

# Algoritmos e Programação I

## Módulo 5 - Lista Encadeada

**Prof<sup>ª</sup>. Elisa de Cássia Silva Rodrigues**

- Definição:

- ▶ Estrutura de dados utilizada para armazenar e organizar uma sequência de elementos do mesmo tipo.

Lista de inteiros

10

20

30

40

- Características:

- ▶ Seus elementos possuem estrutura interna abstraída.
- ▶ Uma lista pode possuir elementos repetidos, ser ordenada ou não.
- ▶ Uma lista pode possuir  $N \geq 0$  elementos ou itens.
- ▶ Se  $N = 0$ , dizemos que a lista é vazia.

A implementação de operações de uma lista depende do tipo de alocação de memória usada (estática ou dinâmica).

# Alocação de Memória

- Definição:

- ▶ Processo de reserva de memória para armazenamento de dados durante a execução de um programa.

- Alocação dinâmica:

- ▶ A memória é **reservada dinamicamente** (em tempo de execução).
- ▶ Na linguagem C, a alocação é feita pela função `malloc()`.
- ▶ Os dados desta área da memória só podem ser acessados por **ponteiros**.

- Vantagens:

- ▶ Variáveis não dependem do escopo.
- ▶ Quantidade total de memória não precisa ser previamente conhecida.
- ▶ Espaço de memória pode ser alterado durante a execução do programa.
- ▶ Programador controla o tempo de vida das variáveis.

- Desvantagens:

- ▶ Os dados não são necessariamente armazenados de forma sequencial.
- ▶ A memória utilizada deve ser alocada e liberada manualmente.
- ▶ **Obs:** esquecer de liberar a memória pode gerar falhas.

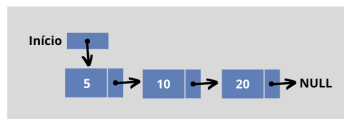
# Lista Encadeada

- Definição:

- ▶ Estrutura de dados do tipo **lista** que é definida utilizando **alocação dinâmica** e **acesso encadeado** dos elementos.

- Características:

- ▶ É necessário armazenar um ponteiro que indica o 1º elemento da lista.
- ▶ Cada elemento possui um dado e um ponteiro para o próximo da lista.
- ▶ Cada elemento é alocado dinamicamente quando é inserido na lista.
- ▶ Se um elemento é removido, a memória alocada para ele é liberada.



- Vantagens:

- ▶ Melhor utilização dos recursos da memória.
- ▶ Não é necessário definir previamente o tamanho da lista.
- ▶ Nem movimentar os elementos nas operações de inserção e remoção.

- Desvantagens:

- ▶ Acesso indireto aos elementos.
- ▶ Necessidade de percorrer a lista para acessar um elemento.

# Lista Encadeada

- Definição do TAD Lista Encadeada:

- ▶ Definir os arquivos `listaEncadeada.h` e `listaEncadeada.c`.
- ▶ Declarar o tipo de dado que irá representar a lista no `arquivo .h`:
- ▶ Definir o tipo de dado que será armazenado dentro da lista (`int`).
- ▶ Declarar a estrutura para representar a lista encadeada no `arquivo .c`:

```
typedef struct elemento *Lista;
```

```
struct elemento{  
    int dado;  
    struct elemento *prox;  
};  
typedef struct elemento Elemento;
```

- Declarar um ponteiro do tipo Lista para acessar o TAD (`main.c`):

```
Lista *li;
```

# Lista Encadeada

- Ilustração dos tipos de dados **Lista** e **Elemento**:

```
typedef struct elemento Elemento;  
typedef struct elemento *Lista;
```



Note que, nesta implementação, a estrutura **Lista** é abstrata (ponteiro para **Elemento**), ou seja, não é definida uma struct lista (nó descritor).



# Lista Encadeada

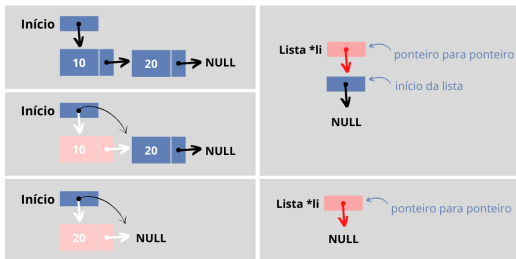
- Definição das operações do TAD Lista Encadeada:
  - ▶ Declaração dos protótipos das funções no arquivo **.h**.
  - ▶ Implementação das funções no arquivo **.c**.
- Operações básicas:
  - ▶ Criação da lista.
  - ▶ Inserção de um elemento na lista.
  - ▶ Remoção de um elemento da lista.
  - ▶ Busca por um elemento da lista.
  - ▶ Destruição da lista.
  - ▶ Informações sobre tamanho da lista.
  - ▶ Informação sobre a lista estar vazia ou cheia.



# Lista Encadeada

## • Destruição da lista:

- ▶ Inicialmente, deve-se liberar a memória alocada para todos os elementos da lista:
  - ★ Liberação da estrutura **Elemento** usando **free()**.
- ▶ Deve-se liberar a memória alocada para o ponteiro do início da lista:
  - ★ Liberação do ponteiro do tipo **Lista** usando **free()**.



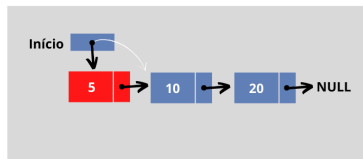
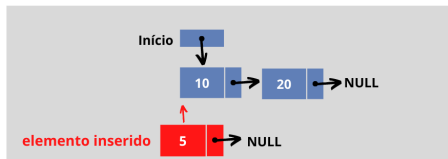
## ● Inserção:

- ▶ Ato de guardar elementos dentro da lista.
- ▶ Tipos de inserção:
  - ★ No início da lista (usada para implementação de Pilha).
  - ★ No meio da lista (usada em listas ordenadas).
  - ★ No final da lista (usada para implementação de Fila).
- ▶ Operação de inserção envolve alocação dinâmica de memória:
  - ★ Necessário verificar se a lista existe (`li != NULL`).
  - ★ Se existir, deve-se verificar se o novo elemento foi alocado corretamente.

# Lista Encadeada

## ● Inserção no início da lista:

- ▶ Envolve a criação de um novo elemento (alocação de memória).
- ▶ Atribui-se o valor do novo elemento ao campo **dado**.
- ▶ O ponteiro **prox** do novo elemento aponta para o 1º elemento da lista.
- ▶ O ponteiro que indica o 1º elemento da lista (**\*li**) aponta para o novo elemento.

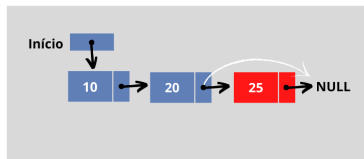
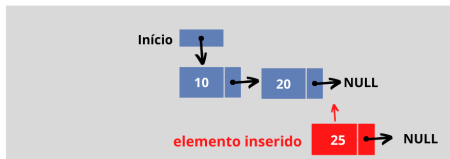




# Lista Encadeada

## ● Inserção no final da lista:

- ▶ Envolve a criação de um novo elemento (alocação de memória).
- ▶ Atribui-se o valor do novo elemento ao campo **dado**.
- ▶ Envolve a busca pelo último elemento da lista (**prox == NULL**).
- ▶ O ponteiro **prox** do novo elemento aponta para **NULL**.
- ▶ O ponteiro **prox** do elemento **anterior** aponta para o novo elemento.



## ● Remoção:

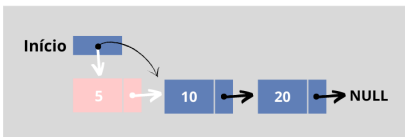
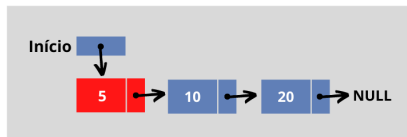
- ▶ Existindo uma lista, e ela possuindo elementos, é possível excluí-los.
- ▶ Tipos de remoção:
  - ★ No início da lista (usada para implementação de Fila e Pilha).
  - ★ No meio da lista (usada para remover um elemento específico).
  - ★ No final da lista.
- ▶ Operação de remoção envolve o teste de lista vazia.
  - ★ Necessário verificar se a lista existe (`li != NULL`).
  - ★ Se existir, deve-se verificar se existem elementos dentro da lista.
  - ★ Ou seja, se a lista não está vazia (`*li != NULL`).



# Lista Encadeada

- Remoção do início da lista:

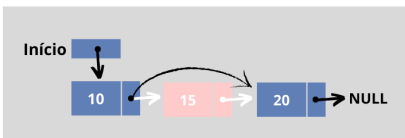
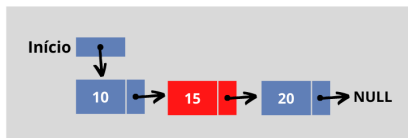
- ▶ Envolve a liberação da memória alocada para o 1º elemento da lista.
- ▶ O ponteiro que indica o 1º elemento (`*li`) aponta para o 2º elemento.
- ▶ Por fim, libera-se a memória do 1º elemento usando a função `free()`.



# Lista Encadeada

## • Remoção do meio da lista:

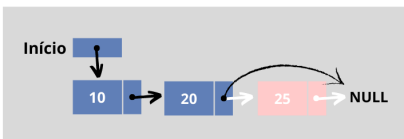
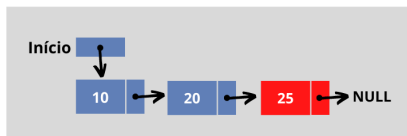
- ▶ Envolve a liberação da memória alocada para o elemento desejado.
- ▶ Envolve a busca pelo elemento **atual** a ser removido.
- ▶ O ponteiro **prox** do elemento **anterior** aponta para o mesmo endereço que o ponteiro **prox** do elemento **atual**.
- ▶ Por fim, libera-se a memória do elemento **atual** usando a função **free()**.



# Lista Encadeada

## • Remoção do final da lista:

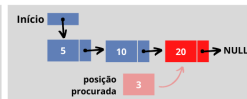
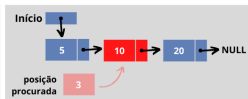
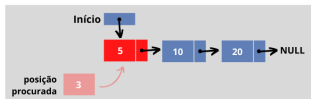
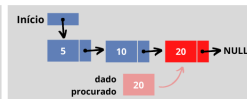
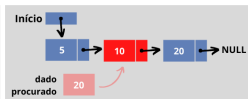
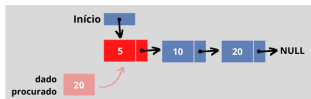
- ▶ Envolve a liberação da memória alocada para o último elemento.
- ▶ Envolve a busca pelo último elemento da lista (**atual**).
- ▶ O ponteiro **prox** do elemento **anterior** aponta para o mesmo endereço que o ponteiro **prox** do elemento **atual** (**NULL**).
- ▶ Por fim, libera-se a memória do elemento **atual** usando a função **free()**.



# Lista Encadeada

## Busca:

- ▶ Envolve percorrer a lista em busca do elemento desejado.
- ▶ Essa operação pode ocorrer de duas formas:
  - ★ Dado um elemento, devolve a posição dele na lista.
  - ★ Dada uma posição, devolve o elemento daquela posição.



# Lista Encadeada

## • Quando usar esse tipo de lista?

- ▶ Quando não é preciso de garantir espaço mínimo para a aplicação.
  - ★ Porque a memória pode ser definida em tempo de execução.
- ▶ Quando o tamanho máximo da lista não é bem definido.
  - ★ Porque a memória é alocada quando um novo elemento é inserido.
- ▶ Quando inserções e remoções são frequentes em listas ordenadas.
  - ★ Porque não é necessário deslocar elementos do vetor.
- ▶ Quando a operação de busca não é muito frequente.
  - ★ Porque é necessário percorrer a lista para encontrar um elemento.

### Implementação:

[https://replit.com/@elisa\\_rodrigues/Modulo5-ListaEncadeada](https://replit.com/@elisa_rodrigues/Modulo5-ListaEncadeada)

❶ BACKES, A. *Estrutura de dados descomplicada em linguagem C*. 2016.

-> **Capítulo 5: Listas**

-> **Material Complementar - Vídeo aulas (10ª a 15ª):**

<https://programacaodescomplicada.wordpress.com/indice/estrutura-de-dados/>