**3.Write a C program for conversion of expression given, from infix to postfix.**

**ALGORITHM**

Step 1: Scan the infix expression from left to right

Step 2: If the scanned character is an operand, append it with final infix to postfix string

Step 3: Else, if the precedence order of the scanned operator is greater than the precedence order of the operator in the stack

Step 4: Else, pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that push the scanned operator to the stack

Step 5: If the scanned character is a ‘(‘ or ‘)’, push it to the stack

Step 6: Repeat the steps until infix expression is scanned

Step 7:Print the output

Step 8: Pop and output from the stack until it is not empty

**PROGRAM**

#include<stdio.h>//standard I/O header file

#include<ctype.h>//standard library header file

char stack[100];//stack implementation

int top=-1;//declaring variables

void push(char x)//push function

{

stack[++top]=x;

}

char pop()//pop function

{

if(top==-1)

return -1;

else

return stack[top--];

}

int priority(char x)//priority function

{

if(x=='(')//condition check

return 0;

if(x=='+'||x=='-')//condition check

return 1;

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

return 2;

return 0;

}

int main()

{

char exp[100];

char \*e,x;

printf("enter the expression\n");

scanf("%s",exp);

printf("\n");//printing statement

e=exp;

while(\*e!='\0')//condition check

{

if(isalnum(\*e))//condition check

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

else if(\*e=='(')//condition check

push(\*e);

else if(\*e==')')//condition check

{

while((x=pop())!='(')//condition check

printf("%c",x);

}

else

{

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

printf("%c",pop());//printing statement

push(\*e);

}

e++;

}

while(top !=-1)//condition check

{

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

}

return 0;

}

**OUTPUT**



