**Expt:3**

**A calculator needs to evaluate a postfix expression. Develop and execute a program in C using a suitable data structure to evaluate a valid postfix expression. Assume that the postfix expression is read as a single line consisting of non-negative single digit operands and binary arithmetic operators. The arithmetic operators are + (add), (subtract), \* (multiply) and/ (divide).**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX 100

// Stack structure

typedef struct {

int top;

char items[MAX];

} Stack;

// Initialize stack

void initStack(Stack \*s) {

s->top = -1;

}

// Check if the stack is empty

int isEmpty(Stack \*s) {

return s->top == -1;

}

// Push an element onto the stack

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

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

printf("Stack Overflow\n");

return;

}

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

}

// Pop an element from the stack

char pop(Stack \*s) {

if (isEmpty(s)) {

printf("Stack Underflow\n");

exit(1);

}

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

}

// Peek the top element of the stack

char peek(Stack \*s) {

if (isEmpty(s)) {

return '\0';

}

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

}

// Function to check if the character is an operator

int isOperator(char ch) {

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

}

// Function to return precedence of operators

int precedence(char op) {

if (op == '+' || op == '-') {

return 1;

} else if (op == '\*' || op == '/') {

return 2;

} else {

return 0;

}

}

// Function to check if the parentheses are balanced

int checkBalancedParentheses(char \*exp) {

Stack s;

initStack(&s);

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

if (exp[i] == '(') {

push(&s, exp[i]);

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

if (isEmpty(&s)) {

return 0;

}

pop(&s);

}

}

return isEmpty(&s);

}

// Function to convert infix to postfix

void infixToPostfix(char \*exp, char \*postfix) {

Stack s;

initStack(&s);

int j = 0;

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

if (isalnum(exp[i])) {

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

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

push(&s, exp[i]);

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

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

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

}

pop(&s); // pop '('

} else if (isOperator(exp[i])) {

while (!isEmpty(&s) && precedence(peek(&s)) >= precedence(exp[i])) {

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

}

push(&s, exp[i]);

}

}

while (!isEmpty(&s)) {

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

}

postfix[j] = '\0';

}

// Main function

int main() {

char infix[MAX], postfix[MAX];

printf("Enter a parenthesized arithmetic expression: ");

scanf("%s", infix);

// Check for balanced parentheses

if (!checkBalancedParentheses(infix)) {

printf("Error: Unbalanced parentheses\n");

return 1;

}

// Convert to postfix

infixToPostfix(infix, postfix);

// Print original and postfix expressions

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

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

return 0;

}