**20) Write a C program to compute TRAILING( ) – operator precedence parser for the given grammar**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

char stack[MAX];

int top = -1;

void push(char c) {

if (top < MAX - 1) {

stack[++top] = c;

}

}

char pop() {

if (top >= 0) {

return stack[top--];

}

return '\0';

}

int isOperator(char c) {

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

}

int precedence(char c) {

if (c == '+' || c == '-') return 1;

if (c == '\*' || c == '/') return 2;

return 0;

}

void infixToPostfix(char \*input) {

char \*token = strtok(input, " ");

while (token != NULL) {

// If the token is an operand, print it

if (!isOperator(token[0])) {

printf("%s ", token);

} else {

// If the token is an operator, pop from the stack

// until the top of the stack has an operator with lower precedence

while (top >= 0 && precedence(stack[top]) >= precedence(token[0])) {

printf("%c ", pop());

}

// Push the current operator to the stack

push(token[0]);

}

token = strtok(NULL, " ");

}

// Pop all the remaining operators in the stack

while (top >= 0) {

printf("%c ", pop());

}

printf("\n");

}

int main() {

char input[MAX];

printf("Enter the expression (space-separated): ");

fgets(input, MAX, stdin);

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

// Convert infix to postfix

infixToPostfix(input);

return 0;

}

