AIM:- Implement stack for expression conversion (infix to postfix)

OBJECTIVE:-1) Understand the Stack Data Structure and its basic operators.

2) Understand the method of defining stack ADT and implement the basic operators.

3) Learn how to create objects from an ADT and invoke member functions.

THEORY:-

In high level programming languages, we use arithmetic expression in its infix form. An expression in infix form contains operators in between operands on which it operates. Parentheses also appear in infix expressions to specify the order of evaluation. During compilation, the compiler converts the infix expression to postfix for easy evaluation, since a postfix expression does not contain any parenthesis. Also, a postfix expression can be evaluated easily by using a stack. Postfix notation has the following virtues: No parenthesis. The priority of the operations is no longer relevant. Enables easy evaluation (evaluated by making a left to right scan, stacking the operands.)

Algorithm:-

1. Scan the infix expression from left to right.

2. If the scanned character is an operand, output it.

3. Else,

…..3.1 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.

…..3.2 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.

Source Code:

#define SIZE 50

#include<iostream>

#include <ctype.h>

using namespace std;

char s[SIZE];

int top=-1; //Global declarations

char push(char elem)

{ //Push onto stack

s[++top]=elem;

}

char pop()

{ //Pop from stack

return(s[top--]);

}

int pr(char elem) //Check operator priority

{

switch(elem)

{

case '#': return 0;

case '(': return 1;

case '+':

case '-': return 2;

case '\*':

case '/': return 3;

}

}

int main()

{

char infix[50],postfix[50],ch,elem;

int i=0,k=0;

printf("\n\nEnter Infix Expression : ");

cin>>infix;

push('#');

while( (ch=infix[i++]) != '\0')

{

if( ch == '(')

push(ch);

else

if(isalnum(ch))

postfix[k++]=ch;

else

if( ch == ')')

{

while( s[top] != '(')

postfix[k++]=pop();

elem=pop(); // Remove (

}

else

{ //Operator priority check

while( pr(s[top]) >= pr(ch) )

postfix[k++]=pop();

push(ch);

}

}

while( s[top] != '#') //Pop till stack empty

postfix[k++]=pop();

postfix[k]='\0'; //Last elem of char sting null for making it a valid string

cout<<"\nPostfix Expression = "<<postfix<<"\n";

return 0;

}

