**Implementando**

#include <stdio.h>

#include <stdlib.h>

// Definindo a estrutura do nó da lista

struct No {

int valor; // Valor armazenado no nó

struct No \*prox; // Ponteiro para o próximo nó

};

// Função para inserir um novo nó no início da lista

struct No\* inserirNoInicio(struct No \*head, int novoValor) {

struct No \*novoNo = (struct No\*)malloc(sizeof(struct No)); // Aloca memória para o novo nó

novoNo->valor = novoValor; // Atribui o valor desejado ao novo nó

novoNo->prox = head; // Faz o ponteiro 'prox' do novo nó apontar para o antigo primeiro nó (head)

return novoNo; // Retorna o novo nó como sendo o novo início (head) da lista

};

// Função para imprimir a lista encadeada

void imprimirLista(struct No \*head) {

struct No \*atual = head; // Cria um ponteiro auxiliar para percorrer a lista

printf("Lista: ");

while (atual != NULL) {

printf("%d -> ", atual->valor); // Imprime o valor do nó atual

atual = atual->prox; // Avança para o próximo nó

}

printf("NULL\n"); // Indica o fim da lista

};

int main() {

struct No \*lista = NULL; // Inicialmente a lista está vazia (NULL)

// Construindo a lista inicial [10 | \*] → [20 | \*] → [30 | NULL]

lista = inserirNoInicio(lista, 30); // Insere 30 no início → lista = [30 | NULL]

lista = inserirNoInicio(lista, 20); // Insere 20 no início → lista = [20 | \*] → [30 | NULL]

lista = inserirNoInicio(lista, 10); // Insere 10 no início → lista = [10 | \*] → [20 | \*] → [30 | NULL]

// Inserindo o valor 5 no início → [5 | \*] → [10 | \*] → [20 | \*] → [30 | NULL]

lista = inserirNoInicio(lista, 5);

// Imprimir a lista final

imprimirLista(lista);

return 0;

}

#include <stdio.h>

#include <stdlib.h>

// Definindo a estrutura do nó

struct No {

int valor;

struct No \*prox;

};

// Função para inserir no fim da lista

struct No\* inserirNoFim(struct No \*head, int novoValor) {

// 1. Alocar memória para o novo nó

struct No \*novoNo = (struct No\*)malloc(sizeof(struct No));

// 2. Atribuir o valor ao novo nó

novoNo->valor = novoValor;

// 3. O novo nó será o último, portanto 'prox' será NULL

novoNo->prox = NULL;

// 4. Se a lista estiver vazia (head == NULL), o novo nó será o primeiro

if (head == NULL) {

return novoNo;

}

// 5. Percorrer a lista até o último nó

struct No \*atual = head;

while (atual->prox != NULL) {

atual = atual->prox;

}

// 6. Fazer o último nó apontar para o novo nó

atual->prox = novoNo;

// 7. Retorna o início da lista (head não muda)

return head;

}

// Função para imprimir a lista encadeada

void imprimirLista(struct No \*head) {

struct No \*atual = head;

printf("Lista: ");

while (atual != NULL) {

printf("%d -> ", atual->valor);

atual = atual->prox;

}

printf("NULL\n");

}

#include <stdio.h>

#include <stdlib.h>

int main() {

struct No \*lista = NULL; // Lista vazia inicialmente

// Construindo a lista inserindo no fim

lista = inserirNoFim(lista, 10); // Lista = [10 | NULL]

lista = inserirNoFim(lista, 20); // Lista = [10 | \*] → [20 | NULL]

lista = inserirNoFim(lista, 30); // Lista = [10 | \*] → [20 | \*] → [30 | NULL]

// Inserindo 40 no fim

lista = inserirNoFim(lista, 40); // Lista = [10 | \*] → [20 | \*] → [30 | \*] → [40 | NULL]

// Imprimindo a lista final

imprimirLista(lista);

return 0;

}

**LISTA ENCADEADA**

#include <stdio.h>

#include <stdlib.h>

// Estrutura do nó

struct No {

int valor;

struct No \*prox;

};

// Função para remover um nó com valor específico

struct No\* removerElemento(struct No \*head, int valorRemover) {

// 1. Verifica se a lista está vazia

if (head == NULL) {

printf("Lista vazia.\n");

return NULL;

}

// 2. Verifica se o valor está no primeiro nó

if (head->valor == valorRemover) {

struct No \*temp = head; // Guarda o nó a ser removido

head = head->prox; // Atualiza o head para o próximo nó

free(temp); // Libera o nó removido

return head; // Retorna o novo head

}

// 3. Caso contrário, percorre a lista procurando o valor

struct No \*anterior = head;

struct No \*atual = head->prox;

while (atual != NULL && atual->valor != valorRemover) {

anterior = atual;

atual = atual->prox;

}

// 4. Se não encontrou o valor

if (atual == NULL) {

printf("Valor %d não encontrado na lista.\n", valorRemover);

return head;

}

// 5. Encontrou o valor: remove o nó

anterior->prox = atual->prox; // Pula o nó a ser removido

free(atual); // Libera a memória do nó

return head; // Retorna o head da lista

}

#include <stdio.h>

#include <stdlib.h>

// Função para imprimir a lista

void imprimirLista(struct No \*head) {

struct No \*atual = head;

printf("Lista: ");

while (atual != NULL) {

printf("%d -> ", atual->valor);

atual = atual->prox;

}

printf("NULL\n");

}

// Função auxiliar para inserir no fim

struct No\* inserirNoFim(struct No \*head, int valor) {

struct No \*novo = (struct No\*)malloc(sizeof(struct No));

novo->valor = valor;

novo->prox = NULL;

if (head == NULL) return novo;

struct No \*atual = head;

while (atual->prox != NULL) {

atual = atual->prox;

}

atual->prox = novo;

return head;

}

int main() {

struct No \*lista = NULL;

// Criando a lista [10 -> 20 -> 30 -> 40]

lista = inserirNoFim(lista, 10);

lista = inserirNoFim(lista, 20);

lista = inserirNoFim(lista, 30);

lista = inserirNoFim(lista, 40);

imprimirLista(lista); // Mostra a lista inicial

// Remover o valor 30

lista = removerElemento(lista, 30);

imprimirLista(lista); // Esperado: 10 -> 20 -> 40 -> NULL

// Tentar remover valor inexistente

lista = removerElemento(lista, 99);

// Remover o primeiro elemento (10)

lista = removerElemento(lista, 10);

imprimirLista(lista); // Esperado: 20 -> 40 -> NULL

return 0;

}