Estruturas de Dados 1

Listas Duplamente Encadeadas

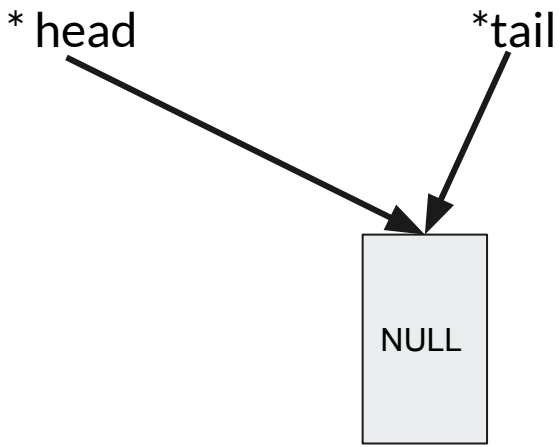
Rafael Fazzolino

Lista Duplamente Encadeada - Definição

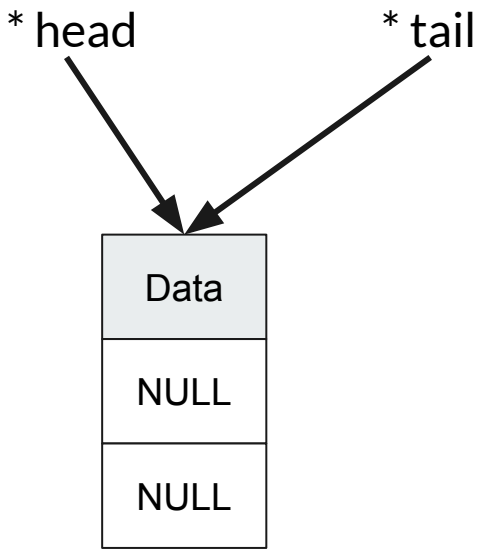
- É uma estrutura composta por nós, onde cada nó armazena um Dado, um ponteiro para o antecessor e um ponteiro para o próximo nó da lista, possibilitando um caminho de ida e volta na lista
- Podemos acessar qualquer elemento a partir de qualquer elemento da lista
- A flexibilidade e poder da lista duplamente encadeada exige a utilização de mais um ponteiro (ant)

Estrutura básica

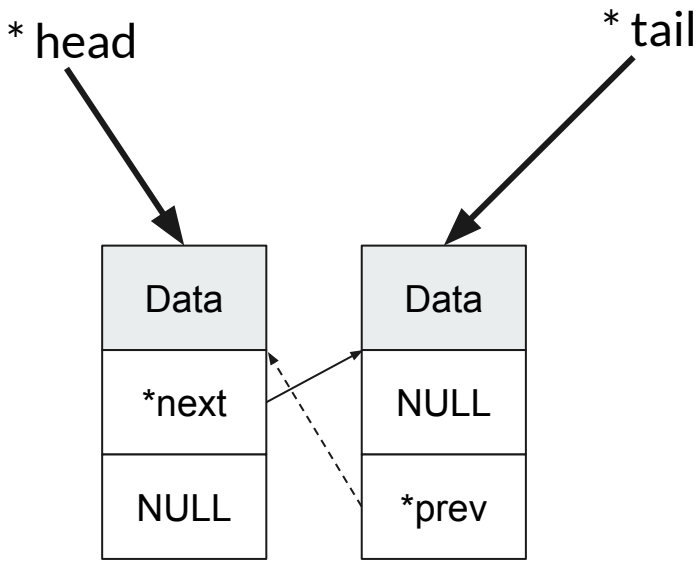
Estrutura básica



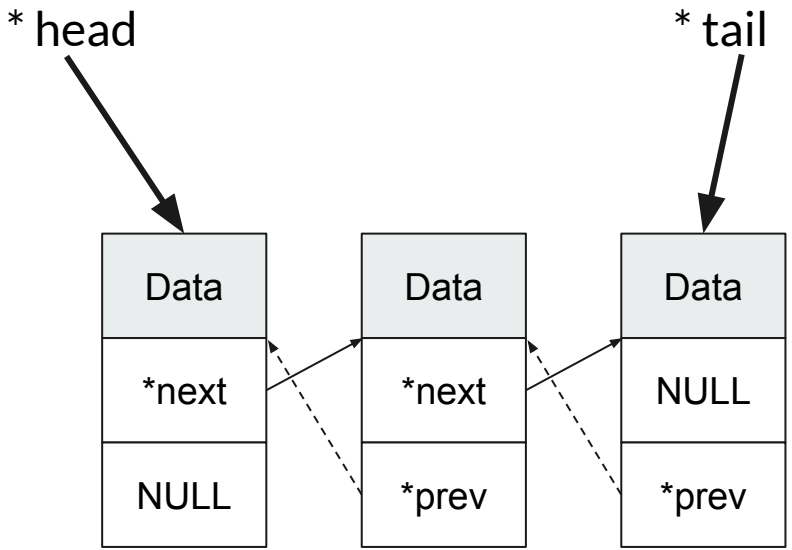
Estrutura básica



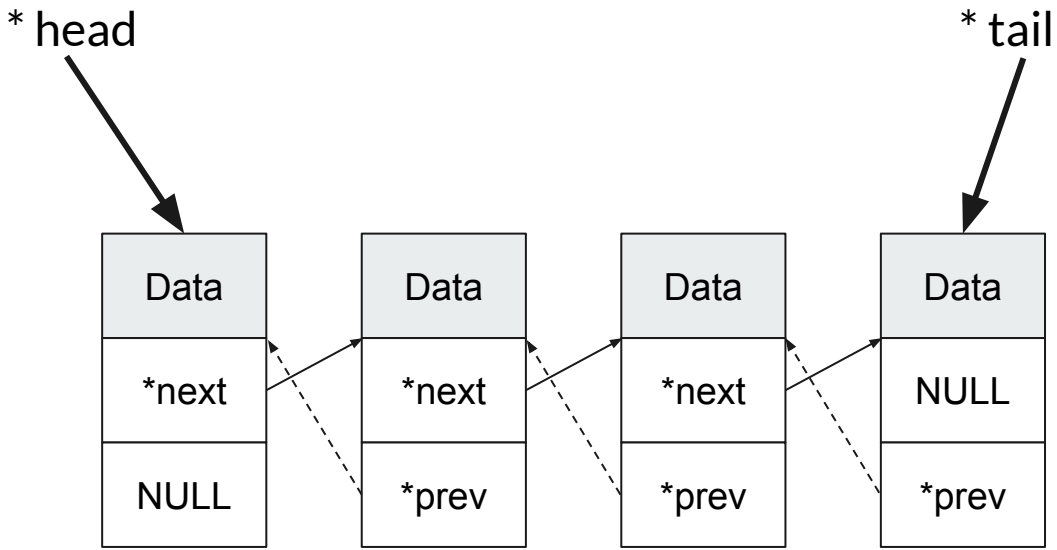
Estrutura básica



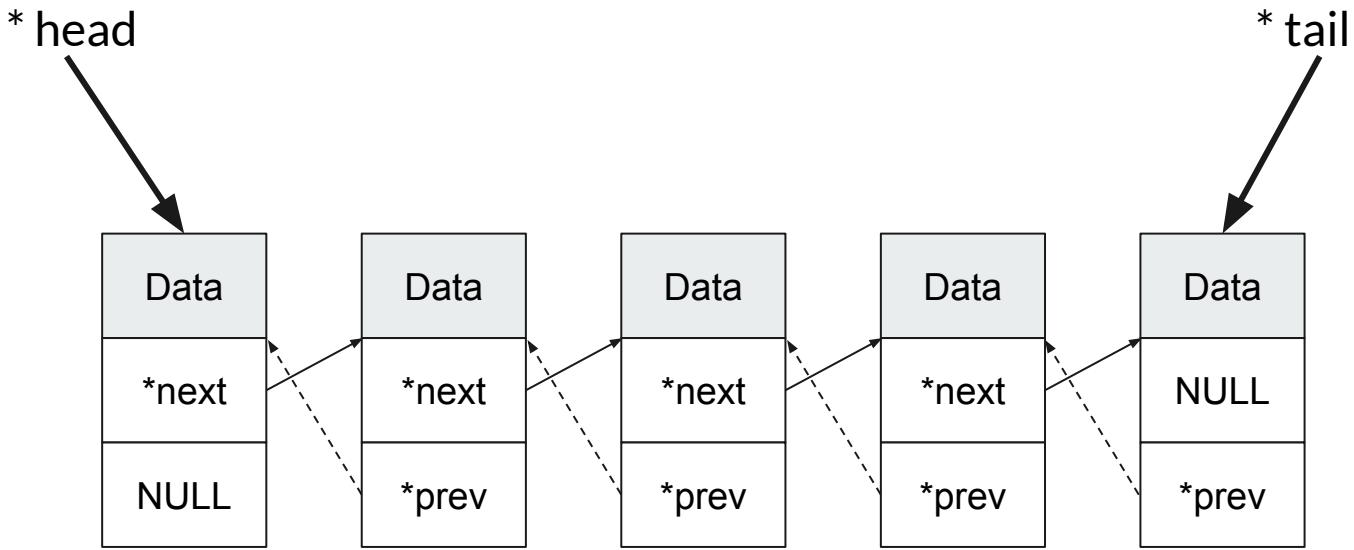
Estrutura básica



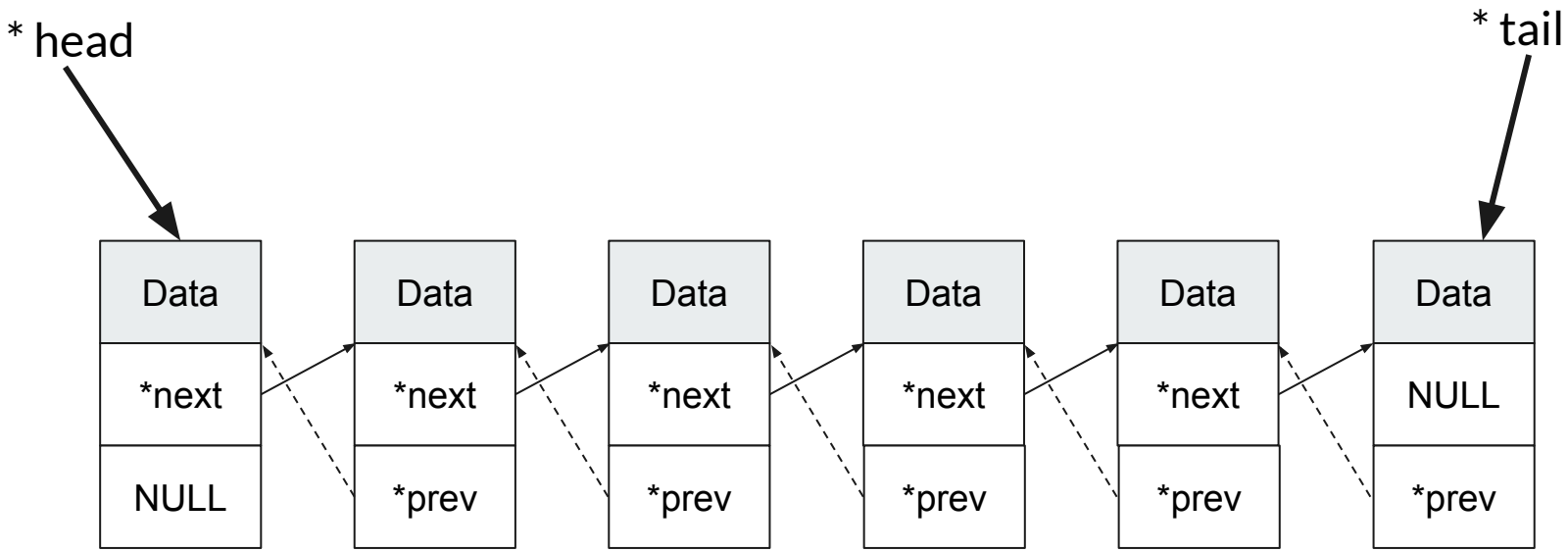
Estrutura básica



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;
    struct node * prev;
}Node;
```

Estrutura básica

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

```
typedef struct lista{
    Node * head;
    Node * tail;
    int size;
}Lista;
```

Funcionamento

- Com uma lista duplamente encadeada, podemos percorrer a lista do início para o final ou do final para o início, por isso temos **head** e **tail**
- Dessa forma, precisamos nos atentar a manter o encadeamento funcionando, gerenciando o **tail**, o **head** e os ponteiros **next** e **prev**

List * create_list()

List * create_list()

- Devemos alocar a memória necessária, atribuir os valores iniciais e retornar a lista:

List * create_list()

- Devemos alocar a memória necessária, atribuir os valores iniciais e retornar a lista:

```
Lista * create_list(){
    Lista * list = (Lista *) malloc(sizeof(Lista));
    list->head = NULL;
    list->tail = NULL;
    list->size = 0;
    return list;
}
```

```
34 void push(Lista * list, Node * node){
35     if(node){
36         /* Se a lista não estiver vazia: */
37         if(list->head){
38             node->next = list->head;
39             list->head->prev = node;
40             list->head = node;
41         }else{/* Se estiver vazia */
42             node->next = list->head;
43             list->head = node;
44             list->tail = node;
45         }
46         list->size++;
47 
48     }else{
49         printf("No invalido!\n");
50     }
51 }
```

Void print(List * list); *início → fim*

```
53 void print_list_normal(Lista * list){
54     Node * aux = list->head;
55     while(aux){
56         printf("%d\n", aux->idade);
57         aux = aux->next;
58     }
59 }
```

Void print(List * list); *fim → inicio*

```
61 void print_list_inversa(Lista * list){
62     Node * aux = list->tail;
63     while(aux){
64         printf("%d\n", aux->idade);
65         aux = aux->prev;
66     }
67 }
```

Exercícios

- Crie um algoritmo que seja capaz de registrar idades em uma lista duplamente encadeada. O programa deve permitir que o usuário insira quantas idades desejar, até o momento em que o usuário entrar com uma idade negativa (idade < 0). Então o programa deve permitir que o usuário escolha a opção de listagem (do primeiro para o último, ou do último para o primeiro).

Exercícios

- Crie a função `void * push_back(List * list, Node * node)`. Esta função insere um novo elemento no final da lista.
- Crie a função `void erase(List * list, Node * node)`. Esta função remove o elemento `node` da lista.