**Godavari College of Engineering, Jalgoan**

**Subject Name:** Data Structure. **Teacher Name:** Prof. S.S.Shete

**Practical No. : 3**  **Date:**

**Class:** S.E **Roll No:**

**Title:** Write a program to convert a given infix expression to postfix form using stacks.

**Aim:** To convert a given infix expression to postfix form using stacks.

**Theory:**

One of the applications of Stack is in the conversion of arithmetic expressions in high-level programming languages into machine readable form. As our computer system can only understand and work on a binary language, it assumes that an arithmetic operation can take place in two operands only e.g., A+B, C\*D, D/A etc. But in our usual form an arithmetic expression may consist of more than one operator and two operands e.g. (A+B)\*C(D/(J+D)). These complex arithmetic operations can be converted into polish notation using stacks which then can be executed in two operands and an operator form.

## Infix Expression:-

It follows the scheme of <operand><operator><operand> i.e. an <operator> is preceded and succeeded by an <operand>. Such an expression is termed infix expression. E.g., A+B

## Postfix Expression:-

It follows the scheme of <operand><operand><operator> i.e. an <operator> is succeeded by both the <operand>. E.g., AB+

**Algorithm** :-

**1.** Scan the infix expression from left to right.  
**2.** If the scanned character is an operand, output it.  
**3.**Else,  
a) If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a ‘(‘ ), push it.  
b) 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. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)  
**4.** If the scanned character is an ‘(‘, push it to the stack.  
**5.** If the scanned character is an ‘)’, pop the stack and and output it until a ‘(‘ is encountered, and discard both the parenthesis.  
**6.** Repeat steps 2-6 until infix expression is scanned.  
**7.** Print the output  
**8.**Pop and output from the stack until it is not empty.

**Program:**

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

#define SIZE 100

char stack[SIZE];

int top = -1;

void push(char item)

{

if(top >= SIZE-1)

printf("\nStack Overflow.");

}

else

{

top = top+1;

stack[top] = item;

}

}

char pop()

{

char item ;

if(top <0)

{

printf("stack under flow: invalid infix expression");

getchar();

exit(1);

}

else

{

item = stack[top];

top = top-1;

return(item);

}

}

int is\_operator(char symbol)

{

if(symbol == '^' || symbol == '\*' || symbol == '/' || symbol == '+' || symbol =='-')

{

return 1;

}

else

{

return 0;

}

}

int precedence(char symbol)

{

if(symbol == '^')

{

return(3);

}

else if(symbol == '\*' || symbol == '/')

{

return(2);

}

else if(symbol == '+' || symbol == '-')

{

return(1);

}

else

{

return(0);

}

}

void InfixToPostfix(char infix\_exp[], char postfix\_exp[])

{

int i, j;

char item;

char x;

push('(');

strcat(infix\_exp,")");

i=0;

j=0;

item=infix\_exp[i];

while(item != '\0')

{

if(item == '(')

{

push(item);

}

else if( isdigit(item) || isalpha(item))

{

postfix\_exp[j] = item;

j++;

}

else if(is\_operator(item) == 1)

{

x=pop();

while(is\_operator(x) == 1 && precedence(x)>= precedence(item))

{

postfix\_exp[j] = x;

j++;

x = pop();

}

push(x);

push(item);

}

else if(item == ')')

{

x = pop();

while(x != '(')

{

postfix\_exp[j] = x;

j++;

x = pop();

}

}

else

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

i++;

item = infix\_exp[i];

}

if(top>0)

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

if(top>0)

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

postfix\_exp[j] = '\0';

}

int main()

{

char infix[SIZE], postfix[SIZE];

printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");

printf("\nEnter Infix expression : ");

gets(infix);

InfixToPostfix(infix,postfix);

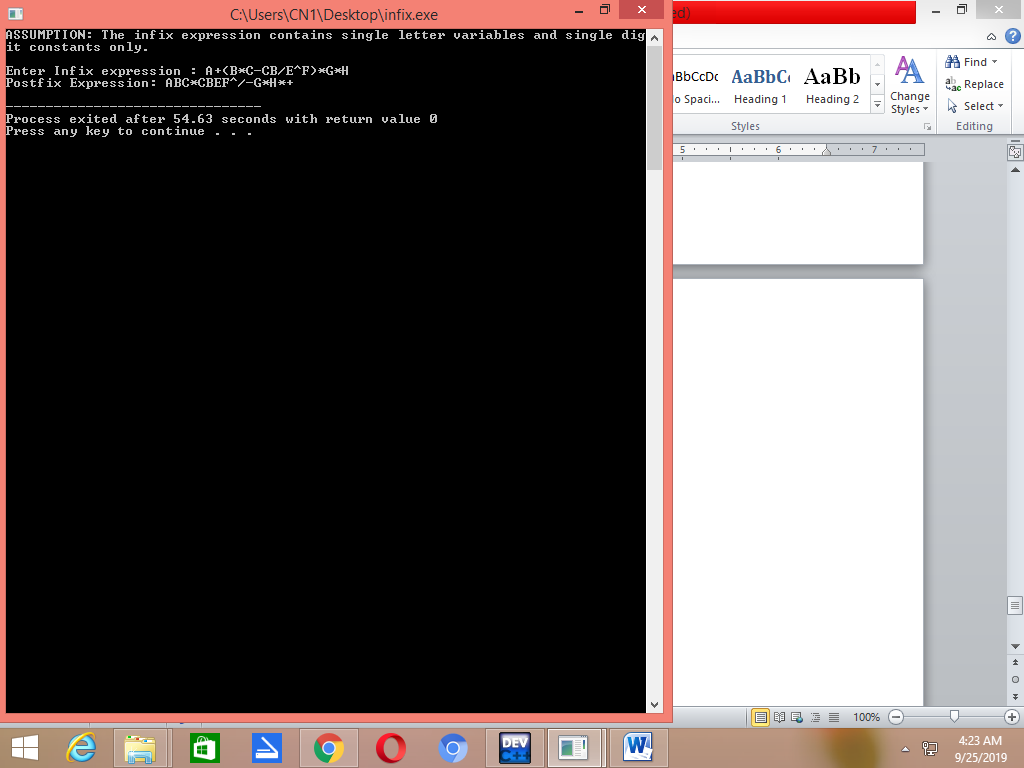
printf("Postfix Expression: ");

puts(postfix);

return 0;

}

**OUTPUT:**



**Conclusion:-**