24/10/2020

**Experiment No:7**

**INFIX TO POSTFIX AND EVALUATION**

**AIM:**

Write a program to convert a given infix expression to its postfix expression and evaluate it.

**DATA STRUCTURES USED:**

Stack

**ALGORITHM:**

Algorithm INFIX\_TO\_POSTFIX()

START

1. TOP = -1, push(‘(‘)

2. While TOP > -1 do

3. ITEM = Readsymbol()

4. X = pop()

5. Case : ITEM = Operand

6. push(X)

7. print ITEM

8. Case : ITEM = ‘)’

9. While X != ‘(’

10. print X

11. X = pop()

12. EndWhile

13. Case : ISP(X) >= ICP(ITEM)

14. While ISP(X) >= ICP(ITEM) do

15. print X

16. X = pop()

17. EndWhile

18. push(X)

19. push(ITEM)

20. Case : ISP(X) < ICP(ITEM)

21. push(X)

22. push(ITEM)

23. Otherwise :

24. Print “Invalid Expression”

25. EndWhile

STOP

Algorithm POSTFIX\_CONVERSION()

START

1. While (TOP >= -1) do

2. ITEM = Readsymbol()

3. Case : ITEM = Operand

4. push(ITEM)

5. Case : ITEM = Operator

6. x2 = pop()

7. x1 = pop()

8. x = Operation(x1, x2, ITEM)

9. push(x)

10. Otherwise :

11. Print “Invalid Expression”

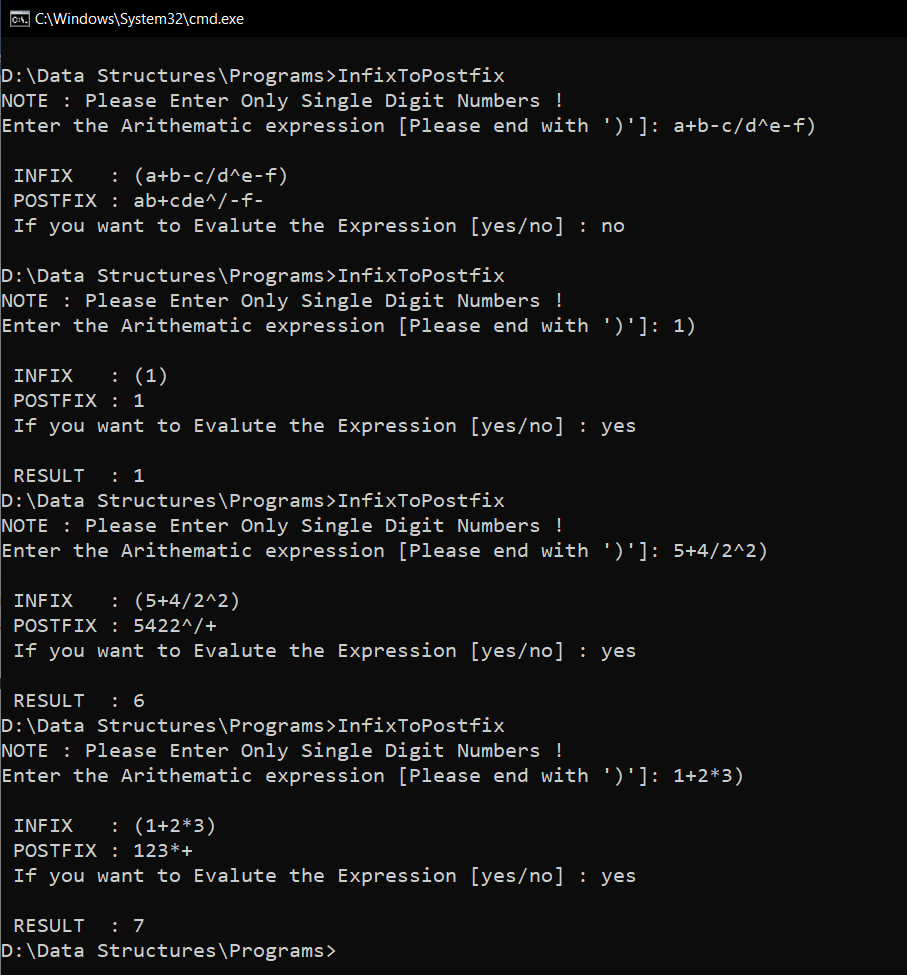
12. EndWhile

STOP

**PROGRAM:**

#include<stdio.h>  
#include<ctype.h>  
#include<math.h>  
#include<string.h>  
  
char stack[50];  
int top = -1;  
void push(char x){  
 stack[++top] = x;  
}  
  
char pop(){  
 if(top == -1)  
 return -1;  
 else  
 return stack[top--];  
}  
  
int Istack[50];  
int Itop = -1;  
  
void Ipush(int x){  
 Istack[++Itop] = x;  
}  
  
int Ipop(){  
 if(Itop == -1)  
 return 0;  
 else  
 return Istack[Itop--];  
}  
  
int ISP(char y){  
 if(y == '(')  
 return 0;  
 if(y == '+' || y == '-')  
 return 1;  
 if(y == '\*' || y == '/')  
 return 4;  
 if(y =='^')  
 return 5;  
 return 0;  
}  
int ICP(char y){  
 if(y == '(')  
 return 0;  
 if(y == '+' || y == '-')  
 return 1;  
 if(y == '\*' || y == '/')  
 return 3;  
 if(y =='^')  
 return 6;  
 return 0;  
}  
  
void main()  
{  
 char input[100];  
 char postfix[100];  
 char \*p,\*t,x;  
 char ans[5]="no";  
 // INFIX TO POSTFIX CONVERSION  
  
 printf("NOTE : Please Enter Only Single Digit Numbers !\n");  
 printf("Enter the Arithematic expression [Please end with \')\']: ");  
 scanf("%s",input);  
 printf("\n");  
 p=input;  
 t=postfix;  
 push('(');  
 printf(" INFIX : (%s",input);  
 printf("\n POSTFIX : ");  
 while(top!=-1){  
 if(isalnum(\*p)){  
  
 printf("%c",\*p);  
 \*t=\*p;  
 t++;  
  
 }else{  
 x=pop();  
 if(\*p == '('){  
  
 push(\*p);  
  
 }else if(\*p == ')'){  
  
 while( x!= '('){  
 printf("%c",x);  
 \*t=x;  
 t++;  
 x=pop();  
 }  
  
 }else if(ISP(x)>= ICP(\*p)){  
  
 while(ISP(x)>=ICP(\*p)){  
 printf("%c",x);  
 \*t=x;  
 t++;  
 x=pop();  
 }  
 push(x);  
 push(\*p);  
  
 }else if(ISP(x) < ICP(\*p)){  
 push(x);  
 push(\*p);  
  
 }else{  
  
 printf("Invalid Expression");  
  
 }  
 }  
  
 p++;  
  
 }  
  
  
 \*t='\0';  
 printf("\n");  
  
  
 // POSTFIX EVALUATION  
 printf(" If you want to Evalute the Expression [yes/no] : ");  
 scanf("%s",ans);  
 if(strcmp(ans,"yes")==0){  
 t = postfix;  
 int a,b,c;  
 while(\*t!='\0'){  
 if(isdigit(\*t)){  
 Ipush(\*t-48);  
 }else{  
 b= Ipop();  
 a= Ipop();  
 switch(\*t)  
 {  
 case '+': c=a+b;  
  
 break;  
  
  
 case '-': c=a-b;  
  
 break;  
  
 case '\*': c=a\*b;  
  
 break;  
  
 case '/': c=a/b;  
  
 break;  
  
 case '^': c=pow(a,b);  
  
 break;  
 }  
  
 Ipush(c);  
 }  
 t++;  
 }  
 int result=Ipop();  
 printf("\n RESULT : %d",result);  
 }  
}

**OUTPUT:**



**RESULT:**

Given infix expression is converted to postfix form and then the result of the

expression is displayed.

Time complexity for infix to postfix conversion = O(n)

Time complexity for postfix evaluation = O(n)