LAB Program (Infix to Postfix) Page No. Date 6 10 25

Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consist of single line charecter operends and the binary operators +, −, * and /.

Program

~~#include <stdio.h>~~

~~#define N 100~~
~~int stack[N];~~
~~int top=-1;~~

Algorithm

```
step 1  START
step 2  Input a string from user
step 3  check the first character of the string, if
        it is a variable push it into the stack
step 4  else if the character is a left porenthesis
        push it into the stack
step 5  if the character is an operand then check
        if the top of stack is higher priority. if
        the top is of higher priority then push otherwise
        pop and print the top and check for priority
        again. for same priority use associative law.
step 6  if the character is a right bracket then
        pop and print the element of stacks until
        the left part of bracket is found
step 7  On encountering the end of input pop and
        print all the element of the stack.
step 8  STOP
```

Page No. Date

Program

```
#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");
    }
    stack[++top] = c;
 }

~~void~~ 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 precendence (char op)
  {
    switch(op)
     {
      case '+':
      case '-':
          return 1;
      case '*':
      case '/':
          return 2;
      case '^':
          return 3;
      case '(':
      case ')': return 0;
     }
   return -1;
 }

int associativity (char op)
  {
    if (op=='^')
       return 1;
    return 0;
  }
```

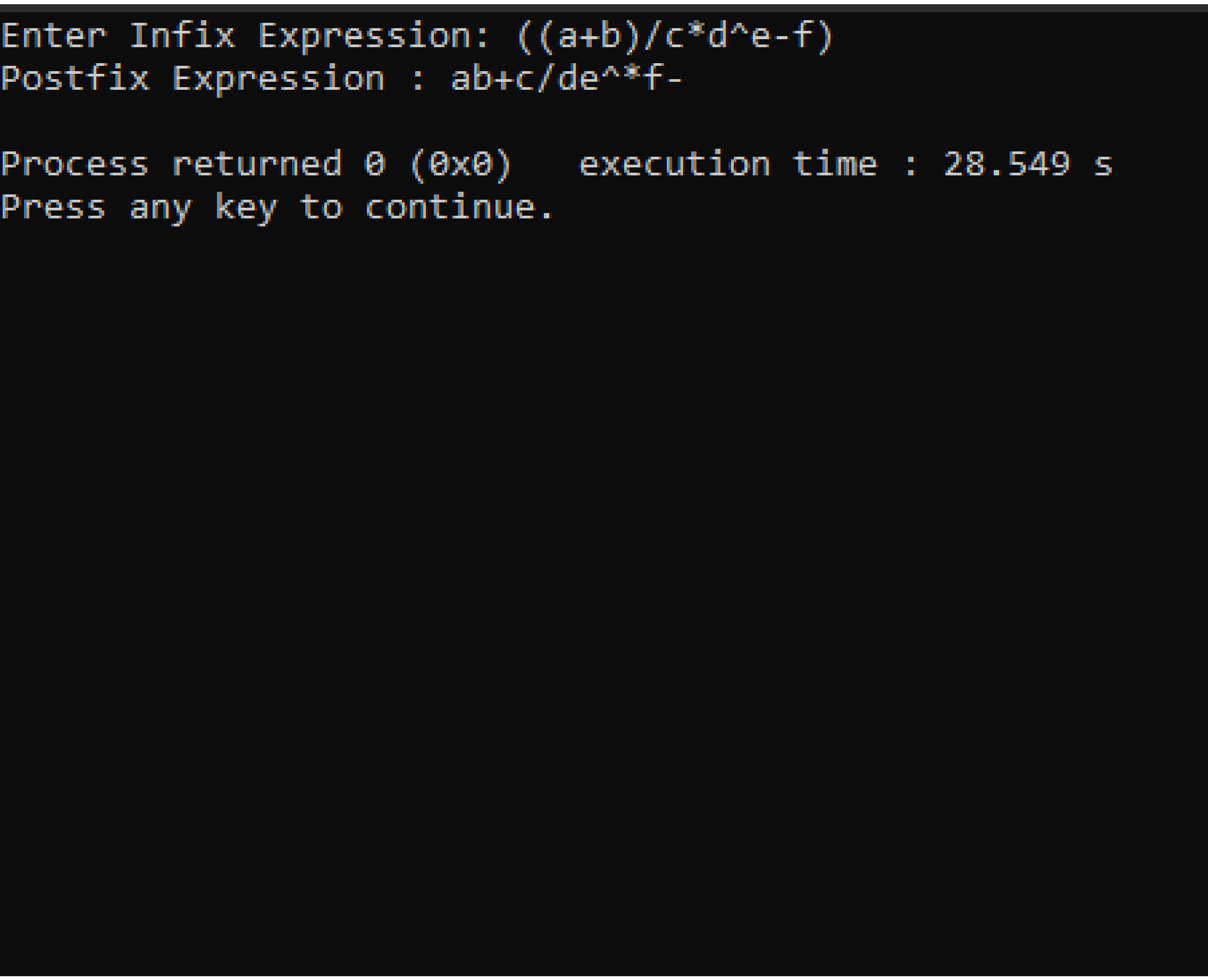
```
void infixToPostfix(char infix[], char postfix[])
  {
    int i, k=0;
    char c;

    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 input : ");
     scanf("%s", infix);

     infixToPostfix (infix, postfix);

     printf(" Postfix Expression : %s\n", postfix);
     return 0;
   }
```

Output

```
Enter Infix Expression: ((a+b)/c*d^e-f)
Postfix Expression : ab+c/de^*f-
```

[illegible]
6/10/25
[illegible]



```
Enter Infix Expression: ((a+b)/c*d^e-f)
Postfix Expression : ab+c/de^*f-

Process returned 0 (0x0)   execution time : 28.549 s
Press any key to continue.
```