**31. Ordenação por Bubble Sort**#include <stdio.h>

void bubbleSort(int arr[], int n) {

int i, j, temp;

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

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

if (arr[j] > arr[j+1]) {

temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

}

void printArray(int arr[], int size) {

for (int i = 0; i < size; i++)

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

printf("\n");

}

int main() {

int arr[] = {64, 34, 25, 12, 22, 11, 90};

int n = sizeof(arr) / sizeof(arr[0]);

bubbleSort(arr, n);

printf("Array ordenado: \n");

printArray(arr, n);

return 0;

}

### 32. Ordenação por Selection Sort #include <stdio.h>

### void selectionSort(int arr[], int n) {

### int i, j, min\_idx, temp;

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

### min\_idx = i;

### for (j = i+1; j < n; j++)

### if (arr[j] < arr[min\_idx])

### min\_idx = j;

### temp = arr[min\_idx];

### arr[min\_idx] = arr[i];

### arr[i] = temp;

### }

### }

### void printArray(int arr[], int size) {

### for (int i = 0; i < size; i++)

### printf("%d ", arr[i]);

### printf("\n");

### }

### int main() {

### int arr[] = {64, 34, 25, 12, 22, 11, 90};

### int n = sizeof(arr) / sizeof(arr[0]);

### selectionSort(arr, n);

### printf("Array ordenado: \n");

### printArray(arr, n);

### return 0;

### }

### 34. Merge Sort #include <stdio.h>

### #include <stdlib.h>

### void merge(int arr[], int l, int m, int r) {

### int i, j, k;

### int n1 = m - l + 1;

### int n2 = r - m;

### int \*L = (int\*)malloc(n1 \* sizeof(int));

### int \*R = (int\*)malloc(n2 \* sizeof(int));

### for (i = 0; i < n1; i++)

### L[i] = arr[l + i];

### for (j = 0; j < n2; j++)

### R[j] = arr[m + 1 + j];

### i = 0;

### j = 0;

### k = l;

### while (i < n1 && j < n2) {

### if (L[i] <= R[j]) {

### arr[k] = L[i];

### i++;

### } else {

### arr[k] = R[j];

### j++;

### }

### k++;

### }

### while (i < n1) {

### arr[k] = L[i];

### i++;

### k++;

### }

### while (j < n2) {

### arr[k] = R[j];

### j++;

### k++;

### }

### free(L);

### free(R);

### }

### void mergeSort(int arr[], int l, int r) {

### if (l < r) {

### int m = l + (r - l) / 2;

### mergeSort(arr, l, m);

### mergeSort(arr, m + 1, r);

### merge(arr, l, m, r);

### }

### }

### void printArray(int arr[], int size) {

### for (int i = 0; i < size; i++)

### printf("%d ", arr[i]);

### printf("\n");

### }

### int main() {

### int arr[] = {64, 34, 25, 12, 22, 11, 90};

### int n = sizeof(arr) / sizeof(arr[0]);

### mergeSort(arr, 0, n - 1);

### printf("Array ordenado: \n");

### printArray(arr, n);

### return 0;

### }

### 35. Quick Sort #include <stdio.h>

### int partition(int arr[], int low, int high) {

### int pivot = arr[high];

### int i = (low - 1);

### for (int j = low; j <= high - 1; j++) {

### if (arr[j] < pivot) {

### i++;

### int temp = arr[i];

### arr[i] = arr[j];

### arr[j] = temp;

### }

### }

### int temp = arr[i + 1];

### arr[i + 1] = arr[high];

### arr[high] = temp;

### return (i + 1);

### }

### void quickSort(int arr[], int low, int high) {

### if (low < high) {

### int pi = partition(arr, low, high);

### quickSort(arr, low, pi - 1);

### quickSort(arr, pi + 1, high);

### }

### }

### void printArray(int arr[], int size) {

### for (int i = 0; i < size; i++)

### printf("%d ", arr[i]);

### printf("\n");

### }

### int main() {

### int arr[] = {64, 34, 25, 12, 22, 11, 90};

### int n = sizeof(arr) / sizeof(arr[0]);

### quickSort(arr, 0, n - 1);

### printf("Array ordenado: \n");

### printArray(arr, n);

### return 0;

### }

### 36. Implementar uma Pilha #include <stdio.h>

### #include <stdlib.h>

### #define MAX 100

### typedef struct {

### int top;

### int arr[MAX];

### } Stack;

### void initStack(Stack \*s) {

### s->top = -1;

### }

### int isFull(Stack \*s) {

### return s->top == MAX - 1;

### }

### int isEmpty(Stack \*s) {

### return s->top == -1;

### }

### void push(Stack \*s, int value) {

### if (isFull(s)) {

### printf("Stack overflow\n");

### return;

### }

### s->arr[++(s->top)] = value;

### }

### int pop(Stack \*s) {

### if (isEmpty(s)) {

### printf("Stack underflow\n");

### return -1;

### }

### return s->arr[(s->top)--];

### }

### int peek(Stack \*s) {

### if (isEmpty(s)) {

### printf("Stack is empty\n");

### return -1;

### }

### return s->arr[s->top];

### }

### int main() {

### Stack s;

### initStack(&s);

### push(&s, 10);

### push(&s, 20);

### printf("Top element is %d\n", peek(&s));

### printf("Popped element is %d\n", pop(&s));

### return 0;

### }

### 37. Implementar uma Fila #include <stdio.h>

### #include <stdlib.h>

### #define MAX 100

### typedef struct {

### int front, rear, size;

### int arr[MAX];

### } Queue;

### void initQueue(Queue \*q) {

### q->front = 0;

### q->rear = -1;

### q->size = 0;

### }

### int isFull(Queue \*q) {

### return q->size == MAX;

### }

### int isEmpty(Queue \*q) {

### return q->size == 0;

### }

### void enqueue(Queue \*q, int value) {

### if (isFull(q)) {

### printf("Queue overflow\n");

### return;

### }

### q->rear = (q->rear + 1) % MAX;

### q->arr[q->rear] = value;

### q->size++;

### }

### int dequeue(Queue \*q) {

### if (isEmpty(q)) {

### printf("Queue underflow\n");

### return -1;

### }

### int value = q->arr[q->front];

### q->front = (q->front + 1) % MAX;

### q->size--;

### return value;

### }

### int main() {

### Queue q;

### initQueue(&q);

### enqueue(&q, 10);

### enqueue(&q, 20);

### printf("Dequeued element is %d\n", dequeue(&q));

### return 0;

### }

### 38. Conversão de Infixa para Pós-fixa #include <stdio.h>

### #include <stdlib.h>

### #include <string.h>

### #define MAX 100

### typedef struct {

### int top;

### char arr[MAX];

### } Stack;

### void initStack(Stack \*s) {

### s->top = -1;

### }

### int isFull(Stack \*s) {

### return s->top == MAX - 1;

### }

### int isEmpty(Stack \*s) {

### return s->top == -1;

### }

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

### if (isFull(s)) {

### printf("Stack overflow\n");

### return;

### }

### s->arr[++(s->top)] = c;

### }

### char pop(Stack \*s) {

### if (isEmpty(s)) {

### printf("Stack underflow\n");

### return '\0';

### }

### return s->arr[(s->top)--];

### }

### char peek(Stack \*s) {

### if (isEmpty(s)) {

### printf("Stack is empty\n");

### return '\0';

### }

### return s->arr[s->top];

### }

### int precedence(char op) {

### switch (op) {

### case '+':

### case '-': return 1;

### case '\*':

### case '/': return 2;

### case '^': return 3;

### default: return 0;

### }

### }

### void infixToPostfix(char \*exp, char \*result) {

### Stack s;

### initStack(&s);

### int k = 0;

### for (int i = 0; exp[i]; i++) {

### char c = exp[i];

### if (c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z') {

### result[k++] = c;

### } else if (c == '(') {

### push(&s, c);

### } else if (c == ')') {

### while (!isEmpty(&s) && peek(&s) != '(') {

### result[k++] = pop(&s);

### }

### pop(&s); // Remove '('

### } else {

### while (!isEmpty(&s) && precedence(peek(&s)) >= precedence(c)) {

### result[k++] = pop(&s);

### }

### push(&s, c);

### }

### }

### while (!isEmpty(&s)) {

### result[k++] = pop(&s);

### }

### result[k] = '\0';

### }

### int main() {

### char infix[MAX] = "a+b\*(c^d-e)^(f+g\*h)-i";

### char postfix[MAX];

### infixToPostfix(infix, postfix);

### printf("Expressão pós-fixa: %s\n", postfix);

### return 0;

### }

### 39. Avaliação de Expressão Pós-fixa #include <stdio.h>

### #include <stdlib.h>

### #include <ctype.h>

### #define MAX 100

### typedef struct {

### int top;

### int arr[MAX];

### } Stack;

### void initStack(Stack \*s) {

### s->top = -1;

### }

### int isFull(Stack \*s) {

### return s->top == MAX - 1;

### }

### int isEmpty(Stack \*s) {

### return s->top == -1;

### }

### void push(Stack \*s, int value) {

### if (isFull(s)) {

### printf("Stack overflow\n");

### return;

### }

### s->arr[++(s->top)] = value;

### }

### int pop(Stack \*s) {

### if (isEmpty(s)) {

### printf("Stack underflow\n");

### return -1;

### }

### return s->arr[(s->top)--];

### }

### int evaluatePostfix(char \*exp) {

### Stack s;

### initStack(&s);

### for (int i = 0; exp[i]; i++) {

### char c = exp[i];

### if (isdigit(c)) {

### push(&s, c - '0');

### } else {

### int val2 = pop(&s);

### int val1 = pop(&s);

### switch (c) {

### case '+': push(&s, val1 + val2); break;

### case '-': push(&s, val1 - val2); break;

### case '\*': push(&s, val1 \* val2); break;

### case '/': push(&s, val1 / val2); break;

### }

### }

### }

### return pop(&s);

### }

### int main() {

### char postfix[MAX] = "231\*+9-";

### printf("Valor da expressão pós-fixa: %d\n", evaluatePostfix(postfix));

### return 0;

### }

### 40. Jogo da Velha #include <stdio.h>

### #define SIZE 3

### void printBoard(char board[SIZE][SIZE]) {

### for (int i = 0; i < SIZE; i++) {

### for (int j = 0; j < SIZE; j++) {

### printf("%c ", board[i][j]);

### }

### printf("\n");

### }

### }

### int checkWin(char board[SIZE][SIZE], char player) {

### for (int i = 0; i < SIZE; i++) {

### if (board[i][0] == player && board[i][1] == player && board[i][2] == player)

### return 1;

### if (board[0][i] == player && board[1][i] == player && board[2][i] == player)

### return 1;

### }

### if (board[0][0] == player && board[1][1] == player && board[2][2] == player)

### return 1;

### if (board[0][2] == player && board[1][1] == player && board[2][0] == player)

### return 1;

### return 0;

### }

### int isFull(char board[SIZE][SIZE]) {

### for (int i = 0; i < SIZE; i++) {

### for (int j = 0; j < SIZE; j++) {

### if (board[i][j] == ' ')

### return 0;

### }

### }

### return 1;

### }

### int main() {

### char board[SIZE][SIZE] = {

### {' ', ' ', ' '},

### {' ', ' ', ' '},

### {' ', ' ', ' '}

### };

### int row, col;

### char player = 'X';

### while (1) {

### printBoard(board);

### printf("Jogador %c, digite a linha e a coluna (0-2): ", player);

### scanf("%d %d", &row, &col);

### if (row < 0 || row >= SIZE || col < 0 || col >= SIZE || board[row][col] != ' ') {

### printf("Jogada inválida! Tente novamente.\n");

### continue;

### }

### board[row][col] = player;

### if (checkWin(board, player)) {

### printBoard(board);

### printf("Jogador %c venceu!\n", player);

### break;

### }

### if (isFull(board)) {

### printBoard(board);

### printf("Empate!\n");

### break;

### }

### player = (player == 'X') ? 'O' : 'X';

### }

### return 0;

### }

### 41. Cifra de César #include <stdio.h>

### #include <string.h>

### #define MAX 100

### void cipherCaesar(char \*text, int shift) {

### for (int i = 0; text[i]; i++) {

### if (text[i] >= 'a' && text[i] <= 'z') {

### text[i] = ((text[i] - 'a' + shift) % 26 + 26) % 26 + 'a';

### } else if (text[i] >= 'A' && text[i] <= 'Z') {

### text[i] = ((text[i] - 'A' + shift) % 26 + 26) % 26 + 'A';

### }

### }

### }

### int main() {

### char text[MAX] = "HelloWorld";

### int shift = 3;

### printf("Texto original: %s\n", text);

### cipherCaesar(text, shift);

### printf("Texto cifrado: %s\n", text);

### return 0;

### }

### 42. Gerador de Números Aleatórios #include <stdio.h>

### #include <stdlib.h>

### #include <time.h>

### int main() {

### srand(time(0));

### int num = rand() % 100 + 1;

### printf("Número aleatório entre 1 e 100: %d\n", num);

### return 0;

### }

### 43. Maior Subsequência Crescente #include <stdio.h>

### #define MAX 100

### int longestIncreasingSubsequence(int arr[], int n) {

### int lis[MAX];

### int max\_lis = 1;

### for (int i = 0; i < n; i++) {

### lis[i] = 1;

### for (int j = 0; j < i; j++) {

### if (arr[i] > arr[j] && lis[i] < lis[j] + 1) {

### lis[i] = lis[j] + 1;

### }

### }

### if (max\_lis < lis[i]) {

### max\_lis = lis[i];

### }

### }

### return max\_lis;

### }

### int main() {

### int arr[] = {10, 22, 9, 33, 21, 50, 41, 60, 80};

### int n = sizeof(arr) / sizeof(arr[0]);

### printf("Comprimento da maior subsequência crescente: %d\n", longestIncreasingSubsequence(arr, n));

### return 0;

### }

### 44. Caminho de um Cavalo no Xadrez #include <stdio.h>

### #define N 8

### int isSafe(int x, int y, int board[N][N]) {

### return (x >= 0 && x < N && y >= 0 && y < N && board[x][y] == -1);

### }

### void printBoard(int board[N][N]) {

### for (int i = 0; i < N; i++) {

### for (int j = 0; j < N; j++)

### printf("%2d ", board[i][j]);

### printf("\n");

### }

### }

### int solveKTUtil(int x, int y, int movei, int board[N][N], int xMove[], int yMove[]) {

### if (movei == N \* N)

### return 1;

### for (int k = 0; k < 8; k++) {

### int newX = x + xMove[k];

### int newY = y + yMove[k];

### if (isSafe(newX, newY, board)) {

### board[newX][newY] = movei;

### if (solveKTUtil(newX, newY, movei + 1, board, xMove, yMove))

### return 1;

### board[newX][newY] = -1;

### }

### }

### return 0;

### }

### void solveKT() {

### int board[N][N];

### for (int i = 0; i < N; i++)

### for (int j = 0; j < N; j++)

### board[i][j] = -1;

### int xMove[] = {2, 1, -1, -2, -2, -1, 1, 2};

### int yMove[] = {1, 2, 2, 1, -1, -2, -2, -1};

### board[0][0] = 0;

### if (solveKTUtil(0, 0, 1, board, xMove, yMove) == 0) {

### printf("Solução não encontrada");

### } else {

### printBoard(board);

### }

### }

### int main() {

### solveKT();

### return 0;

### }

### 45. Solucionador de Sudoku #include <stdio.h>

### #define N 9

### int isSafe(int board[N][N], int row, int col, int num) {

### for (int x = 0; x < N; x++) {

### if (board[row][x] == num || board[x][col] == num || board[row - row % 3 + x / 3][col - col % 3 + x % 3] == num) {

### return 0;

### }

### }

### return 1;

### }

### int solveSudoku(int board[N][N], int row, int col) {

### if (row == N - 1 && col == N) {

### return 1;

### }

### if (col == N) {

### row++;

### col = 0;

### }

### if (board[row][col] != 0) {

### return solveSudoku(board, row, col + 1);

### }

### for (int num = 1; num <= N; num++) {

### if (isSafe(board, row, col, num)) {

### board[row][col] = num;

### if (solveSudoku(board, row, col + 1))

### return 1;

### board[row][col] = 0;

### }

### }

### return 0;

### }

### void printBoard(int board[N][N]) {

### for (int i = 0; i < N; i++) {

### for (int j = 0; j < N; j++)

### printf("%d ", board[i][j]);

### printf("\n");

### }

### }

### int main() {

### int board[N][N] = {

### {5, 3, 0, 0, 7, 0, 0, 0, 0},

### {6, 0, 0, 1, 9, 5, 0, 0, 0},

### {0, 9, 8, 0, 0, 0, 0, 6, 0},

### {8, 0, 0, 0, 6, 0, 0, 0, 3},

### {4, 0, 0, 8, 0, 3, 0, 0, 1},

### {7, 0, 0, 0, 2, 0, 0, 0, 6},

### {0, 6, 0, 0, 0, 0, 2, 8, 0},

### {0, 0, 0, 4, 1, 9, 0, 0, 5},

### {0, 0, 0, 0, 8, 0, 0, 7, 9}

### };

### if (solveSudoku(board, 0, 0))

### printBoard(board);

### else

### printf("No solution exists\n");

### return 0;

### }

### 46. Algoritmo de Dijkstra #include <stdio.h>

### #include <limits.h>

### #define V 9

### int minDistance(int dist[], int sptSet[]) {

### int min = INT\_MAX, min\_index;

### for (int v = 0; v < V; v++) {

### if (sptSet[v] == 0 && dist[v] <= min) {

### min = dist[v];

### min\_index = v;

### }

### }

### return min\_index;

### }

### void printSolution(int dist[], int n) {

### printf("Vértice \t Distância da origem\n");

### for (int i = 0; i < n; i++)

### printf("%d \t\t %d\n", i, dist[i]);

### }

### void dijkstra(int graph[V][V], int src) {

### int dist[V];

### int sptSet[V];

### for (int i = 0; i < V; i++) {

### dist[i] = INT\_MAX;

### sptSet[i] = 0;

### }

### dist[src] = 0;

### for (int count = 0; count < V - 1; count++) {

### int u = minDistance(dist, sptSet);

### sptSet[u] = 1;

### for (int v = 0; v < V; v++) {

### if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v]) {

### dist[v] = dist[u] + graph[u][v];

### }

### }

### }

### printSolution(dist, V);

### }

### int main() {

### int graph[V][V] = {

### {0, 4, 0, 0, 0, 0, 0, 8, 0},

### {4, 0, 8, 0, 0, 0, 0, 11, 0},

### {0, 8, 0, 7, 0, 4, 0, 0, 2},

### {0, 0, 7, 0, 9, 14, 0, 0, 0},

### {0, 0, 0, 9, 0, 10, 0, 0, 0},

### {0, 0, 4, 14, 10, 0, 2, 0, 0},

### {0, 0, 0, 0, 0, 2, 0, 1, 6},

### {8, 11, 0, 0, 0, 0, 1, 0, 7},

### {0, 0, 2, 0, 0, 0, 6, 7, 0}

### };

### dijkstra(graph, 0);

### return 0;

### }

### 47. Algoritmo de Kruskal #include <stdio.h>

### #include <stdlib.h>

### #define V 4

### typedef struct {

### int u, v, w;

### } Edge;

### int compare(const void \*a, const void \*b) {

### return ((Edge\*)a)->w - ((Edge\*)b)->w;

### }

### int find(int parent[], int i) {

### if (parent[i] == -1)

### return i;

### return find(parent, parent[i]);

### }

### void unionSets(int parent[], int x, int y) {

### int xset = find(parent, x);

### int yset = find(parent, y);

### if (xset != yset)

### parent[xset] = yset;

### }

### void kruskal(Edge edges[], int n) {

### int parent[V];

### for (int i = 0; i < V; i++)

### parent[i] = -1;

### qsort(edges, n, sizeof(Edge), compare);

### int mst\_wt = 0;

### for (int i = 0; i < n; i++) {

### int u = edges[i].u;

### int v = edges[i].v;

### int w = edges[i].w;

### int x = find(parent, u);

### int y = find(parent, v);

### if (x != y) {

### printf("%d - %d: %d\n", u, v, w);

### mst\_wt += w;

### unionSets(parent, x, y);

### }

### }

### printf("Peso total da Árvore Geradora Mínima: %d\n", mst\_wt);

### }

### int main() {

### Edge edges[] = {

### {0, 1, 10},

### {0, 2, 6},

### {0, 3, 5},

### {1, 3, 15},

### {2, 3, 4}

### };

### int n = sizeof(edges) / sizeof(edges[0]);

### kruskal(edges, n);

### return 0;

### }

### 48. Enigma de Ritter #include <stdio.h>

### #include <string.h>

### #define MAX 100

### void ritterEncrypt(char \*input, char \*output, int key) {

### int len = strlen(input);

### for (int i = 0; i < len; i++) {

### output[i] = input[i] ^ key;

### }

### output[len] = '\0';

### }

### int main() {

### char input[MAX] = "HelloWorld";

### char encrypted[MAX];

### char decrypted[MAX];

### int key = 23;

### ritterEncrypt(input, encrypted, key);

### printf("Texto criptografado: %s\n", encrypted);

### ritterEncrypt(encrypted, decrypted, key);

### printf("Texto descriptografado: %s\n", decrypted);

### return 0;

### }

### 49. Programa de Gerenciamento de Biblioteca #include <stdio.h>

### #include <stdlib.h>

### #include <string.h>

### #define MAX\_BOOKS 100

### #define TITLE\_LEN 50

### typedef struct {

### char title[TITLE\_LEN];

### int id;

### } Book;

### void addBook(Book books[], int \*count, char \*title, int id) {

### if (\*count >= MAX\_BOOKS) {

### printf("Biblioteca cheia!\n");

### return;

### }

### strcpy(books[\*count].title, title);

### books[\*count].id = id;

### (\*count)++;

### }

### void listBooks(Book books[], int count) {

### for (int i = 0; i < count; i++) {

### printf("ID: %d, Título: %s\n", books[i].id, books[i].title);

### }

### }

### int main() {

### Book books[MAX\_BOOKS];

### int count = 0;

### int option;

### char title[TITLE\_LEN];

### int id;

### while (1) {

### printf("1. Adicionar livro\n2. Listar livros\n3. Sair\n");

### printf("Escolha uma opção: ");

### scanf("%d", &option);

### getchar(); // Remove newline character

### switch (option) {

### case 1:

### printf("Digite o título do livro: ");

### fgets(title, TITLE\_LEN, stdin);

### title[strcspn(title, "\n")] = '\0'; // Remove newline character

### printf("Digite o ID do livro: ");

### scanf("%d", &id);

### addBook(books, &count, title, id);

### break;

### case 2:

### listBooks(books, count);

### break;

### case 3:

### exit(0);

### }

### }

### return 0;

### }

### 50. Conversor de Temperatura #include <stdio.h>

### #define C\_TO\_F(c) ((c) \* 9 / 5 + 32)

### #define F\_TO\_C(f) (((f) - 32) \* 5 / 9)

### int main() {

### float celsius = 25.0;

### float fahrenheit = 77.0;

### printf("%.2f Celsius é igual a %.2f Fahrenheit\n", celsius, C\_TO\_F(celsius));

### printf("%.2f Fahrenheit é igual a %.2f Celsius\n", fahrenheit, F\_TO\_C(fahrenheit));

### return 0;

### }