**DESAFIO 1 — VETORES + ORDENAÇÃO**

**Enunciado:** Crie um programa que leia 10 números inteiros e os ordene em **ordem decrescente**. Imprima o vetor ordenado.

#include <stdio.h>

void ordenarDecrescente(int vetor[], int tamanho) {

int i, j, temp;

for (i = 0; i < tamanho - 1; i++) {

for (j = 0; j < tamanho - i - 1; j++) {

if (vetor[j] < vetor[j + 1]) {

// Troca os elementos

temp = vetor[j];

vetor[j] = vetor[j + 1];

vetor[j + 1] = temp;

}

}

}

}

int main() {

int numeros[10];

int i;

printf("Digite 10 números inteiros:\n");

for (i = 0; i < 10; i++) {

printf("Número %d: ", i + 1);

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

}

ordenarDecrescente(numeros, 10);

printf("\nVetor ordenado em ordem decrescente:\n");

for (i = 0; i < 10; i++) {

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

}

printf("\n");

return 0;

}

**DESAFIO 2 — STRUCTS + LISTA ENCADEADA**

**Enunciado:** Crie uma lista encadeada de **alunos**, com: Nome (char[40]) Nota (float)

O programa deve permitir: 1Inserir aluno **ordenado por nome (ordem alfabética)** 2 Imprimir a lista de alunos

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Aluno {

char nome[40];

float nota;

struct Aluno\* proximo;

} Aluno;

Aluno\* criarAluno(const char\* nome, float nota) {

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

if (!novo) {

printf("Erro ao alocar memória!\n");

exit(1);

}

strncpy(novo->nome, nome, 39);

novo->nome[39] = '\0';

novo->nota = nota;

novo->proximo = NULL;

return novo;

}

void inserirOrdenado(Aluno\*\* cabeca, const char\* nome, float nota) {

Aluno\* novo = criarAluno(nome, nota);

// Caso 1: Lista vazia ou novo aluno deve ser o primeiro

if (\*cabeca == NULL || strcmp(nome, (\*cabeca)->nome) < 0) {

novo->proximo = \*cabeca;

\*cabeca = novo;

} else {

// Caso 2: Encontrar a posição correta

Aluno\* atual = \*cabeca;

while (atual->proximo != NULL && strcmp(nome, atual->proximo->nome) > 0) {

atual = atual->proximo;

}

novo->proximo = atual->proximo;

atual->proximo = novo;

}

}

void imprimirLista(Aluno\* cabeca) {

Aluno\* atual = cabeca;

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

printf("---------------------------------\n");

printf("Nome\t\tNota\n");

printf("---------------------------------\n");

while (atual != NULL) {

printf("%-20s%.1f\n", atual->nome, atual->nota);

atual = atual->proximo;

}

printf("---------------------------------\n");

}

void liberarLista(Aluno\* cabeca) {

Aluno\* atual = cabeca;

while (atual != NULL) {

Aluno\* temp = atual;

atual = atual->proximo;

free(temp);

}

}

int main() {

Aluno\* lista = NULL;

int opcao;

char nome[40];

float nota;

do {

printf("\nMenu:\n");

printf("1. Inserir aluno (ordenado por nome)\n");

printf("2. Imprimir lista de alunos\n");

printf("3. Sair\n");

printf("Escolha uma opção: ");

scanf("%d", &opcao);

switch (opcao) {

case 1:

printf("Digite o nome do aluno: ");

scanf(" %39[^\n]", nome); // Lê até 39 caracteres ou até nova linha

printf("Digite a nota do aluno: ");

scanf("%f", &nota);

inserirOrdenado(&lista, nome, nota);

break;

case 2:

imprimirLista(lista);

break;

case 3:

printf("Encerrando o programa...\n");

break;

default:

printf("Opção inválida!\n");

}

} while (opcao != 3);

liberarLista(lista);

return 0;

}

**DESAFIO 3 — PILHA DE NÚMEROS**

**Enunciado:** Implemente uma pilha de números inteiros e: Armazene 5 números digitados pelo usuário Em seguida, esvazie a pilha, mas **mostrando o somatório dos valores que foram retirados da pilha**.

#include <stdio.h>

#include <stdlib.h>

#define TAMANHO\_MAX 5

typedef struct {

int dados[TAMANHO\_MAX];

int topo;

} Pilha;

void inicializarPilha(Pilha \*p) {

p->topo = -1;

}

int pilhaVazia(Pilha \*p) {

return (p->topo == -1);

}

int pilhaCheia(Pilha \*p) {

return (p->topo == TAMANHO\_MAX - 1);

}

void empilhar(Pilha \*p, int valor) {

if (pilhaCheia(p)) {

printf("Erro: Pilha cheia!\n");

return;

}

p->dados[++(p->topo)] = valor;

}

int desempilhar(Pilha \*p) {

if (pilhaVazia(p)) {

printf("Erro: Pilha vazia!\n");

return -1;

}

return p->dados[(p->topo)--];

}

int main() {

Pilha p;

inicializarPilha(&p);

int i, num, soma = 0;

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

for (i = 0; i < TAMANHO\_MAX; i++) {

printf("Número %d: ", i + 1);

scanf("%d", &num);

empilhar(&p, num);

}

printf("\nDesempilhando e somando os valores:\n");

while (!pilhaVazia(&p)) {

num = desempilhar(&p);

printf("Valor retirado: %d\n", num);

soma += num;

}

printf("\nSomatório dos valores: %d\n", soma);

return 0;

}

**DESAFIO 4 — FILA COM PRIORIDADE SIMPLES**

**Enunciado:** Implemente uma fila com prioridade simulada: Pacientes com **idade >= 60** devem entrar na **frente da fila**. Pacientes < 60 entram no final.

Peça o nome e idade de 5 pacientes e imprima a ordem de atendimento.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_PACIENTES 5

typedef struct Paciente {

char nome[50];

int idade;

struct Paciente \*proximo;

} Paciente;

typedef struct {

Paciente \*inicio;

Paciente \*fim;

} Fila;

void inicializarFila(Fila \*f) {

f->inicio = NULL;

f->fim = NULL;

}

int filaVazia(Fila \*f) {

return (f->inicio == NULL);

}

void inserirPrioritario(Fila \*f, const char \*nome, int idade) {

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

if (!novo) {

printf("Erro ao alocar memória!\n");

exit(1);

}

strcpy(novo->nome, nome);

novo->idade = idade;

novo->proximo = NULL;

if (filaVazia(f)) {

f->inicio = novo;

f->fim = novo;

} else if (idade >= 60) {

// Insere no início (prioridade)

novo->proximo = f->inicio;

f->inicio = novo;

} else {

// Insere no final (sem prioridade)

f->fim->proximo = novo;

f->fim = novo;

}

}

void imprimirFila(Fila \*f) {

Paciente \*atual = f->inicio;

int posicao = 1;

printf("\nOrdem de atendimento:\n");

printf("-----------------------------\n");

printf("Posição | Nome | Idade\n");

printf("-----------------------------\n");

while (atual != NULL) {

printf("%-7d | %-10s | %d\n", posicao++, atual->nome, atual->idade);

atual = atual->proximo;

}

printf("-----------------------------\n");

}

void liberarFila(Fila \*f) {

Paciente \*atual = f->inicio;

while (atual != NULL) {

Paciente \*temp = atual;

atual = atual->proximo;

free(temp);

}

f->inicio = NULL;

f->fim = NULL;

}

int main() {

Fila fila;

inicializarFila(&fila);

char nome[50];

int idade, i;

printf("Cadastro de 5 pacientes:\n");

for (i = 0; i < MAX\_PACIENTES; i++) {

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

printf("Nome: ");

scanf(" %49[^\n]", nome); // Lê até 49 caracteres ou até nova linha

printf("Idade: ");

scanf("%d", &idade);

inserirPrioritario(&fila, nome, idade);

}

imprimirFila(&fila);

liberarFila(&fila);

return 0;

}

**DESAFIO 5— LISTA DUPLAMENTE ENCADEADA CIRCULAR — Inserir no Fim e Remover por Valor**

**Enunciado:** Implemente uma **Lista Duplamente Encadeada Circular** que permita: Inserir no fim Remover um elemento pelo valor Imprimir a lista circular (faça um laço que pare quando voltar ao primeiro elemento)

#include <stdio.h>

#include <stdlib.h>

typedef struct No {

int valor;

struct No\* anterior;

struct No\* proximo;

} No;

typedef struct {

No\* inicio;

} ListaCircular;

// Inicializa a lista circular

void inicializarLista(ListaCircular\* lista) {

lista->inicio = NULL;

}

// Verifica se a lista está vazia

int listaVazia(ListaCircular\* lista) {

return (lista->inicio == NULL);

}

// Cria um novo nó

No\* criarNo(int valor) {

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

if (!novo) {

printf("Erro ao alocar memória!\n");

exit(1);

}

novo->valor = valor;

novo->anterior = NULL;

novo->proximo = NULL;

return novo;

}

// Insere um elemento no fim da lista circular

void inserirNoFim(ListaCircular\* lista, int valor) {

No\* novo = criarNo(valor);

if (listaVazia(lista)) {

lista->inicio = novo;

novo->proximo = novo;

novo->anterior = novo;

} else {

No\* ultimo = lista->inicio->anterior;

novo->proximo = lista->inicio;

novo->anterior = ultimo;

ultimo->proximo = novo;

lista->inicio->anterior = novo;

}

}

// Remove um elemento pelo valor (remove a primeira ocorrência encontrada)

int removerPorValor(ListaCircular\* lista, int valor) {

if (listaVazia(lista)) {

printf("Lista vazia! Nada para remover.\n");

return 0;

}

No\* atual = lista->inicio;

do {

if (atual->valor == valor) {

// Se for o único nó da lista

if (atual->proximo == atual) {

lista->inicio = NULL;

} else {

atual->anterior->proximo = atual->proximo;

atual->proximo->anterior = atual->anterior;

// Se estiver removendo o início, atualiza o início

if (atual == lista->inicio) {

lista->inicio = atual->proximo;

}

}

free(atual);

return 1; // Remoção bem sucedida

}

atual = atual->proximo;

} while (atual != lista->inicio);

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

return 0;

}

// Imprime a lista circular

void imprimirLista(ListaCircular\* lista) {

if (listaVazia(lista)) {

printf("Lista vazia.\n");

return;

}

No\* atual = lista->inicio;

printf("Lista Circular: [");

do {

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

atual = atual->proximo;

if (atual != lista->inicio) {

printf(", ");

}

} while (atual != lista->inicio);

printf("]\n");

}

// Libera a memória alocada para a lista

void liberarLista(ListaCircular\* lista) {

if (listaVazia(lista)) return;

No\* atual = lista->inicio;

No\* temp;

do {

temp = atual;

atual = atual->proximo;

free(temp);

} while (atual != lista->inicio);

lista->inicio = NULL;

}

int main() {

ListaCircular lista;

inicializarLista(&lista);

int opcao, valor;

do {

printf("\nMenu:\n");

printf("1. Inserir no fim\n");

printf("2. Remover por valor\n");

printf("3. Imprimir lista\n");

printf("4. Sair\n");

printf("Escolha uma opção: ");

scanf("%d", &opcao);

switch (opcao) {

case 1:

printf("Digite o valor a ser inserido: ");

scanf("%d", &valor);

inserirNoFim(&lista, valor);

break;

case 2:

printf("Digite o valor a ser removido: ");

scanf("%d", &valor);

removerPorValor(&lista, valor);

break;

case 3:

imprimirLista(&lista);

break;

case 4:

printf("Encerrando o programa...\n");

break;

default:

printf("Opção inválida!\n");

}

} while (opcao != 4);

liberarLista(&lista);

return 0;

}