#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MAX\_SIZE 100

// Stack definition

struct Stack {

char items[MAX\_SIZE];

int top;

};

// Initialize the stack

void initialize(struct Stack \*stack) {

stack->top = -1;

}

// Check if the stack is empty

int isEmpty(struct Stack \*stack) {

return stack->top == -1;

}

// Check if the stack is full

int isFull(struct Stack \*stack) {

return stack->top == MAX\_SIZE - 1;

}

// Push an item onto the stack

void push(struct Stack \*stack, char item) {

if (!isFull(stack)) {

stack->items[++stack->top] = item;

} else {

printf("Stack overflow\n");

}

}

// Pop an item from the stack

char pop(struct Stack \*stack) {

if (!isEmpty(stack)) {

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

} else {

printf("Stack underflow\n");

return '\0';

}

}

// Get the top item from the stack without removing it

char peek(struct Stack \*stack) {

if (!isEmpty(stack)) {

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

} else {

printf("Stack is empty\n");

return '\0';

}

}

// Check if the character is an operator

int isOperator(char c) {

return (c == '+' || c == '-' || c == '\*' || c == '/');

}

// Get the precedence of an operator

int getPrecedence(char c) {

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

return 1;

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

return 2;

return 0; // For other characters (operands)

}

// Convert infix expression to postfix

void infixToPostfix(char infix[], char postfix[]) {

struct Stack stack;

initialize(&stack);

int i, j;

char c, popped;

for (i = 0, j = -1; infix[i]; i++) {

c = infix[i];

if (isalnum(c)) {

postfix[++j] = c;

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

push(&stack, c);

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

while (!isEmpty(&stack) && peek(&stack) != '(') {

popped = pop(&stack);

postfix[++j] = popped;

}

if (!isEmpty(&stack) && peek(&stack) != '(') {

printf("Invalid expression\n");

return;

} else {

pop(&stack); // Remove the '(' from the stack

}

} else if (isOperator(c)) {

while (!isEmpty(&stack) && getPrecedence(c) <= getPrecedence(peek(&stack))) {

popped = pop(&stack);

postfix[++j] = popped;

}

push(&stack, c);

}

}

while (!isEmpty(&stack)) {

popped = pop(&stack);

postfix[++j] = popped;

}

postfix[++j] = '\0';

}

// Evaluate postfix expression

int evaluatePostfix(char postfix[]) {

struct Stack stack;

initialize(&stack);

int i, operand1, operand2, result;

for (i = 0; postfix[i]; i++) {

char c = postfix[i];

if (isdigit(c)) {

push(&stack, c - '0'); // Convert char digit to integer

} else if (isOperator(c)) {

operand2 = pop(&stack);

operand1 = pop(&stack);

switch (c) {

case '+':

result = operand1 + operand2;

break;

case '-':

result = operand1 - operand2;

break;

case '\*':

result = operand1 \* operand2;

break;

case '/':

if (operand2 == 0) {

printf("Division by zero\n");

return 0;

}

result = operand1 / operand2;

break;

}

push(&stack, result);

}

}

return pop(&stack);

}

int main() {

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

printf("Enter an infix expression: ");

scanf("%s", infix);

infixToPostfix(infix, postfix);

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

int result = evaluatePostfix(postfix);

printf("Result of expression: %d\n", result);

return 0;

}

A screenshot of a computer code

Description automatically generated