11) c program to evaluate an arithmetic expression

#include <stdio.h>

int main() {

double num1 = 10.0;

double num2 = 5.0;

char operator = '/'; // Change this to any operator you want to test

double result;

printf("Input values:\n");

printf("First number: %.2lf\n", num1);

printf("Operator: %c\n", operator);

printf("Second number: %.2lf\n", num2);

switch(operator) {

case '+':

result = num1 + num2;

printf("%.2lf + %.2lf = %.2lf\n", num1, num2, result);

break;

case '-':

result = num1 - num2;

printf("%.2lf - %.2lf = %.2lf\n", num1, num2, result);

break;

case '\*':

result = num1 \* num2;

printf("%.2lf \* %.2lf = %.2lf\n", num1, num2, result);

break;

case '/':

if (num2 != 0) {

result = num1 / num2;

printf("%.2lf / %.2lf = %.2lf\n", num1, num2, result);

} else {

printf("Error: Division by zero is not allowed.\n");

}

break;

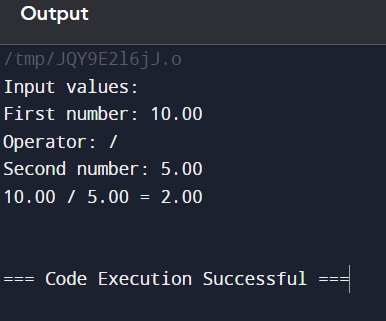
default:

printf("Error: Invalid operator.\n");

}

return 0;

}



12) c program to balance symbols in given expression

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h> // For strlen

#define MAX 100

typedef struct {

char items[MAX];

int top;

} Stack;

void initStack(Stack \*s) {

s->top = -1;

}

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

if (s->top >= MAX - 1) return false;

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

return true;

}

char pop(Stack \*s) {

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

return s->items[(s->top)--];

}

bool isEmpty(Stack \*s) {

return s->top < 0;

}

bool areBalanced(const char \*expression) {

Stack stack;

initStack(&stack);

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

char ch = expression[i];

if (ch == '(' || ch == '{' || ch == '[') {

if (!push(&stack, ch)) return false;

} else if (ch == ')' || ch == '}' || ch == ']') {

char top = pop(&stack);

if ((ch == ')' && top != '(') ||

(ch == '}' && top != '{') ||

(ch == ']' && top != '[')) {

return false;

}

}

}

return isEmpty(&stack);

}

int main() {

const char \*expression = "{[()]}"; // Change this to test different

printf("Expression: %s\n", expression);

if (areBalanced(expression)) {

printf("The expression is balanced.\n");

} else {

printf("The expression is not balanced.\n");

}

return 0;

}

A screenshot of a computer program

Description automatically generated

13) recursive function in c to implement tower of Hanoi problem

#include <stdio.h>

void towerOfHanoi(int n, char fromPeg, char toPeg, char auxPeg) {

if (n == 1) {

printf("Move disk 1 from peg %c to peg %c\n", fromPeg, toPeg);

return;

}

towerOfHanoi(n - 1, fromPeg, auxPeg, toPeg);

printf("Move disk %d from peg %c to peg %c\n", n, fromPeg, toPeg);

towerOfHanoi(n - 1, auxPeg, toPeg, fromPeg);

}

int main() {

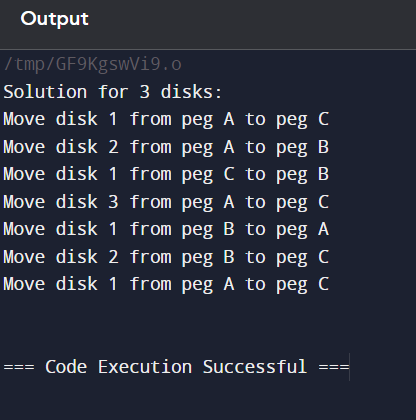
int n = 3; // Number of disks

printf("Solution for %d disks:\n", n);

towerOfHanoi(n, 'A', 'C', 'B');

return 0;

}



14) recursive function in c to factorial of a given number

#include <stdio.h>

int factorial(int n) {

if (n == 0 || n == 1) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

int main() {

int number = 5; // Hard-coded number to find factorial

printf("Factorial of %d is %d\n", number, factorial(number));

return 0;

}

A screenshot of a computer

Description automatically generated

15) implement a queue using an array

#include <stdio.h>

#include <stdlib.h>

#define MAX 100 // Maximum number of elements in the queue

typedef struct {

int items[MAX];

int front, rear, size;

} Queue;

void initQueue(Queue \*q) {

q->front = 0;

q->rear = -1;

q->size = 0;

}

int isEmpty(Queue \*q) {

return q->size == 0;

}

int isFull(Queue \*q) {

return q->size == MAX;

}

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

if (isFull(q)) {

printf("Queue is full. Cannot enqueue %d\n", value);

return;

}

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

q->items[q->rear] = value;

q->size++;

printf("Enqueued %d\n", value);

}

int dequeue(Queue \*q) {

if (isEmpty(q)) {

printf("Queue is empty. Cannot dequeue\n");

return -1; // Indicate an error

}

int value = q->items[q->front];

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

q->size--;

return value;

}

int searchQueue(Queue \*q, int value) {

if (isEmpty(q)) return -1; // Queue is empty

for (int i = 0; i < q->size; i++) {

if (q->items[(q->front + i) % MAX] == value) {

return i; // Return position if found

}

}

return -1; // Return -1 if not found

}

void printQueue(Queue \*q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return;

}

printf("Queue elements: ");

for (int i = 0; i < q->size; i++) {

printf("%d ", q->items[(q->front + i) % MAX]);

}

printf("\n");

}

int main() {

Queue q;

initQueue(&q);

enqueue(&q, 10);

enqueue(&q, 20);

enqueue(&q, 30);

printQueue(&q);

int value = 20;

int pos = searchQueue(&q, value);

if (pos != -1) {

printf("Element %d found at position %d\n", value, pos);

} else {

printf("Element %d not found\n", value);

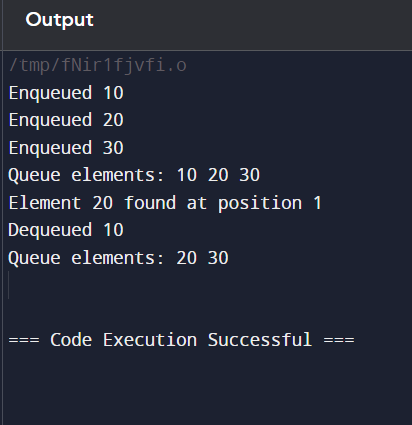
}

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

printQueue(&q);

return 0;

}



16) implement a queue using a linked list

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node \*next;

} Node;

typedef struct {

Node \*front, \*rear;

} Queue;

void initQueue(Queue \*q) {

q->front = q->rear = NULL;

}

int isEmpty(Queue \*q) {

return q->front == NULL;

}

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

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

newNode->data = value;

newNode->next = NULL;

if (isEmpty(q)) {

q->front = q->rear = newNode;

} else {

q->rear->next = newNode;

q->rear = newNode;

}

printf("Enqueued %d\n", value);

}

int dequeue(Queue \*q) {

if (isEmpty(q)) {

printf("Queue is empty. Cannot dequeue\n");

return -1; // Indicate an error

}

Node \*temp = q->front;

int value = temp->data;

q->front = q->front->next;

if (q->front == NULL) {

q->rear = NULL;

}

free(temp);

return value;

}

int searchQueue(Queue \*q, int value) {

Node \*current = q->front;

int position = 0;

while (current != NULL) {

if (current->data == value) {

return position; // Return position if found

}

current = current->next;

position++;

}

return -1; // Return -1 if not found

}

void printQueue(Queue \*q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return;

}

Node \*current = q->front;

printf("Queue elements: ");

while (current != NULL) {

printf("%d ", current->data);

current = current->next;

}

printf("\n");

}

int main() {

Queue q;

initQueue(&q);

enqueue(&q, 10);

enqueue(&q, 20);

enqueue(&q, 30);

printQueue(&q);

int value = 20;

int pos = searchQueue(&q, value);

if (pos != -1) {

printf("Element %d found at position %d\n", value, pos);

} else {

printf("Element %d not found\n", value);

}

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

printQueue(&q);

return 0;

}

