Estruturas de Dados 1

Pilhas

Rafael Fazzolino

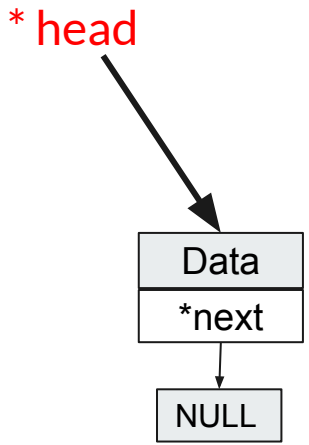
Pilha - Definição

- É uma estrutura composta por Nós, onde cada Nó armazena uma informação e um ponteiro para o próximo Nó da pilha, assim como a lista
- A principal diferença é a regra básica para inserção e remoção de um elemento:
  - **Em uma pilha, o primeiro elemento a ser inserido sempre deverá ser o último elemento a ser removido, e vice versa**
    - **LIFO: Last In First Out**

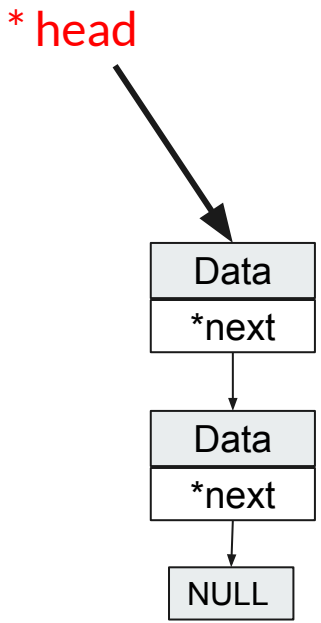
Estrutura básica

Estrutura básica

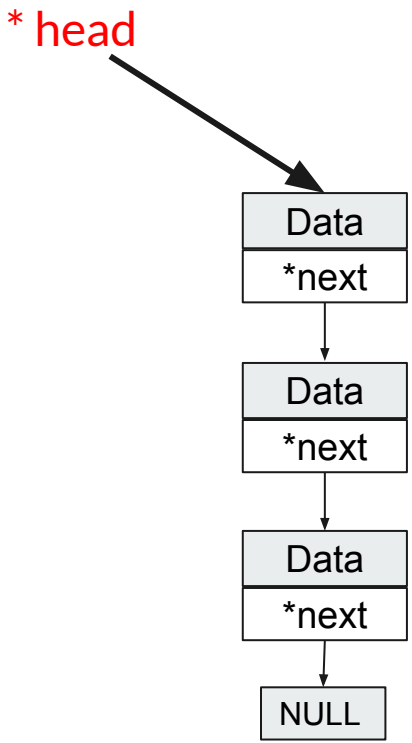
Estrutura básica



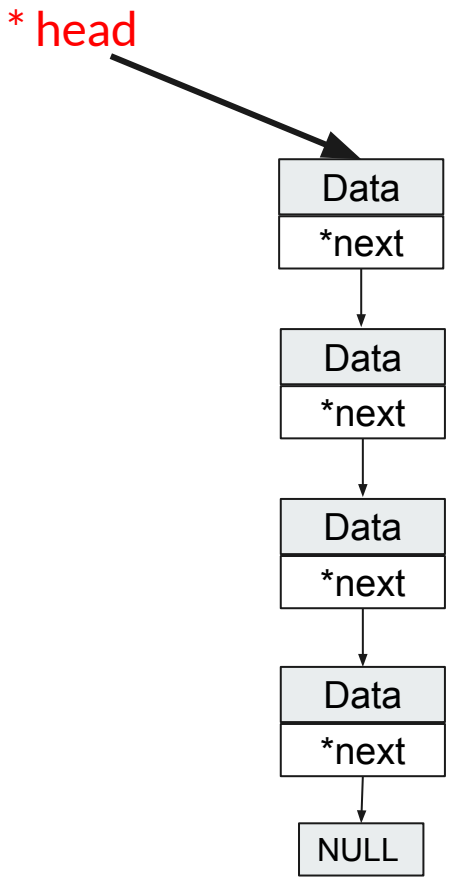
Estrutura básica



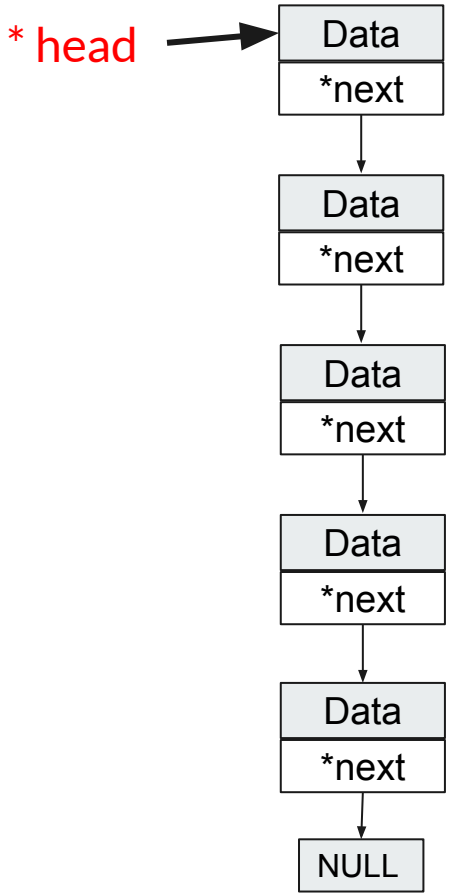
Estrutura básica



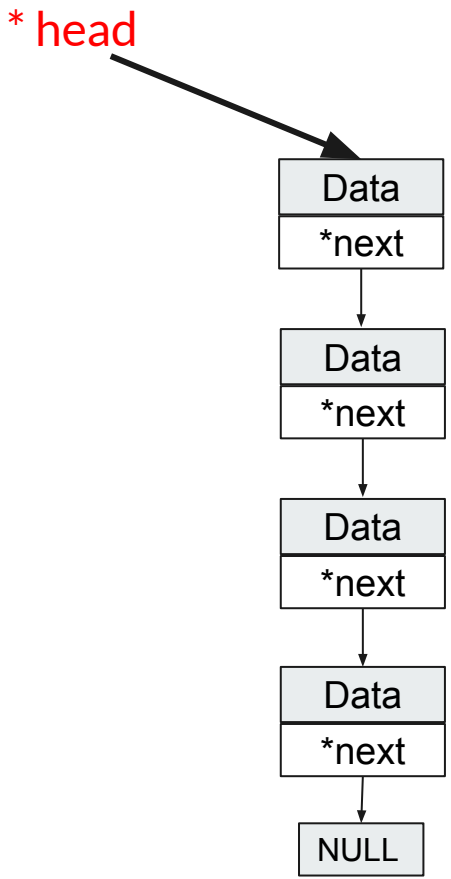
Estrutura básica



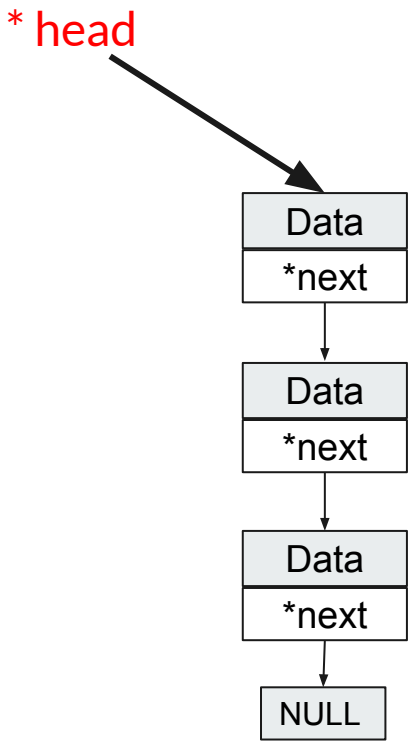
Estrutura básica



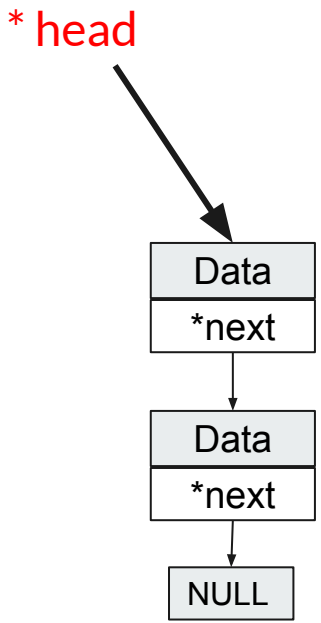
Estrutura básica



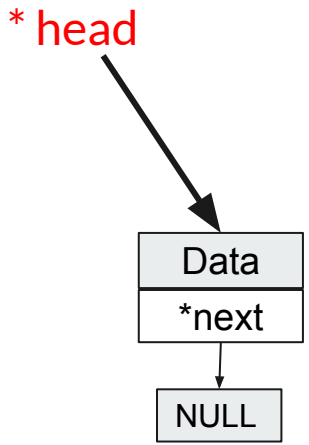
Estrutura básica



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 pilha também pode ser definida a partir de estruturas, por exemplo:

```
typedef struct stack{
    int size;
    Node * top;
}Stack;
```

Funções principais

Funções principais

```
Stack * create_stack();
void push(Stack * stack, Node * node);
void pop(Stack * stack);
bool is_empty(Stack * stack);
void clear(Stack * stack); //Limpa toda a pilha, removendo todos os elementos
void top(Stack * stack); // imprime (ou retorna) o elemento no topo da lista
```

Stack * create_stack();

Stack * create_stack();

Objetivo: Criar uma stack vazia

- Para criar uma lista, devemos:

Stack * create_stack();

Objetivo: Criar uma stack vazia

- Para criar uma pilha, devemos:
  - Alocar espaço de memória suficiente para tal
    - Stack * stack = (Stack *) malloc(sizeof(Stack));

Stack * create_stack();

Objetivo: Criar uma stack vazia

- Para criar uma pilha, devemos:
  - Alocar espaço de memória suficiente para tal
    - Stack * stack = (Stack *) malloc(sizeof(Stack));
  - Inicializar o topo da pilha como NULL
    - stack->top = NULL;

Stack * create_stack();

Objetivo: Criar uma stack vazia

- Para criar uma pilha, devemos:
  - Alocar espaço de memória suficiente para tal
    - Stack * stack = (Stack *) malloc(sizeof(Stack));
  - Inicializar o topo da pilha como NULL
    - stack->top = NULL;
  - Inicializar o tamanho como 0
    - stack->size = 0;

Stack * create_stack();

```
22 Stack * create_stack(){
23     Stack * stack = (Stack *) malloc(sizeof(Stack));
24     stack->size = 0;
25     stack->top = NULL;
26     return stack;
27 }
```

Objetivo: Inserir elemento no topo da pilha

void push(Stack * stack, Node * node);

Objetivo: Inserir elemento no topo da pilha

void push(Stack * stack, Node * node);

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

Objetivo: Inserir elemento no topo da pilha

void push(Stack * stack, Node * node);

- Para incluir um elemento no topo da pilha, deve-se, primeiramente, ter um Node alocado e pronto para inserção
- Então devemos fazer com que o node->next aponte para stack->top

Objetivo: Inserir elemento no topo da pilha

void push(Stack * stack, Node * node);

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

Objetivo: Inserir elemento no topo da pilha

void push(Stack * stack, Node * node);

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

void push(Stack * stack, Node * node);

```
43  void push(Stack * stack, Node * node){
44      if(node){
45          node->next = stack->top;
46          stack->top = node;
47          stack->size++;
48      }
49  }
```

Objetivo: Verificar se a pilha está vazia

bool is_empty(Stack * stack);

Objetivo: Verificar se a pilha está vazia

bool is_empty(Stack * stack);

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

bool is_empty(Stack * stack);

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

```
29  bool is_empty(Stack * stack){
30      return stack->size == 0;
31  }
```

Objetivo: Imprimir ou retornar o topo da pilha

void top(Stack * stack);

- Imprimir ou retornar o topo da pilha:

Objetivo: Imprimir ou retornar o topo da pilha

void top(Stack * stack);

- Imprimir ou retornar o topo da pilha:
  - Verificar se a pilha está vazia, se estiver, informar que está vazia e retornar

Objetivo: Imprimir ou retornar o topo da pilha

void top(Stack * stack);

- Imprimir ou retornar o topo da pilha:
  - Verificar se a pilha está vazia, se estiver, informar que está vazia e retornar
  - Imprimir ou retornar o elemento no topo da pilha:
    - return stack->top

void top(Stack * stack);

```
52 void top(Stack * stack){
53     if(stack->top){
54         printf("TOP: %d\n", stack->top->idade);
55     }else{
56         printf("Pilha Vazia!\n");
57     }
58 }
```

Objetivo: Remover o elemento no topo da pilha

void pop(Stack * stack);

Objetivo: Remover o elemento no topo da pilha

void pop(Stack * stack);

- Para remover o primeiro elemento, precisamos:

Objetivo: Remover o elemento no topo da pilha

void pop(Stack * stack);

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

Objetivo: Remover o elemento no topo da pilha

void pop(Stack * stack);

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

Objetivo: Remover o elemento no topo da pilha

void pop(Stack * stack);

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

Objetivo: Remover o elemento no topo da pilha

void pop(Stack * stack);

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

Objetivo: Remover o elemento no topo da pilha

void pop(Stack * stack);

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

void pop(Stack * stack);

```
33 void pop(Stack * stack){
34     if(is_empty(stack)){
35         return;
36     }
37     Node * aux = stack->top;
38     stack->top = aux->next;
39     stack->size--;
40     free(aux);
41 }
```

Exercícios

- Faça um programa que permita ao usuário simular o comportamento de uma pilha. Para isso, faça um programa com o seguinte menu:
  - Inserir aluno (nome)
  - Remover topo
  - Visualizar tamanho da pilha
  - Imprimir pilha completa
  - Pesquisar aluno
  - Limpar pilha (remover todos os alunos)

Exercício da lista 5