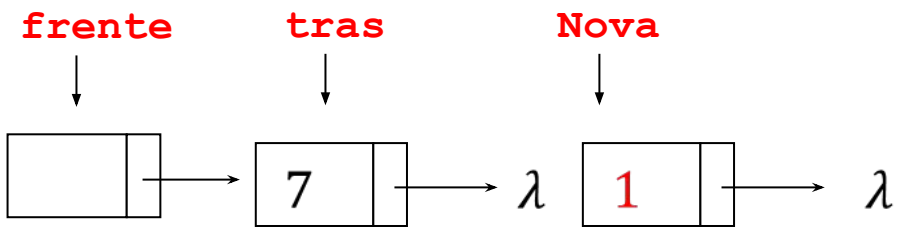
tamanho = 1

Class FilaEncadeada - Enfileira

- Enfileira
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento no final da fila
  - Incrementa o tamanho

```
FilaEncadeada F;
TipoItem x;

x.SetChave(1);
F.Enfileira(x);
```

```
void FilaEncadeada::Enfileira(TipoItem item) {
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    tras->prox = nova;
    tras = nova;
    tamanho++;
}
```



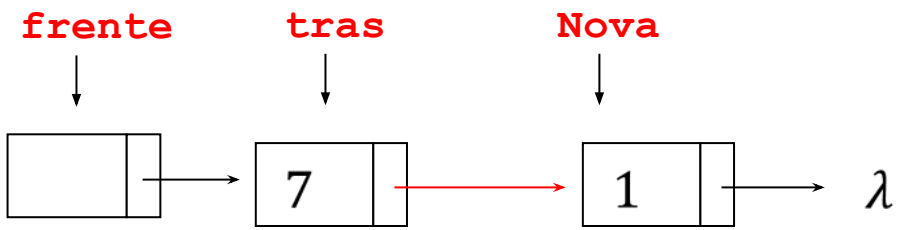
tamanho = 1

Class FilaEncadeada - Enfileira

- Enfileira
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento no final da fila
  - Incrementa o tamanho

```
FilaEncadeada F;
TipoItem x;

x.SetChave(1);
F.Enfileira(x);
```

```
void FilaEncadeada::Enfileira(TipoItem item) {
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    tras->prox = nova;
    tras = nova;
    tamanho++;
}
```



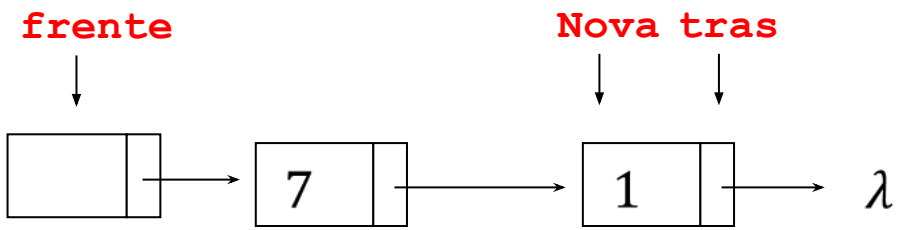
tamanho = 1

Class FilaEncadeada - Enfileira

- Enfileira
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento no final da fila
  - Incrementa o tamanho

```
FilaEncadeada F;
TipoItem x;

x.SetChave(1);
F.Enfileira(x);
```

```
void FilaEncadeada::Enfileira(TipoItem item) {
     TipoCelula *nova;

     nova = new TipoCelula();
     nova->item = item;
     tras->prox = nova;
     tras = nova;
     tamanho++;
}
```



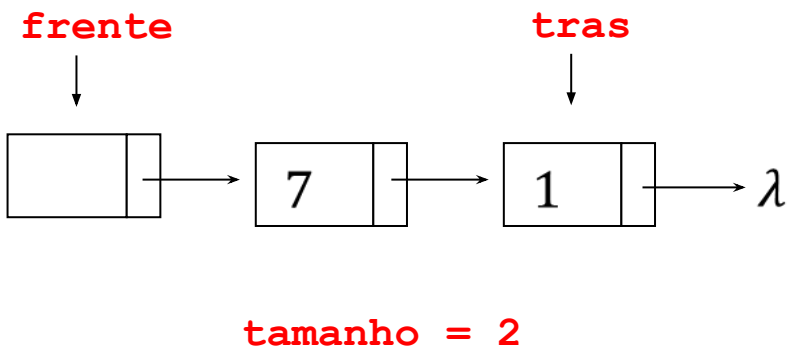
tamanho = 1

Class FilaEncadeada - Enfileira

- Enfileira
  - Cria uma nova célula
  - Coloca o item nessa célula
  - Faz o encadeamento no final da fila
  - Incrementa o tamanho

```
FilaEncadeada F;
TipoItem x;

x.SetChave(1);
F.Enfileira(x);
```

```
void FilaEncadeada::Enfileira(TipoItem item) {
    TipoCelula *nova;

    nova = new TipoCelula();
    nova->item = item;
    tras->prox = nova;
    tras = nova;
    tamanho++;
}
```

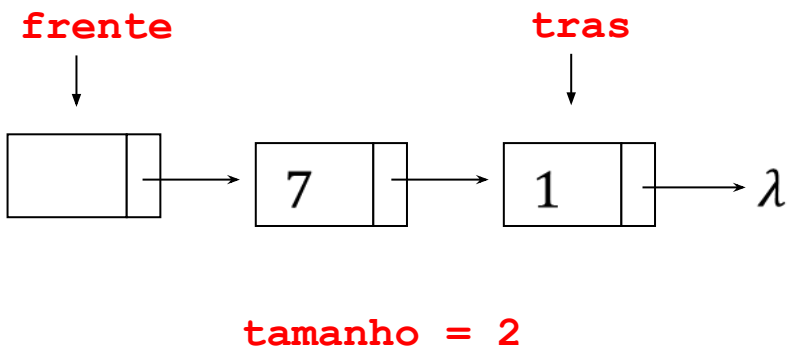


Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```

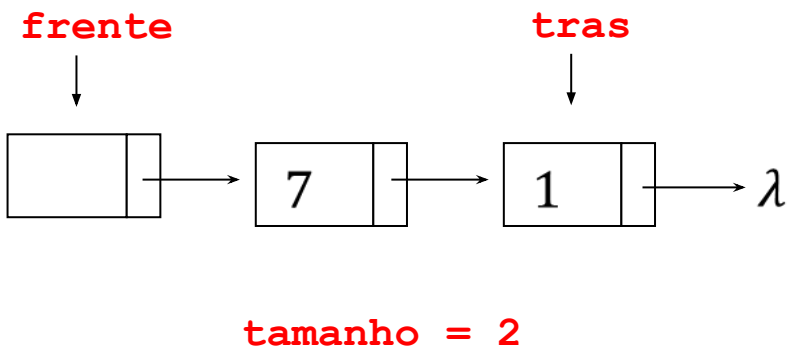


Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```

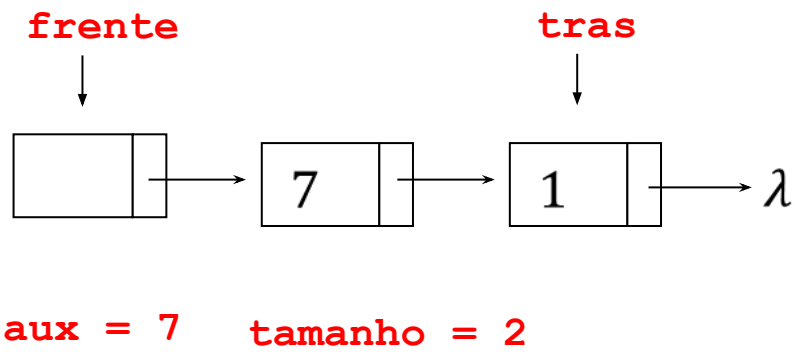


Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```

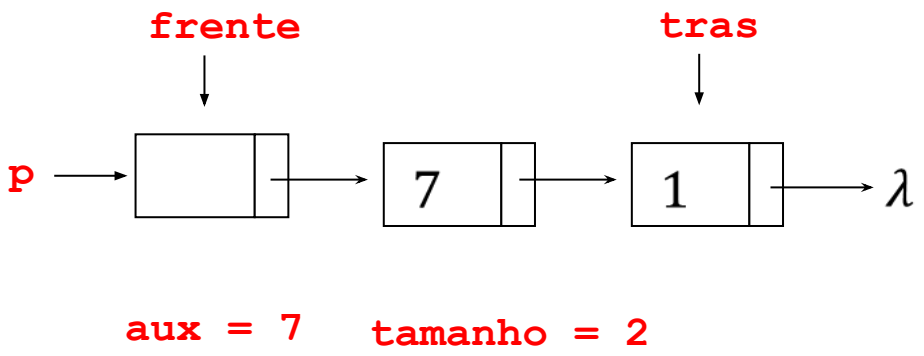


Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```

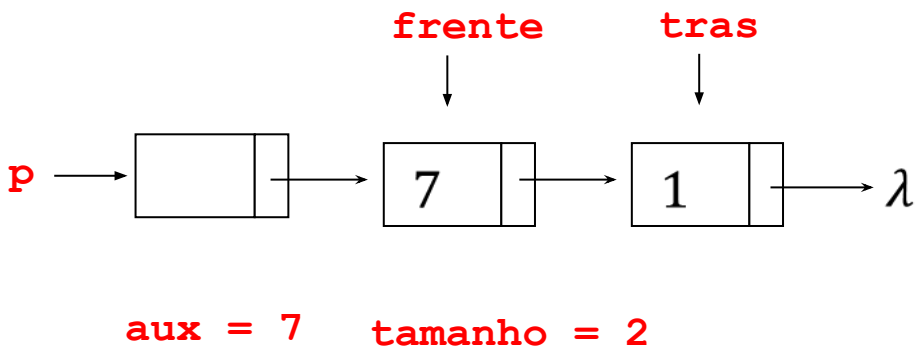


Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```

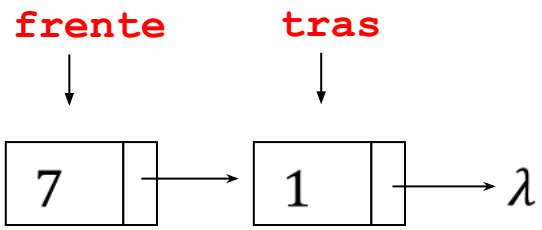


Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```



aux = 7 **tamanho = 2**

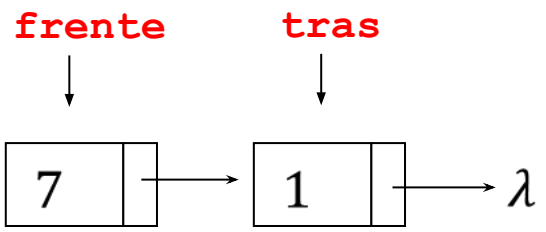
Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```

7



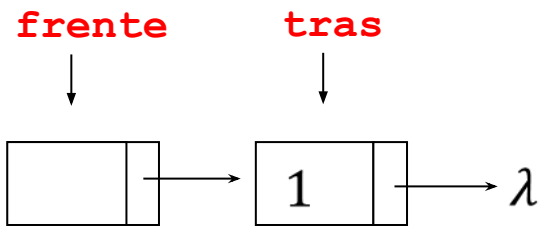
tamanho = 1

Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```



7

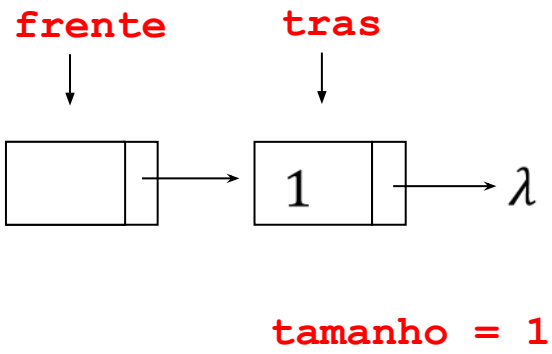
tamanho = 1

Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```

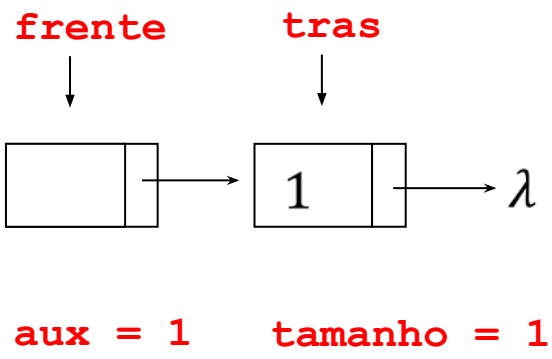


Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item ;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```

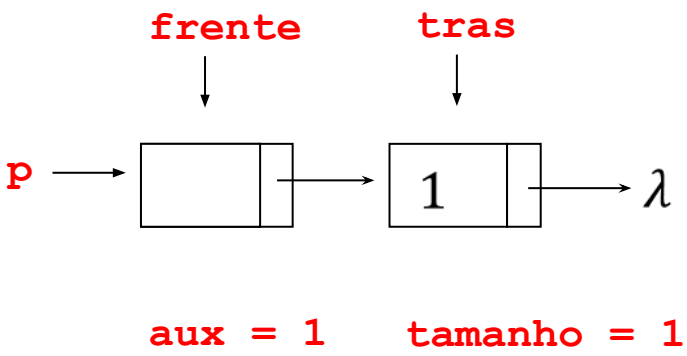


Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```

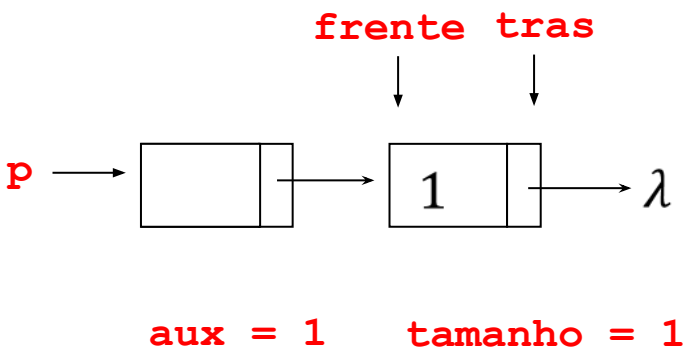


Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```

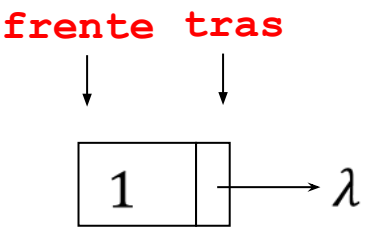


Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```



aux = 1 tamanho = 1

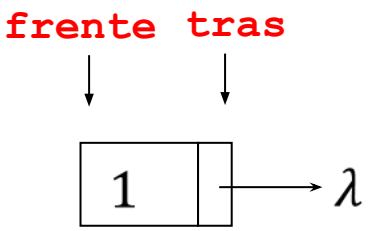
Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```

1



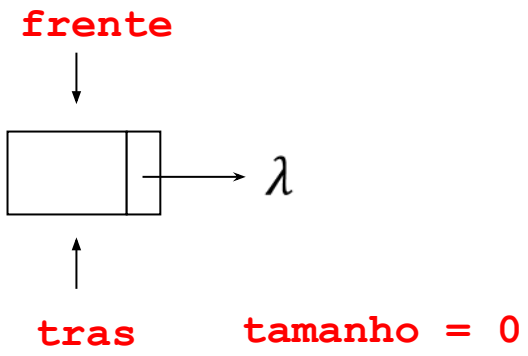
tamanho = 0

Class FilaEncadeada - Desenfileira

- Desenfileira
  - Retorna o elemento que está na primeira posição (`frente->prox`)
  - Remove a célula cabeça, e a primeira célula passa a ser a cabeça

```
FilaEncadeada F;
TipoItem x;

F.desenfileira(x);
x.Imprime()
```

```
TipoItem FilaEncadeada::Desenfileira() {
    TipoCelula *p;
    TipoItem aux;

    if (tamanho == 0)
        throw "Fila está vazia!";

    aux = frente->prox->item;
    p = frente;
    frente = frente->prox;
    delete p;
    tamanho--;
    return aux;
}
```

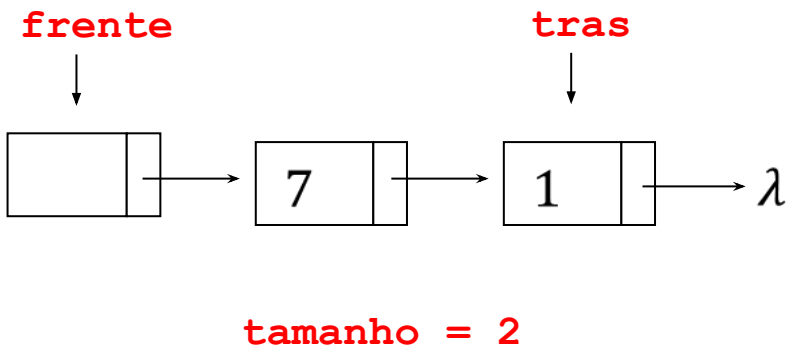


Class FilaEncadeada - Limpa

- Limpa
  - Remove todas as células da lista, restando apenas a célula cabeça
  - Seta o tamanho para 0
  - Atualiza o apontador tras

```
void FilaEncadeada::Limpa(){
    TipoCelula *p;

    p = frente->prox;
    while(p!=NULL){
        frente->prox = p->prox;
        delete p;
        p = frente->prox;
    }
    tamanho = 0;
    tras = frente;
}
```

```
FilaEncadeada F;
TipoItem x;

F.Limpa()
```

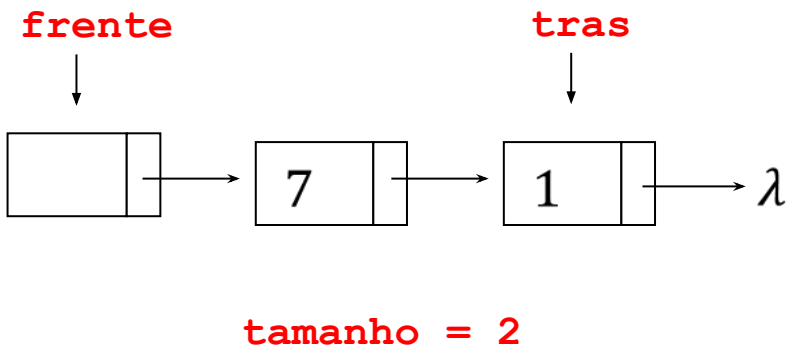


Class FilaEncadeada - Limpa

- Limpa
  - Remove todas as células da lista, restando apenas a célula cabeça
  - Seta o tamanho para 0
  - Atualiza o apontador tras

```
FilaEncadeada F;
TipoItem x;

F.Limpa()
```

```
void FilaEncadeada::Limpa(){
    TipoCelula *p;

    p = frente->prox;
    while(p!=NULL){
        frente->prox = p->prox;
        delete p;
        p = frente->prox;
    }
    tamanho = 0;
    tras = frente;
}
```



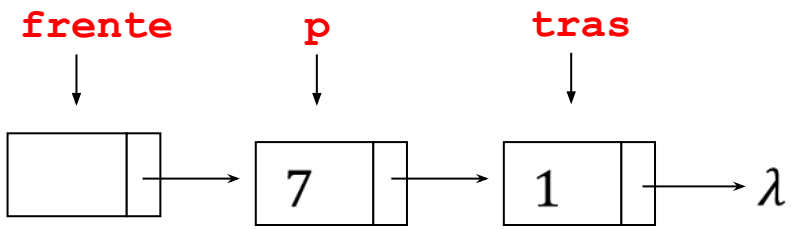
tamanho = 2

Class FilaEncadeada - Limpa

- Limpa
  - Remove todas as células da lista, restando apenas a célula cabeça
  - Seta o tamanho para 0
  - Atualiza o apontador tras

```
FilaEncadeada F;
TipoItem x;

F.Limpa()
```

```
void FilaEncadeada::Limpa(){
    TipoCelula *p;

    p = frente->prox;
    while(p!=NULL){
        frente->prox = p->prox;
        delete p;
        p = frente->prox;
    }
    tamanho = 0;
    tras = frente;
}
```



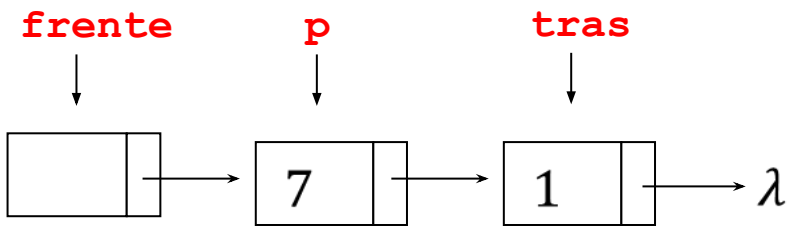
tamanho = 2

Class FilaEncadeada - Limpa

- Limpa
  - Remove todas as células da lista, restando apenas a célula cabeça
  - Seta o tamanho para 0
  - Atualiza o apontador tras

```
FilaEncadeada F;
TipoItem x;

F.Limpa()
```

```
void FilaEncadeada::Limpa(){
    TipoCelula *p;

    p = frente->prox;
    while(p!=NULL){
        frente->prox = p->prox;
        delete p;
        p = frente->prox;
    }
    tamanho = 0;
    tras = frente;
}
```



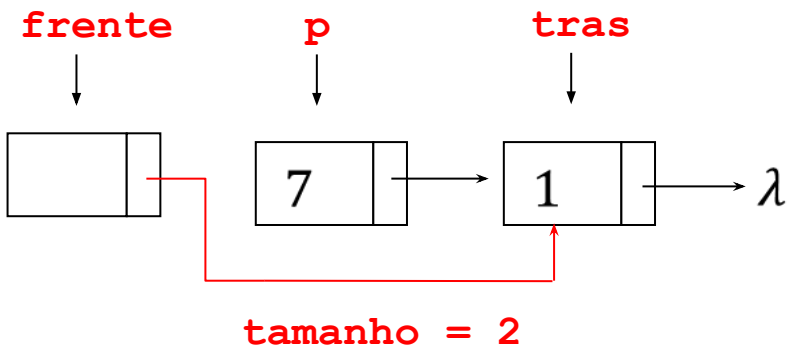
tamanho = 2

Class FilaEncadeada - Limpa

- Limpa
  - Remove todas as células da lista, restando apenas a célula cabeça
  - Seta o tamanho para 0
  - Atualiza o apontador tras

```
void FilaEncadeada::Limpa(){
    TipoCelula *p;

    p = frente->prox;
    while(p!=NULL){
        frente->prox = p->prox;
        delete p;
        p = frente->prox;
    }
    tamanho = 0;
    tras = frente;
}
```

```
FilaEncadeada F;
TipoItem x;

F.Limpa()
```

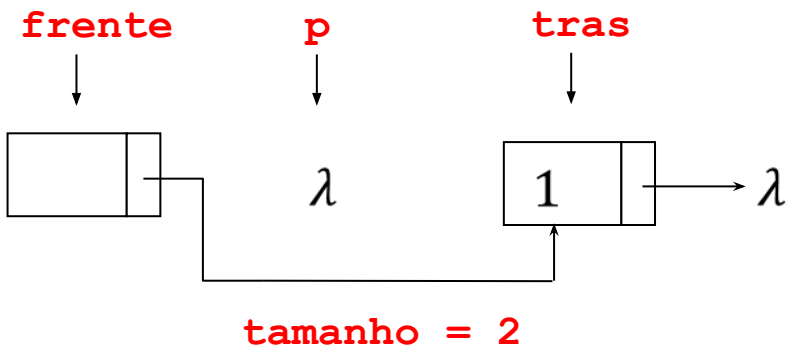


Class FilaEncadeada - Limpa

- Limpa
  - Remove todas as células da lista, restando apenas a célula cabeça
  - Seta o tamanho para 0
  - Atualiza o apontador tras

```
void FilaEncadeada::Limpa(){
    TipoCelula *p;

    p = frente->prox;
    while(p!=NULL){
        frente->prox = p->prox;
        delete p;
        p = frente->prox;
    }
    tamanho = 0;
    tras = frente;
}
```

```
FilaEncadeada F;
TipoItem x;

F.Limpa()
```

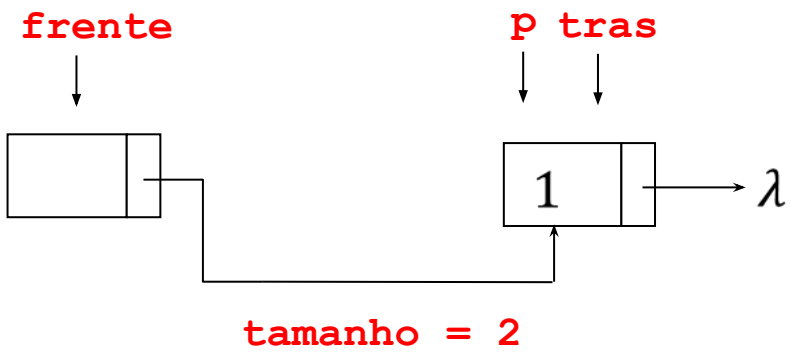


Class FilaEncadeada - Limpa

- Limpa
  - Remove todas as células da lista, restando apenas a célula cabeça
  - Seta o tamanho para 0
  - Atualiza o apontador tras

```
void FilaEncadeada::Limpa(){
    TipoCelula *p;

    p = frente->prox;
    while(p!=NULL){
        frente->prox = p->prox;
        delete p;
        p = frente->prox;
    }
    tamanho = 0;
    tras = frente;
}
```

```
FilaEncadeada F;
TipoItem x;

F.Limpa()
```

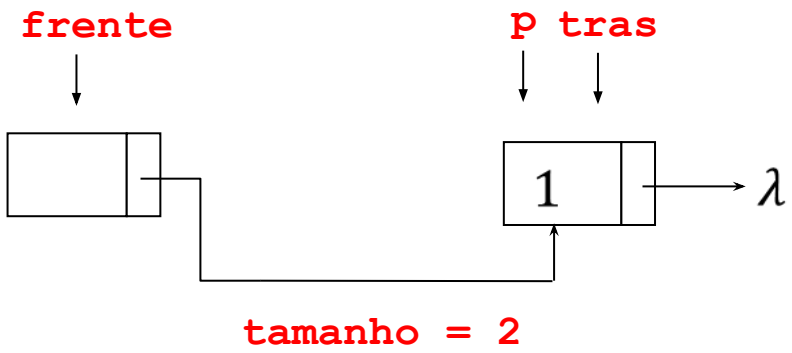


Class FilaEncadeada - Limpa

- Limpa
  - Remove todas as células da lista, restando apenas a célula cabeça
  - Seta o tamanho para 0
  - Atualiza o apontador tras

```
void FilaEncadeada::Limpa(){
    TipoCelula *p;

    p = frente->prox;
    while(p!=NULL){
        frente->prox = p->prox;
        delete p;
        p = frente->prox;
    }
    tamanho = 0;
    tras = frente;
}
```

```
FilaEncadeada F;
TipoItem x;

F.Limpa()
```

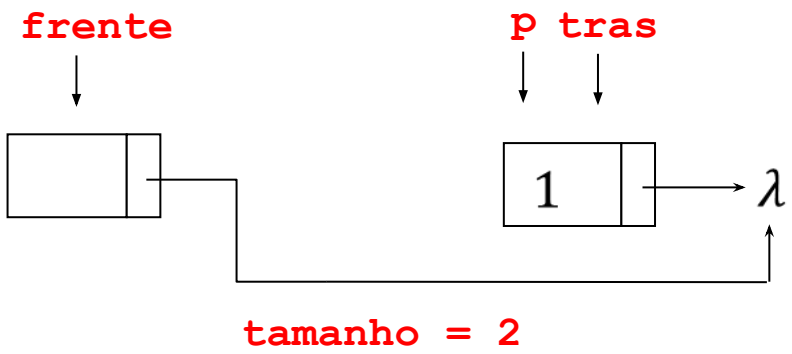


Class FilaEncadeada - Limpa

- Limpa
  - Remove todas as células da lista, restando apenas a célula cabeça
  - Seta o tamanho para 0
  - Atualiza o apontador tras

```
void FilaEncadeada::Limpa(){
    TipoCelula *p;

    p = frente->prox;
    while(p!=NULL){
        frente->prox = p->prox;
        delete p;
        p = frente->prox;
    }
    tamanho = 0;
    tras = frente;
}
```

```
FilaEncadeada F;
TipoItem x;

F.Limpa()
```

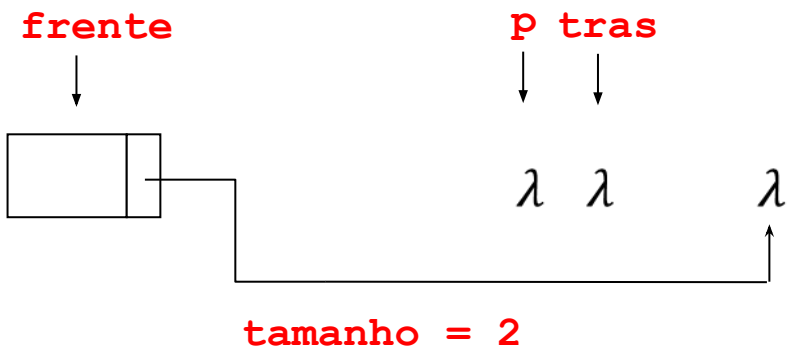


Class FilaEncadeada - Limpa

- Limpa
  - Remove todas as células da lista, restando apenas a célula cabeça
  - Seta o tamanho para 0
  - Atualiza o apontador tras

```
void FilaEncadeada::Limpa(){
    TipoCelula *p;

    p = frente->prox;
    while(p!=NULL){
        frente->prox = p->prox;
        delete p;
        p = frente->prox;
    }
    tamanho = 0;
    tras = frente;
}
```

```
FilaEncadeada F;
TipoItem x;

F.Limpa()
```

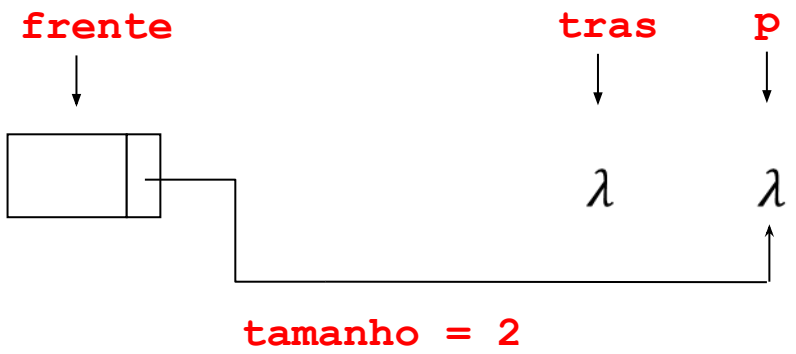


Class FilaEncadeada - Limpa

- Limpa
  - Remove todas as células da lista, restando apenas a célula cabeça
  - Seta o tamanho para 0
  - Atualiza o apontador tras

```
void FilaEncadeada::Limpa(){
    TipoCelula *p;

    p = frente->prox;
    while(p!=NULL){
        frente->prox = p->prox;
        delete p;
        p = frente->prox ;
    }
    tamanho = 0;
    tras = frente;
}
```

```
FilaEncadeada F;
TipoItem x;

F.Limpa()
```

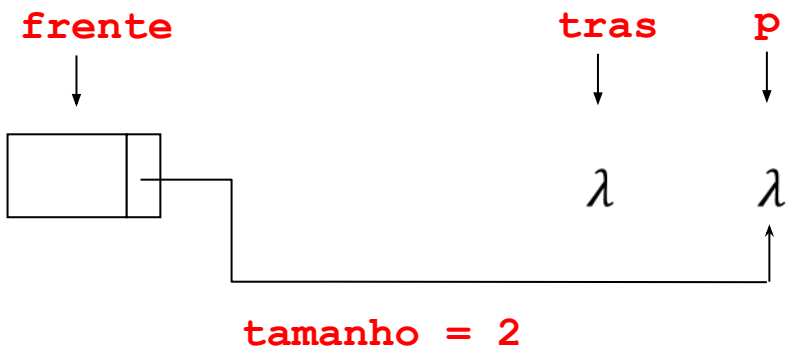


Class FilaEncadeada - Limpa

- Limpa
  - Remove todas as células da lista, restando apenas a célula cabeça
  - Seta o tamanho para 0
  - Atualiza o apontador tras

```
void FilaEncadeada::Limpa(){
    TipoCelula *p;

    p = frente->prox;
    while(p!=NULL){
        frente->prox = p->prox;
        delete p;
        p = frente->prox ;
    }
    tamanho = 0;
    tras = frente;
}
```

```
FilaEncadeada F;
TipoItem x;

F.Limpa()
```

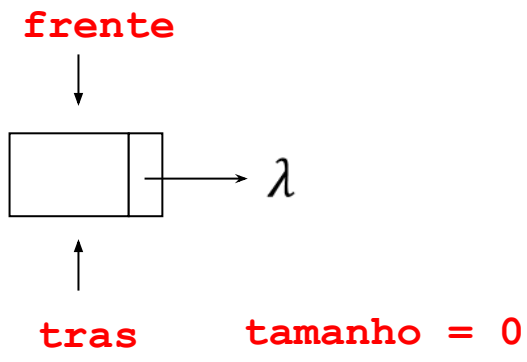


Class FilaEncadeada - Limpa

- Limpa
  - Remove todas as células da lista, restando apenas a célula cabeça
  - Seta o tamanho para 0
  - Atualiza o apontador tras

```
FilaEncadeada F;
TipoItem x;

F.Limpa()
```

```
void FilaEncadeada::Limpa(){
    TipoCelula *p;

    p = frente->prox;
    while(p!=NULL){
        frente->prox = p->prox;
        delete p;
        p = frente->prox ;
    }
    tamanho = 0;
    tras = frente;
}
```



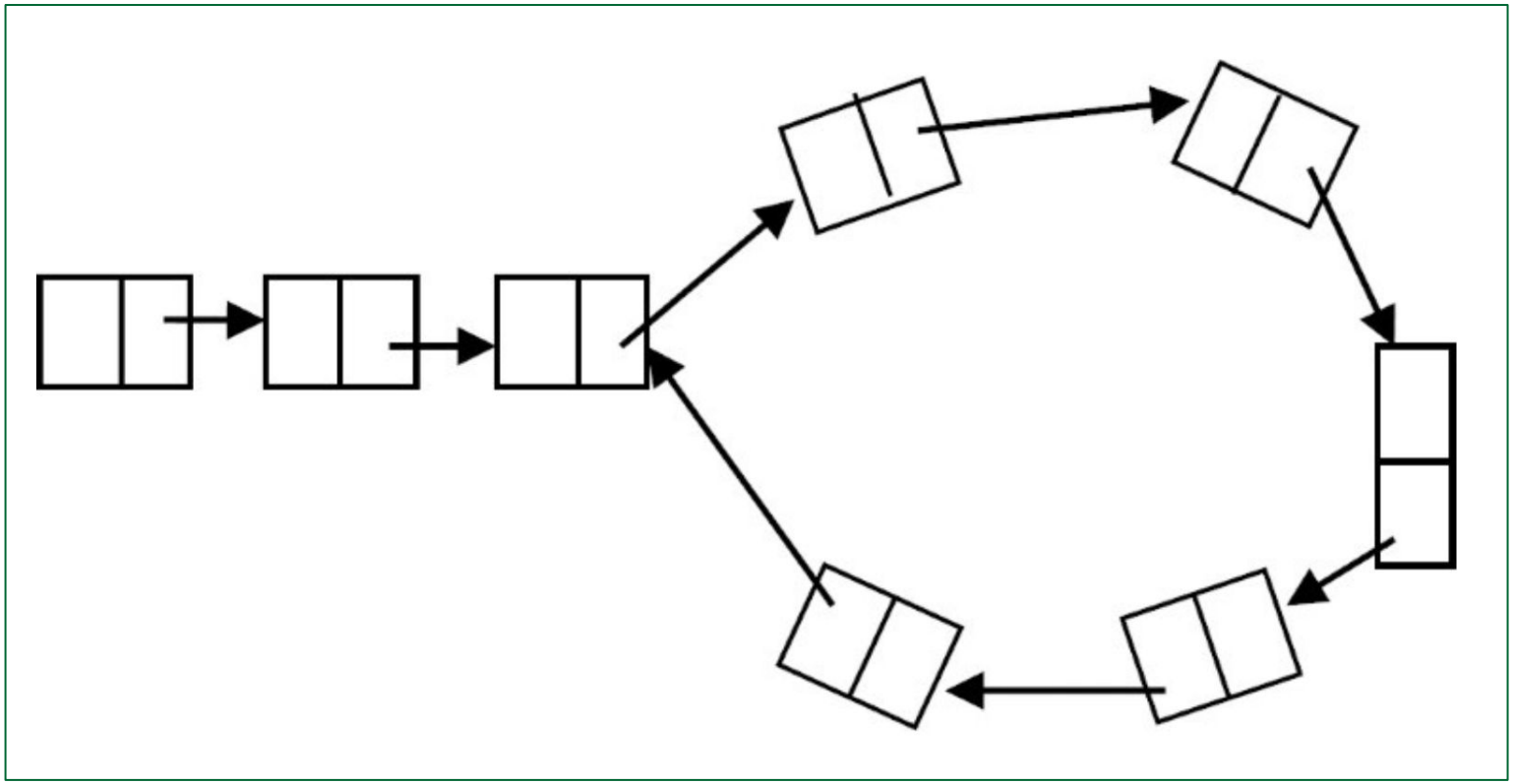
Fila Arranjo x Fila Encadeada

	Fila Arranjo	Fila Encadeada
Construtor	**O(1)**	**O(1)**
Destrutor	**-**	O(n)
Enfileira	**O(1)**	**O(1)**
Desenfileira	**O(1)**	**O(1)**
Limpa	**O(1)**	O(n)
Tamanho	Fixo	**Dinâmico**
Memória Extra	**Não**	Sim
Implementação Simples	**Sim**	Não

Em geral, por sua simplicidade, a implementação por arranjo é mais indicada, a menos que a questão de tamanho dinâmico seja de fundamental importância

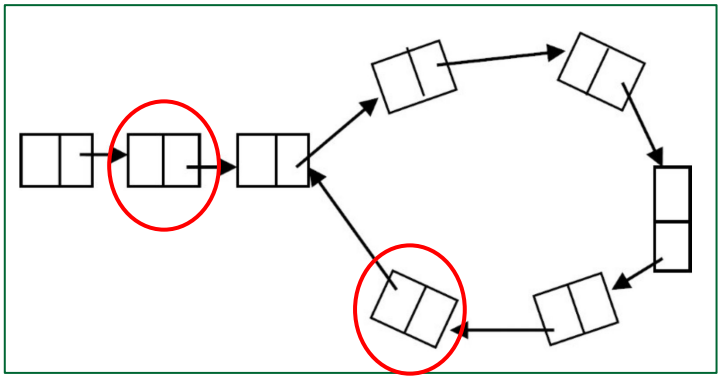
Lista - exemplo de uso

Verifique se uma lista contém um ciclo ou termina em um ponteiro nulo



Lista - exemplo de uso

Verifique se uma lista contém um ciclo ou termina em um ponteiro nulo



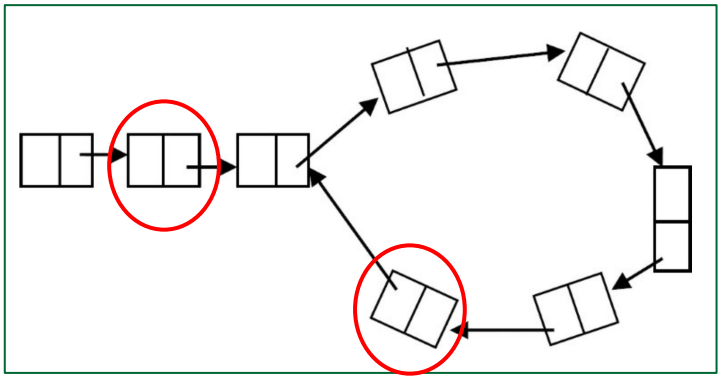
Força bruta

Nesta lista existem dois nós que possuem o mesmo ponteiro *próximo*

Isso não acontece em uma lista sem ciclo

Lista - exemplo de uso

Verifique se uma lista contém um ciclo ou termina em um ponteiro nulo



Força bruta

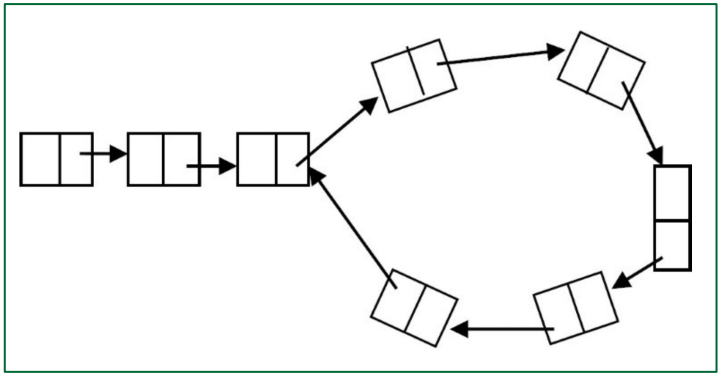
Precisamos de uma maneira eficiente de guardar e consultar os endereços já vistos

Tempo: n * (custo de consultar um endereço)

Espaço: O(n)

Lista - exemplo de uso

Verifique se uma lista contém um ciclo ou termina em um ponteiro nulo



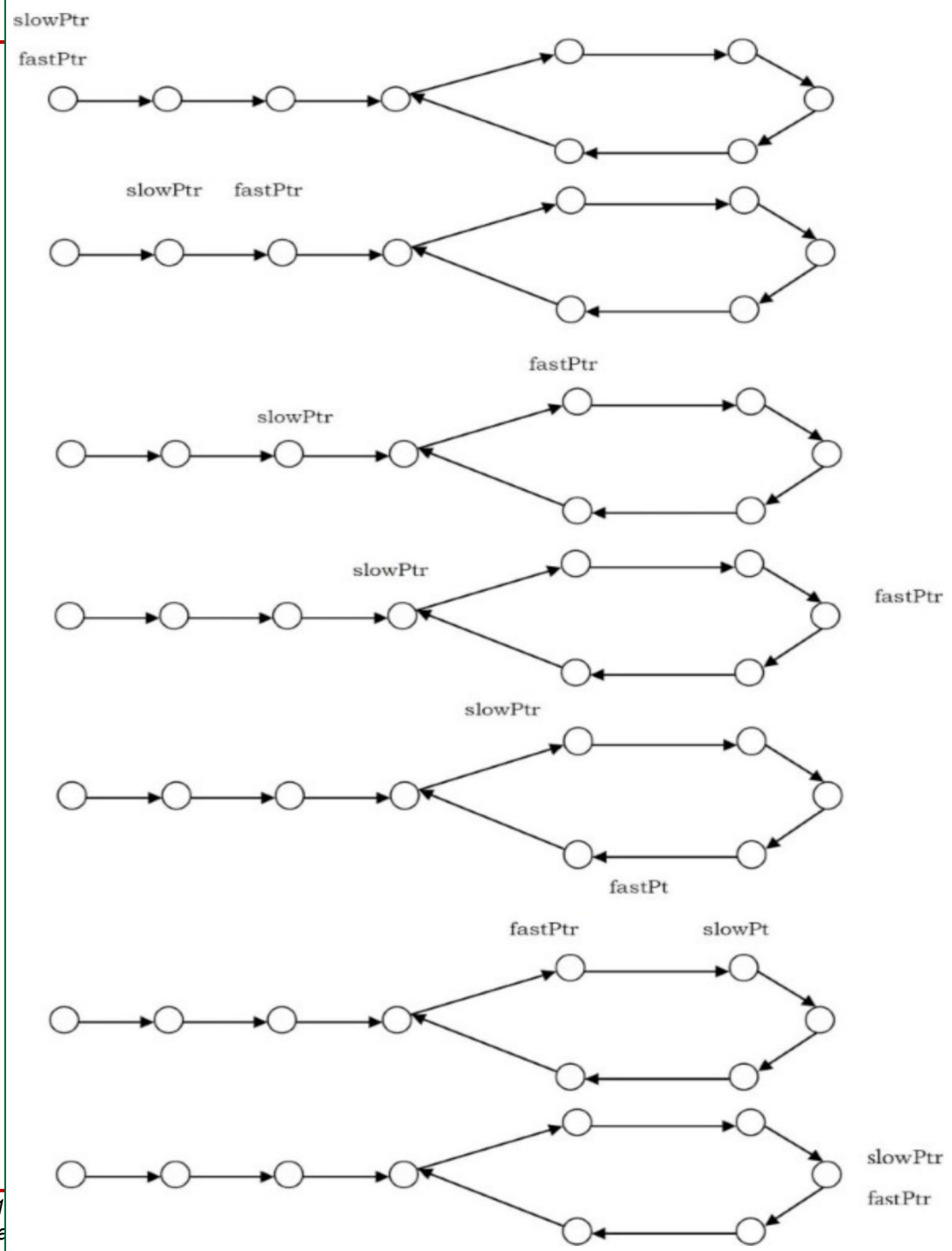
Tempo: O(n) Espaço: O(1)?

Algoritmo de Floyd (lebre/tartaruga)

- dois ponteiros que percorrem a lista em velocidades diferentes

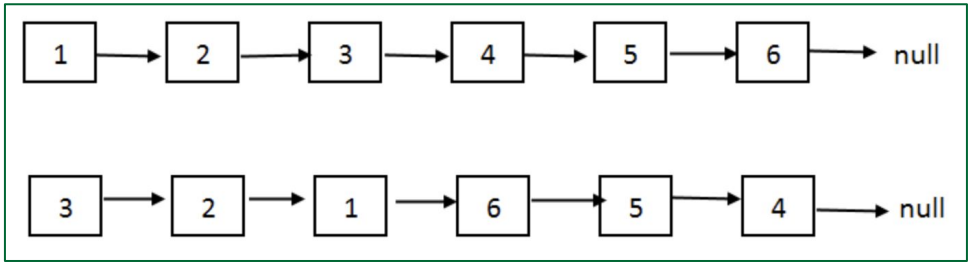
    slowPtr - tartaruga: um ponteiro por vez

    fastPtr - lebre: dois ponteiros por vez


slowPtr
fastPtr
slowPtr
fastPtr
fastPtr
slowPtr
slowPtr
fastPtr
slowPtr
fastPt
fastPtr
slowPt
slowPtr
fastPtr

Listas

Como inverter uma lista encadeada? T:O(n) E:O(1)



Como encontrar o elemento do meio da lista?

T: O(n) E: O(1)

Pilha - exemplo de uso

Balanceamento de símbolos

Example	Valid?	Description
(A+B)+(C-D)	Yes	The expression has a balanced symbol
((A+B)+(C-D)	No	One closing brace is missing
((A+B)+[C-D])	Yes	Opening and immediate closing braces correspond
((A+B)+[C-D]}	No	The last closing brace does not correspond with the first opening parenthesis

- cria pilha vazia
- enquanto (não fim da entrada)
  - se curr_char não símbolo a ser balanceado, ignorar
  - se curr_char = (,[,{ , empilha
  - se curr_char =),],}
    - se pilha vazia retorna erro
    - senão desempilha
      - se char desempilhado não casa com curr_char
        - retorna erro
- Se pilha não vazia: retorna erro

Pilha

Implemente a função *get_minimum* em O(1)

retorna o menor elemento da pilha corrente

Filas

Como implementar uma fila usando duas pilhas? Qual a complexidade das operações:

enfileira?

desenfileira?

Qual a estrutura mais apropriada para imprimir os elementos de uma fila em ordem inversa?