Min Stack (LeetCode 155) - Detailed Explanation

Problem Statement:

Design a stack that supports the following operations in O(1) time complexity:

- push(int val): Pushes the element val onto the stack.

- pop(): Removes the element on the top of the stack.

- top(): Gets the top element of the stack.

- getMin(): Retrieves the minimum element in the stack.

Optimized Approach Using Mathematics:

Instead of using an extra stack to keep track of the minimum element, we can use a single stack with some mathematical tricks.

Data Members:

1. stack<long long> st; (To store elements)

2. long long min; (To store the minimum element at any point)

Logic Behind the Approach:

- We maintain a 'min' variable that always holds the current minimum.

- If we push a value greater than the current 'min', we push it normally.

- If we push a value less than or equal to 'min', we do the following:

** - Store '2*x - min' instead of 'x' to encode the previous 'min' inside the stack.**

- Update 'min = x'.

- When popping:

- If 'st.top() < min', it means the popped value was encoded, so we retrieve the previous 'min' using the formula 'min = 2*min - st.top()'.

Code Implementation:

```
class MinStack {
  public:
      stack<long long> st;
      long long min;
      MinStack() {
          min = LLONG_MAX;
      }

      void push(int val) {
         long long x = (long long)val;
         if(st.size()==0){
          st.push(x);
          min = x;
         }
         else if(x>min){
          st.push(x);
         }
         else{
          st.push(2*x-min);
          min = x;
         }
      }

      void pop() {
         if(st.top()<min){
          min = 2*min - st.top();
         }
         st.pop();
      }

      int top() {
          if(st.top()<min) return (int)min;
          else return (int)st.top();
      }
```

```
    int getMin() {
        return (int)min;
    }
};
```

Test Cases:

Test Case 1:

Input:

MinStack obj;

obj.push(3);

obj.push(5);

cout << obj.getMin() << endl; // Output: 3

obj.push(2);

obj.push(1);

cout << obj.getMin() << endl; // Output: 1

obj.pop();

cout << obj.getMin() << endl; // Output: 2

Test Case 2:

Input:

MinStack obj;

obj.push(10);

obj.push(20);

obj.push(5);

obj.push(30);

cout << obj.getMin() << endl; // Output: 5

```
obj.pop();
cout << obj.getMin() << endl;  // Output: 5
obj.pop();
cout << obj.getMin() << endl;  // Output: 10
```

Time and Space Complexity Analysis:

- push(): O(1) (Constant time due to direct stack operations)
- pop(): O(1) (Constant time due to direct stack operations)
- top(): O(1) (Direct access to stack top)
- getMin(): O(1) (Direct return of min value)

Space Complexity: O(N) (In the worst case, we store all elements with some encoding but still within the same stack)