Date: 06-10-25

2. WAP 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:

initialise empty stack

initialise an empty string postfix

for i in infix expression

→ if i is operand, append to postfix

→ if i is left parenthesis push to stack

→ if i is right parenthesis

- pop and append from stack until left parenthesis encount
- pop left parenthesis from stack.

→ if ch is operator push into stack

→ if i is an operator (+,-,*,/) if top >= i, pop, append postf

while stack not empty, pop from stack, append to postfix

Return postfix.

MG 6/10/25.

• Precedence function:

return 1 for +,- return 2 for *,/.

CODE:

```
#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];
}

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

Page No.
Date:

```
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 a vaid parenthesized infix
                expression ");
        scanf ("%s", infix);
        infix To Postfix (infix, postfix);
        printf ("Postfix Expression : %s\n", postfix);
        return 0;
}
```

Output:

```
Enter a vaid parenthesized infix expression: a*b*(c+d)
Postfix expression: ab*cd+*

Enter a vaid parenthesized infix expression: a/b+(cd*e)
Postfix expression: ab/cde*e

Enter a vaid parenthesized infix expression
(a+(b*c-(a/e*f)+g)+h)
Postfix expression: abc*de/f*-g++h+
```

M4
6/10/23