(AB(C*+D))* (C+) AB*/*

Date __/__/__
Page

LAB PROGRAM-2

W.A.P to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply), and / (divide).

Algorithm:

- Create an empty stack for the operator.
- Create an empty string for postfix operation.
- Read the infix expression.
- for each character ch in the infix expression:

a) If ch is an operand:
   Add ch to postfix string.
b) Else if ch is '(':
   push '(' onto stack.
c) Else if ch is ')':
   while top of stack is not '(':
   pop from stack and add to postfix.
   Pop '(' from stack.
d) If ch is an operator, push it onto stack.
~~Else if ch~~ is an operator
   while stack is not empty.
   ANI)
   top of stack >= ch:
   pop from stack and add to postfix.
   push ch onto stack

e). while stack is not empty:
   Pop from stack & add to post fix.
- Print postfix expression.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define MAX 100

char stack[MAX];
int top = -1;

void push(char c) {
    if (top == MAX-1) {
        printf("Stack overflow\n");
        return;
    }
    stack[++top] = c;
}

char pop() {
    if (top == -1) {
        printf("Stack underflow\n");
        return -1;
    }
    return stack[top--];
}

char peek() {
    if (top == -1) {
        return -1;
    }
    return stack[top];
}
```

Date __/__/__
Page _____

```
int precedence(char op){
    switch(op){
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        case '(':
            return 0;
    }
    return -1;
}

int associativity(char op){
    if(op == '^'){
        return 1;
    }
    return 0;
}

void infixtopostfix(char infix[], char postfix[]){
    int i, k=0;
    char c;
    for(i=0; infix[i] != '\0'; i++){
        c = infix[i];
        if(isalnum(c)){
            postfix[k++] = c;
        }
        else if(c == '('){
            push(c);
        }
```

```
else if (c == ')') {
    while (peek() != '(') {
        postfix[k++] = pop();
    }
    pop();
}
else {
    while (top != -1) && ((precedence(peek()) >
                precedence(c)) ||
            (precedence(peek()) = precedence(c)
                && associativity(c) == 0)))
        {
            postfix[k++] = pop();
        }
        push(c);
    }
}
while(top != -1) {
    postfix[k++] = pop();
}
postfix[k] = '\0';
}

int main() {
    char infix[MAX], postfix[MAX];
    printf("Enter infix expression: ");
    scanf("%s", infix);

    infixtopostfix(infix, postfix);
    printf("postfix expression : %s \n", postfix);
    return 0;
}
```

Date __/__/__
Page _____

output

Enter infix expression : a*b(c+d)
postfix expression : abcd+*

enter infix expression : a/b+c(d*e)
postfix expression : ab/cde*+

Enter infix expression : (a+b*c-(d/e*f)+g)+h)
postfix expression : abc*de/f*-g+h+

8/10/25

INDEX

Name ______________________ Std ________ Sec ________

Roll No. ______ Subject ______________ School/College ______________

School/College Tel. No. ______________ Parents Tel. No. ______________

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
1.	29/09/25	Push, pop, peek		✓ (10)
2.	06/10/25	Infix to postfix		6/10/25 (10)