**NAME: ASHWINI SHIVAJI RATHOD**

**CLASS:SE**

**DIV:3**

**Experiment No 2: Conversion of Infix to postfix expression using stack ADT**

**Aim:** To convert infix expression to postfix expression using stack ADT

**Objective:**

1) Understand the use of stack

2) Understand how to import an ADT in an application program

3) Understand the instantiation of stack ADT in an application program

4) Understand how the member function of an ADT are accessed in an application program

**Theory:**

**Infix:**

Infix expressions are readable and solvable by humans. We can easily distinguish the order of operators, and also can use the parenthesis to solve that part first during solving mathematical expressions. The computer cannot differentiate the operators and parenthesis easily, that's why postfix conversion is needed.

**Postfix:**

To convert infix expression to postfix expression, we will use the stack data structure. By scanning the infix expression from left to right, when we will get any operand, simply add them to the postfix form, and for the operator and parenthesis, add them in the stack maintaining the precedence of them.

**Algorithm:**

**1.Import and Initialize:**

* Import stdio.h and ctype.h.
* Declare the character stack stack and initialize top to -1.

**2.Functions:**

* push(char x): Increment top and push x onto stack.
* pop(): Return and remove the top character if the stack isn't empty.

**3.Priority:**

* priority(char x): Return precedence of operators, 0 for others.

**4.Main:**

* Declare exp array and character pointer e.
* Read infix expression into exp.
* Loop through characters using e.
* Alphanumeric: Print.
* '(': Push onto stack.
* ')': Pop and print until '('.

**5.Operator:**

* Pop and print higher/equal precedence operators.
* Push current operator.
* Move e to next character.

**6.Finish:**

* Pop and print remaining operators.
* return 0.

**Code:**

#include<stdio.h>

#include<ctype.h>

char stack[100];

int top=-1;

void push(char x)

{

stack[++top]=x;

}

char pop()

{

if(top==-1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x== '(')

return 0;

if(x=='+'|| x=='-')

return 1;

if(x=='\*'||x=='/')

return 2;

return 0;

}

int main()

{

char exp[100];

char \*e,x;

printf("Enter the expression: ");

scanf("%s",exp);

printf("\n");

e=exp;

while(\*e!='\0')

{

if(isalnum(\*e))

printf("%c",\*e);

else if(\*e=='(')

push(\*e);

else if(\*e== ')')

{

while((x=pop()) !='(')

printf("%c",x);

}

else

{

while(priority(stack[top])>=priority(\*e))

printf("%c",pop());

push(\*e);

}

e++;

}

while(top!=-1)

{

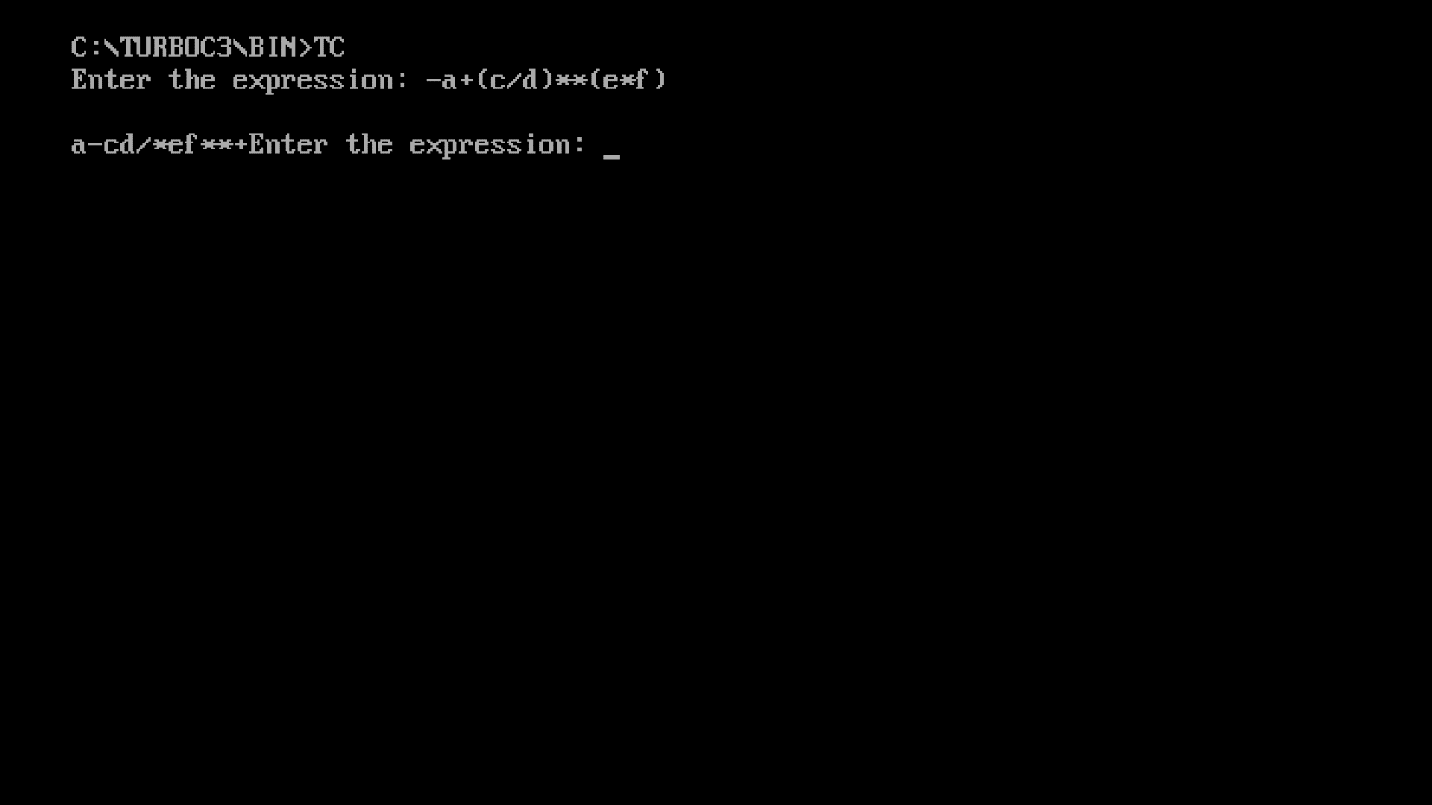
printf("%c",pop());

}

return 0;

}

**Output:**

****

**Conclusion:**

To convert infix expression to postfix expression, use the stack data structure. Scan the infix expression from left to right. Whenever we get an operand, add it to the postfix expression and if we get an operator or parenthesis add it to the stack by maintaining their precedence.