TOWER OF HANOI:

#include <stdio.h>

void move(int n, int source, int destination, int intermediate) {

if (n == 1) {

printf("Move disk 1 from shaft %d to shaft %d\n", source, destination);

return;

}

move(n - 1, source, intermediate, destination);

printf("Move disk %d from shaft %d to shaft %d\n", n, source, destination);

move(n - 1, intermediate, destination, source);

}

int main() {

int n = 4;

int source = 1;

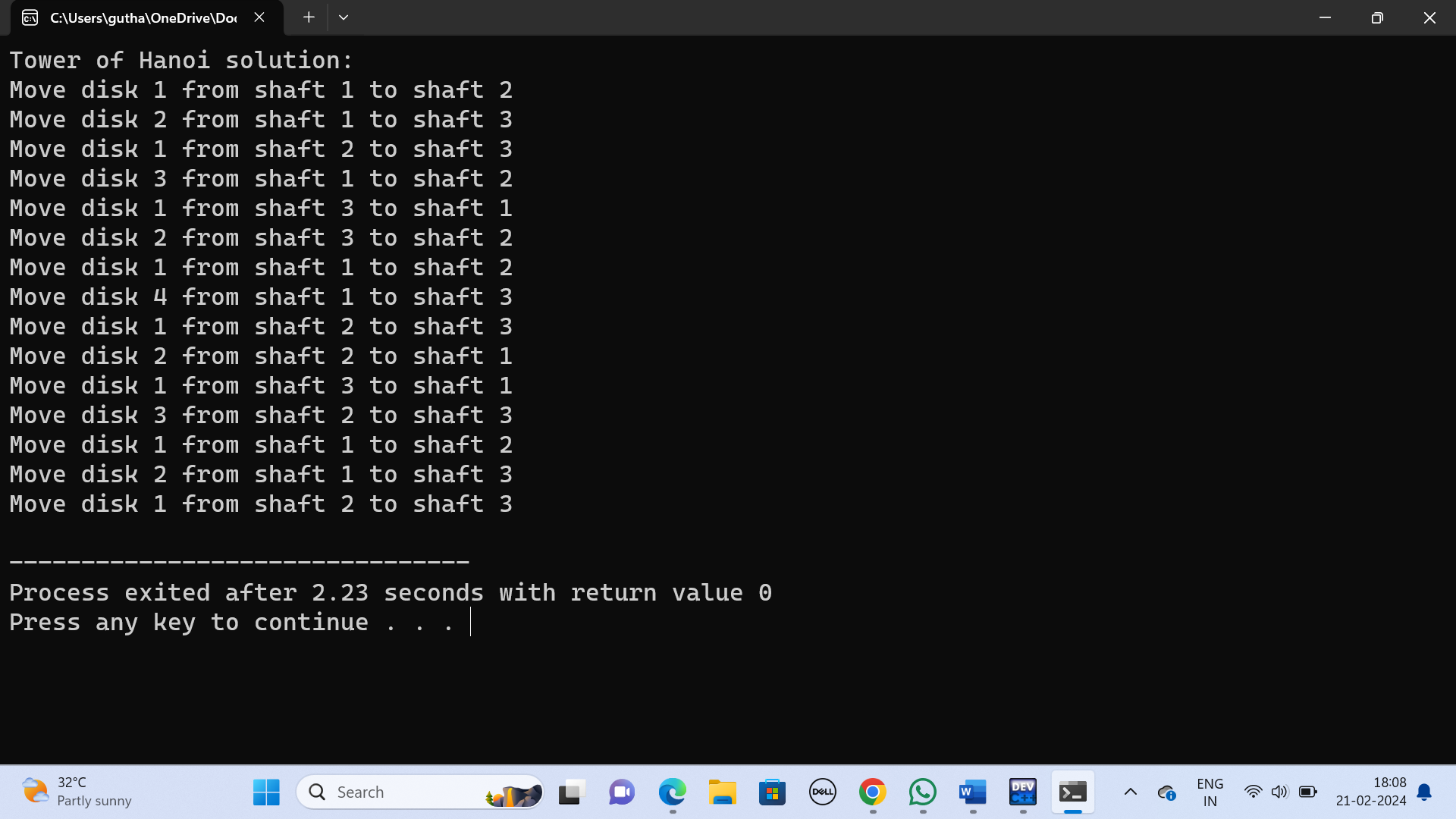
int destination = 3;

int intermediate = 2;

printf("Tower of Hanoi solution:\n");

move(n, source, destination, intermediate);

return 0;

}

CONVERSION OF INFINIX TO POSTFIX:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_SIZE 100

typedef struct {

char stack[MAX\_SIZE];

int top;

} Stack;

void initialize(Stack \*s) {

s->top = -1;

}

int is\_empty(Stack \*s) {

return s->top == -1;

}

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

if (s->top == MAX\_SIZE - 1) {

printf("Stack overflow\n");

exit(1);

}

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

}

char pop(Stack \*s) {

if (is\_empty(s)) {

printf("Stack underflow\n");

exit(1);

}

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

}

int precedence(char c) {

if (c == '+' || c == '-')

return 1;

if (c == '\*' || c == '/')

return 2;

return 0;

}

void infix\_to\_postfix(char \*infix, char \*postfix) {

Stack s;

initialize(&s);

int i, j;

char c;

for (i = 0, j = 0; infix[i] != '\0'; i++) {

if (isalnum(infix[i])) {

postfix[j++] = infix[i];

} else if (infix[i] == '(') {

push(&s, infix[i]);

} else if (infix[i] == ')') {

while (!is\_empty(&s) && (c = pop(&s)) != '(') {

postfix[j++] = c;

}

} else {

while (!is\_empty(&s) && precedence(infix[i]) <= precedence(s.stack[s.top])) {

postfix[j++] = pop(&s);

}

push(&s, infix[i]);

}

}

while (!is\_empty(&s)) {

postfix[j++] = pop(&s);

}

postfix[j] = '\0';

}

int main() {

char infix[MAX\_SIZE], postfix[MAX\_SIZE];

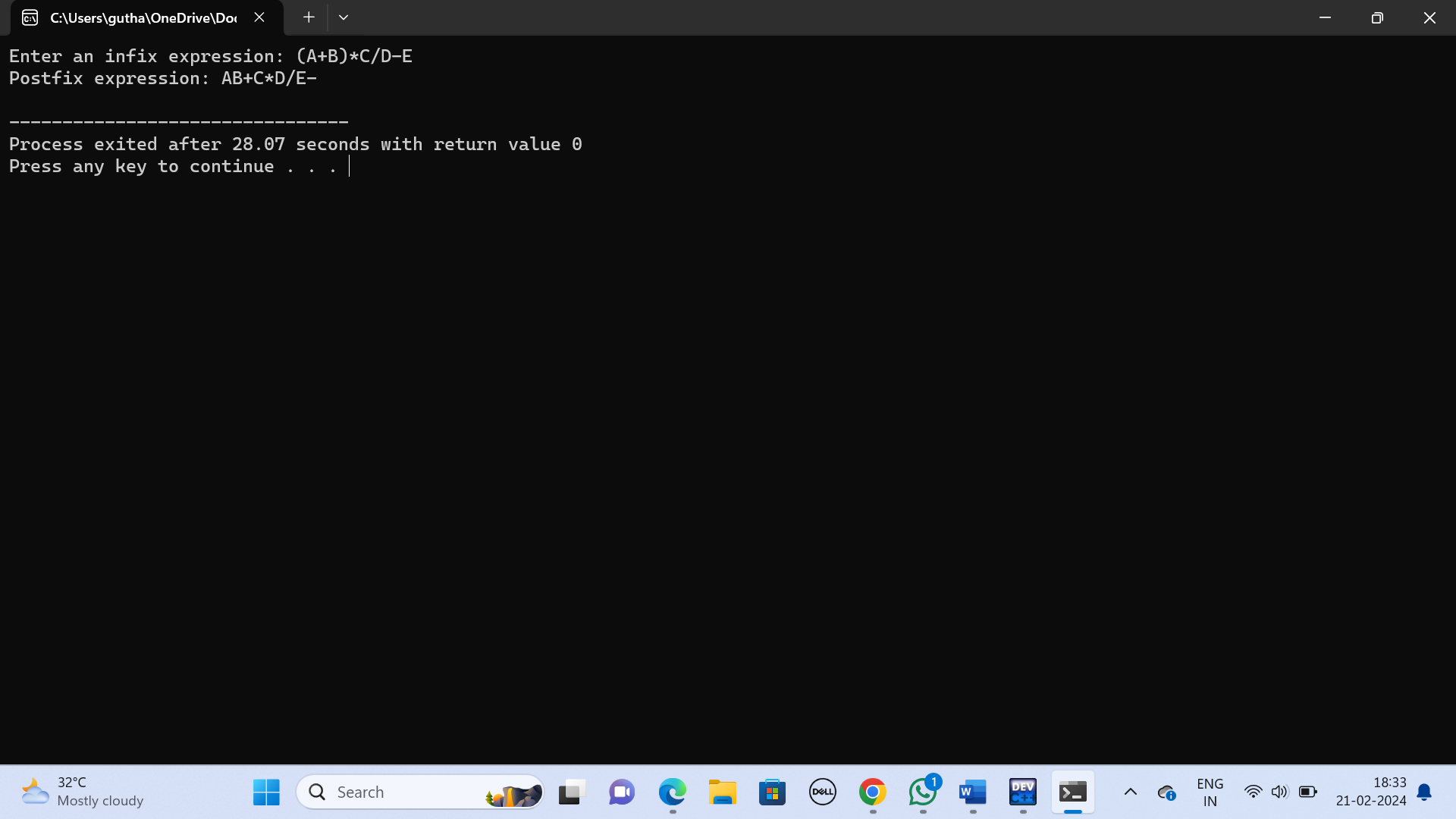
printf("Enter an infix expression: ");

scanf("%s", infix);

infix\_to\_postfix(infix, postfix);

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

return 0;

}

IMPLEMENTATION OF STACK:

#include<stdio.h>

int top=-1,stack[100],n;

int isfull(){

if(top==n-1)

return 1;

else

return 0;

};

int isempty(){

if(top==-1)

return 1;

else

return 0;

};

void push(){

if(isfull()==1)

printf("overflow\n");

else{

printf("enter the number :\n");

top++;

scanf("%d",&stack[top]);

}

};

void pop(){

if(isempty()==1)

printf("underflow the stack is empty\n");

else{

printf("%d",stack[top]);

top--;

}

};

void peek(){

if(isempty()==1)

printf("underflow the stack is empty\n");

else

printf("%d\n",stack[top]);

};

void size(){

printf("%d",top+1);

}

int main(){

int i=0;

printf("enter the number of elements in the stack :");

scanf("%d",&n);

printf("Chose one from the below options\n");

printf("\n1.Push\n2.Pop\n3.Peek\n4.Exit");

while(i!=4){

printf("\n Enter your option \n");

scanf("%d",&i);

switch(i)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

peek();

break;

}

case 4:

{

printf("Exiting");

break;

}

default:

{

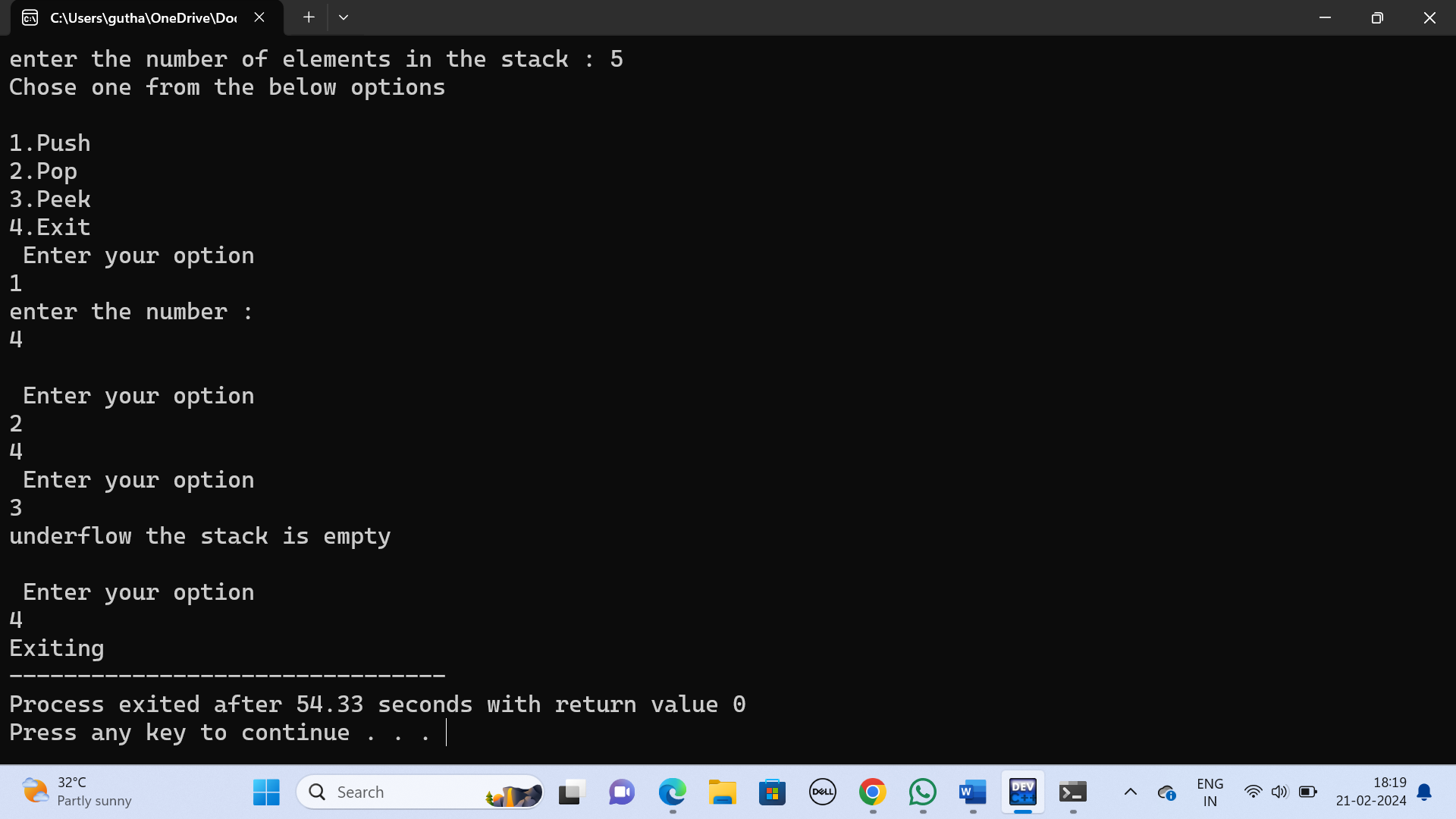
printf("Please Enter valid option\n");

}

}

}

return 0;

}

EVALUATION OF EXPRESSIONS:

#include <stdio.h>

#include <stdlib.h>

// Function to evaluate expressions

double evaluateExpression(char expression[]) {

char operators[100];

double operands[100];

int opCount = 0;

int numCount = 0;

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

if (expression[i] == '+' || expression[i] == '-' || expression[i] == '\*' || expression[i] == '/') {

operators[opCount++] = expression[i];

} else if (expression[i] >= '0' && expression[i] <= '9') {

// Convert character to integer

double num = expression[i] - '0';

// Continue reading digits to get the complete number

while (expression[i + 1] >= '0' && expression[i + 1] <= '9') {

num = num \* 10 + (expression[i + 1] - '0');

i++;

}

operands[numCount++] = num;

}

}

double result = operands[0];

for (int i = 0; i < opCount; i++) {

switch (operators[i]) {

case '+':

result += operands[i + 1];

break;

case '-':

result -= operands[i + 1];

break;

case '\*':

result \*= operands[i + 1];

break;

case '/':

if (operands[i + 1] != 0) {

result /= operands[i + 1];

} else {

printf("Error: Division by zero\n");

exit(EXIT\_FAILURE);

}

break;

default:

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

exit(EXIT\_FAILURE);

}

}

return result;

}

int main() {

char expression[100];

printf("Enter an expression: ");

fgets(expression, sizeof(expression), stdin);

// Remove newline character from the input

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

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

expression[i] = '\0';

break;

}

}

double result = evaluateExpression(expression);

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

return 0;

}

