11) Write a C program to evaluate an arithmetic operation

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX 100

typedef struct {

int top;

double items[MAX];

} Stack;

void initializeStack(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, double value) {

if (isFull(s)) {

printf("Stack is full!\n");

} else {

s->items[++s->top] = value;

}

}

double pop(Stack \*s) {

if (isEmpty(s)) {

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

return -1;

} else {

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

}

}

double peek(Stack \*s) {

if (isEmpty(s)) {

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

return -1;

} else {

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

}

}

double evaluateExpression(char \*expression) {

Stack stack;

initializeStack(&stack);

char \*token = expression;

while (\*token != '\0') {

if (isspace(\*token)) {

token++;

continue;

}

if (isdigit(\*token) || (\*token == '.' && isdigit(\*(token + 1)))) {

char number[MAX];

int i = 0;

while (isdigit(\*token) || \*token == '.') {

number[i++] = \*token++;

}

number[i] = '\0';

push(&stack, atof(number));

} else {

double operand2 = pop(&stack);

double operand1 = pop(&stack);

switch (\*token) {

case '+':

push(&stack, operand1 + operand2);

break;

case '-':

push(&stack, operand1 - operand2);

break;

case '\*':

push(&stack, operand1 \* operand2);

break;

case '/':

if (operand2 != 0)

push(&stack, operand1 / operand2);

else

printf("Division by zero!\n");

break;

default:

printf("Invalid operator: %c\n", \*token);

}

token++;

}

}

return pop(&stack);

}

int main() {

char expression[MAX];

printf("Enter an arithmetic expression in Reverse Polish Notation (RPN):\n");

fgets(expression, MAX, stdin);

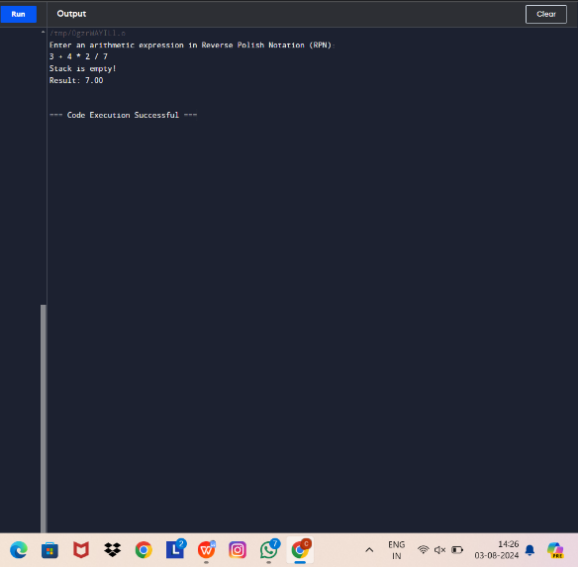
double result = evaluateExpression(expression);

printf("Result: %.2lf\n", result);

return 0;

}

Output:



12) Write a C program to balance symbols in a given expression

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

typedef struct {

int top;

char items[MAX];

} Stack;

void initializeStack(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 value) {

if (isFull(s)) {

printf("Stack is full!\n");

} else {

s->items[++s->top] = value;

}

}

char pop(Stack \*s) {

if (isEmpty(s)) {

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

return '\0';

} else {

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

}

}

char peek(Stack \*s) {

if (isEmpty(s)) {

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

return '\0';

} else {

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

}

}

int isMatchingPair(char left, char right) {

if (left == '(' && right == ')') return 1;

if (left == '{' && right == '}') return 1;

if (left == '[' && right == ']') return 1;

return 0;

}

int areSymbolsBalanced(char \*expression) {

Stack stack;

initializeStack(&stack);

for (int i = 0; i < strlen(expression); i++) {

char current = expression[i];

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

push(&stack, current);

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

if (isEmpty(&stack)) {

return 0;

} else {

char popped = pop(&stack);

if (!isMatchingPair(popped, current)) {

return 0;

}

}

}

}

return isEmpty(&stack);

}

int main() {

char expression[MAX];

printf("Enter an expression: ");

fgets(expression, MAX, stdin);

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

if (areSymbolsBalanced(expression)) {

printf("Symbols are balanced.\n");

} else {

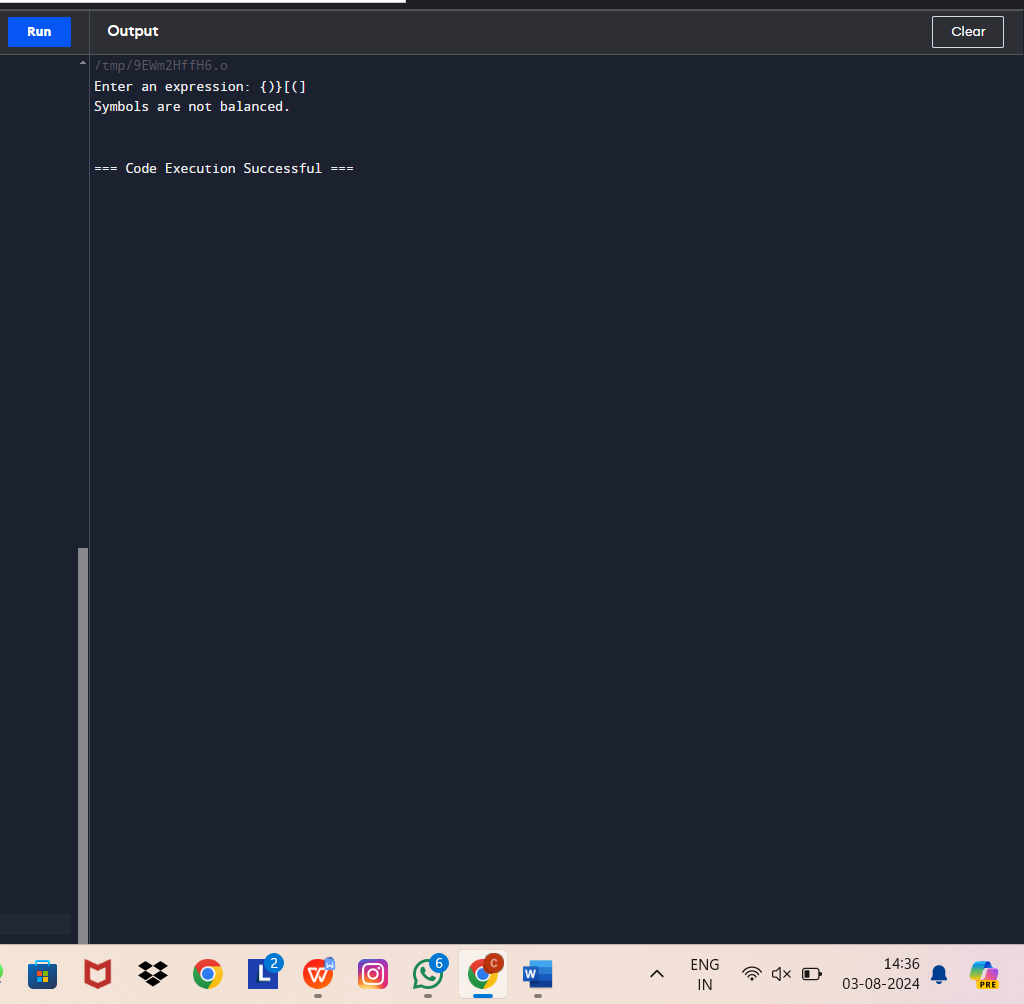
printf("Symbols are not balanced.\n");

}

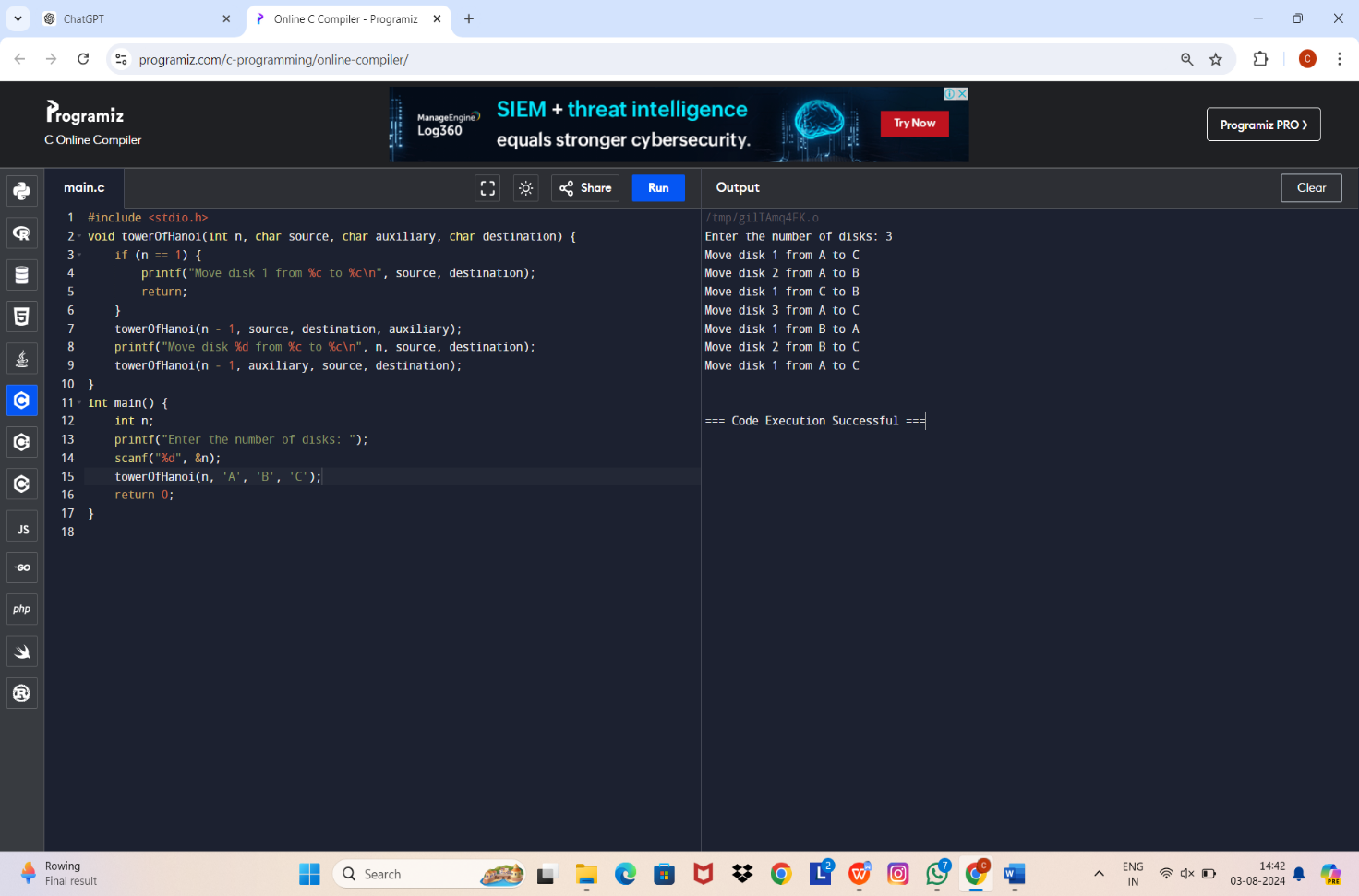
return 0;

}

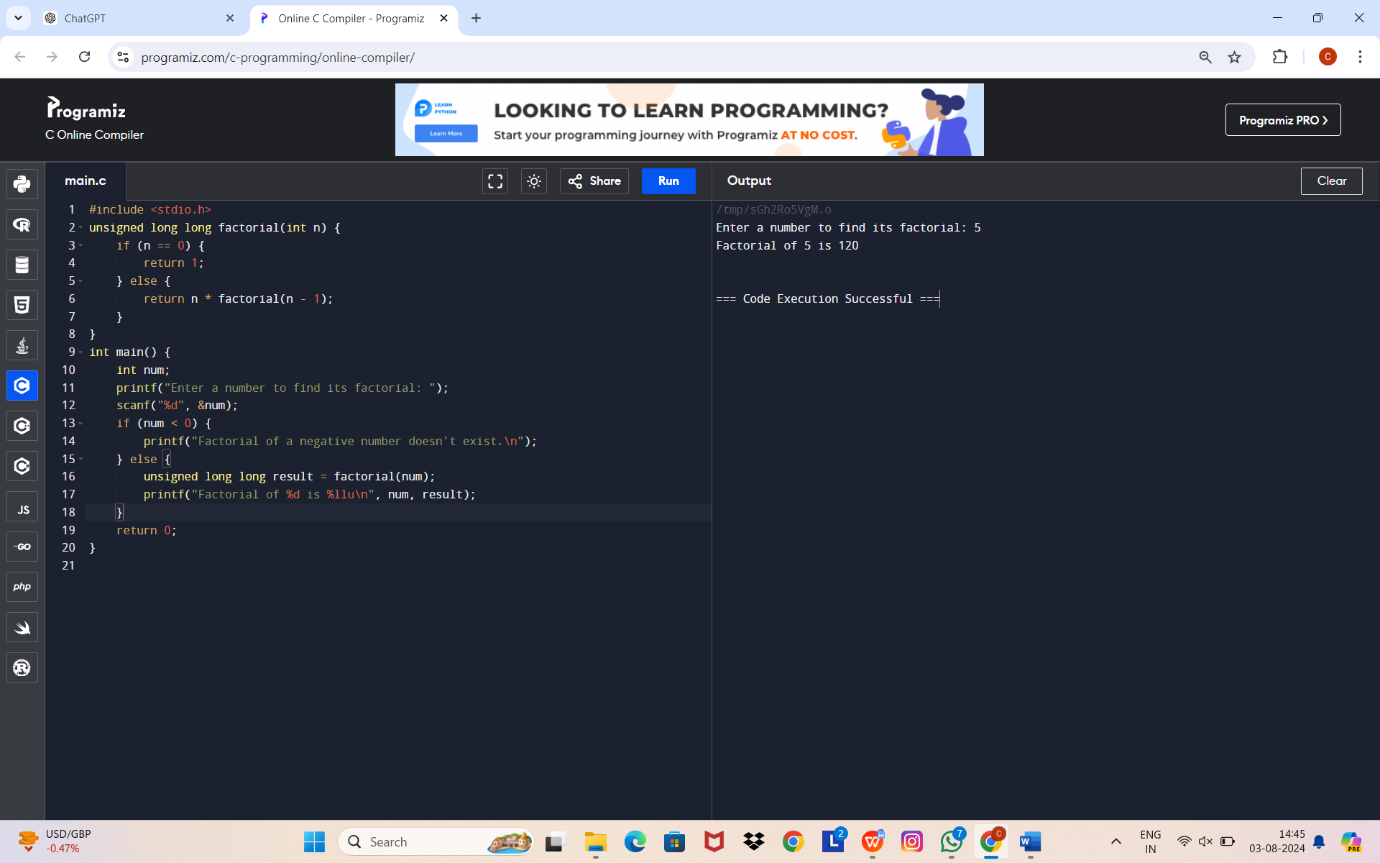
Output:



13) Write a recursive function in C to implement Tower of Hanoi Problem



14) Write a recursive function in C to find the factorial of a number



15) Implement a queue using an array

#include <stdio.h>

#include <stdlib.h>

#define SIZE 5 // Define the maximum size of the queue

typedef struct {

int items[SIZE];

int front;

int rear;

} Queue;

// Function to initialize the queue

void initializeQueue(Queue\* q) {

q->front = -1;

q->rear = -1;

}

// Function to check if the queue is full

int isFull(Queue\* q) {

return (q->rear == SIZE - 1);

}

// Function to check if the queue is empty

int isEmpty(Queue\* q) {

return (q->front == -1 || q->front > q->rear);

}

// Function to add an element to the queue

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

if (isFull(q)) {

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

return;

}

if (q->front == -1) {

q->front = 0;

}

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

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

}

// Function to remove an element from the queue

int dequeue(Queue\* q) {

if (isEmpty(q)) {

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

return -1;

}

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

if (q->front > q->rear) {

q->front = q->rear = -1;

}

printf("Dequeued: %d\n", value);

return value;

}

// Function to display the queue elements

void display(Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty!\n");

return;

}

printf("Queue elements are: ");

for (int i = q->front; i <= q->rear; i++) {

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

}

printf("\n");

}

int main() {

Queue q;

initializeQueue(&q);

enqueue(&q, 25);

enqueue(&q, 37);

enqueue(&q, 90);

display(&q);

dequeue(&q);

display(&q);

enqueue(&q, 15);

enqueue(&q, 40);

enqueue(&q, 12);

display(&q);

dequeue(&q);

dequeue(&q);

dequeue(&q);

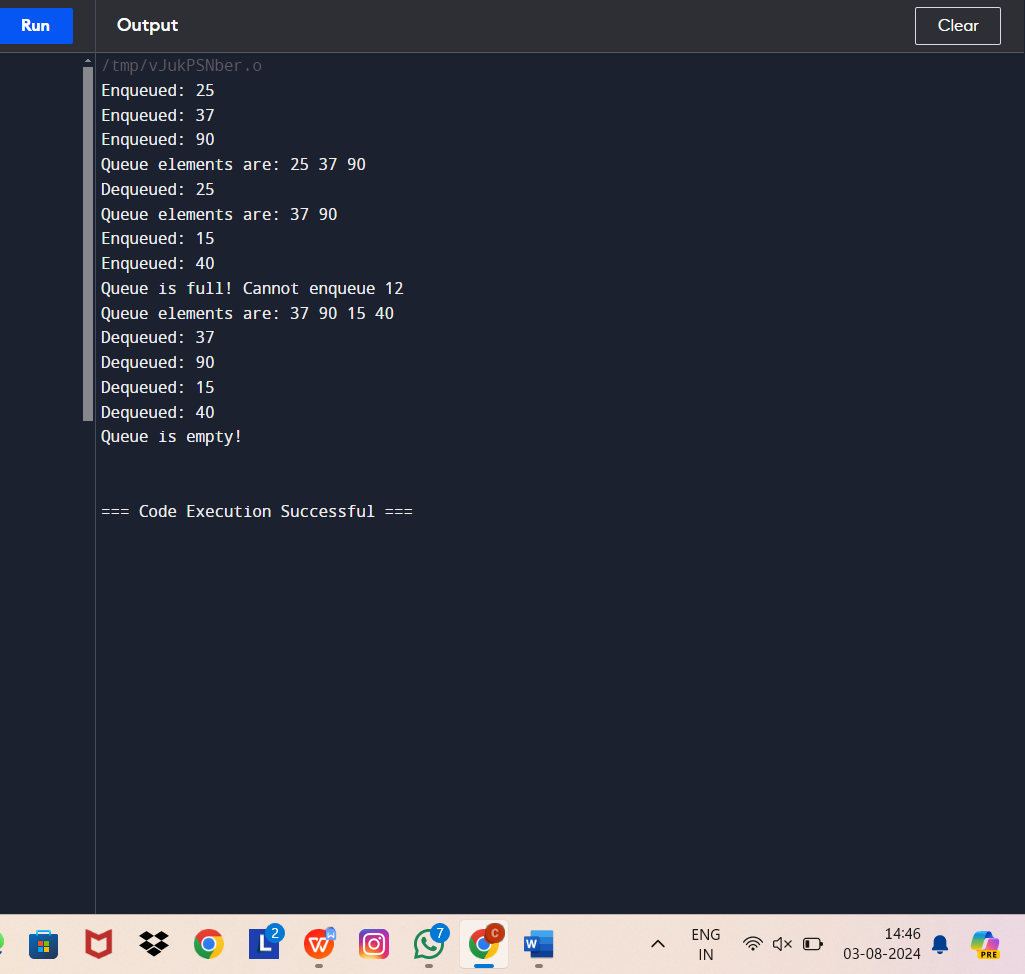
dequeue(&q);

display(&q);

return 0;

}

Output:



16) Implement a queue using linked list

#include <stdio.h>

#include <stdlib.h>

// Define a node in the linked list

typedef struct Node {

int data;

struct Node\* next;

} Node;

// Define the queue structure

typedef struct {

Node\* front;

Node\* rear;

} Queue;

// Function to initialize the queue

void initializeQueue(Queue\* q) {

q->front = NULL;

q->rear = NULL;

}

// Function to check if the queue is empty

int isEmpty(Queue\* q) {

return q->front == NULL;

}

// Function to add an element to the queue

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

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

if (!newNode) {

printf("Memory allocation error!\n");

return;

}

newNode->data = value;

newNode->next = NULL;

if (isEmpty(q)) {

q->front = newNode;

q->rear = newNode;

} else {

q->rear->next = newNode;

q->rear = newNode;

}

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

}

// Function to remove an element from the queue

int dequeue(Queue\* q) {

if (isEmpty(q)) {

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

return -1;

}

Node\* temp = q->front;

int value = temp->data;

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

if (q->front == NULL) {

q->rear = NULL;

}

free(temp);

printf("Dequeued: %d\n", value);

return value;

}

// Function to display the queue elements

void display(Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty!\n");

return;

}

Node\* current = q->front;

printf("Queue elements are: ");

while (current != NULL) {

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

current = current->next;

}

printf("\n");

}

int main() {

Queue q;

initializeQueue(&q);

enqueue(&q, 25);

enqueue(&q, 37);

enqueue(&q, 90);

display(&q);

dequeue(&q);

display(&q);

enqueue(&q, 15);

enqueue(&q, 40);

enqueue(&q, 12);

display(&q);

dequeue(&q);

dequeue(&q);

dequeue(&q);

dequeue(&q);

display(&q);

return 0;

}

Output:

