**PRACTICAL-4**

**Aim: Implement a program that enters Infix expression, verify validity and convert it into Postfix expression (if valid).**

• Prerequisite: Stack data structure operations

• Input: Size of the data structure, circular queue operation.

• Output: Circular queue contents after every operation.

Sample Input 1: Enter infix expression for conversion: a+b#

Sample Output 1: ab+

Sample Input 2: Enter infix expression for conversion: a+b+#

Sample Output 2: NOT A VALID EXPRESSION

**ALGORITHM:**

1)  Examine the one by one character from the input.

2)  If it is operand, append in output string.

3)  If it is opening parenthesis, push it on stack.

4)  If it is an operator, then

i) If stack is empty, push operator on stack.

ii) If the top of stack is opening parenthesis, push operator on stack

iii) If operator has higher priority than the top of stack, push operator on stack.

iv) Else pop the operator from the stack and append in output string, repeat step 4

5)  If it is a closing parenthesis, pop operators from stack and append them in output string until an opening parenthesis is encountered. pop and discard the opening parenthesis.

6)  If there is more input characters then go to step 1

7)  If there is no more input, pop the remaining operators and append in output string.

**1. [Initialize stack]**

TOP ← 1 , PUSH (S, TOP, ’(‘)

**2. [Initialize output string]**

POLISH ← NULL

**3. [Get first input symbol]**

NEXT ← NEXTCHAR(INFIX)

**4. [Translate the infix expression]**

Repeat thru step 7 while NEXT ≠' (‘

**5. Remove symbols with greater precedence from stack]**

Repeat while f(NEXT) **<g**(S[TOP])

TEMP ← POP(S,TOP)

POLISH ← POLISH **○** TEMP

**6. [Are there matching parentheses?]**

If f (NEXT) ≠ **g** (S[TOP]) then Call PUSH (S,TOP,NEXT)

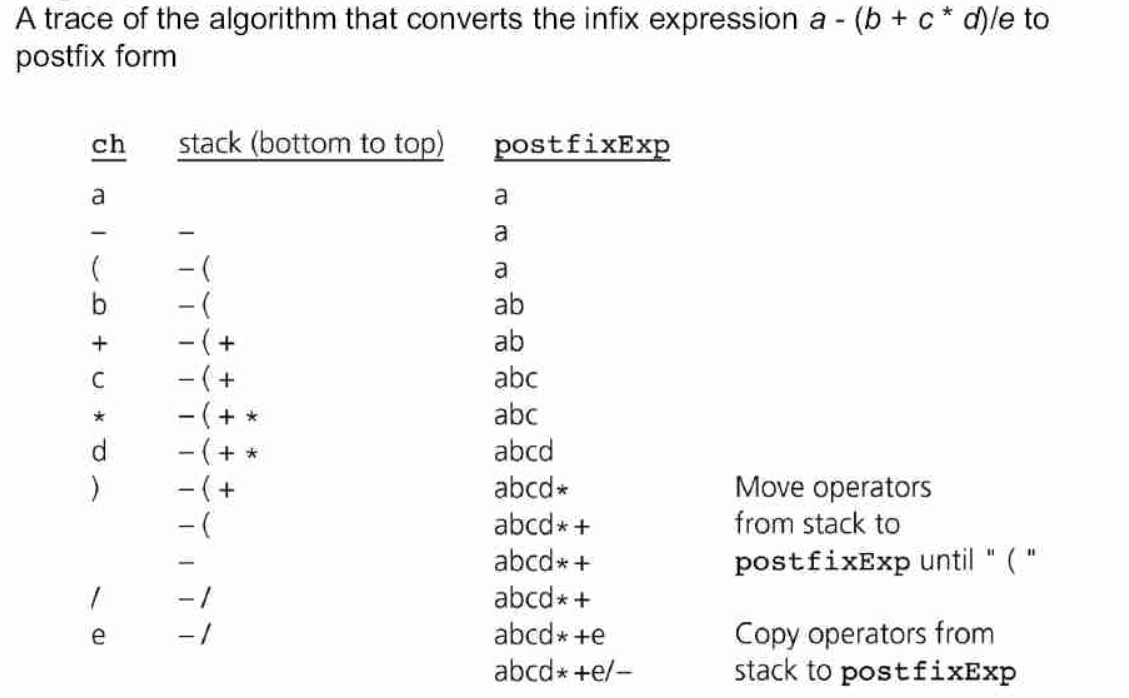
else POP(S,TOP)

**7. [Get next input symbol]**

NEXT ← NEXTCHAR(INFIX)

**8. Print Output String**

**TRACING WITH EXAMPLE:**



**CODE:**

#include<stdio.h>

#include<string.h>

#include<math.h>

#include<stdlib.h>

#define BLANK ' '

#define TAB '\t'

#define MAX 50

void push(long int symbol);

long int pop();

void infix\_to\_postfix();

int priority(char symbol);

int isEmpty();

int white\_space(char );

char infix[MAX], postfix[MAX];

long int stack[MAX];

int top;

int main()

{

long int value;

top=-1;

printf("Enter infix : ");

gets(infix);

infix\_to\_postfix();

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

return 0;

}

void infix\_to\_postfix()

{

unsigned int i,p=0;

char next;

char symbol;

for(i=0;i<strlen(infix);i++)

{

symbol=infix[i];

if(!white\_space(symbol))

{

switch(symbol)

{

case '(':

push(symbol);

break;

case ')':

while((next=pop())!='(')

postfix[p++] = next;

break;

case '+':

case '-':

case '\*':

case '/':

case '%':

case '^':

while( !isEmpty( ) && priority(stack[top])>= priority(symbol) )

postfix[p++]=pop();

push(symbol);

break;

default: /\*if an operand comes\*/

postfix[p++]=symbol;

}

}

}

while(!isEmpty( ))

postfix[p++]=pop();

postfix[p]='\0'; /\*End postfix with'\0' to make it a string\*/

}/\*End of infix\_to\_postfix()\*/

int priority(char symbol)

{

switch(symbol)

{

case '(':

return 0;

case '+':

case '-':

return 1;

case '\*':

case '/':

case '%':

return 2;

case '^':

return 3;

default :

return 0;

}

}/\*End of priority()\*/

void push(long int symbol)

{

if(top>MAX)

{

printf("Stack overflow\n");

exit(1);

}

stack[++top]=symbol;

}/\*End of push()\*/

long int pop()

{

if( isEmpty() )

{

printf("Stack underflow\n");

exit(1);

}

return (stack[top--]);

}/\*End of pop()\*/

int isEmpty()

{

if(top==-1)

return 1;

else

return 0;

}/\*End of isEmpty()\*/

int white\_space(char symbol)

{

if( symbol == BLANK || symbol == TAB )

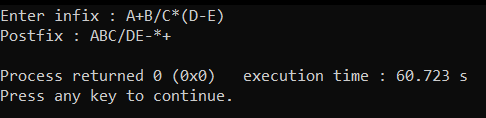
return 1;

else

return 0;

}/\*End of white\_space()\*/

**OUTPUT:**

****

**CONCLUSION:**

We learnt how to implement the conversion of infix to postfix with the help of c programming language.