**Prefix Expression Calculator**

The **prefix expression**calculator evaluates the prefix string data using stacks,  it first checks if the expression is prefix before evaluating the left most character, if the first charactor matches any operator +,-,\*,/ then it returns true  meaning our prefix expression is valid . Once the prefix passes all necessary validations. we first reverse the prefix string and make it a postfix expression  then iterate all the charactors in the prefix expression  looking for appropriate operators and operands.

If we find a match of an operand we push it into our stack

   if(isdigit(PreExpression[i])==true)

           {

               char c = PreExpression[i];

               //convert any char to integer type

               int n = c-'0';

               DataStack.push(n);

           }

and if we find an operator we evaluate the values we stored into our stack by popping(pushing the value to the top of the stack) them and assigning them into necessary variables and applying the returned operator to the variables then asign the the results of our operation to our variable and push the  variable to the stack .

**Note:**Remember stacks use **LIFO(last in first out)**which means that the value we push last in our stack will be the first one to be retrieved when we pop our values

 if(PreExpression[i]=='+')

               {

                   int x = DataStack.top();

                   DataStack.pop();

                   int y = DataStack.top();

                   DataStack.pop();

                   result = x+y;

           DataStack.push(result);

               }

We repeat this process until we're done reading  all the charactors from our expression.

  for( int i = 0; i < nLength; i++)

       {.........................

Then when we're done our top most value from our stack will be our results

     cout<<"Final evaluation:"<<DataStack.top()<<endl;

 Array based stack performs better with our prefixExpression evaluation compared to our linked list based stack. One point is because we can access any element in an array without going through every item in the list