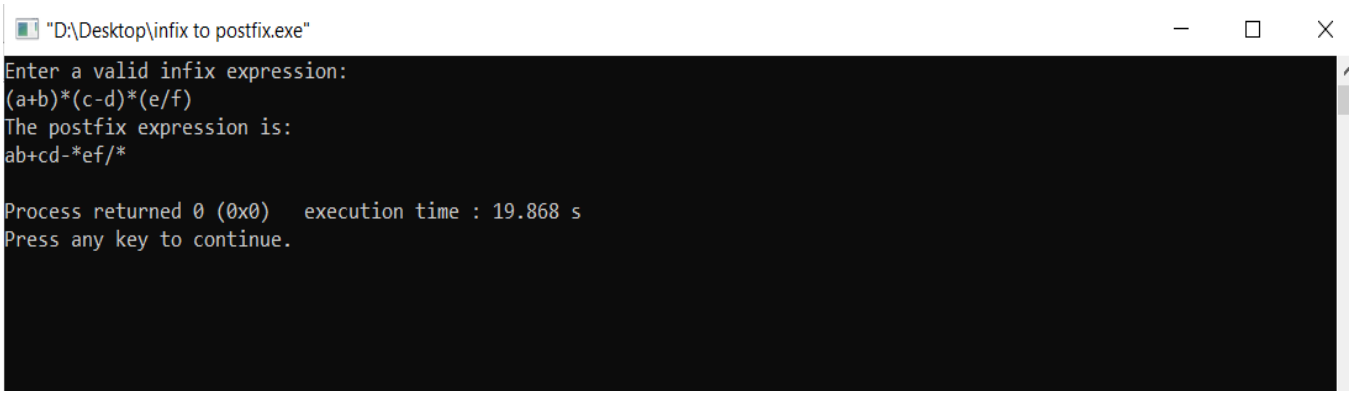
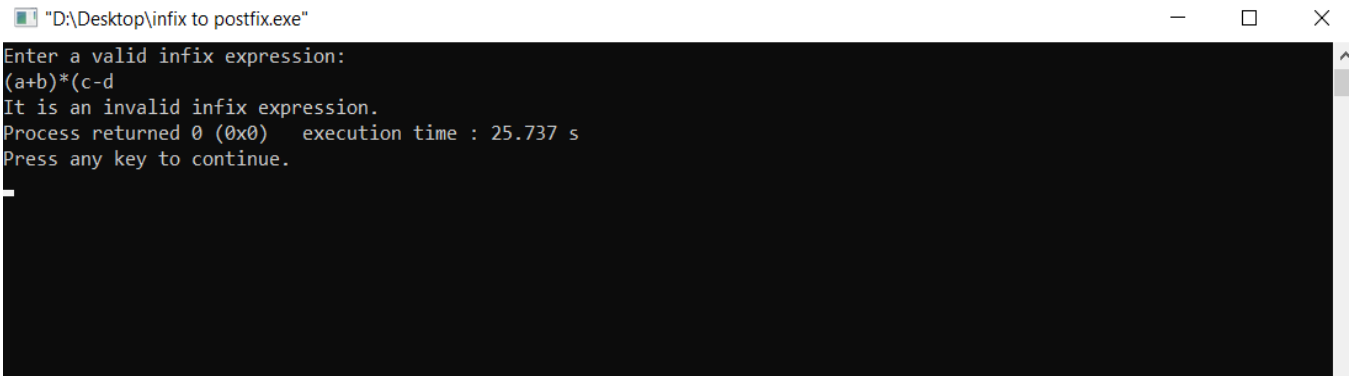
WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + , - , * and /

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int fun1(char symbol)
{
switch(symbol)
{
case '+' :
case '-' : return 2;
case '*' :
case '/' : return 4;
case '^' :
case '$' : return 5;
case '(' : return 0;
case '#' : return -1;
default : return 8;
}
}
int fun2(char symbol)
{
switch(symbol)
{
case '+' :
case '-' : return 1;
case '*' :
case '/' : return 3;
case '^' :
```

```
case '$' : return 6;
case '(' : return 9;
case ')' : return 0;
default : return 7;
}
}
void infix_postfix(char infix[],char postfix[])
{
int top,j,i;
char s[30];
char symbol;
top=-1;
s[++top]='#';
j=0;
for(i=0;i<strlen(infix);i++)
{
symbol=infix[i];
while(fun1(s[top])>fun2(symbol))
{
postfix[j]=s[top--];
j++;
}
if(fun1(s[top])!=fun2(symbol))
{
s[++top]=symbol;
}
else
top--;
}
```

```
while(s[top]!='#')
{
postfix[j++]=s[top--];}
postfix[j]='\0';
}
void main()
{
char infix[20],postfix[20];
int a=0,b=0,k;
printf("Enter a valid infix expression:\n");
scanf("%s",infix);
for(k=0;k<strlen(infix);k++)
{
if(infix[k]=='(')
a++;
else if(infix[k]==')')
b++;
else
continue;
}
if(a!=b)
{
printf("It is an invalid infix expression.");
exit(0);
}
infix_postfix(infix,postfix);
printf("The postfix expression is: \n");
printf("%s\n",postfix);
}
```


"D:\Desktop\infix to postfix.exe"
Enter a valid infix expression:
(a+b)*(c-d)*(e/f)
The postfix expression is:
ab+cd-*ef/*
Process returned 0 (0x0)   execution time : 19.868 s
Press any key to continue.


"D:\Desktop\infix to postfix.exe"
Enter a valid infix expression:
(a+b)*(c-d
It is an invalid infix expression.
Process returned 0 (0x0)   execution time : 25.737 s
Press any key to continue.