**Assignment 8**

**Objective:**

Write a program to converts the given arithmetic expression into post-fix notation and evaluate it.

**Code:**

using System;

usingSystem.Collections.Generic;

usingSystem.Linq;

usingSystem.Text;

usingSystem.Threading.Tasks;

namespace ToCS\_Assignment\_8

{

classProgram

{

//This function specify the precendence of operators

publicstaticintPrecedence(charchr)

{

if (chr == '^') //Power have highest precedence

return 3;

elseif (chr == '\*' || chr == '/')

return 2;

elseif (chr == '+' || chr == '-')

return 1;

else

return -1;

}

//This function converts the infix expression into postfix expression

publicstaticstringInfix\_To\_Postfix()

{

charchr; //chr holds the i'th character of string/expression

intbrace\_conut = 0; //This counts the occurance of '(' & ')', if it is //even the expression is 'valid' else 'invalid'

intoperator\_count = 0; //This checks that no two operators come togather like (6-+2)

string postfix = "", expression;

Stack<char> stack = new Stack<char>(); //Initialize the stack

Console.Write("Enter an mathematical expression : ");

expression = Console.ReadLine(); //Get the mathematical expression from user

expression = "(" + expression + ")"; //here we enclose the expression braces

for (inti = 0; i<expression.Length; i++)//This loop run till length of string/expression

{

chr = expression[i]; //i'th character of string is assigned to 'chr '

if (char.IsLetterOrDigit(chr)) //If 'chr' is letter or digit put it into stack

{

postfix += chr;

operator\_count = 0; //This specify that i'thchr is not operator

}

elseif (chr == '(') //Push if chr is '(', we are pushing this cause

{ //if in future chr= ')' appears

stack.Push(chr);

brace\_conut++; //We should know were to stop Poping from stack

operator\_count = 0;

}

elseif (chr == ')')//If chr = ')' we Pop the values from stack until '(' this appears

{

while (stack.Count> 0 &&stack.Peek() != '(')

postfix += stack.Pop(); //Here we add operators to the postfix expression

if (stack.Count> 0)

stack.Pop(); //This extra pop is for '(', cause we don't want '(' in postfix expression

brace\_conut++;

operator\_count = 0;

}

else

{

operator\_count++;

if (operator\_count == 2)

{

Console.WriteLine("\nInvalid Expression!!!\n");

postfix = "";

brace\_conut = 2;

break;

}

//If Precedence of chr is greater than the precendence of last added Operator then push

if (Precedence(chr) > Precedence(stack.Peek()))

stack.Push(chr);

//If Precedence of chr is less than the precendence of last added Operator then

else

{

//pop untill precedence of chr is greater than stack top operator/element

while (Precedence(chr) <= Precedence(stack.Peek()))

postfix += stack.Pop(); //Here we add operators to the postfix expression

stack.Push(chr);

}

}

}

if (brace\_conut % 2 != 0 &&brace\_conut!=0)

{

postfix = "";

Console.WriteLine("\nInvalid Expression!!!\n");

}

returnpostfix; //Return Postfix expression

}

//This function finds the value of postfix expression

publicstaticdoublePostfix\_To\_Value(string s)

{

//Here the logic is, we put the numbers in stack and when encounter with any operator

//Pop stack two times do respective Operation and push the result back into stack

charchr;

double result = 0, first\_Operand, second\_Operand=0;

Stack<string> stack = new Stack<string>();

for (inti = 0; i<s.Length; i++)

{

chr = s[i]; //Here chr contains the i'th character of string

if (char.IsLetterOrDigit(chr)) //if chr is letter of digit we put this into stack

stack.Push(chr.ToString());

else

{

first\_Operand = double.Parse(stack.Pop()); //first\_Operand contains the first Poped

if (stack.Count>0)

second\_Operand = double.Parse(stack.Pop()); //second\_Operand contains the second Poped

if (chr == '+') //If Operator is + do addition

result = second\_Operand + first\_Operand;

elseif (chr == '-') //If Operator is - do subtraction

result = second\_Operand - first\_Operand;

elseif (chr == '\*') //If Operator is \* do multiplicatio

result = second\_Operand \* first\_Operand;

elseif (chr == '/') //If Operator is / do division

result = second\_Operand / first\_Operand;

elseif (chr == '^')

result = Math.Pow(second\_Operand, first\_Operand);

stack.Push(result.ToString()); //Put the result back in stack

}

}

returnresult; //Return result

}

staticvoid Main(string[] args)

{

string expression;

double result;

while (true)

{

expression = Infix\_To\_Postfix();

result = Postfix\_To\_Value(expression);

Console.WriteLine("Postfix : " + expression);

Console.WriteLine("Value : " + result);

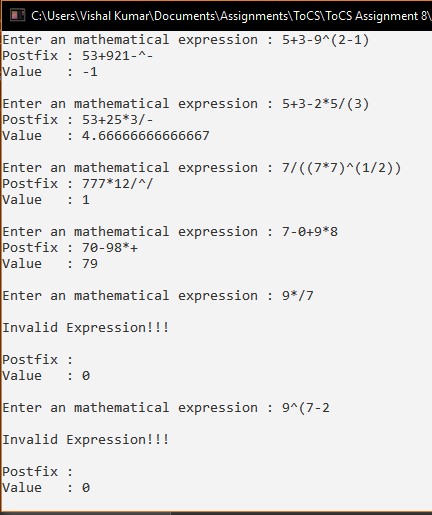
Console.ReadLine();

}

}

}

}

****