



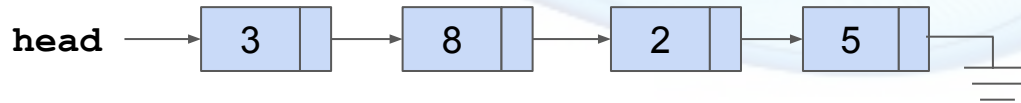
Lista Encadeada

Aula 03

Marcos Silvano Almeida
marcossilvano@utfpr.edu.br
Departamento de Computação
UTFPR Campo Mourão

Lista Encadeada

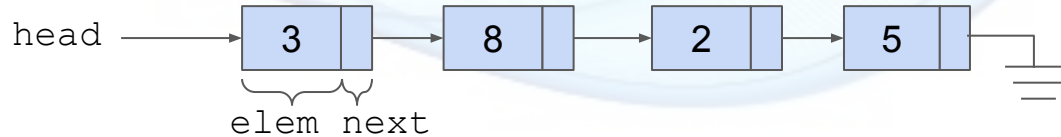
- Encadeamento de nós (Nodes) que armazenam os dados
 - Acesso somente sequencial
 - Não permite acesso aleatório aos seus elementos.
 - Posições/Índices podem ser simulados, mas acesso continua sequencial
 - Bom para para inserções/remoções em qualquer posição da lista
 - Disponível em C++ pela STL como `std::forward_list`
 - `std::List` é a Lista Duplamente Encadeada
 - Operações: criar, destruir, inserir, acessar, pesquisar, remover, imprimir, ...
- É estruturada como um encadeamento de Nodes



O tipo Lista Encadeada e seus Nodes (C/C++)



```
struct LinkedList {  
    Node* head; // aponta para o primeiro elemento  
};  
  
struct Node {  
    int key;      // identificador do elemento  
    Node* next;  // aponta para o próximo elemento (encadeamento)  
};
```



O tipo Lista Encadeada e seus Nodes (C++)



- A manipulação da Lista Encadeada ocorre por meio das **operações** (funções)

// Utilizando Lista Encadeada em C++

```
LinkedList* list1 = new LinkedList();
```

```
// head -> NULL
```

```
list1->push_front(5);
```

```
// head -> [5]
```

```
list1->push_front(9);
```

```
// head -> [9] -> [5]
```

```
list1->push_front(15);
```

```
// head -> [15] -> [9] -> [5]
```

```
list1->print();
```

```
// head -> [15] -> [9] -> [5]
```

```
list1->pop_front();
```

```
// head -> [9] -> [5]
```

O tipo Lista Encadeada e seus Nodes (C vs C++)



- A manipulação da Lista Encadeada ocorre por meio das **operações** (funções)

```
// Utilizando Lista Encadeada em C++
LinkedList* list1 = new LinkedList();

list1->push_front(5);
list1->push_front(9);
list1->push_front(15);

list1->print();

list1->pop_front();
```

```
// Utilizando Lista Encadeada em C
LinkedList* list1 = list_create();

list_push_front(list1, 5);
list_push_front(list1, 9);
list_push_front(list1, 15);

list_print(list1);

list_pop_front(list1); ...
```

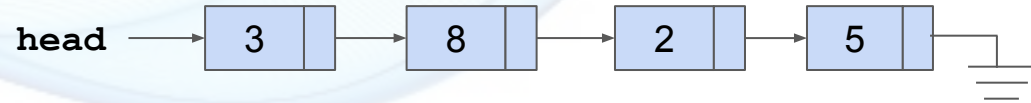
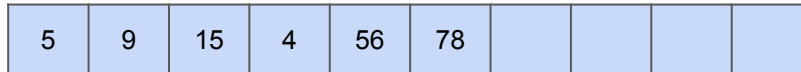
Lista Sequencial vs Encadeada

Lista Sequencial	Lista Encadeada
✓ Acesso sequencial	✓ Acesso sequencial
✓ Acesso aleatório	✗ Acesso aleatório
✗ Redimensionamento custoso	✓ Fácil redimensionamento
✗ Inserções/Remoções no início	✓ Inserções/Remoções no início
✓ Inserções/Remoções no final	✓ Inserções/Remoções no final (se tiver endereço)
✗ Inserções/Remoções no meio	✓ Inserções/Remoções no meio (se tiver endereço)

capacity: 10

size: 6

data





Operações Básicas sobre a Lista Encadeada

Classe Lista Encadeada (C++)



```
#pragma once

class Node {
public:
    int key;
    Node* next;
};

class LinkedList {
private:
    Node* head;
public:
    LinkedList();
    ~LinkedList();
    bool push_front(int key);
    bool pop_front();
    int get(int pos);
    void print();
};
```

```
#include "linkedlist.h"
#include <cstdio>

LinkedList::LinkedList() {
    //...
}

LinkedList::~~LinkedList() {
    //...
}

bool LinkedList::push_front(int key) {
    //...
}

bool LinkedList::pop_front() {
    //...
}

int LinkedList::get(int pos) {
    //...
}

void LinkedList::print() {
    //...
}
```


Operações Básicas da Lista

<https://visualgo.net/en/list>



```
LinkedList(); // Construtor: prepara lista.
~LinkedList(); // Destrutor: libera cada um dos Nodes.

void print(); // Imprime o conteúdo da lista.

int size(); // Retorna o tamanho da lista.
bool empty(); // Informa se lista vazia.
bool full(); // Informa se lista cheia.

void push_front(int key); // Insere elemento no início da lista.
bool pop_front(); // Remove elemento do início da lista.

// Insere e remove elemento no final (último elemento)
void push_back(int Key);
bool pop_back();
```

Dica: geralmente é mais fácil começar a pensar no processo pelo “caso médio” e, depois, adicionar as exceções.

Operações Básicas da Lista

 <https://visualgo.net/en/list>



```
Node* find(int key);  
Node* get(int pos);  
NULL)
```

```
// Operações baseadas em Nodes
```

```
bool insert_after(int key, Node* pos);
```

```
bool remove_after(int key, Node* pos);
```

```
// Operações baseadas em posições ou valores
```

```
bool insert(int pos);
```

```
bool remove(int pos);
```

```
bool remove(int key);
```

```
bool insert_sorted(int key);
```

```
// Encontra Node pela chave (ou NULL)
```

```
// Encontra Node por posição (ou
```

```
// Insere novo Node após o
```

```
// Node informado
```

```
// Remove o Node imediatamente após o
```

```
// Node informado
```

```
// Insere elemento na posição informada
```

```
// Remove elemento na posição informada
```

```
// Remove elemento de valor informado
```

```
// Insere elemento em ordem crescente.
```



Estratégias de Implementação de Lista Encadeada

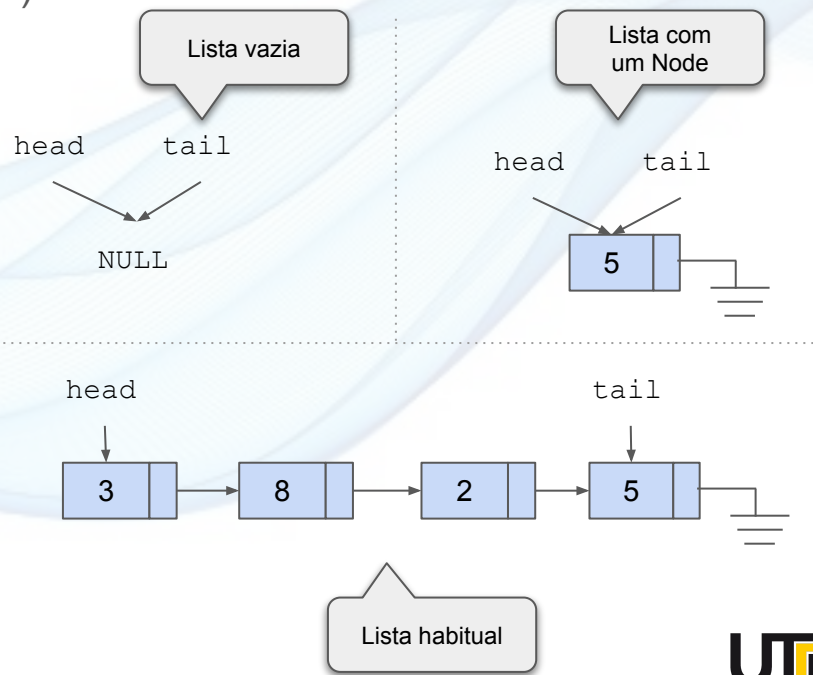
Estratégias de implementação de Lista Encadeada



- Adicionar **ponteiro para cauda** da lista
 - Torna bastante eficiente as inserções ao final da lista
 - Acarreta em mais código de controle, especialmente nas operações de modificação da lista (inserir e remover).

```
struct LinkedList {  
    Node* head; // primeiro Node  
    Node* tail; // último Node  
};
```

```
struct Node {  
    int key;  
    Node* next;  
};
```



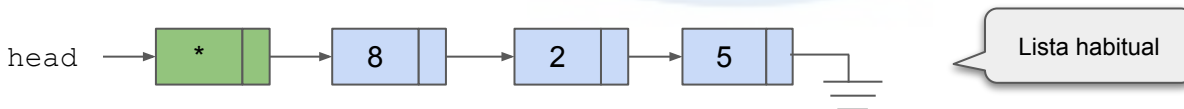
Estratégias de implementação de Lista Encadeada



- Emprego de **sentinela**
 - Elemento "coringa" como primeiro da lista



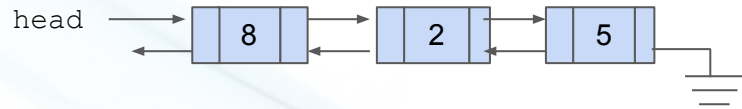
- Simplifica a implementação, pois **"head"** nunca precisa ser atualizado
 - Na implementação tradicional, as operações de modificação da lista (inserir e remover) requerem verificações para atualizar o ponteiro **"head"**. Com o emprego do elemento sentinela, não há necessidade de atualizar **"head"**, simplificando a implementação.



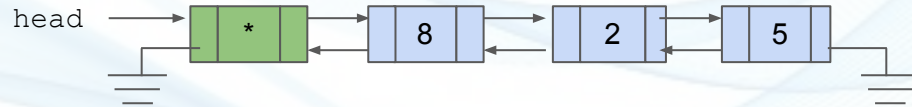
Variações do TAD Lista Encadeada

- Lista Duplamente Encadeada

- Cada Node possui dois ponteiros, prev e next, que apontam, respectivamente, ao Node anterior e ao próximo, na lista.



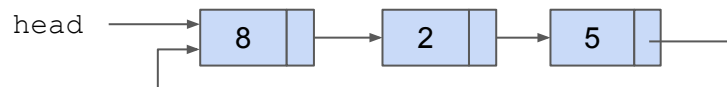
Lista Duplamente encadeada



Lista Duplamente encadeada com Sentinela

- Lista Circular

- Simples ou Duplamente Encadeada



Lista Encadeada Circular