Aula 12 - Técnicas de Encadeamento (Circular)

Prof. Me. Claudiney R. Tinoco

profclaudineytinoco@gmail.com

Faculdade de Computação (FACOM)
Bacharelado em Ciência da Computação (BCC)
Bacharelado em Sistemas de Informação (BSI)

Algoritmos e Estruturas de Dados 1 (AED1)
GBC024 - GSI006

Introdução

- Diferentes técnicas de encadeamento podem ser aplicadas a fim de **gerar algoritmos mais simples e/ou eficientes**

Introdução

- Diferentes técnicas de encadeamento podem ser aplicadas a fim de **gerar algoritmos mais simples e/ou eficientes**

- **Técnicas mais usuais:**
  - Uso do nó cabeçalho
  - Encadeamento circular
  - Encadeamento duplo

Introdução

- Diferentes técnicas de encadeamento podem ser aplicadas a fim de **gerar algoritmos mais simples e/ou eficientes**

- **Técnicas mais usuais:**
  - Uso do nó cabeçalho
  - **Encadeamento circular**
  - Encadeamento duplo

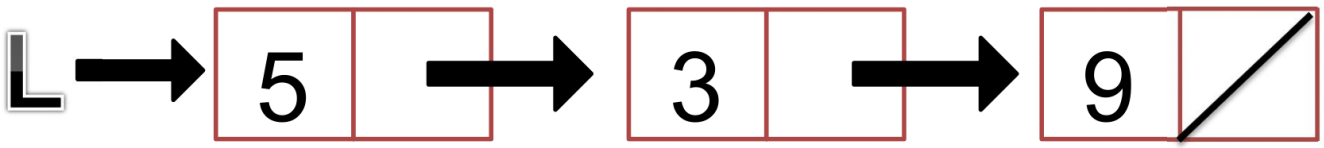
Encadeamento Circular

- Permite o **acesso direto** ao 1º e ao último elementos da lista

Encadeamento Circular

- Permite o **acesso direto** ao 1º e ao último elementos da lista

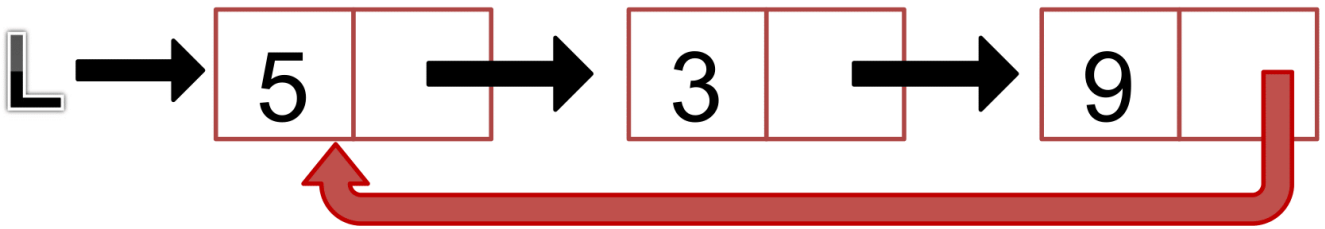
- **Mudança na implementação:**



Encadeamento Circular

- Permite o **acesso direto** ao 1º e ao último elementos da lista

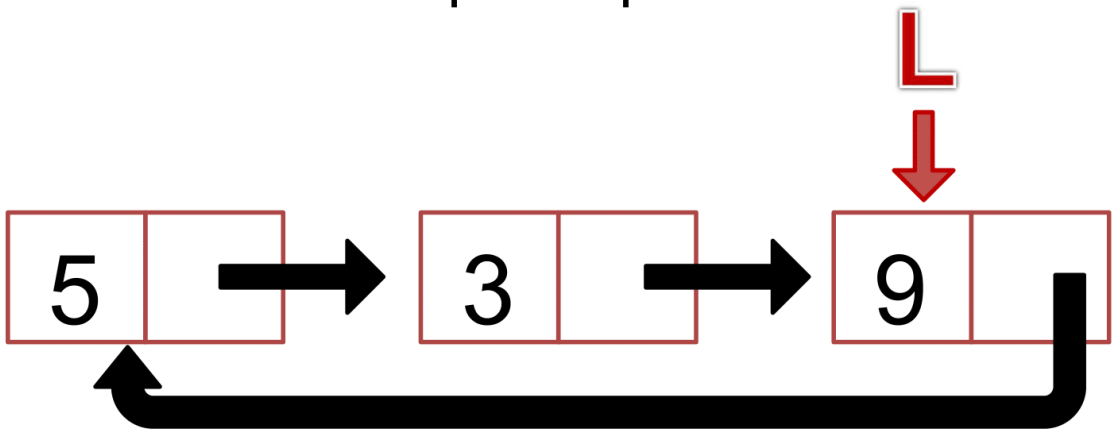
- **Mudança na implementação:**
  - Último nó aponta para o 1º nó da lista



Encadeamento Circular

- Permite o **acesso direto** ao 1º e ao último elementos da lista

- **Mudança na implementação:**
  - Último nó aponta para o 1º nó da lista
  - O ponteiro da Lista aponta para o último nó



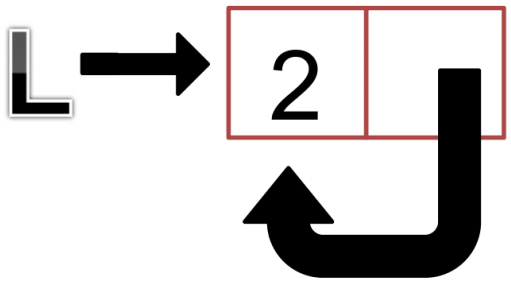
Encadeamento Circular

- **Lista vazia** (**Ex:** L= { })

Encadeamento Circular

- **Lista vazia** (**Ex:** L= { })

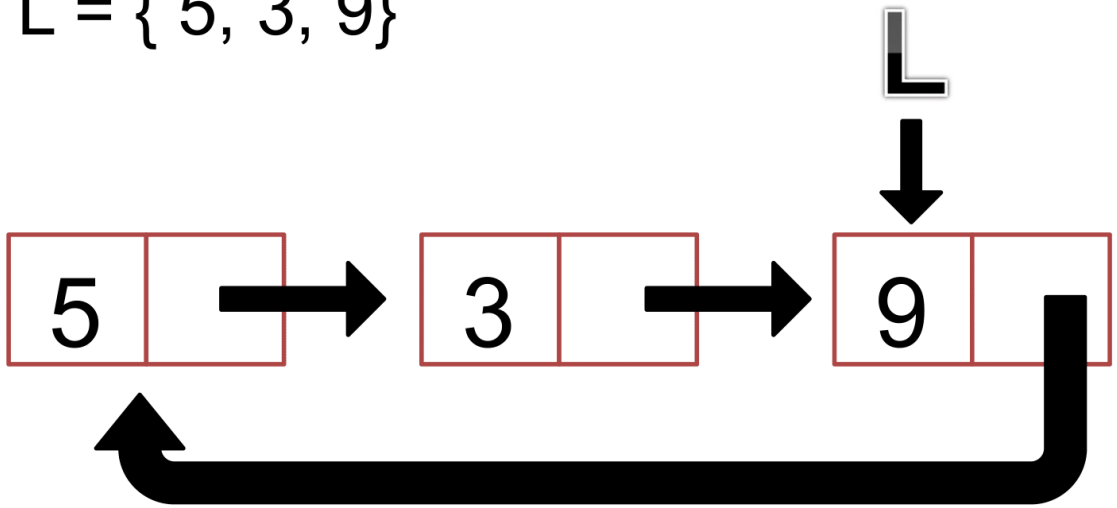
- **Lista com 1 único elemento** (**Ex:** L= {2})



Encadeamento Circular

- **Lista com mais de um elemento:**

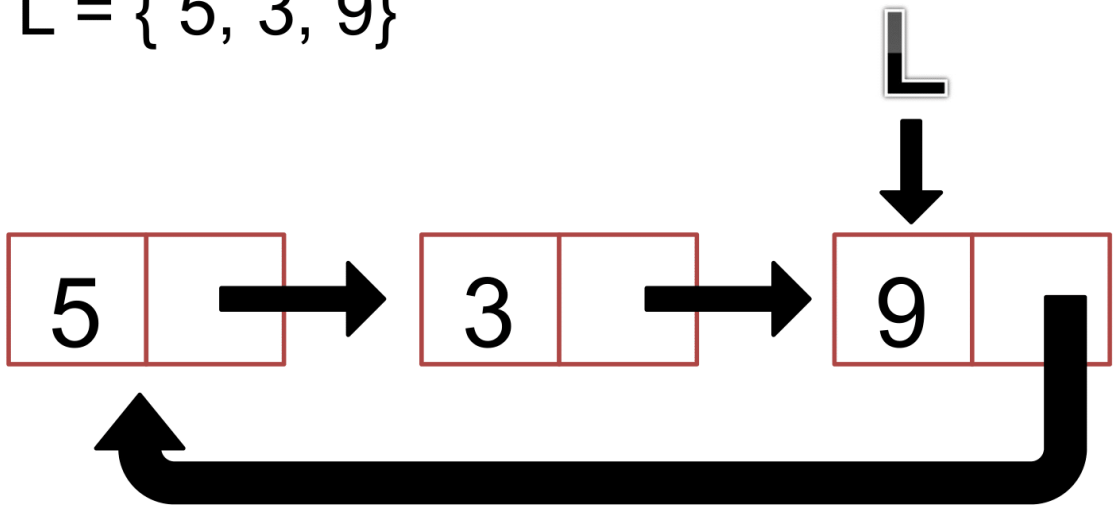
  Ex: L = { 5, 3, 9}



Encadeamento Circular

– **Lista com mais de um elemento:**

Ex: L = { 5, 3, 9}

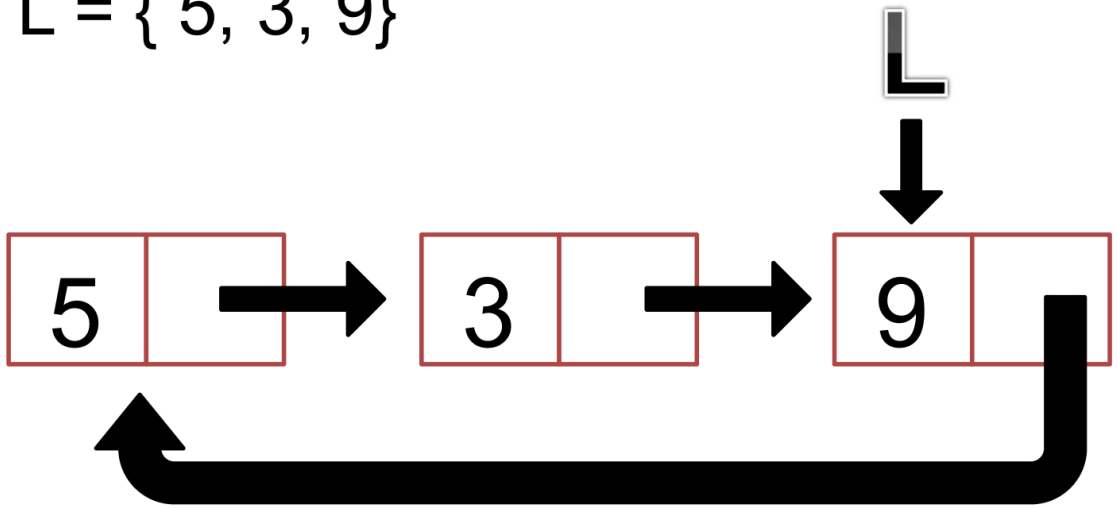


Informação do último nó: ***L->info***

Encadeamento Circular

– **Lista com mais de um elemento:**

Ex: L = { 5, 3, 9}



Informação do último nó: ***L->info***

Informação do 1o nó: ***L->prox->info***

Encadeamento Circular

- Facilita a implementação das operações necessária para a **Fila** (**FIFO**):

Encadeamento Circular

- Facilita a implementação das operações necessária para a **Fila** (**FIFO**):
  - **Insere no final** da Lista

Encadeamento Circular

- Facilita a implementação das operações necessária para a **Fila** (**FIFO**):
  - **Insere no final** da Lista
  - **Remoção no início** da Lista

Encadeamento Circular

- Facilita a implementação das operações necessária para a **Fila** (**FIFO**):
  - **Insere no final** da Lista
  - **Remoção no início** da Lista

- **Estrutura de representação permanece a MESMA** utilizada no encadeamento simples:

Encadeamento Circular

- Facilita a implementação das operações necessária para a **Fila** (**FIFO**):
  - **Insere no final** da Lista
  - **Remoção no início** da Lista

- **Estrutura de representação permanece a MESMA** utilizada no encadeamento simples:

Encadeamento Circular

- Facilita a implementação das operações necessária para a **Fila** (**FIFO**):
  - **Insere no final** da Lista
  - **Remoção no início** da Lista

- **Estrutura de representação permanece a MESMA** utilizada no encadeamento simples:

Estrutura de Representação em C

- Declaração da estrutura nó inteiro no **lista.c**:

```
struct no {
        int info;
        struct no * prox;
};
```

- Definição do tipo de dado lista no **lista.h**:

```
typedef struct no * Lista;
```

Encadeamento Circular

- Operações ***cria_lista*** **e** ***lista_vazia*** **também são IDÊNTICAS** ao encadeamento simples

Encadeamento Circular

- Operações ***cria_lista*** **e** ***lista_vazia*** **também são IDÊNTICAS** ao encadeamento simples

```
Lista cria_lista() {
    return NULL;
}
```

Encadeamento Circular

- Operações ***cria_lista*** **e** ***lista_vazia*** **também são IDÊNTICAS** ao encadeamento simples

```
Lista cria_lista() {
    return NULL;
}
```

```
int lista_vazia (Lista lst) {
    if (lst == NULL)
        return 1;
    else
        return 0;
}
```

Encadeamento Circular

- Analisaremos as seguintes operações para o **TAD lista não-ordenada**:
  - Inserir no final
  - Remover no início

Insere no Final

- Existem **3 cenários** possíveis:
  - Lista vazia
  - Lista com um único nó
  - Lista com mais de um nó

Insere no Final

- Lista vazia:

Ex: inserir 5

Insere no Final

- Lista vazia:
  - Aloca o novo nó

Ex: inserir 5

Insere no Final

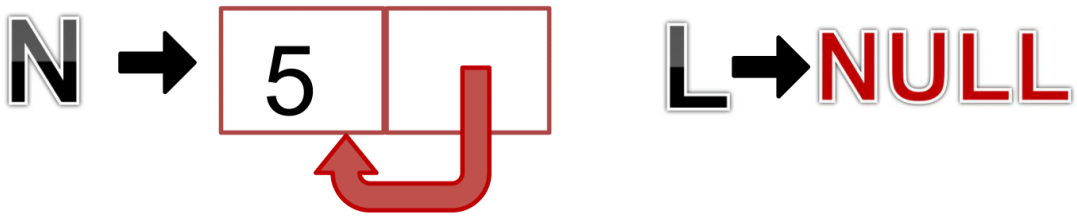
- Lista vazia:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento

Ex: inserir 5

Insere no Final

- Lista vazia:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento
    - Campo ***prox*** aponta para o 1º nó (o próprio nó)

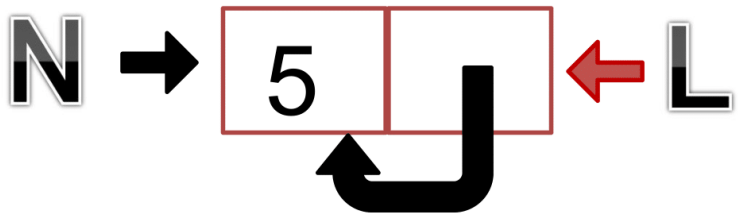
Ex: inserir 5



Insere no Final

- Lista vazia:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento
    - Campo ***prox*** aponta para o 1º nó (o próprio nó)
  - Faz a lista apontar para o último nó, ou seja, o **novo nó**

Ex: inserir 5

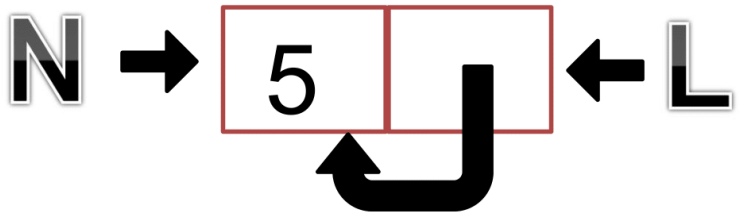


Insere no Final

- Lista vazia:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento
    - Campo ***prox*** aponta para o 1º nó (o próprio nó)
  - Faz a lista apontar para o último nó da lista (**novo nó**)

Ex: inserir 5

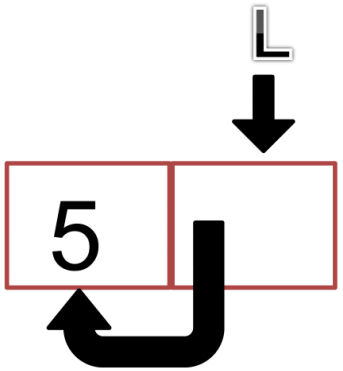
retorna 1



Insere no Final

- Lista com um único nó:

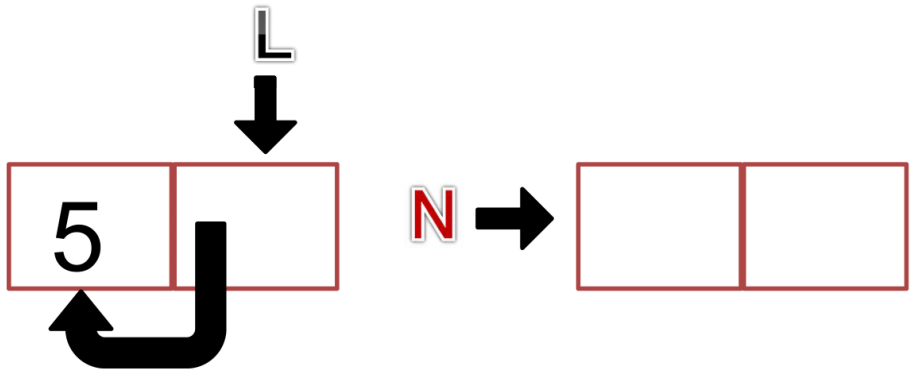
Ex: inserir 2



Insere no Final

- Lista com um único nó:
  - Aloca o novo nó

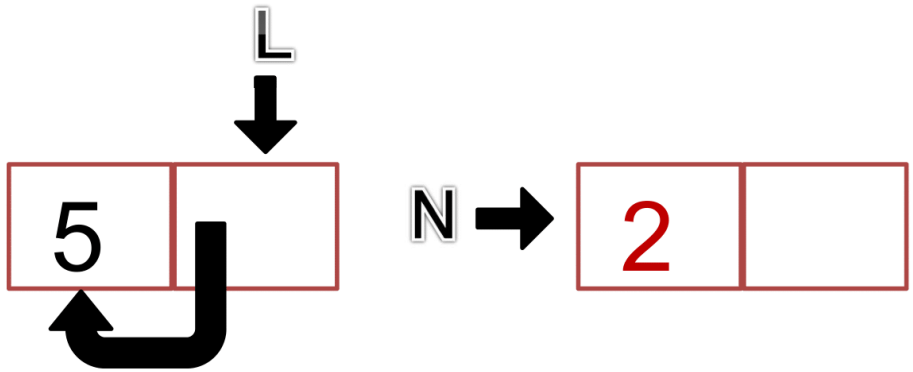
Ex: inserir 2



Insere no Final

- Lista com um único nó:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento

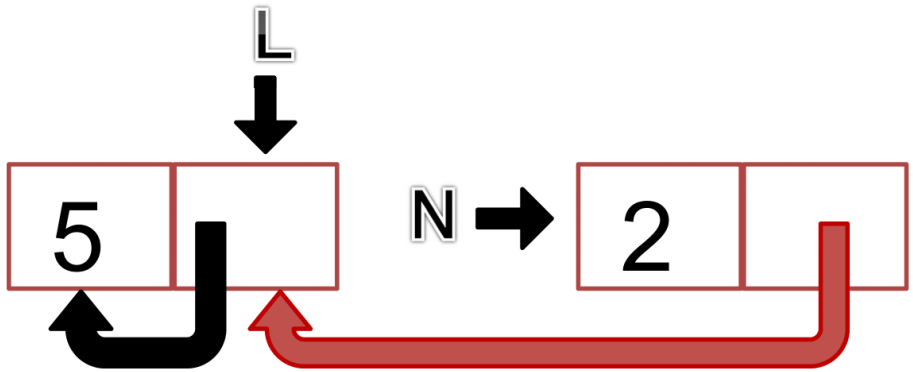
Ex: inserir 2



Insere no Final

- Lista com um único nó:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento
    - Campo ***prox*** aponta para o 1º nó (***L->prox***)

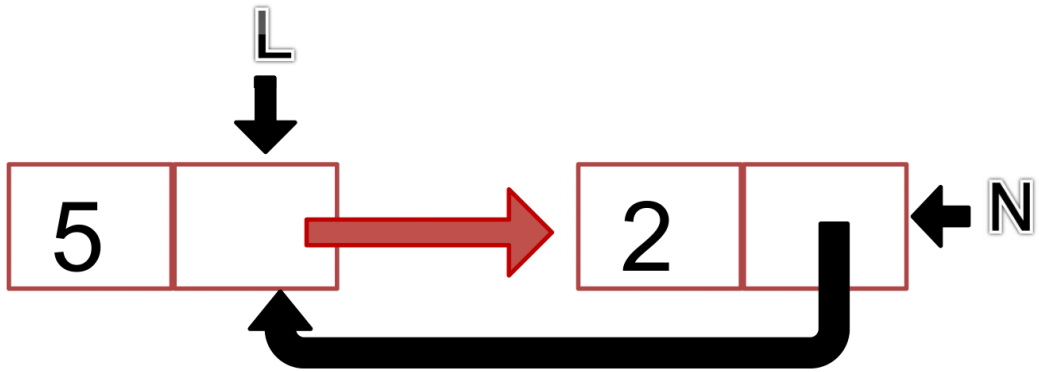
Ex: inserir 2



Insere no Final

- Lista com um único nó:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento
    - Campo ***prox*** aponta para o 1º nó (***L->prox***)
  - Faz a o último nó apontar para o novo nó

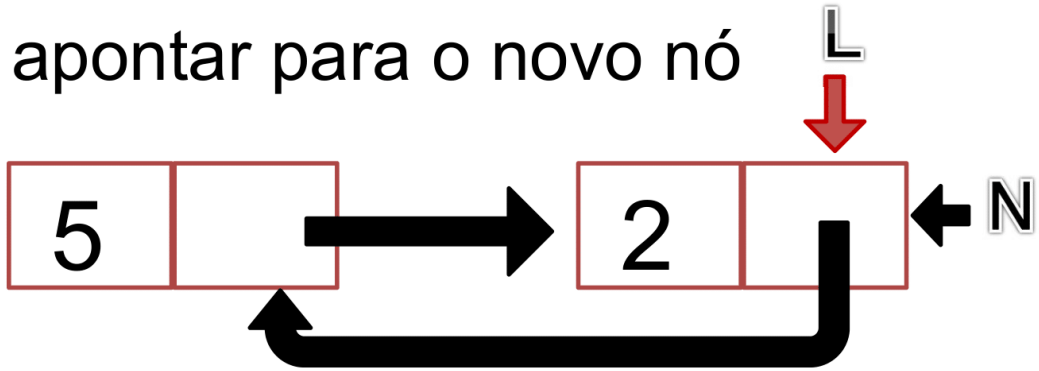
Ex: inserir 2



Insere no Final

- Lista com um único nó:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento
    - Campo ***prox*** aponta para o 1º nó (***L->prox***)
  - Faz a o último nó apontar para o novo nó
  - Faz a lista apontar para o novo nó

Ex: inserir 2

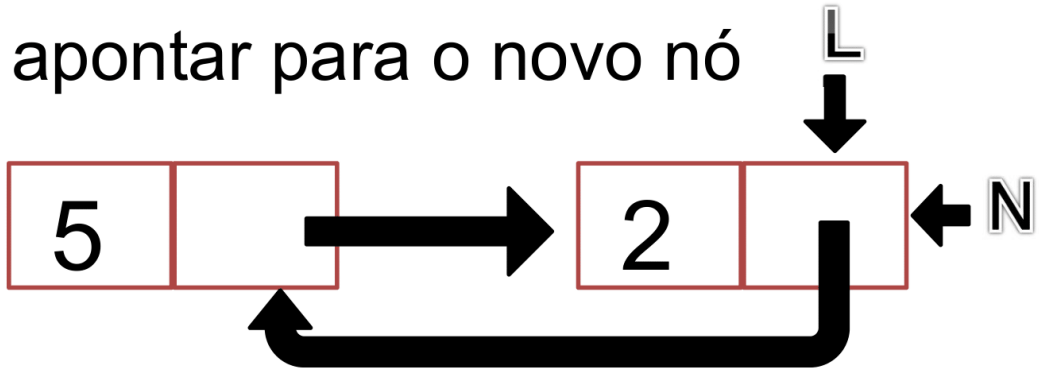


Insere no Final

- Lista com um único nó:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento
    - Campo ***prox*** aponta para o 1º nó (***L->prox***)
  - Faz a o último nó apontar para o novo nó
  - Faz a lista apontar para o novo nó

Ex: inserir 2

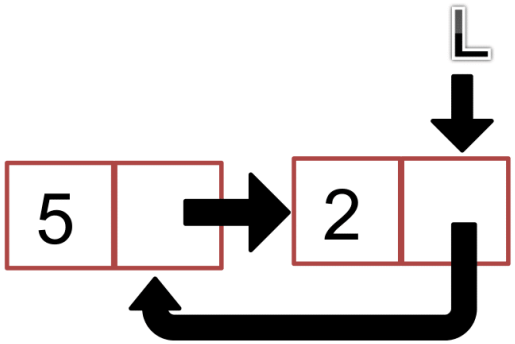
retorna 1



Insere no Final

- Lista com mais de um nó:

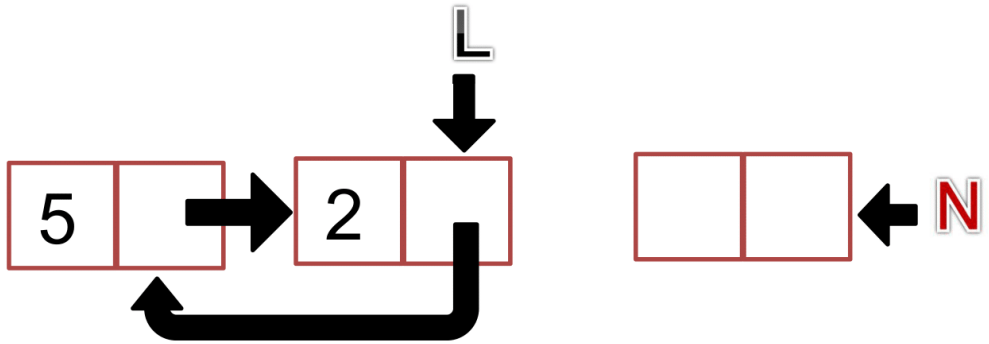
Ex: inserir 8



Insere no Final

- Lista com mais de um nó:
  - Aloca o novo nó

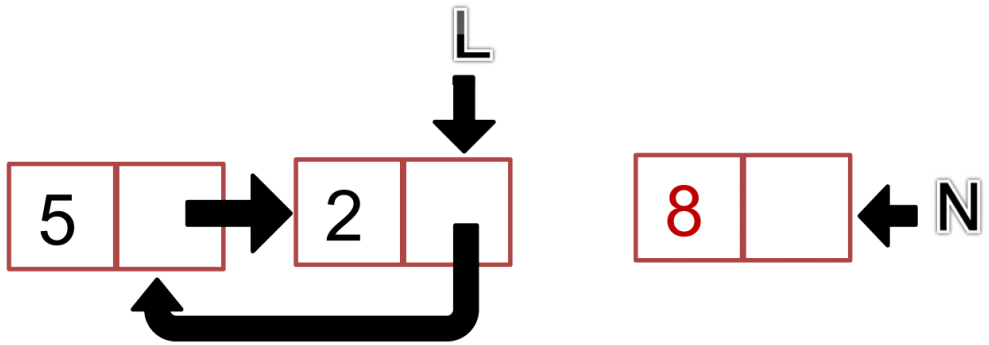
Ex: inserir 8



Insere no Final

- Lista com mais de um nó:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento

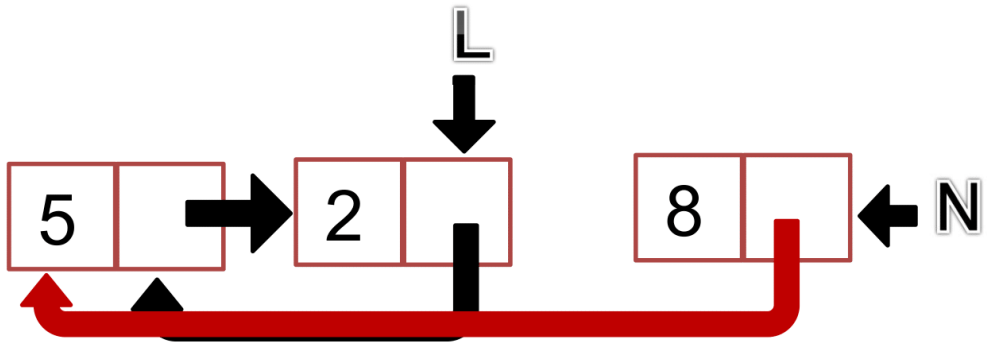
Ex: inserir 8



Insere no Final

- Lista com mais de um nó:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento
    - Campo ***prox*** aponta para o 1º nó (***L->prox***)

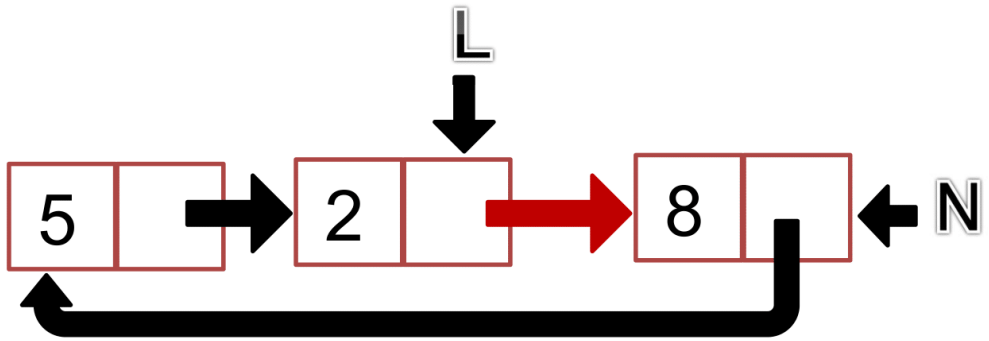
Ex: inserir 8



Insere no Final

- Lista com mais de um nó:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento
    - Campo ***prox*** aponta para o 1º nó (***L->prox***)
  - Faz a o último nó apontar para o novo nó

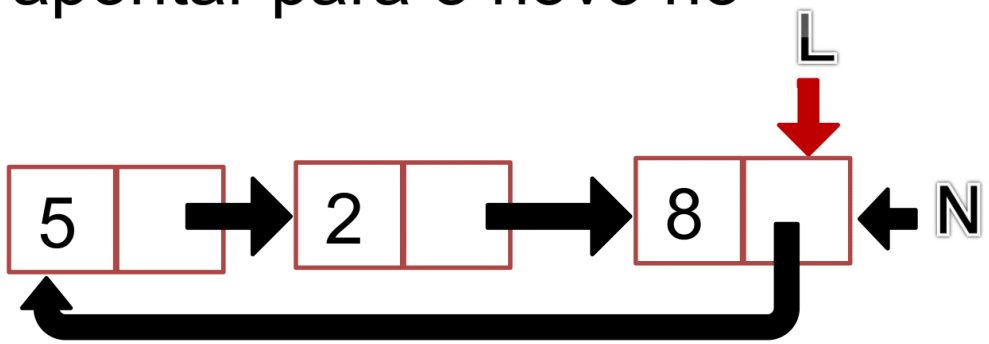
Ex: inserir 8



Insere no Final

- Lista com mais de um nó:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento
    - Campo ***prox*** aponta para o 1º nó (***L->prox***)
  - Faz a o último nó apontar para o novo nó
  - Faz a lista apontar para o novo nó

Ex: inserir 8

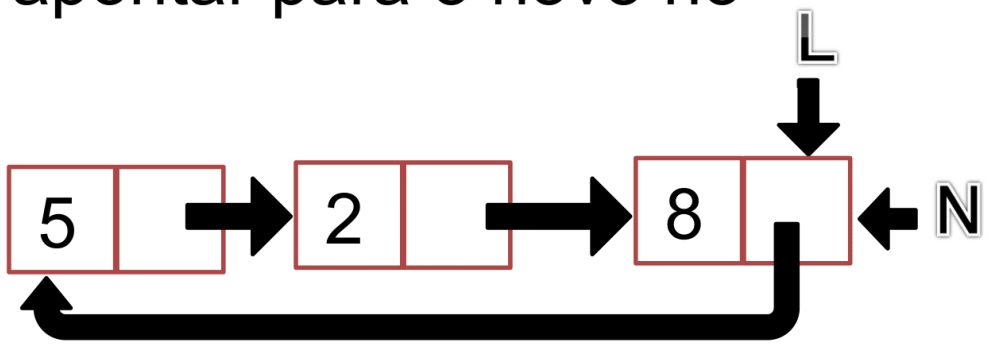


Insere no Final

- Lista com mais de um nó:
  - Aloca o novo nó
  - Preenche os campos do novo nó:
    - Campo ***info*** com o valor do elemento
    - Campo ***prox*** aponta para o 1º nó (***L->prox***)
  - Faz a o último nó apontar para o novo nó
  - Faz a lista apontar para o novo nó

Ex: inserir 8

retorna 1



Insere no Final

- **Implementação em C:**

```
int insere_final (Lista *lst, int elem) {
    // Aloca um novo nó e preenche campo info
    Lista N = (Lista) malloc(sizeof(struct no));
    if (N == NULL) { return 0; } // Falha: nó não alocado
    N->info = elem; // Insere o conteúdo (valor do elem)

    // Trata lista vazia
    if (lista_vazia(*lst) == 1) {
        N->prox = N; // Faz o novo nó apontar para ele mesmo
        *lst = N; // Faz a lista apontar para o novo nó (último nó)
    }
    ...
```

Insere no Final

- **Implementação em C:**

```
int insere_final (Lista *lst, int elem) {
    ...

    // Trata lista com elementos (1 ou +)
    else {
        N->prox = (*lst)->prox; // Faz o novo nó apontar o 1º nó
        (*lst)->prox = N; // Faz o último nó apontar para o novo nó
        *lst = N; // Faz a lista apontar para o novo nó (último nó)
    }
    return 1;
}
```

Remove no Início

- Existem **3 cenários** possíveis:
  - Lista vazia
  - Lista com um único nó
  - Lista com mais de um nó

Remove no Início

- Lista vazia:

Ex:

Remove no Início

- Lista vazia:
  - Não existe o elemento

Ex:

Remove no Início

- Lista vazia:
  - Não existe o elemento
  - Retorna ZERO (**operação falha**)

Ex:

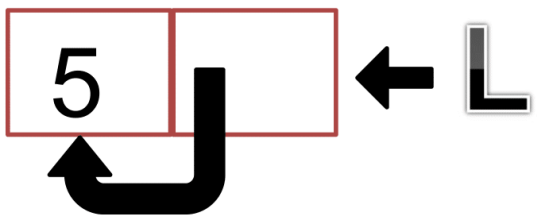
retorna 0

≠ Elemento

Remove no Início

- Lista com um único nó:

Ex:



Remove no Início

- Lista com um único nó:
  - Libera o espaço alocado para o nó apontado pela lista

free(L)

Ex:

retorna 1

Remove no Início

- Lista com um único nó:
  - Libera o espaço alocado para o nó apontado pela lista
  - Faz a lista apontar para **NULL**

Ex:

Remove no Início

- Lista com um único nó:
  - Libera o espaço alocado para o nó apontado pela lista
  - Faz a lista apontar para **NULL**

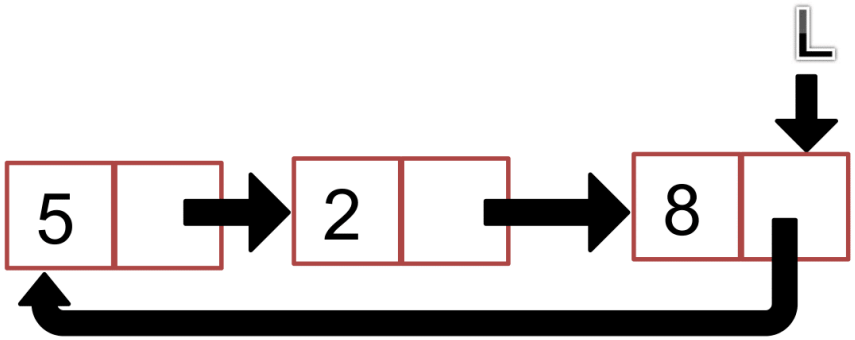
Ex:

retorna 1

Remove no Início

- Lista com mais de um nó:

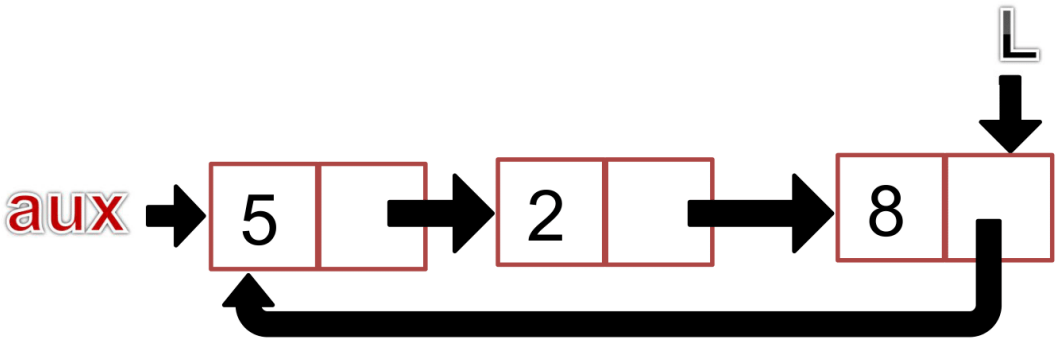
Ex:



Remove no Início

- Lista com mais de um nó:
  - Faz um **ponteiro auxiliar** apontar para o 1º nó da lista (***aux = L->prox***)

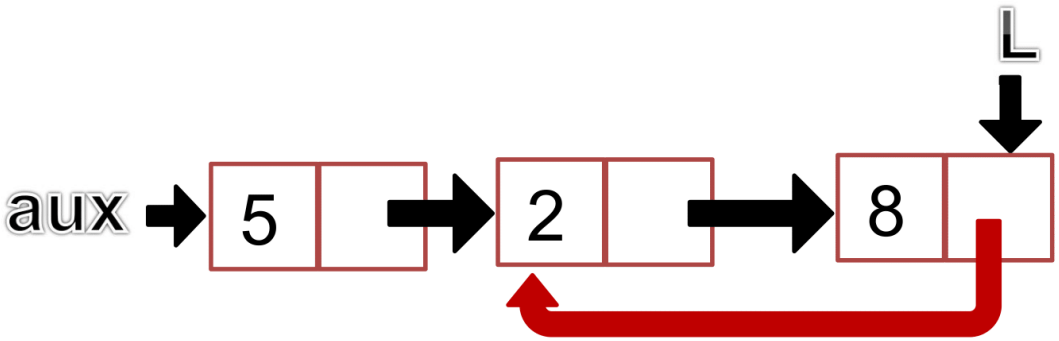
Ex:



Remove no Início

- Lista com mais de um nó:
  - Faz um ponteiro auxiliar apontar para o 1º nó da lista (***aux = L->prox***)
  - Faz o último nó apontar para o 2º nó da lista (***L->prox = aux->prox***)

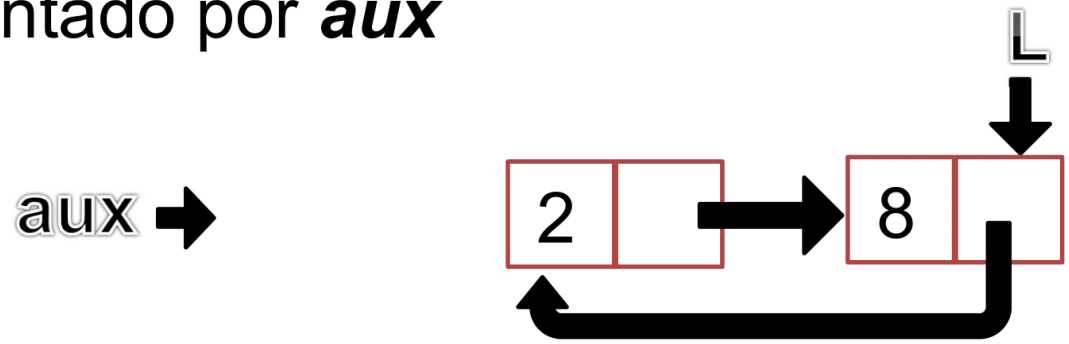
Ex:



Remove no Início

- Lista com mais de um nó:
  - Faz um ponteiro auxiliar apontar para o 1º nó da lista (***aux = L->prox***)
  - Faz o último nó apontar para o 2º nó da lista (***L->prox = aux->prox***)
  - Libera o espaço alocado para o nó apontado por ***aux***

Ex:

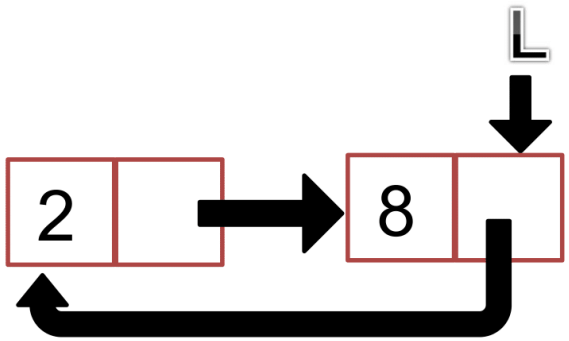


Remove no Início

- Lista com mais de um nó:
  - Faz um ponteiro auxiliar apontar para o 1º nó da lista (***aux = L->prox***)
  - Faz o último nó apontar para o 2º nó da lista (***L->prox = aux->prox***)
  - Libera o espaço alocado para o nó apontado por ***aux***

Ex:

retorna 1



Remove no Início

- **Implementação em C:**

```
int remove_inicio (Lista *lst, int *elem) {
    // Trata lista vazia
    if (lista_vazia(*lst) == 1)
        return 0;
    Lista aux = (*lst)->prox; // Faz aux apontar para 1º nó
    *elem = aux->info; // Retorna valor do nó a ser removido
    if (*lst == (*lst)->prox) // Trata lista com 1 único nó
        *lst = NULL;
    else // Trata lista com + de 1 elemento
        (*lst)->prox = aux->prox;
    free(aux);
    return 1;
}
```

Remove no Início

- **Implementação em C:**

```
int remove_inicio (Lista *lst, int *elem) {
    // Trata lista vazia
    if (lista_vazia(*lst) == 1)
        return 0;
    Lista aux = (*lst)->prox; // Faz aux apontar para 1º nó
    *elem = aux->info; // Retorna valor do nó a ser removido
    if (*lst == (*lst)->prox) // Trata lista com 1 único nó
        *lst = NULL;
    else // Trata lista com + de 1 elemento
        (*lst)->prox = aux->prox;
    free(aux);
    return 1;
}
```

Remove no Início

- **Implementação em C:**

```
int remove_inicio (Lista *lst, int *elem) {
    // Trata lista vazia
    if (lista_vazia(*lst) )== 1
        return 0;
    Lista aux = (*lst)->prox; // Faz aux apontar para 1º nó
    *elem = aux->info; // Retorna valor do nó a ser removido
    if (*lst == (*lst)->prox) // Trata lista com 1 único nó
        *lst = NULL;
    else // Trata lista com + de 1 elemento
        (*lst)->prox = aux->prox;
    free(aux);
    return 1;
}
```

Remove no Início

- **Implementação em C:**

```
int remove_inicio (Lista *lst, int *elem) {
    // Trata lista vazia
    if (lista_vazia(*lst) == 1)
        return 0;
    Lista aux = (*lst)->prox; // Faz aux apontar para 1º nó
    *elem = aux->info; // Retorna valor do nó a ser removido
    if (*lst == (*lst)->prox) // Trata lista com 1 único nó
        *lst = NULL;
    else // Trata lista com + de 1 elemento
        (*lst)->prox = aux->prox;
    free(aux);
    return 1;
}
```

Remove no Início

- **Implementação em C:**

```
int remove_inicio (Lista *lst, int *elem) {
    // Trata lista vazia
    if (lista_vazia(*lst) == 1)
        return 0;
    Lista aux = (*lst)->prox; // Faz aux apontar para 1º nó
    *elem = aux->info; // Retorna valor do nó a ser removido
    if (*lst == (*lst)->prox) // Trata lista com 1 único nó
        *lst = NULL;
    else // Trata lista com + de 1 elemento
        (*lst)->prox = aux->prox;
    free(aux);
    return 1;
}
```

Referências

✓ Básica

- **CELES, W., CERQUEIRA, R. e RANGEL, J. L. "Introdução a estruturas de dados". Campus Elsevier, 2004.**
- TENENBAUM, A. M., LANGSAM, Y. e AUGENSTEIN, M.J. *"Estrutura de Dados Usando C"*. Makron Books.

✓ Extra

- BACKES, André. *"Programação Descomplicada Linguagem C"*. Projeto de extensão que disponibiliza vídeo-aulas de C e Estruturas de Dados. Disponível em: `https://www.youtube.com/user/progdescomplicada`. Acessado em: 25/04/2022.

✓ Baseado nos materiais dos seguintes professores:

- Prof. André Backes (UFU)
- Prof. Bruno Travençolo (UFU)
- Prof. Luiz Gustavo de Almeida Martins (UFU)

Dúvidas?

Prof. Me. Claudiney R. Tinoco
profclaudineytinoco@gmail.com

Faculdade de Computação (FACOM)
Universidade Federal de Uberlândia (UFU)