Lista encadeada – 12/05

Ex1

#include <stdio.h>

#include <stdlib.h>

// Definindo a estrutura de 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\* inserirInicio(struct No \*head, int novovalor) {

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

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

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

    return novo;                   // 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

    while (atual != NULL) {        // Enquanto não chegar no final (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

}

Ex 2

int main() {

    struct No \*lista = NULL;       // Inicialmente a lista está vazia [NULL]

    // Vamos inserir os valores 10, 20 e 30 no início da lista

    lista = inserirInicio(lista, 10); // Insere 10 no início -> [10] -> [NULL]

    lista = inserirInicio(lista, 20); // Insere 20 no início -> [20] -> [10] -> [NULL]

    lista = inserirInicio(lista, 30); // Insere 30 no início -> [30] -> [20] -> [10] -> [NULL]

    // Agora vamos inserir o valor 5 no início e queremos [5] -> [30] -> [20] -> [10] -> [NULL]

    lista = inserirInicio(lista, 5);

    imprimirLista(lista);

    return 0;

}

Exercício 3

#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 nó

    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. Retornar a cabeça 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");

}

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]

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

    // Imprimindo a lista final

    imprimirLista(lista);

    return 0;

}

Exercício 4

#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;

        head = head->prox;             // Guarda o nó a ser removido

        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 a head da lista

}

// 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 (reaproveitando o que já vimos)

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

    lista = removerElemento(lista, 10);

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

    return 0;

}

Exercício 5(Percorrer a Lista)

#include <stdio.h>

#include <stdlib.h>

// Estrutura do nó

struct No {

int valor;

struct No\* prox;

};

// Função para percorrer e imprimir a lista

void imprimirLista(struct No\* head) {

struct No\* atual = head; // Cria um ponteiro auxiliar que começa no head

while (atual != NULL) { // Enquanto o ponteiro não for NULL (fim da lista)

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

}

// Função auxiliar para inserir no fim (já conhecida)

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 -> NULL]

lista = inserirNoFim(lista, 10);

lista = inserirNoFim(lista, 20);

lista = inserirNoFim(lista, 30);

// Imprimindo a lista

imprimirLista(lista);

return 0;

}

Exercício 6 - Buscar um Elemento

#include <stdio.h>

#include <stdlib.h>

// Estrutura do nó

struct No {

int valor;

struct No \*prox;

};

// Função para buscar um valor na lista

struct No\* buscarElemento(struct No \*head, int valorBusca) {

struct No \*atual = head; // Ponteiro auxiliar para percorrer

while (atual != NULL) { // Percorre até o fim da lista

if (atual->valor == valorBusca) { // Se encontrou o valor

return atual; // Retorna o endereço do nó encontrado

}

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

}

return NULL; // Se não encontrar, retorna NULL

}

// 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 (reaproveitando)

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 -> NULL

lista = inserirNoFim(lista, 10);

lista = inserirNoFim(lista, 20);

lista = inserirNoFim(lista, 30);

imprimirLista(lista);

// Buscar valor 20

struct No \*resultado = buscarElemento(lista, 20);

if (resultado != NULL) {

printf("Valor %d encontrado no endereço %p.\n", resultado->valor, (void\*)resultado);

} else {

printf("Valor 20 não encontrado.\n");

}

// Buscar valor inexistente

resultado = buscarElemento(lista, 99);

if (resultado != NULL) {

printf("Valor %d encontrado no endereço %p.\n", resultado->valor, (void\*)resultado);

} else {

printf("Valor 99 não encontrado.\n");

}

return 0;

}