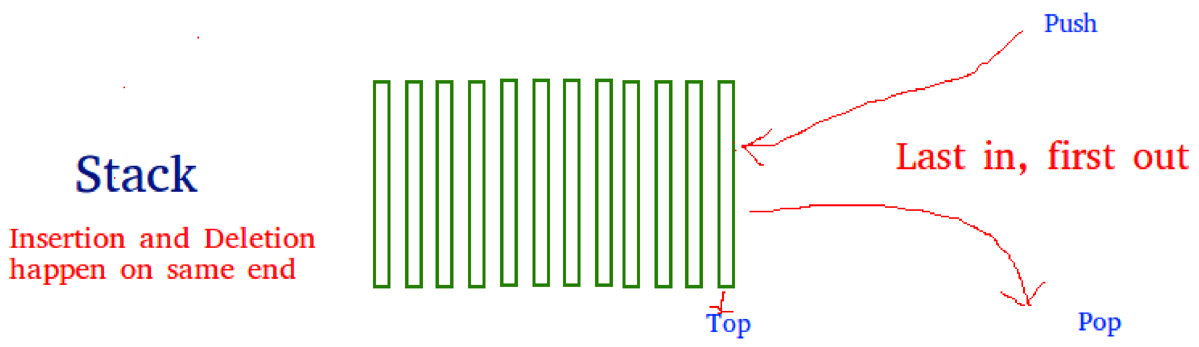
Assignment no	**10**
Aim	**Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions: 1. Operands and operator, both must be single character. 2. Input Postfix expression must be in a desired format. 3. Only '+', '-', '*' and '/ ' operators are expected.**
Objective	To understand the stack data structure and its use in real-time applications To understand , expression conversion as infix to postfix and its evaluation using stack
Outcome	To implement primitive operations on stack. To make use of stack for converting infix expression to postfix.
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Eclipse with C++ plugin

Theory:-
Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).



There are many real-life examples of a stack. Consider an example of plates stacked over one another in the canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO(Last In First Out)/FILO(First In Last Out) order.

Consider a simple expression: A + B .This notation is called Infix Notation.

A + B in Postfix notation is A B +

As you might have noticed, in this form the operator is placed after the operands (Hence the name 'post'). Postfix is also known as Reverse Polish Notation. Similarly for Infix, the operator is placed inside the operands. Likewise the equivalent expression in prefix would be + A B, where the operator precedes the operands.

Examples of Infix, Prefix, and Postfix		
Infix Expression	**Prefix Expression**	**Postfix Expression**
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +

An Expression with Parentheses		
Infix Expression	**Prefix Expression**	**Postfix Expression**
(A + B) * C	* + A B C	A B + C *

ADT:

```
class Stack
{
        finite ordered list with 0 or more elements.
        int top;
public:
        Stack();//create an empty stack and initializes top;
        void push( ele);//to insert element into stack;
        void pop();//to remove element from stack;
        int top();//returns value of top
        Isempty();//check wheather stack is empty or not
        Isfull();
}
```

Algorithm:-

```
1. Scan infix expression Q from left to right and repeat step 2 to 5 for each element of Q until the STACK is empty.
2. If an operand is encountered add it to P(answer).
3. If a left parenthesis is encountered push it onto the STACK.
4. If an operator is encountered, then
    • Repeatedly pop from STACK and add to P each operator which has same precedence as or higher precedence than the operator encountered.
    • Push the encountered operator onto the STACK.
5. If a right parenthesis is encountered, then
    • Repeatedly pop from the STACK and add to P each operator until a left parenthesis is encountered.
    • Remove the left parenthesis; do not add it to P.
6. Exit
```

```
  for (each character ch in the infix expression){
    switch(ch){
      case operand:     // append operand to end of PE
        postfixExp = postfixExp + ch
        break
      case '(':         // save '(' on stack
        aStack.push(ch)
        break
      case ')':         // pop stack until matching '('
        while (top of stack is not '('){
          postfixExp = postfixExp + (top of aStack)
          aStack.pop()
        }  // end while
        aStack.pop()    // remove the '('
        break
      case operator:    // process stack operators of
                        // greater precedence
        while (!aStack.isEmpty() and
               top of stack is not '(' and
               precedence(ch) <= precedence(top of aStack)){
          postfixExp = postfixExp + (top of aStack)
          aStack.pop()
        } // end while
        aStack.push(ch)    // save new operator
        break
    } // end switch
  } // end for
 // append to postfixExp the operators remaining on the stack
 while(!aStack.isEmpty()){
   postfixExp = postfixExp + (top of aStack)
   aStack.pop()
 } // end while
```

Evaluating Postfix expression:

Start with an empty stack.
We scan P from left to right.
While (we have not reached the end of P)
If an operand is found push it onto the stack End-If
If an operator is found
    Pop the stack and call the value A
    Pop the stack and call the value B
    Evaluate B op A using the operator just found.
    Push the resulting value onto the stack
• End-If
End-While Pop the stack (this is the final value)
Complexity:
    Time complexity = O(n).

Conclusion:

Implemented C++ program for expression conversion as infix to postfix and its evaluation using stack

Review Questions:

1. What is Stack and where it can be used?
2. What are Infix, prefix, Postfix notations?
3. Why Are Stacks Useful?
4. Explain what are Infix, Prefix and Postfix Expressions?
5. Why do we need Prefix and Postfix notations?