Collage : Vishwakarma Institute of Technology
Course Name : Data Structure in C
Name : Vedika Vikas Sontakke
Roll no : 37
PRN NO 12220206

Assignment No – 3 : Implement a stack for following expression conversion.

a. Infix to Prefix

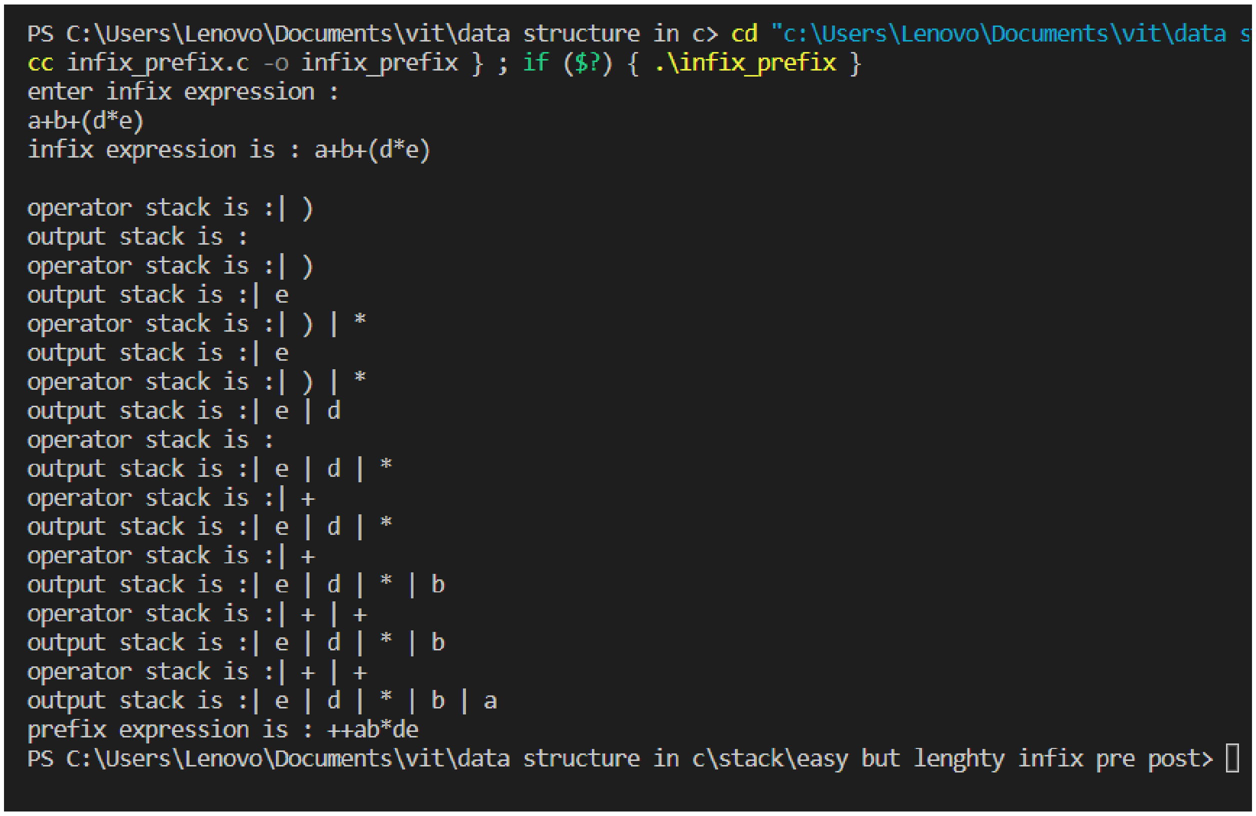
Program :

```
#include<stdio.h>
#include<string.h>

// operator stack
void pushopr(char);
char popopr();
char peepopr();
void displayopr();

//output stack
void pushout(char);
void displayout();

int getpriority(char);

char opr[25] = {'\0'};
char out[25] = {'\0'};
int topopr = -1;
int topout = -1;

void main()
{
   char infix[25] = {'\0'} , ele , pop_ele;

   printf("enter infix expression : \n");
   scanf("%s",infix);
```

```c
   printf("infix expression is : %s\n" , infix);

   int i = strlen(infix)-1;

   while(i >= 0)
   {
     ele = infix[i];
     if(ele == ')') pushopr(ele);
     else if(ele == '('){
        while(peepopr() != ')'){
            pop_ele = popopr();
            pushout(pop_ele);
        }
        popopr();
     }
     else if(ele == '^' || ele == '*' || ele == '/' || ele == '+' || ele == '-')
     {
         if(topopr >= 0) {
            while(getpriority(peepopr()) > getpriority(ele) && topopr != -1) {
                pop_ele = popopr();
                pushout(pop_ele);
            }
         }
         pushopr(ele);
     }
     else
         pushout(ele);

     displayopr();
     displayout();
     i--;
   }
   if(topopr != -1){
     while(topopr != -1) {
        pop_ele = popopr();
        pushout(pop_ele);
     }
   }

   printf("\nprefix expression is : %s \n", strrev(out));
}

void pushopr(char ele){
    if(topopr == 24) printf("\n operator stack is full\n");
    else opr[++topopr] = ele;
```

```c
}

char popopr()
{
    if(topopr != -1) return opr[topopr--];
}

char peepopr()
{
   if(topopr != -1) return opr[topopr];

}

void displayopr()
{
    printf("\noperator stack is :");
    for(int i=0 ; i<=topopr ; i++) printf("| %c ", opr[i]);
}

int getpriority(char ele)
{
   switch(ele)
   {
     case '^' : return 3;
     case '*' :
     case '/' : return 2;
     case '+' :
     case '-' : return 1;
   }
   return -1;
}

void pushout(char ele)
{
    if(topout == 24) printf("output stack is full\n");
    else out[++topout] = ele;
}

void displayout()
{
    printf("\noutput stack is :");
    for(int i=0 ; i<=topout ; i++)  printf("| %c ", out[i]);
}
```

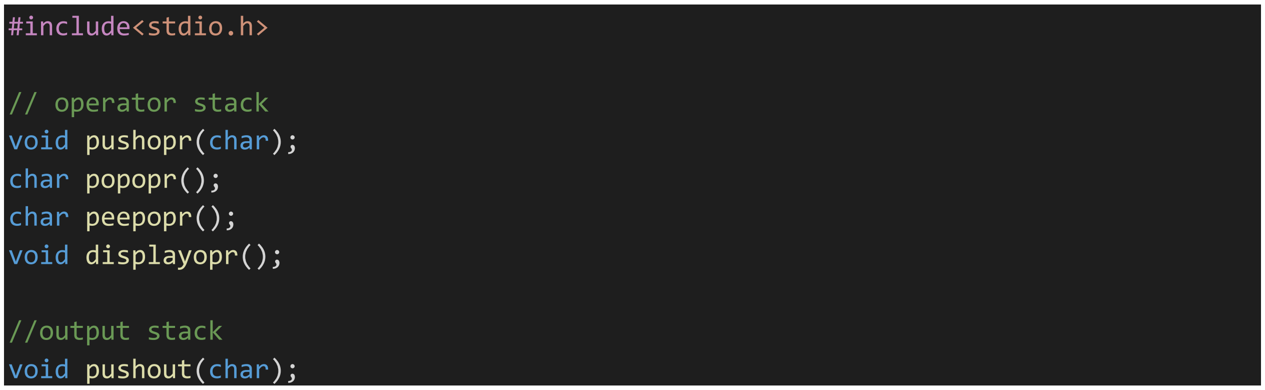
Output :



```
PS C:\Users\Lenovo\Documents\vit\data structure in c> cd "c:\Users\Lenovo\Documents\vit\data s
cc infix_prefix.c -o infix_prefix } ; if ($?) { .\infix_prefix }
enter infix expression :
a+b+(d*e)
infix expression is : a+b+(d*e)

operator stack is :| )
output stack is :
operator stack is :| )
output stack is :| e
operator stack is :| ) | *
output stack is :| e
operator stack is :| ) | *
output stack is :| e | d
operator stack is :
output stack is :| e | d | *
operator stack is :| +
output stack is :| e | d | *
operator stack is :| +
output stack is :| e | d | * | b
operator stack is :| + | +
output stack is :| e | d | * | b
operator stack is :| + | +
output stack is :| e | d | * | b | a
prefix expression is : ++ab*de
PS C:\Users\Lenovo\Documents\vit\data structure in c\stack\easy but lenghty infix pre post> 
```

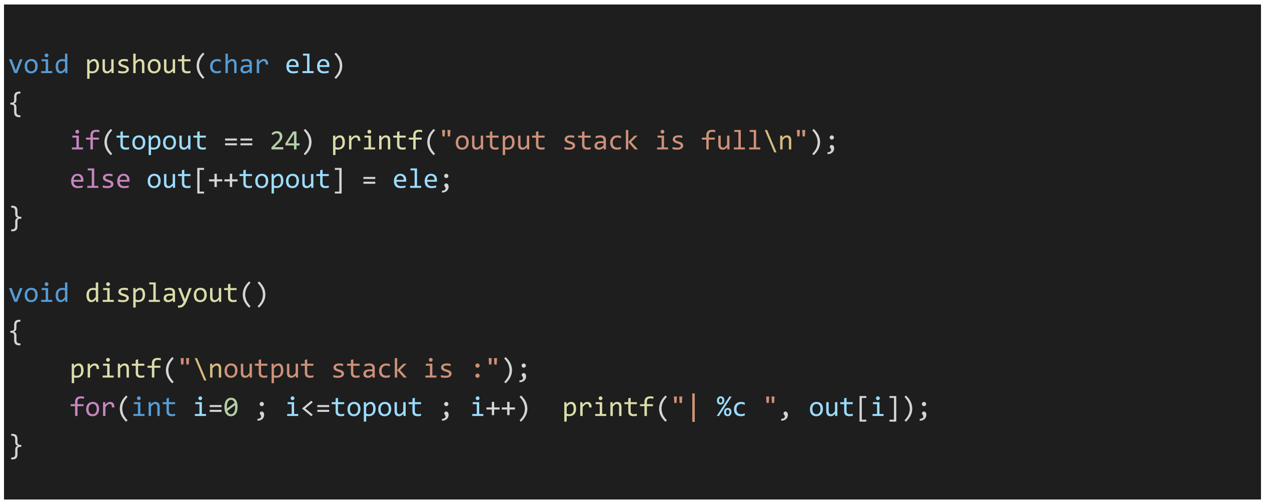
b. Infix to postfix

Program :



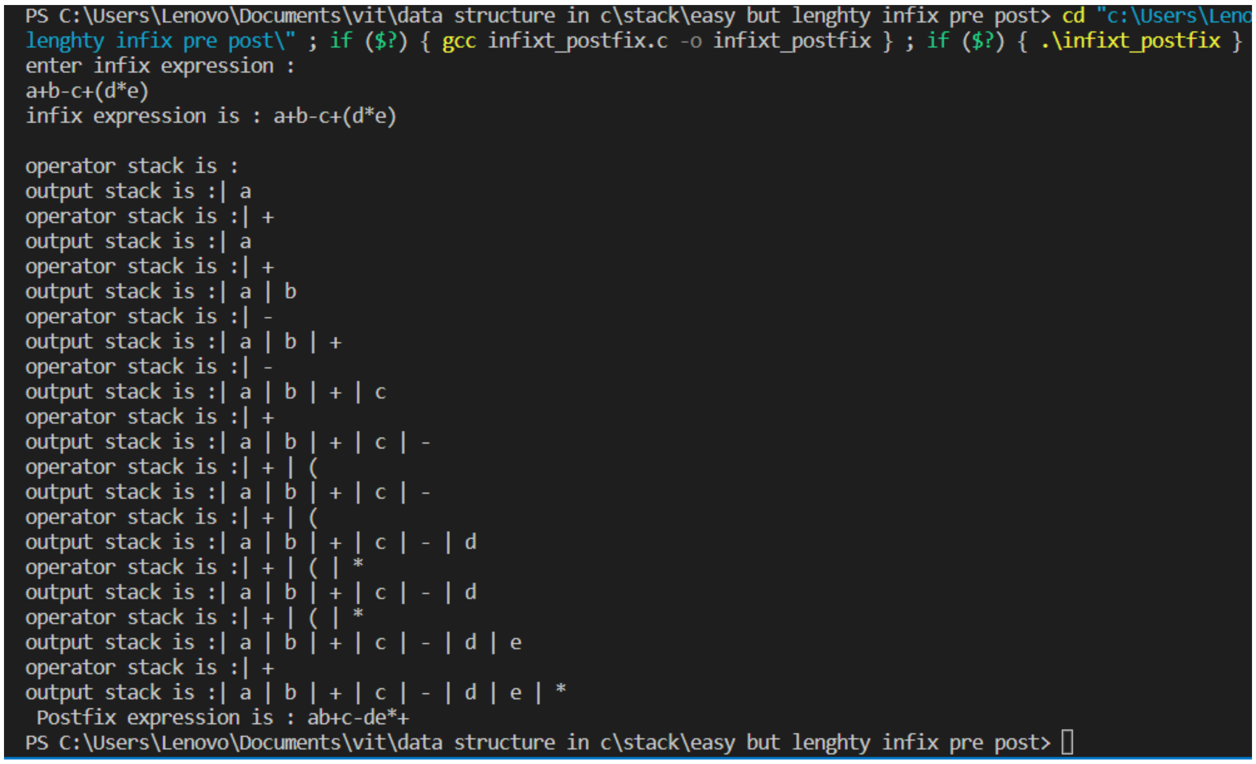
```c
#include<stdio.h>

// operator stack
void pushopr(char);
char popopr();
char peepopr();
void displayopr();

//output stack
void pushout(char);
void displayout();

int getpriority(char);

char opr[25] = {'\0'};
char out[25] = {'\0'};
int topopr = -1;
int topout = -1;

void main()
{
   char infix[25] = {'\0'} , ele , pop_ele;
   int i=0;

   printf("enter infix expression : \n");
   scanf("%s",infix);
   printf("infix expression is : %s\n" , infix);

   while(infix[i] != '\0')
   {
     ele = infix[i];
     if(ele == '(') pushopr(ele);
     else if(ele == ')'){
        while(peepopr() != '('){
            pop_ele = popopr();
            pushout(pop_ele);
        }
        popopr();
     }
     else if(ele == '^' || ele == '*' || ele == '/' || ele == '+' || ele == '-')
     {
         if(topopr >= 0) {
            while(getpriority(peepopr()) >= getpriority(ele)) {
                pop_ele = popopr();
                pushout(pop_ele);
            }
         }
         pushopr(ele);
     }
     else
         pushout(ele);

     displayopr();
     displayout();
```

```c
     i++;
   }
   if(topopr != -1){
     while(topopr != -1) {
        pop_ele = popopr();
        pushout(pop_ele);
     }
   }
   printf("\n Postfix expression is : %s \n", out);
}

void pushopr(char ele){
    if(topopr == 24) printf("\n operator stack is full\n");
    else opr[++topopr] = ele;
}

char popopr()
{
    if(topopr != -1) return opr[topopr--];
}

char peepopr()
{
   if(topopr != -1) return opr[topopr];

}

void displayopr()
{
    printf("\noperator stack is :");
    for(int i=0 ; i<=topopr ; i++) printf("| %c ", opr[i]);
}

int getpriority(char ele)
{
   switch(ele)
   {
     case '^' : return 3;
     case '*' :
     case '/' : return 2;
     case '+' :
     case '-' : return 1;
   }
   return -1;
}
```



```c
void pushout(char ele)
{
    if(topout == 24) printf("output stack is full\n");
    else out[++topout] = ele;
}

void displayout()
{
    printf("\noutput stack is :");
    for(int i=0 ; i<=topout ; i++)  printf("| %c ", out[i]);
}
```

Output :



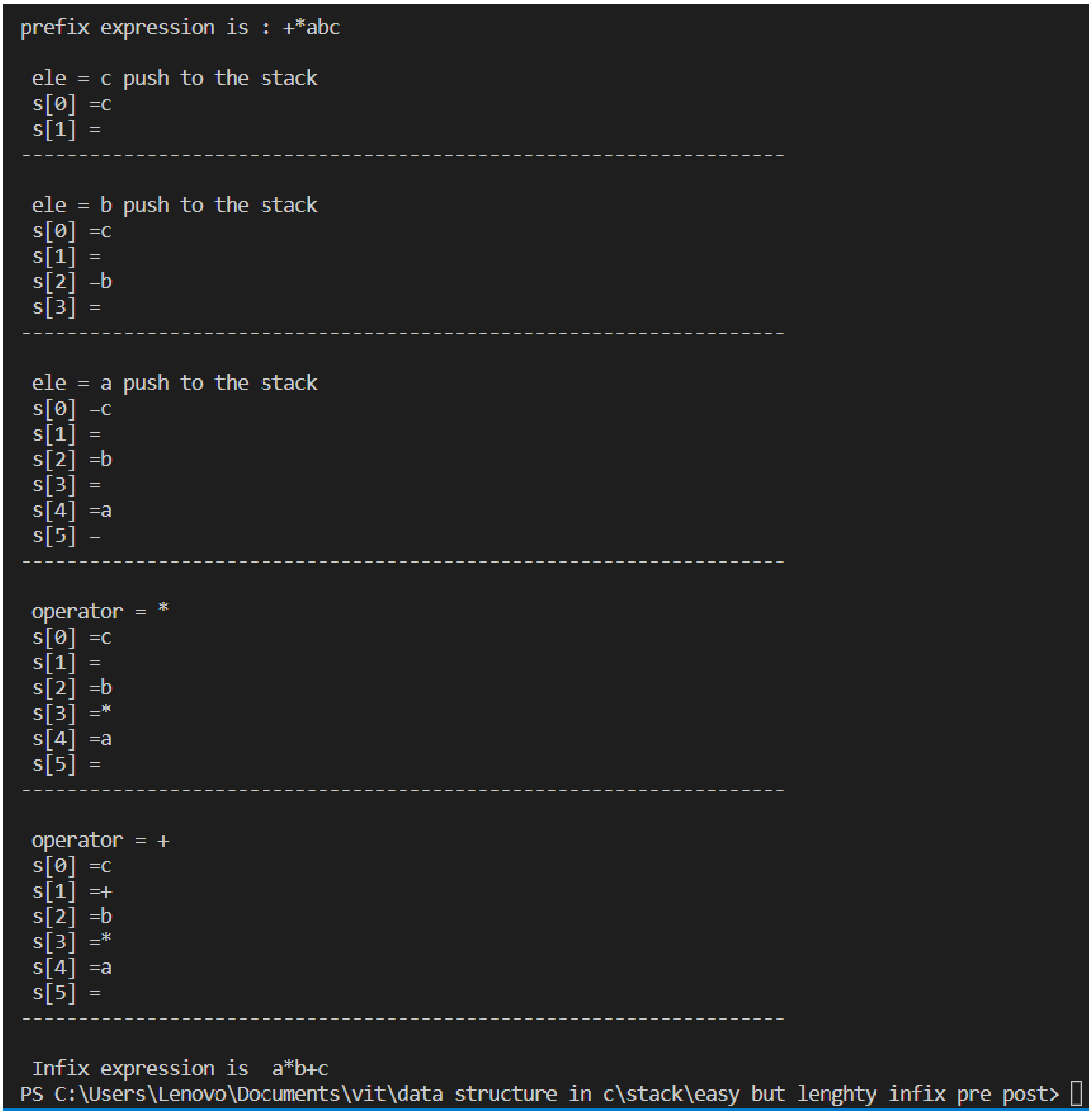
```
PS C:\Users\Lenovo\Documents\vit\data structure in c\stack\easy but lenghty infix pre post> cd "c:\Users\Leno
lenghty infix pre post\" ; if ($?) { gcc infixt_postfix.c -o infixt_postfix } ; if ($?) { .\infixt_postfix }
enter infix expression :
a+b-c+(d*e)
infix expression is : a+b-c+(d*e)

operator stack is :
output stack is :| a
operator stack is :| +
output stack is :| a
operator stack is :| +
output stack is :| a | b
operator stack is :| -
output stack is :| a | b | +
operator stack is :| -
output stack is :| a | b | + | c
operator stack is :| +
output stack is :| a | b | + | c | -
operator stack is :| + | (
output stack is :| a | b | + | c | -
operator stack is :| + | (
output stack is :| a | b | + | c | - | d
operator stack is :| + | ( | *
output stack is :| a | b | + | c | - | d
operator stack is :| + | ( | *
output stack is :| a | b | + | c | - | d | e
operator stack is :| +
output stack is :| a | b | + | c | - | d | e | *
 Postfix expression is : ab+c-de*+
PS C:\Users\Lenovo\Documents\vit\data structure in c\stack\easy but lenghty infix pre post> 
```

c. Prefix to Infix

Program :

```c
#include<stdio.h>
#include<string.h>
#include<ctype.h>

char stack[25] = {'\0'};
int top = -1;
char ch;

void push(char);
char pop();
void display();

void main()
{
   char prefix[25] = {'\0'} , ele;
   int j ;

   printf("enter prefix expression : \n");
   scanf("%s",prefix);
   printf("prefix expression is : %s\n" , prefix);

   int i = strlen(prefix)-1;

   while(i>=0)
   {
      ele = prefix[i];
       if((isalnum(ele)))
         {
            push(ele);
            push(' ');
            printf("\n ele = %c push to the stack",ele);
         }
         else
         {
             printf("\n operator = %c" , ele);
             for(j= top-1 ; j>=0 ; j--)
             {
                if(stack[j]==' ')
                {
                    stack[j] = ele;
                    break;
                }
             }
         }
    display();
    printf("\n-------------------------------------------------------------------
\n");

      i--;
   }

    printf("\n Infix expression is %s ", strrev(stack));
}


void push(char ele)
{
    if(top == 24) printf("stack is overflow\n");
    else stack[++top] = ele;
}

char pop()
{
    if(top == -1) printf("stack is empty\n");
    else return stack[top--];
}

void display()
{
    int i;
    for(int i=0 ; i<=top ; i++) printf("\n s[%d] =%c", i, stack[i]);
}
```

Output :



```
prefix expression is : +*abc

 ele = c push to the stack
 s[0] =c
 s[1] =
-------------------------------------------------------------------

 ele = b push to the stack
 s[0] =c
 s[1] =
 s[2] =b
 s[3] =
-------------------------------------------------------------------

 ele = a push to the stack
 s[0] =c
 s[1] =
 s[2] =b
 s[3] =
 s[4] =a
 s[5] =
-------------------------------------------------------------------

 operator = *
 s[0] =c
 s[1] =
 s[2] =b
 s[3] =*
 s[4] =a
 s[5] =
-------------------------------------------------------------------

 operator = +
 s[0] =c
 s[1] =+
 s[2] =b
 s[3] =*
 s[4] =a
 s[5] =
-------------------------------------------------------------------

 Infix expression is  a*b+c
PS C:\Users\Lenovo\Documents\vit\data structure in c\stack\easy but lenghty infix pre post>
```

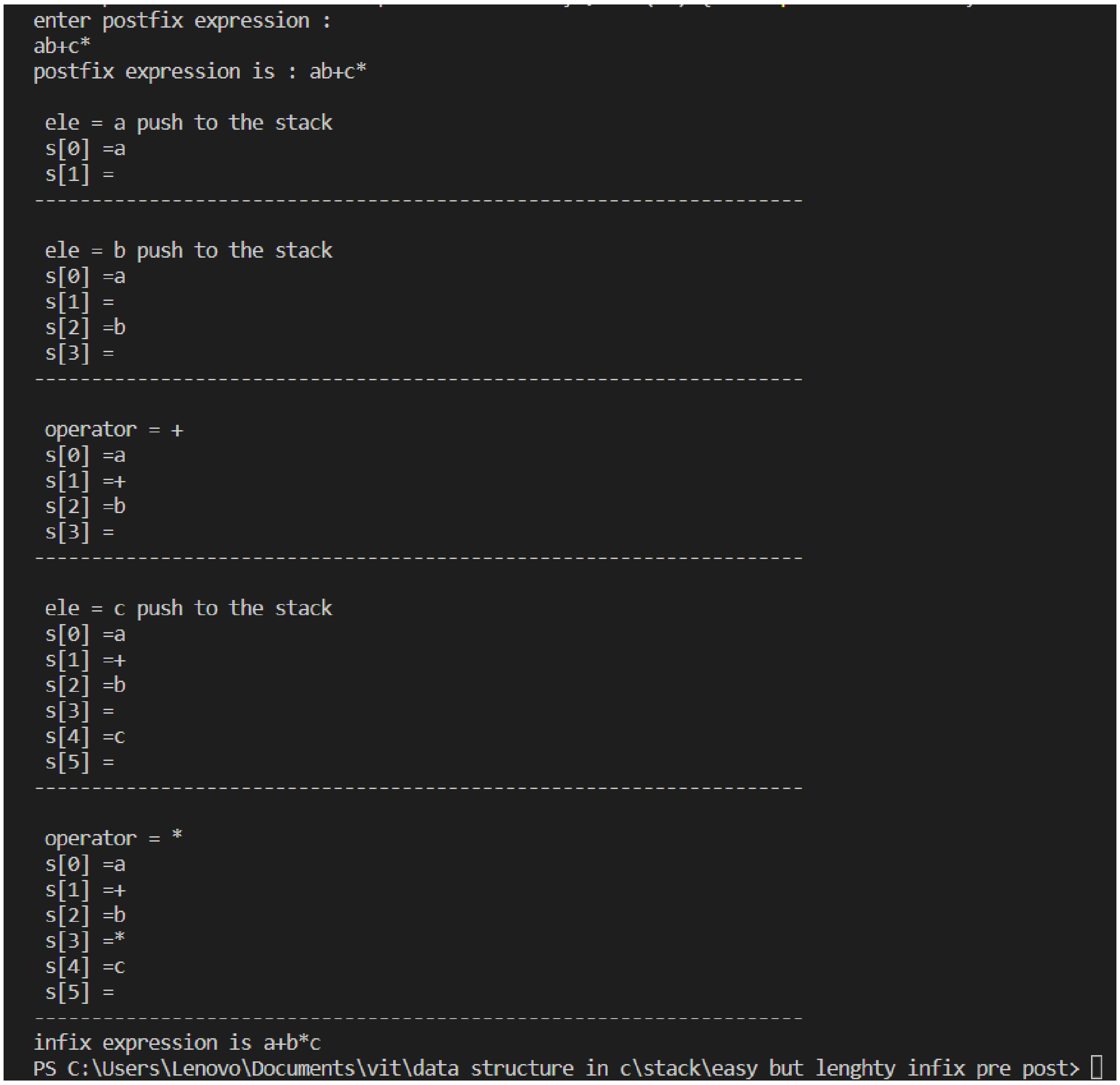
d. Postfix to Infix

Program :

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
```

```c
void push(char);
char pop();
void display();

char stack[25]={'\0'};
int top = -1;
char ch;

int main()
{
    char postfix[25] = {'\0'} , ele;
    int i=0 , j ;

   printf("enter postfix expression : \n");
   scanf("%s",postfix);
   printf("postfix expression is : %s\n" , postfix);

    while(i < strlen(postfix))
    {
         ele = postfix[i];
         if((isalnum(ele)))
         {
            push(ele);
            push(' ');
            printf("\n ele = %c push to the stack",ele);
         }
         else
         {
             printf("\n operator = %c" , ele);
             for(j= top-1 ; j>=0 ; j--)
             {
                if(stack[j]==' ')
                {
                    stack[j] = ele;
                    break;
                }
             }
         }
         display();
         printf("\n-------------------------------------------------------------
-----\n");
         i++;
    }
    printf("infix expression is %s", stack);
}

void push(char ele)
{
    if(top == 24) printf("stack is overflow\n");
    else stack[++top] = ele;
}

char pop()
{
    if(top == -1) printf("stack is empty\n");
    else return stack[top--];
}

void display()
{
    int i;
    for(int i=0 ; i<=top ; i++) printf("\n s[%d] =%c", i, stack[i]);
}
```

Output :



```
enter postfix expression :
ab+c*
postfix expression is : ab+c*

 ele = a push to the stack
 s[0] =a
 s[1] =
-------------------------------------------------------------------

 ele = b push to the stack
 s[0] =a
 s[1] =
 s[2] =b
 s[3] =
-------------------------------------------------------------------

 operator = +
 s[0] =a
 s[1] =+
 s[2] =b
 s[3] =
-------------------------------------------------------------------

 ele = c push to the stack
 s[0] =a
 s[1] =+
 s[2] =b
 s[3] =
 s[4] =c
 s[5] =
-------------------------------------------------------------------

 operator = *
 s[0] =a
 s[1] =+
 s[2] =b
 s[3] =*
 s[4] =c
 s[5] =
-------------------------------------------------------------------
infix expression is a+b*c
PS C:\Users\Lenovo\Documents\vit\data structure in c\stack\easy but lenghty infix pre post>
```