INDEX

Sl No.	Date	Title	Page No	Teacher sign
1.	29/09/25	Push, Pop, Peek	1.	✓ (10)
2.	6/10/25	Infix to Postfix		~~✓~~ (10) 6/10

Date: 6/10/25

LAB PROGRAM-2

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

Algorithm:

1. Initialise an empty stack S
2. Initialise an empty string postfix
3. for each symbol ch in the infix expression
   a. if (ch is an operand, append it to postfix
   b. if ch is '(', push it onto S.
   c. if ch is ')'
      - pop and append symbols from S to postfix until '(' is encountered
      - pop '(' from S (do not append it)
   d. if ch is an operator push it onto S,
   e. if ch is an operator (*, +, -, /)
      while S is not empty, pop from S and append, to postfix, and
      top of stack >= ch.

Pop from the stack and add to postfix

Push ch into stack.

While stack is not empty:

Pop from stack and add to postfix print postfix.

6/10

Code:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
char stack[MAX];
int top = -1;
// Push function
void push(char c) {
    if (top == MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack[++top] = c;
}
```

Pop function

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

// peek function

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

// Function to return precedence of operators int precedence (char op){

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

```
// function to return associativity
// 0 = left to right,  1 = right to left
int associativity (char op) {
    if (op == '^')
        return 1;  // R to L
    return 0;  // L to R
}

// function to convert infix to postfix.
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)){
            // operand → directly to postfix
            Postfix [k++] = c;
        }
        else if (c == '('){
            Push(c);
        }
        else if (c == ')'){
            while(peek() != '('){
                Postfix [k++] = pop();
            }
            Pop (); // discard '('
        }
```

```
else {
    // operator
    while (top != -1 && ((precedence(peek()) > precedence(c)) ||
           (precedence(peek()) == precedence(c) &&
            associativity(c) == 0))) { // L to R
        postfix[k++] = pop();
    }
    push(c);
  }
}
// Pop remaining operators.
while (top != -1) {
    postfix[k++] = pop();
}
postfix[k] = '\0';
}

int main() {
    char infix[MAX], postfix[MAX];
    Printf("Enter a valid parenthesized infix expression : ");
    Scanf("%s", infix);
    infix to postfix(infix, postfix);
    Printf("Postfix Expression : %s\n", postfix);
    return 0;
}
```

o/p

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+

6/10/18