**BLOCO 1 – VETORES**

**Exercício 1:** Faça um programa que leia um vetor de 10 números inteiros. Em seguida, remova todos os números pares do vetor (desloque os elementos) e exiba o vetor atualizado.

#include <stdio.h>

int main() {

int vetor[10];

int i, j, novo\_tamanho = 0;

// Leitura dos valores

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

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

printf("Número %d: ", i+1);

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

}

// Remoção dos números pares

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

if(vetor[i] % 2 != 0) { // Se for ímpar

vetor[novo\_tamanho] = vetor[i];

novo\_tamanho++;

}

}

// Exibição do vetor atualizado

printf("\nVetor sem números pares:\n");

for(i = 0; i < novo\_tamanho; i++) {

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

}

printf("\n");

return 0;

}

**BLOCO 2 – STRUCTS**

**Exercício 2:** Defina uma struct Aluno com nome, nota1, nota2. Leia os dados de 5 alunos e imprima os nomes dos alunos com média >= 7.0.

#include <stdio.h>

#include <string.h>

// Definição da struct Aluno

struct Aluno {

char nome[50];

float nota1;

float nota2;

};

int main() {

struct Aluno alunos[5];

int i;

// Leitura dos dados dos alunos

printf("Cadastro de Alunos\n");

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

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

printf("Nome: ");

scanf(" %[^\n]", alunos[i].nome); // Lê até encontrar nova linha

printf("Nota 1: ");

scanf("%f", &alunos[i].nota1);

printf("Nota 2: ");

scanf("%f", &alunos[i].nota2);

}

// Exibição dos alunos com média >= 7.0

printf("\nAlunos com média >= 7.0:\n");

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

float media = (alunos[i].nota1 + alunos[i].nota2) / 2;

if(media >= 7.0) {

printf("- %s (Média: %.2f)\n", alunos[i].nome, media);

}

}

return 0;

}

**BLOCO 3 – LISTAS ENCADEADAS**

**Exercício 3:** Implemente uma lista encadeada simples que permita: Inserir ordenadamente (valores em ordem crescente) Imprimir a lista

#include <stdio.h>

#include <stdlib.h>

// Estrutura do nó da lista encadeada

typedef struct No {

int valor;

struct No\* proximo;

} No;

// Estrutura da lista encadeada ordenada

typedef struct {

No\* cabeca;

} ListaEncadeadaOrdenada;

// Função para criar um novo nó

No\* criar\_no(int valor) {

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

if (novo == NULL) {

printf("Erro ao alocar memória!\n");

exit(1);

}

novo->valor = valor;

novo->proximo = NULL;

return novo;

}

// Função para inicializar a lista

void inicializar\_lista(ListaEncadeadaOrdenada\* lista) {

lista->cabeca = NULL;

}

// Função para inserir ordenadamente na lista

void inserir\_ordenado(ListaEncadeadaOrdenada\* lista, int valor) {

No\* novo = criar\_no(valor);

// Caso 1: Lista vazia ou novo valor menor que o primeiro

if (lista->cabeca == NULL || valor < lista->cabeca->valor) {

novo->proximo = lista->cabeca;

lista->cabeca = novo;

} else {

// Caso 2: Encontrar a posição correta para inserção

No\* atual = lista->cabeca;

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

atual = atual->proximo;

}

novo->proximo = atual->proximo;

atual->proximo = novo;

}

}

// Função para imprimir a lista

void imprimir\_lista(ListaEncadeadaOrdenada\* lista) {

No\* atual = lista->cabeca;

while (atual != NULL) {

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

atual = atual->proximo;

}

printf("NULL\n");

}

// Função para liberar a memória da lista

void liberar\_lista(ListaEncadeadaOrdenada\* lista) {

No\* atual = lista->cabeca;

while (atual != NULL) {

No\* temp = atual;

atual = atual->proximo;

free(temp);

}

lista->cabeca = NULL;

}

int main() {

ListaEncadeadaOrdenada lista;

inicializar\_lista(&lista);

inserir\_ordenado(&lista, 5);

inserir\_ordenado(&lista, 2);

inserir\_ordenado(&lista, 7);

inserir\_ordenado(&lista, 3);

inserir\_ordenado(&lista, 1);

printf("Lista ordenada: ");

imprimir\_lista(&lista);

liberar\_lista(&lista);

return 0;

}

**BLOCO 4 – PILHAS**

**Exercício 4:** Usando pilha (implementada com lista encadeada), leia uma palavra e verifique se ela é um palíndromo (ex: "arara").

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

// Estrutura do nó da pilha

typedef struct Node {

char data;

struct Node\* next;

} Node;

// Estrutura da pilha

typedef struct {

Node\* top;

} Stack;

// Inicializa a pilha

void initStack(Stack\* s) {

s->top = NULL;

}

// Empilha um caractere

void push(Stack\* s, char c) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

if (!newNode) {

printf("Erro de alocação de memória\n");

exit(1);

}

newNode->data = tolower(c);

newNode->next = s->top;

s->top = newNode;

}

// Desempilha um caractere

char pop(Stack\* s) {

if (s->top == NULL) return '\0';

Node\* temp = s->top;

char c = temp->data;

s->top = temp->next;

free(temp);

return c;

}

// Verifica se é palíndromo

int isPalindrome(const char\* str) {

Stack s;

int i; // Declaração movida para fora do for para compatibilidade com C89

initStack(&s);

/\* Empilha todos os caracteres alfabéticos \*/

for (i = 0; str[i] != '\0'; i++) {

if (isalpha(str[i])) {

push(&s, str[i]);

}

}

/\* Verifica desempilhando \*/

for (i = 0; str[i] != '\0'; i++) {

if (isalpha(str[i])) {

if (tolower(str[i]) != pop(&s)) {

return 0; // Não é palíndromo

}

}

}

return 1; // É palíndromo

}

int main() {

char input[100];

int i; // Declaração movida para fora do for

printf("Digite uma palavra ou frase: ");

fgets(input, sizeof(input), stdin);

/\* Remove a quebra de linha \*/

for (i = 0; input[i] != '\0'; i++) {

if (input[i] == '\n') {

input[i] = '\0';

break;

}

}

if (isPalindrome(input)) {

printf("\"%s\" é um palíndromo!\n", input);

} else {

printf("\"%s\" não é um palíndromo.\n", input);

}

return 0;

}

**BLOCO 5 – FILAS**

**Exercício 5:** Usando uma fila (implementada com lista encadeada), simule o atendimento de clientes. Cada cliente possui um número. Insira 5 clientes na fila e atenda-os na ordem.

#include <stdio.h>

#include <stdlib.h>

// Estrutura para representar um cliente

typedef struct Cliente {

int numero;

struct Cliente\* proximo;

} Cliente;

// Estrutura para representar a fila

typedef struct {

Cliente\* inicio;

Cliente\* fim;

} Fila;

// Inicializa a fila

void inicializarFila(Fila\* f) {

f->inicio = NULL;

f->fim = NULL;

}

// Verifica se a fila está vazia

int filaVazia(Fila\* f) {

return (f->inicio == NULL);

}

// Insere um cliente no final da fila (enqueue)

void inserirCliente(Fila\* f, int num) {

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

if (!novo) {

printf("Erro ao alocar memória!\n");

exit(1);

}

novo->numero = num;

novo->proximo = NULL;

if (filaVazia(f)) {

f->inicio = novo;

f->fim = novo;

} else {

f->fim->proximo = novo;

f->fim = novo;

}

printf("Cliente %d chegou e entrou na fila.\n", num);

}

// Remove e retorna o cliente do início da fila (dequeue)

int atenderCliente(Fila\* f) {

if (filaVazia(f)) {

printf("Fila vazia! Nenhum cliente para atender.\n");

return -1;

}

Cliente\* temp = f->inicio;

int num = temp->numero;

f->inicio = f->inicio->proximo;

if (f->inicio == NULL) {

f->fim = NULL;

}

free(temp);

return num;

}

// Exibe o estado atual da fila

void mostrarFila(Fila\* f) {

if (filaVazia(f)) {

printf("Fila vazia.\n");

return;

}

Cliente\* atual = f->inicio;

printf("Fila atual: ");

while (atual != NULL) {

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

atual = atual->proximo;

}

printf("\n");

}

int main() {

Fila fila;

inicializarFila(&fila);

// Inserindo 5 clientes na fila

inserirCliente(&fila, 1);

inserirCliente(&fila, 2);

inserirCliente(&fila, 3);

inserirCliente(&fila, 4);

inserirCliente(&fila, 5);

mostrarFila(&fila);

// Atendendo os clientes na ordem

printf("\nIniciando atendimento...\n");

while (!filaVazia(&fila)) {

int clienteAtendido = atenderCliente(&fila);

printf("Atendendo cliente %d\n", clienteAtendido);

mostrarFila(&fila);

}

return 0;

}