**BLOCO 1 – VETORES**

**Exercício 1:** Declare um vetor de 5 inteiros. Peça para o usuário digitar os 5 números e, ao final, exiba todos os números digitados.

#include <stdio.h>

int main() {

int vetor[5]; // Vetor para armazenar 5 inteiros

int i; // Variável de controle do loop

printf("Digite 5 números inteiros:\n");

// Preenche o vetor com valores digitados pelo usuário

for(i = 0; i < 5; i++) {

printf("Número %d: ", i + 1);

scanf("%d", &vetor[i]);

}

printf("\nNúmeros digitados:\n");

// Exibe os valores armazenados no vetor

for(i = 0; i < 5; i++) {

printf("%d ", vetor[i]);

}

printf("\n"); // Quebra de linha final

return 0;

}

**Exercício 2:** Dado um vetor de 6 números inteiros já preenchido, faça uma função que encontre e retorne o maior valor.

#include <stdio.h>

// Função para encontrar o maior valor em um vetor

int encontrarMaior(int vetor[], int tamanho) {

int maior = vetor[0];

int i;

for(i = 1; i < tamanho; i++) {

if(vetor[i] > maior) {

maior = vetor[i];

}

}

return maior;

}

int main() {

int numeros[6] = {15, 8, 23, 4, 42, 17}; // Vetor pré-preenchido

int maior\_valor;

maior\_valor = encontrarMaior(numeros, 6);

printf("O maior valor no vetor é: %d\n", maior\_valor);

return 0;

}

**BLOCO 2 – STRUCTS**

**Exercício 3:** Defina uma struct Aluno com nome (char[40]) e nota (float). Faça um programa que leia os dados de 1 aluno e imprima.

#include <stdio.h>

#include <string.h>

// Definição da struct Aluno

struct Aluno {

char nome[40];

float nota;

};

int main() {

struct Aluno aluno; // Cria uma variável do tipo Aluno

// Leitura dos dados

printf("Cadastro de Aluno\n");

printf("Nome: ");

fgets(aluno.nome, 40, stdin);

aluno.nome[strcspn(aluno.nome, "\n")] = '\0'; // Remove o \n do final

printf("Nota: ");

scanf("%f", &aluno.nota);

// Exibição dos dados

printf("\nDados do Aluno:\n");

printf("Nome: %s\n", aluno.nome);

printf("Nota: %.2f\n", aluno.nota);

return 0;

}

**Exercício 4:** Usando struct, leia e exiba os dados de 3 produtos: nome e preço.

#include <stdio.h>

#include <string.h>

// Definição da struct Produto

struct Produto {

char nome[40];

float preco;

};

int main() {

struct Produto produtos[3]; // Array de 3 produtos

int i;

// Leitura dos dados

printf("Cadastro de Produtos\n");

for(i = 0; i < 3; i++) {

printf("\nProduto %d:\n", i+1);

printf("Nome: ");

getchar(); // Limpa o buffer do teclado

fgets(produtos[i].nome, 40, stdin);

produtos[i].nome[strcspn(produtos[i].nome, "\n")] = '\0';

printf("Preço: R$ ");

scanf("%f", &produtos[i].preco);

}

// Exibição dos dados

printf("\nLista de Produtos:\n");

for(i = 0; i < 3; i++) {

printf("\nProduto %d:\n", i+1);

printf("Nome: %s\n", produtos[i].nome);

printf("Preço: R$ %.2f\n", produtos[i].preco);

}

return 0;

}

**BLOCO 3 – LISTAS ENCADEADAS**

**Exercício 5:** Implemente uma lista encadeada simples que permita inserir valores no início e imprimir a lista.

#include <stdio.h>

#include <stdlib.h>

// Estrutura do nó da lista

typedef struct No {

int valor;

struct No\* proximo;

} No;

// Função para inserir no início

void inserirInicio(No\*\* cabeca, int valor) {

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

novo->valor = valor;

novo->proximo = \*cabeca;

\*cabeca = novo;

}

// Função para imprimir a lista

void imprimirLista(No\* cabeca) {

No\* atual = cabeca;

printf("Lista: ");

while(atual != NULL) {

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

atual = atual->proximo;

}

printf("\n");

}

// Função para liberar a memória

void liberarLista(No\* cabeca) {

No\* atual = cabeca;

while(atual != NULL) {

No\* temp = atual;

atual = atual->proximo;

free(temp);

}

}

int main() {

No\* lista = NULL; // Lista vazia

// Inserindo elementos

inserirInicio(&lista, 3);

inserirInicio(&lista, 2);

inserirInicio(&lista, 1);

// Imprimindo a lista

imprimirLista(lista);

// Liberando memória

liberarLista(lista);

return 0;

}

**Exercício 6:** Implemente uma pilha com lista encadeada. Faça operações de push e pop.

#include <stdio.h>

#include <stdlib.h>

// Estrutura do nó da pilha

typedef struct NoPilha {

int valor;

struct NoPilha\* proximo;

} NoPilha;

// Função push (empilhar)

void push(NoPilha\*\* topo, int valor) {

NoPilha\* novo = (NoPilha\*)malloc(sizeof(NoPilha));

novo->valor = valor;

novo->proximo = \*topo;

\*topo = novo;

}

// Função pop (desempilhar)

int pop(NoPilha\*\* topo) {

if(\*topo == NULL) {

printf("Pilha vazia!\n");

return -1; // Valor de erro

}

NoPilha\* temp = \*topo;

int valor = temp->valor;

\*topo = (\*topo)->proximo;

free(temp);

return valor;

}

// Função para imprimir a pilha

void imprimirPilha(NoPilha\* topo) {

NoPilha\* atual = topo;

printf("Pilha (topo->base): ");

while(atual != NULL) {

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

atual = atual->proximo;

}

printf("\n");

}

int main() {

NoPilha\* pilha = NULL;

// Empilhando valores

push(&pilha, 10);

push(&pilha, 20);

push(&pilha, 30);

imprimirPilha(pilha);

// Desempilhando

printf("Pop: %d\n", pop(&pilha));

printf("Pop: %d\n", pop(&pilha));

printf("Pop: %d\n", pop(&pilha));

// Tentando desempilhar pilha vazia

printf("Pop: %d\n", pop(&pilha));

return 0;

}

**BLOCO 5 – FILAS**

**Exercício 7:** Implemente uma fila usando lista encadeada. Faça operações de enqueue e dequeue

#include <stdio.h>

#include <stdlib.h>

// Estrutura do nó da fila

typedef struct NoFila {

int valor;

struct NoFila\* proximo;

} NoFila;

// Estrutura da fila (contendo ponteiros para início e fim)

typedef struct {

NoFila\* frente;

NoFila\* tras;

} Fila;

// Inicializa uma fila vazia

void inicializarFila(Fila\* fila) {

fila->frente = NULL;

fila->tras = NULL;

}

// Verifica se a fila está vazia

int estaVazia(Fila\* fila) {

return (fila->frente == NULL);

}

// Operação enqueue (inserir no final)

void enqueue(Fila\* fila, int valor) {

NoFila\* novoNo = (NoFila\*)malloc(sizeof(NoFila));

if(novoNo == NULL) {

printf("Erro: memória insuficiente\n");

exit(1);

}

novoNo->valor = valor;

novoNo->proximo = NULL;

if(estaVazia(fila)) {

fila->frente = fila->tras = novoNo;

} else {

fila->tras->proximo = novoNo;

fila->tras = novoNo;

}

}

// Operação dequeue (remover do início)

int dequeue(Fila\* fila) {

if(estaVazia(fila)) {

printf("Erro: Fila vazia\n");

return -1; // Valor de erro

}

NoFila\* temp = fila->frente;

int valor = temp->valor;

fila->frente = fila->frente->proximo;

// Se a fila ficou vazia após a remoção

if(fila->frente == NULL) {

fila->tras = NULL;

}

free(temp);

return valor;

}

// Imprime o conteúdo da fila

void imprimirFila(Fila\* fila) {

if(estaVazia(fila)) {

printf("Fila vazia\n");

return;

}

NoFila\* atual = fila->frente;

printf("Fila (frente -> tras): ");

while(atual != NULL) {

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

atual = atual->proximo;

}

printf("\n");

}

// Libera a memória alocada para a fila

void liberarFila(Fila\* fila) {

while(!estaVazia(fila)) {

dequeue(fila);

}

}

int main() {

Fila fila;

inicializarFila(&fila);

// Testando as operações

enqueue(&fila, 10);

enqueue(&fila, 20);

enqueue(&fila, 30);

imprimirFila(&fila);

printf("Dequeue: %d\n", dequeue(&fila));

printf("Dequeue: %d\n", dequeue(&fila));

imprimirFila(&fila);

enqueue(&fila, 40);

enqueue(&fila, 50);

imprimirFila(&fila);

printf("Dequeue: %d\n", dequeue(&fila));

printf("Dequeue: %d\n", dequeue(&fila));

printf("Dequeue: %d\n", dequeue(&fila)); // Tentando remover de fila vazia

liberarFila(&fila);

return 0;

}