Estruturas de Dados 1

Listas Encadeadas - Parte 1

Rafael Fazzolino

Lista Encadeada - Definição

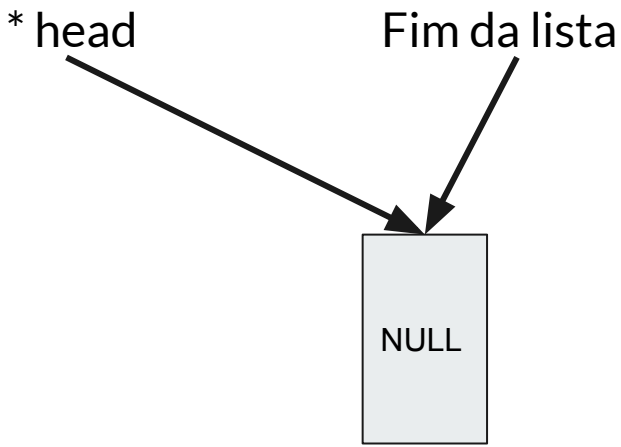
- É uma estrutura composta por Nós, onde cada Nó armazena uma informação e um ponteiro para o próximo Nó da lista.
- A partir do primeiro Nó da lista (*head*), pode-se acessar todos os demais.
- É uma boa alternativa aos Vetores

Vetores x Listas Encadeadas

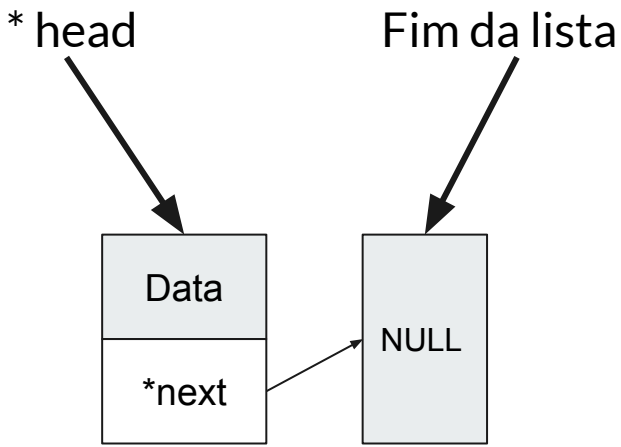
- **Vetores:**
  - Vantagem do acesso imediato
  - Complexidade alta na inserção e remoção de elementos no vetor
- **Lista:**
  - Acesso mais lento, pois precisa navegar pelos Nós até encontrar o desejado
  - Vantagem da inserção e remoção com eficiência

Estrutura básica

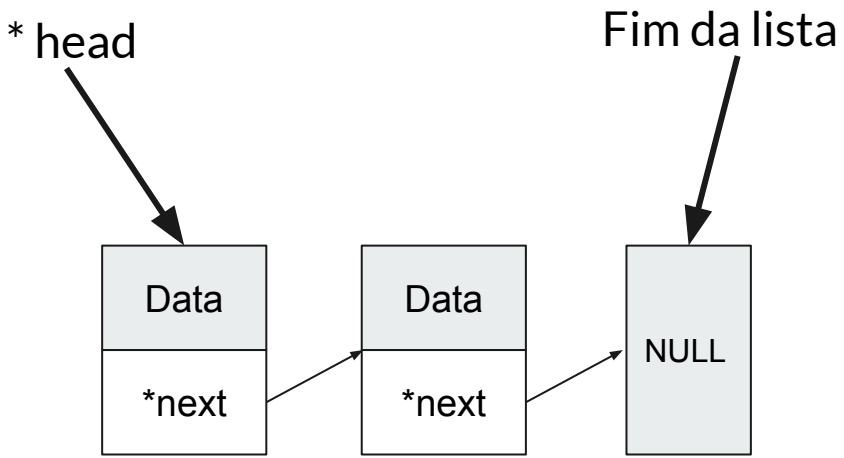
Estrutura básica



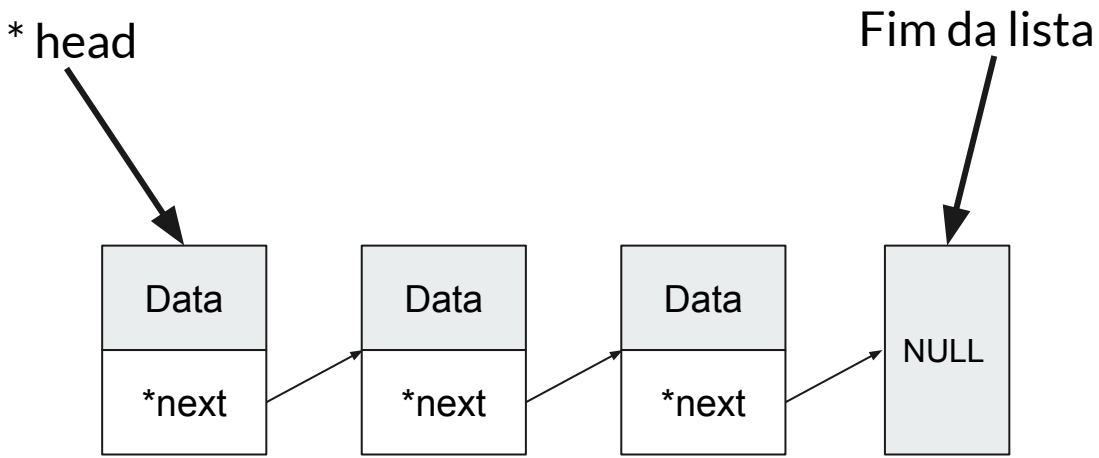
Estrutura básica



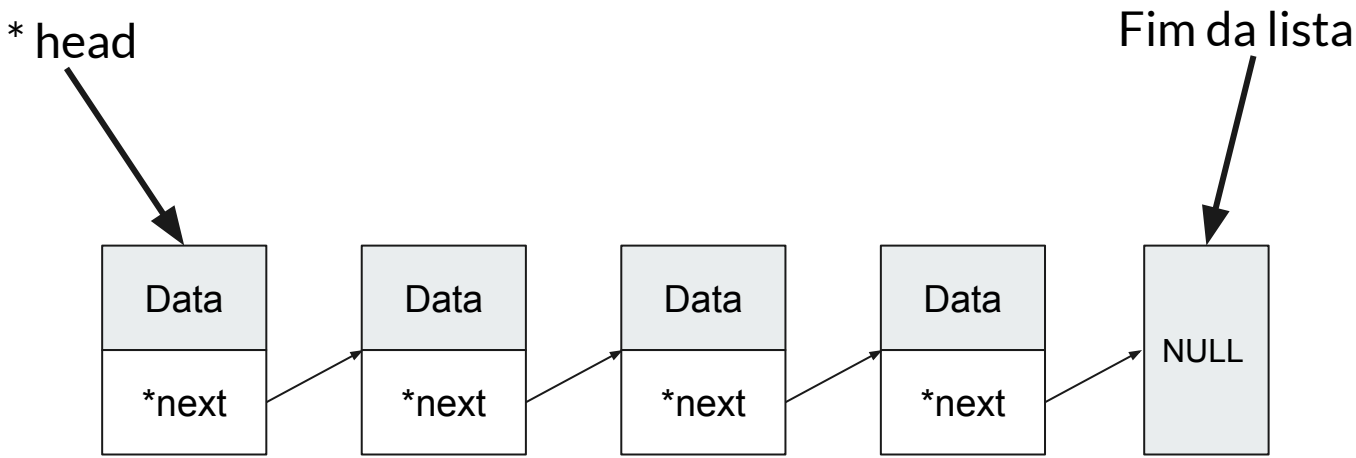
Estrutura básica



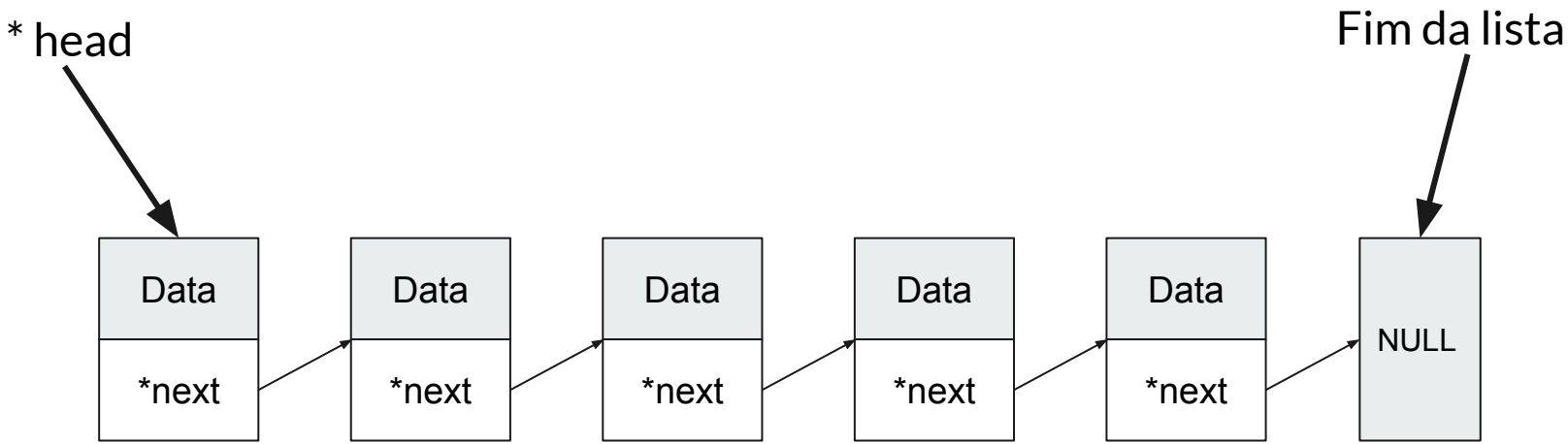
Estrutura básica



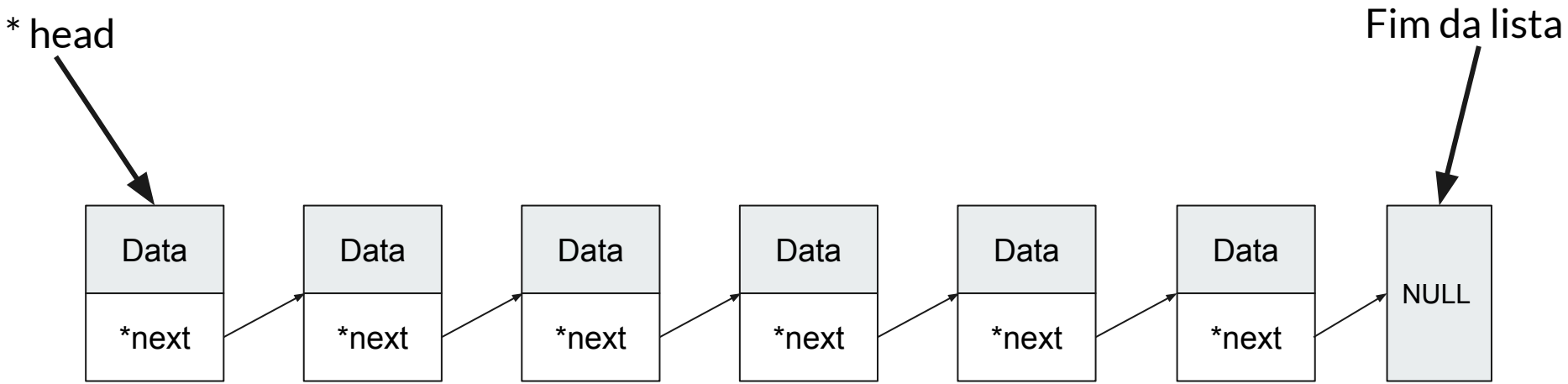
Estrutura básica



Estrutura básica



Estrutura básica



Estrutura básica

- Cada nó pode ser definido como uma estrutura em C
- Por exemplo, imagine o seguinte Nó contendo uma idade:

Estrutura básica

- Cada nó pode ser definido como uma estrutura em C
- Por exemplo, imagine o seguinte Nó contendo uma idade:

```
typedef struct node{
    int idade;
    struct node * next;
}Node;
```

Estrutura básica

- A lista também pode ser definida a partir de estruturas, por exemplo:

```
typedef struct list{
    Node * head;
    int size;
}List;
```

Funções principais

Funções principais

- List * create_list()
- void push(List * list, Node * node)
- bool is_empty(List * list)
- void print_list(List * list)
- void pop(List * list)
- int index_of(List * list, Node * node)
- Node * at_pos(List * list, int index)
- void insert(List * list, Node * node, int index)
- void erase(List * list, int index)

List * create_list();

List * create_list();

Objetivo: Criar uma lista vazia

- Para criar uma lista, devemos:

List * create_list();

Objetivo: Criar uma lista vazia

- Para criar uma lista, devemos:
  - Alocar espaço de memória suficiente para tal
    - List * list = (List *) malloc(sizeof(List));

List * create_list();

Objetivo: Criar uma lista vazia

- Para criar uma lista, devemos:
  - Alocar espaço de memória suficiente para tal
    - List * list = (List *) malloc(sizeof(List));
  - Inicializar a cabeça da lista como NULL
    - list->head = NULL;

List * create_list();

Objetivo: Criar uma lista vazia

- Para criar uma lista, devemos:
  - Alocar espaço de memória suficiente para tal
    - List * list = (List *) malloc(sizeof(List));
  - Inicializar a cabeça da lista como NULL
    - list->head = NULL;
  - Inicializar o tamanho da lista em zero
    - list->size = 0;

List * create_list();

```c
/*Função para criar uma lista vazia*/
List * create_list(){
    List * list = (List *) malloc(sizeof(List));
    list->head = NULL;
    list->size = 0;

    return list;
}
```

Objetivo: Inserir elemento no início da lista

void push(List * list, Node * node);

Objetivo: Inserir elemento no início da lista

void push(List * list, Node * node);

- Para incluir um elemento no início da lista, deve-se, primeiramente, ter um Node alocado e pronto para inserção

Objetivo: Inserir elemento no início da lista

void push(List * list, Node * node);

- Para incluir um elemento no início da lista, deve-se, primeiramente, ter um Node alocado e pronto para inserção
- Então devemos fazer com que o node->next aponte para list->head

Objetivo: Inserir elemento no início da lista

void push(List * list, Node * node);

- Para incluir um elemento no início da lista, deve-se, primeiramente, ter um Node alocado e pronto para inserção
- Então devemos fazer com que o node->next aponte para list->head
- E list->head passa a ser o node, já que este foi inserido no início da lista

Objetivo: Inserir elemento no início da lista

void push(List * list, Node * node);

- Para incluir um elemento no início da lista, deve-se, primeiramente, ter um Node alocado e pronto para inserção
- Então devemos fazer com que o node->next aponte para list->head
- E list->head passa a ser o node, já que este foi inserido no início da lista
- Precisamos sempre lembrar de incrementar o tamanho da lista

void push(List * list, Node * node);

```
/*Função para inserir um novo elemento no início da lista*/
void push(List * list, Node * node){
    node->next = list->head;
    list->head = node;
    list->size++;
}
```

Objetivo: Verificar se a lista está vazia

bool is_empty(List * list);

Objetivo: Verificar se a lista está vazia

bool is_empty(List * list);

- Podemos verificar se a lista está vazia apenas analisando o *size* da lista:

bool is_empty(List * list);

- Podemos verificar se a lista está vazia apenas analisando o *size* da lista:

```
/*Função que verifica se função está vazia*/
bool is_empty(List * list){
    if(list->size == 0)
        return true;
    else
        return false;
}
```

Objetivo: Imprimir lista completa

void print_list(List * list);

- Para imprimir a lista inteira, precisamos:
  - Verificar se a lista está vazia, se estiver, informar que está vazia e retornar

Objetivo: Imprimir lista completa

void print_list(List * list);

- Para imprimir a lista inteira, precisamos:
  - Verificar se a lista está vazia, se estiver, informar que está vazia e retornar
  - Começar da primeira posição da lista e ir iterando até encontrar um nó valendo NULL

Objetivo: Imprimir lista completa

void print_list(List * list);

- Para imprimir a lista inteira, precisamos:
  - Verificar se a lista está vazia, se estiver, informar que está vazia e retornar
  - Começar da primeira posição da lista e ir iterando até encontrar um nó valendo NULL
  - Imprimir os dados desejados a cada iteração

void print_list(List * list);

```c
/*Função para imprimir a lista*/
void print_list(List * list){
    if(is_empty(list)){
        printf("Lista Vazia!\n");
        return;
    }
    Node * pointer = list->head;
    while(pointer != NULL){
        printf("Idade: %d\n", pointer->idade);
        pointer = pointer->next;
    }
}
```

Objetivo: Remover o primeiro elemento da lista

void pop(List * list);

Objetivo: Remover o primeiro elemento da lista

void pop(List * list);

- Para remover o primeiro elemento, precisamos:

Objetivo: Remover o primeiro elemento da lista

void pop(List * list);

- Para remover o primeiro elemento, precisamos:
  - Verificar se a lista já está vazia, se sim, não fazemos nada

Objetivo: Remover o primeiro elemento da lista

void pop(List * list);

- Para remover o primeiro elemento, precisamos:
  - Verificar se a lista já está vazia, se sim, não fazemos nada
  - Utilizar um ponteiro *aux* para guardar a referência do elemento que será removido da lista

Objetivo: Remover o primeiro elemento da lista

void pop(List * list);

- Para remover o primeiro elemento, precisamos:
  - Verificar se a lista já está vazia, se sim, não fazemos nada
  - Utilizar um ponteiro *aux* para guardar a referência do elemento que será removido da lista
  - Fazer com que *list->head* aponte *aux->next*

Objetivo: Remover o primeiro elemento da lista

void pop(List * list);

- Para remover o primeiro elemento, precisamos:
  - Verificar se a lista já está vazia, se sim, não fazemos nada
  - Utilizar um ponteiro *aux* para guardar a referência do elemento que será removido da lista
  - Fazer com que *list->head* aponte *aux->next*
  - Liberar a memória do nó apontado por aux

Objetivo: Remover o primeiro elemento da lista

void pop(List * list);

- Para remover o primeiro elemento, precisamos:
  - Verificar se a lista já está vazia, se sim, não fazemos nada
  - Utilizar um ponteiro *aux* para guardar a referência do elemento que será removido da lista
  - Fazer com que *list->head* aponte *aux->next*
  - Liberar a memória do nó apontado por aux
  - Decrementar o tamanho da lista

void pop(List * list);

```
/*Função para remover o primeiro elemento da lista*/
void pop(List * list){

    if(is_empty(list)){
        return;
    }
    Node * aux = list->head;
    list->head = aux->next;
    free(aux);
    list->size--;
}
```

Exercícios

- Crie um algoritmo que seja capaz de registrar notas em uma lista de notas. O programa deve permitir que o usuário insira quantas notas desejar, até o momento em que o usuário entrar com uma nota negativa (nota < 0). O programa deve listar as notas registradas e perguntar ao usuário quantas notas ele deseja remover. Receba a quantidade de itens a serem removidos (pop) e remova-os. Depois imprima toda a lista novamente.

Exercícios

- Crie a função `Node * at_pos(List * list, int index)`. Esta função retorna o Nó presente na posição `index` da lista `list`.
- Crie a função `int index_of(List * list, Node * node)`. Esta função retorna o índice do Nó `node` na lista `list`.