

**Project Report**

Course Code: **CSE207**

Course Title: **Data Structures**

Section: **1**

**Submitted To**

**Dr. Shamim H Ripon**

Associate Professor

Department of CSE

East West University

**Submitted By**

**Md. Ferdous**

**Submission date: August 4, 2019**

**Problem Statement**

**Postfix and Prefix Calculator**

You have been using one of classic calculator, the HP-35, for a long time. It was the first handheld calculator manufactured by Hewlett Packard in 1972. However, after a disastrous accident (dropped it in a sink), it is no longer functional. You miss this calculator so much You finally decided to implement its special form of postfix calculation yourself. For constructing the postfix calculator, you have to do the following:

1. Use stack for converting infix expression to postfix and prefix expression.
2. Use stack for calculating value of postfix expression.

In this project, you have to do the following:

* 1. Take infix expression as input.
  2. Convert postfix and prefix expression.
  3. Evaluate value of postfix and prefix expression.

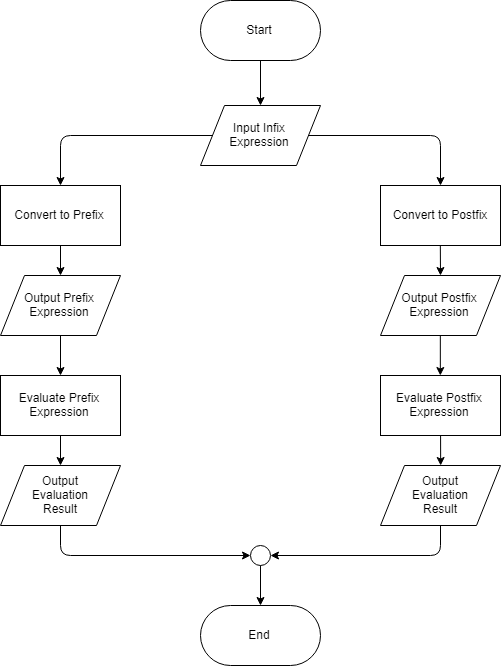
**Solve**

In this project, we did the following:

1. Take any infix expression as input.
2. Convert postfix and prefix show expression.
3. Evaluate value of postfix and prefix conversion.

The purpose of this project is to make a calculator, which can convert any infix expression into prefix and postfix expression. Generally, infix expression is used to solve any mathematical problem, but a computer could not do it, it has to use prefix or postfix expression, which it does automatically. In this project, we did the same thing but by developing a program manually.

**Flow Chart:**



**System Design**

This program has four-featured function.

1. PREFIX
2. POSTFIX
3. EVALUATE\_PREFIX
4. EVALUATE\_POSTFIX

* **PREFIX function:**

This function is to convert the infix expression into prefix expression. After the execution of this function, if returns a string which is the required prefix expression.

**The followed algorithm for this function is given below:**

* Reverse the string, which is representing an infix expression receiving as parameter on this function.
* Initially set the precedence of the operator to zero.
* Use a loop to read the characters one by one from that string.

For each character do the following in the loop:

* When the character is an operand
  + Add it to the prefix string that is used to store the corresponding prefix expression.
* When the character is a left parenthesis “(”
  + Push the character to the stack of characters.
* When the character is a right parenthesis “)”
  + Repeatedly push a character to prefix and pop that character from stack until “(“ is encountered in stack. Then pop “(“ from stack.
  + Change the precedence.
* When the character is an operator
  + Check the precedence is higher than the operator precedence or not.
  + If yes, repeatedly add the top character of the stack to the prefix string that is used to store the corresponding prefix expression and change the precedence until the condition break.
  + Push the operator to the stack and change the precedence.
* End of loop.
* Repeatedly push a character to prefix and pop that character from stack until the stack is empty;
* Reverse the string and return that string.
* **POSRFIX function:**

This function is to convert the infix expression into postfix expression. After the execution of this function, if returns a string which is the required postfix expression.

**The followed algorithm for this function is given below:**

* Initially set the precedence of the operator to zero.
* Use a loop to read the characters one by one from the string, which is representing an infix expression receiving as parameter on this function.

For each character do the following in the loop:

* When the character is an operand
  + Add it to the postfix string that is used to store the corresponding postfix expression.
* When the character is a left parenthesis “(”
  + Push the character to the stack of characters.
* When the character is a right parenthesis “)”
  + Repeatedly push a character to postfix and pop that character from stack until “(“ is encountered in stack. Then pop “(“ from stack.
  + Change the precedence.
* When the character is an operator
  + Check the precedence is equal or higher than the operator precedence or not.
  + If yes, repeatedly add the top character of the stack to the postfix string that is used to store the corresponding postfix expression and change the precedence until the condition break.
  + Push the operator to the stack and change the precedence.
* End of loop.
* Repeatedly push a character to postfix and pop that character from stack until the stack is empty.
* Return the string.
* **EVALUATE\_PREFIX function:**

This function is to evaluate the prefix expression. After the execution of this function, if returns a value which is the evaluated value of the prefix expression.

**The followed algorithm for this function is given below:**

* Use a loop to read the characters one by one from the last of the string, which is representing a prefix expression receiving as parameter on this function.

For each character do the following in the loop:

* When the character is an operand
* Push it to the stack.
* When the character is an operator
* Pop two values from the stack and make operation of the operator from first value to second value.
* Push the result to that stack.
* End of loop.
* Return the last value of the stack.
* **EVALUATE\_POSTFIX function:**

This function is to evaluate the postfix expression. After the execution of this function, if returns a value which is the evaluated value of the postfix expression.

**The followed algorithm for this function is given below:**

* Use a loop to read the characters one by one from the string, which is representing a postfix expression receiving as parameter on this function.

For each character do the following in the loop:

* When the character is an operand
* Push it to the stack.
* When the character is an operator
* Pop two values from the stack and make operation of the operator from second value to first value.
* Push the result to that stack.
* End of loop.
* Return the last value of the stack.

**NB: In the main function, we have all function of above and showed the returned values from these function.**

**Sample Input and Output:**

* **Input:**

**9+8-(6/3)-4+(4^2-1)**

* **Output:**

**The PREFIX Expression is: +--+98/634-^421**

**The POSTFIX Expression is: 98+63/-4-42^1-+**

**The PREFIX Evaluation result is: 26**

**The POSTFIX Evaluation result is: 26**

**Limitations and Future Scope:**

This program has some major limitations, such as:

1. It only works for one digit numbers (0 – 9), it can’t work using two or more digit numbers.
2. It is a very blunt program as there is no uses of various and enjoyable graphics.

Although it has some limitations, this project can be modified by:

1. Using a higher level language
2. Using a graphical user interface
3. Using more methods than basic arithmetic methods.