Listas ordenadas

PROFESSOR: DIEGO RICARDO KROHL
diego.krohl@ifc.edu.br

Lista ordenada

- Uma lista ordenada é uma coleção de elementos dispostos em uma sequência específica, onde a ordem dos elementos é significativa e predeterminada de acordo com algum critério;
- Essa ordenação pode ser feita de diversas maneiras, dependendo do contexto e dos critérios estabelecidos;

Lista ordenada - Características

- A lista pode ser ordenada numericamente, alfabeticamente, cronologicamente, ou por qualquer outro critério relevante. Por exemplo, uma lista de números pode ser ordenada do menor para o maior, enquanto uma lista de nomes pode ser ordenada alfabeticamente;
- Na programação, listas ordenadas são estruturas de dados onde os elementos são mantidos em uma ordem específica.

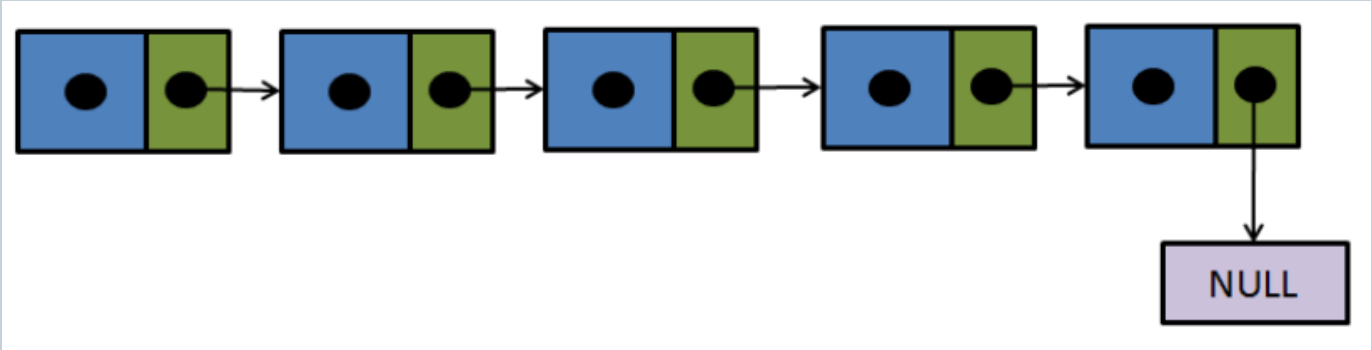
Exemplos

- **Lista de Tarefas**: Uma lista de tarefas ordenada por prioridade, onde as tarefas mais importantes ou urgentes aparecem primeiro;
- **Agenda Telefônica**: Uma lista de contatos ordenada alfabeticamente pelo nome ou sobrenome;
- **Dados Cronológicos**: Uma lista de eventos ou registros ordenados por data e hora.

Na programação

- **Algoritmos de Ordenação**: Existem vários algoritmos para ordenar listas, como Quick Sort, Merge Sort, Bubble Sort, entre outros. Cada algoritmo tem suas próprias vantagens e desvantagens em termos de eficiência e complexidade;
- **Uso em Bancos de Dados:** Em bancos de dados, dados podem ser recuperados de forma ordenada usando cláusulas como ORDER BY em SQL, permitindo que os resultados sejam apresentados em uma ordem específica.

Listas encadeadas



Conceito

- Listas encadeadas podem ser implementadas utilizando vetores ou, mais comumente, utilizando **Alocação Dinâmica de Memória (ADM);**

- Na alocação encadeada**, a ordem lógica das informações pode ser (e geralmente é) diferente da ordem física.**

Conceito

- Exemplo encadeamento da palavra BANANA

Primeiro →

	Info	Elo
0	A	4
1		
2	B	10
3		
4	N	8
5		
6	N	0
7		
8	A	-1
9		
10	A	6

- Na alocação encadeada, as informações **SEMPRE** serão acessadas conforme a ordem lógica, nunca pela ordem física.

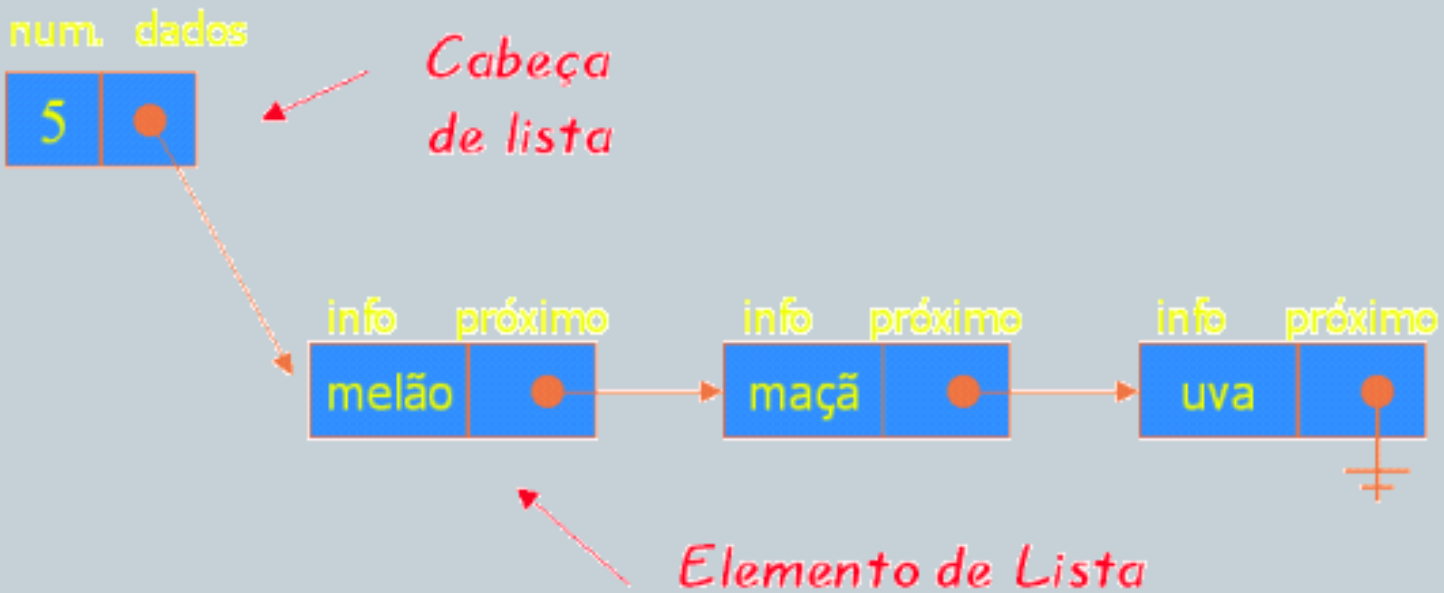
Conceito

- As listas que utilizam encadeamento são divididas basicamente em três tipos:
  - Listas de encadeamento simples:
    - Neste tipo de listas o encadeamento acontece em apenas uma direção;
  - Listas de encadeamento duplo:
    - Neste tipo de lista o encadeamento acontece em duas direções; e

Conceito

- Listas de encadeamento circular:
  - Neste tipo de lista, o último elemento é ligado com o primeiro, produzindo um efeito circular;
  - Podem ser implementadas tanto utilizando encadeamento simples, quanto encadeamento duplo.

Listas de encadeamento simples



Conceito

- Este tipo de lista também é conhecido como Listas Encadeadas Simples;
- Neste tipo de lista, o encadeamento acontece em apenas uma direção;
- São comumente implementadas utilizando ADM;
- A principal vantagem é a facilidade de implementação;
- A principal desvantagem é que não é possível visitar os nodos em sentido oposto.

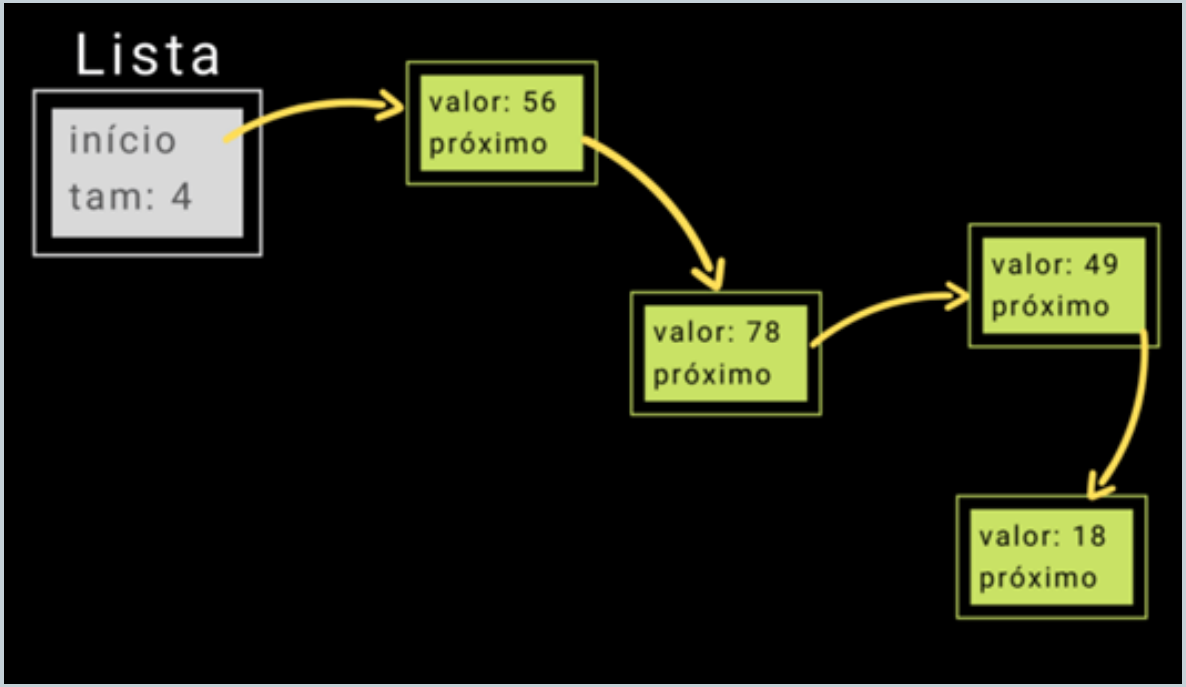
Conceito

- Para o encadeamento acontecer (utilizando ADM), o **nodo (nó)** precisa conter pelo menos duas informações: **INFO** e **ELO**:
  - **INFO:** campo que contém as informações que são armazenadas na lista e;
  - **ELO:** campo que faz o encadeamento das informações, responsável pela definição da ordem lógica.

Conceito

- O campo **INFO**, por sua vez, pode ser dividido em *n* outros campos.
  - O campo ELO sempre conterá o endereço de memória (o índice no caso de vetores) do próximo nodo na ordem lógica.

Listas encadeadas utilizando ADM



Alocação Dinâmica de Memória (ADM)

- Relembrando:
  - **ADM** é o processo que aloca (solicita/reserva) memória em tempo de execução;
  - É utilizado quando não se sabe ao certo quanto de memória será necessário para realizar determinada operação;
  - Essencial para evitar o desperdício de memória.

Lista Encadeada Simples com ADM

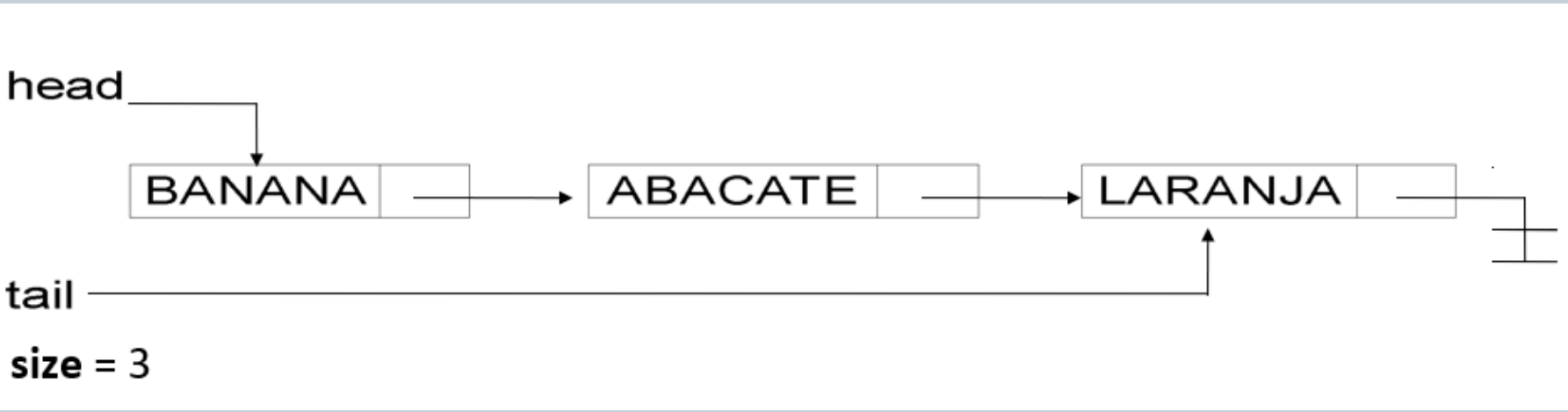
- As Listas Encadeadas, utilizando ADM, são compostas por elementos individuais, cada um ligado por um único ponteiro;
- Cada elemento consiste em duas partes: um **membro de dados** (INFO), e um **ponteiro próximo** (ELO).

Lista Encadeada Simples com ADM

- A lista encadeada é formada definindo-se o ponteiro **next (ELO)** de cada elemento para que ele aponte para o elemento que se segue (o próximo);
- O elemento **next** do último elemento é definido para **NULL**, indicando o fim da lista;
- O elemento no início da lista é denominado de cabeça da lista (**head**) e o elemento no final é denominado de cauda da lista (**tail**).

Lista Encadeada Simples com ADM

- Exemplo:



Lista Encadeada Simples com ADM

- **IMPORTANTE**: Para acessar um elemento em uma lista encadeada simples, inicia-se pela cabeça da lista (***head***) e percorre a lista por meio dos ponteiros próximo (***next***) dos elementos sucessivos, movendo-se de elemento a elemento até que o elemento desejado seja encontrado ou até que o elemento cauda (***tail***) seja atingido.

Lista Encadeada Simples com ADM

As principais funções em listas encadeadas simples são:

- Cria lista;
- Cria elemento/nodo;
- Insere elemento na lista;
- Remove elemento da lista;
- Percorre a lista (encontra elementos).

Estrutura da Lista Encadeada Simples

A estrutura principal da lista encadeada simples, utilizando ADM, é composta por:

- Elemento *head*;
- Elemento *tail*;
- Tamanho da lista (um número inteiro que indica quantos elementos, nodos, compõem a lista).

Algoritmos de inserção e remoção em lista encadeada simples

Algoritmo descritivo da função insere

1 - Definindo a Estrutura da Lista Encadeada:
    - Criar uma estrutura que represente um nó da lista, contendo o dado e um ponteiro para o próximo nó.
2 - Criando um Novo Nó:
    - Alocar memória para um novo nó.
    - Atribuir o valor desejado ao campo de dados do nó.
    - Inicializar o ponteiro *next* do novo nó para NULL.
3 - Inserindo o Nó no Início da Lista:
    - Criar o novo nó.
    - Ajustar o ponteiro do novo nó para apontar para o nó que era a cabeça da lista.
    - Atualizar a cabeça da lista para *next* que aponte para o novo nó.
4 - Inserindo o Nó no Final da Lista:
    - Criar o novo nó.
    - Percorrer a lista até encontrar o último nó.
    - Ajustar o ponteiro *next* do último nó para apontar para o novo nó.
5 - Inserindo o Nó em uma Posição Específica:
    - Criar o novo nó.
    - Percorrer a lista até o nó anterior à posição desejada.
    - Ajustar o ponteiro *next* do novo nó para apontar para o nó seguinte.
    - Ajustar o ponteiro *next* do nó anterior para apontar para o novo nó.

Pseudocódigo da função insere

```
insercao(lista, elemento_pivo, dado)
inicio
        Nodo *novo_elemento;
        novo_elemento = criaNovoElemento(dado);

        se (elemento_pivo == NULL)
                se (listaVazia())
                        lista -> tail = novo_elemento;

                novo_elemento -> next = lista->head
                lista -> head = novo_elemento;
        senao
                se (elemento_pivo -> next == NULL)
                        lista -> tail = novo_elemento;

                novo_elemento -> next = elemento_pivo -> next;
                elemento_pivo -> next = novo_elemento;

        fimse
        atualizaTamanhoDaLista();
fim
```

Algoritmo descritivo da função remove

1 - Definindo a Estrutura da Lista Encadeada:
- A estrutura já deve estar definida, representando um nó da lista com campos para os dados e um ponteiro para o próximo nó.

2 - Removendo o Nó do Início da Lista:
- Verificar se a lista está vazia (cabeça é NULL). Se estiver vazia, não há nada para remover.
- Armazenar o ponteiro para o nó que será removido (a cabeça da lista).
- Atualizar a cabeça da lista para o próximo nó.
- Liberar a memória do nó removido.

3 - Removendo o Nó do Final da Lista:
- Verificar se a lista está vazia. Se estiver vazia, não há nada para remover.
- Percorrer a lista até encontrar o penúltimo nó (o nó cujo ponteiro next aponta para o último nó).
- Armazenar o ponteiro para o nó que será removido (o último nó).
- Ajustar o ponteiro next do penúltimo nó para NULL.
- Liberar a memória do nó removido.

4 - Removendo um Nó em uma Posição Específica:
- Verificar se a lista está vazia. Se estiver vazia, não há nada para remover.
- Percorrer a lista até encontrar o nó anterior à posição desejada.
- Armazenar o ponteiro para o nó que será removido.
- Ajustar o ponteiro next do nó anterior para apontar para o próximo nó do que será removido.
- Liberar a memória do nó removido.

5 - Lidando com Casos Especiais:
- Caso a lista tenha apenas um nó, ao removê-lo, atualizar a cabeça da lista para NULL.
- Caso a posição a ser removida seja inválida (por exemplo, fora do intervalo da lista), retornar um erro ou mensagem apropriada.

Pseudocódigo da função remove

```
remocao(lista, elemento_pivo)
inicio
        Nodo* elemento_antigo;
        se (listaVazia())
                retornaListaVazia();

        se (elemento_pivo == NULL)
                elemento_antigo = lista -> head;
                lista -> head = lista -> head -> next;

                se (lista -> head == NULL)
                        lista -> tail = NULL;
        senão
                se (elemento_pivo -> next == NULL)
                        retornaFimDaLista();

                elemento_antigo = elemento_pivo -> next
                elemento_pivo -> next = elemento_pivo -> next->next
                se (elemento_pivo -> next == NULL)
                        lista -> tail = elemento_pivo;
        fimse

        liberaMemoria(elemento_antigo);

        atualizaTamanhoDaLista();
fim
```

Implementação prática de Lista Encadeada Simples em C

Implementação de lista encadeada simples em C

- Implementar uma lista encadeada simples em C utilizando os conceitos e algoritmos apresentados.

- **Instruções**:
  - O tipo de dados dos elementos da lista podem ser valores inteiros, alfanuméricos (strings), ou ponteiros para outros tipos abstratos de dados;
  - O ideal é que a lista já seja construída para armazenar **tipos abstratos de dados**.

Principais funções

- **Cria lista:**
  - função que cria a lista e aloca a memória necessária para a lista;
- **Cria elemento:**
  - função que cria um elemento (nodo) novo, aloca memória necessária e atribui um valor para este novo elemento;
- **Insere elemento na lista:**
  - função que insere este novo elemento em uma lista seguindo o algoritmo apresentado nos slides;
- **Remove elemento da lista:**
  - função que insere este novo elemento em uma lista seguindo o algoritmo apresentado nos slides;
- **Percorre a lista para encontrar elementos (pesquisa):**
  - função que percorre a lista, a partir da cabeça (head) até encontrar o elemento que está sendo buscado ou atingir o fim da lista;

Representação em C

Representação de Elementos em Lista encadeada Simples usando C

```
typedef struct sElemento{
        struct sElemento *next;
        int dado; //Depende do tipo de dados que se deseja trabalhar
} Elemento;
```

Representação de Lista encadeada Simples em C

```
typedef struct sLista{
        struct sElemento *head;
        struct sElemento *tail;
        int size;
} Lista;
```