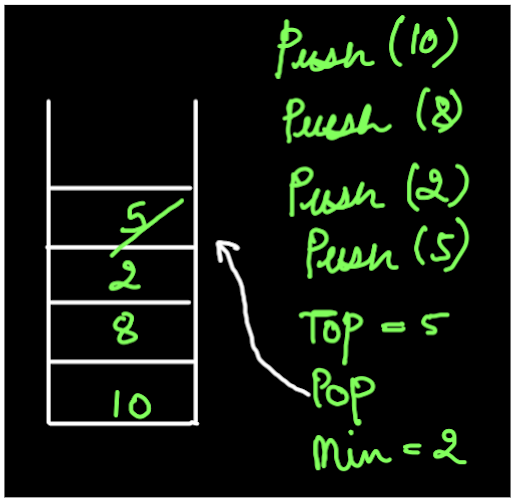
**1. Problem Discussion :**

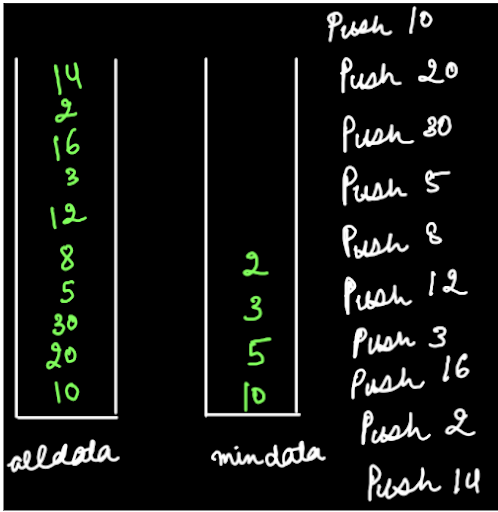
Here we are required to code a function min which should return the smallest element available in the stack and if not available return -1 after printing "Stack underflow" in constant time i.e. time complexity should be O( 1 ). The basic logic for the rest of the functions like pop( ), push( ), size( ),top( ) remains as it was with some minor modifications. Let's discuss it using an example in figure 1.

****

If we push 10, 8 ,2 and 5 then they are normally added in a LIFO manner . Then, calling the top function gives us the value 5 and the pop function deletes this value 5 from the stack. The min function returns 2 as it is the smallest of all the numbers in the stack. Let's strategize the solution for this problem.

**2. Approach :**

Reader, notice in the unfilled solution, we are given two data members of type Stack "allData" and "minData" .They will be used to implement our problem.

****

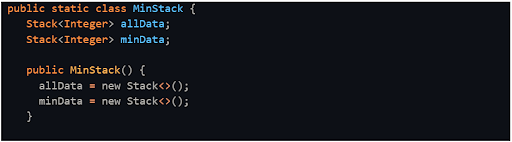
Let's see our main strategy. push( )- allData: We push the elements in the "allData" stack every time the push function is called. minData: We push the given element in the "minData" stack when either the "allData" stack is empty or the value of the given element is less than the peek value at "minData". In figure 2, when push 10 was called then both the stacks were empty therefore 10 was added to both the stacks. Then, 20 and 30 were only pushed in the "allData" stack because they don't fulfill the condition for "minData". After, when 5 was pushed , it was pushed in both the stacks as 5 is less than the peek of "minData" i.e. 5<10. Similarly the rest of the elements were pushed in the stacks using our logic. We suggest you watch the video "Minimum Stack-1" to see how this pushing works. Let's now have a look at what will be the working of the functions. top( ) -For this function we return the top of the "allData" stack since it has all the elements pushed into it. min( ) -For this function return the top value of the "minData" stack because only the elements which are smaller than the previous smallest values are stored in the stack. Hence the top of this stack will have the least value out of all the minimum values. pop( )- Everytime pop function is called we pop the top element from the "allData" stack. Also, if the top value of both "allData" and "minData" is the same then that element will be popped from the "minData" stack too because then it can't remain a minimum value if it doesn't exist.

**3. Code**

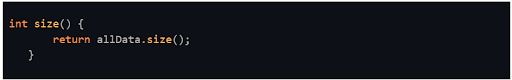
import java.io.\*; import java.util.\*; public class Main { public static class MinStack { Stack< Integer> allData; Stack< Integer> minData; public MinStack() { allData = new Stack< >(); minData = new Stack< >(); } int size() { return allData.size(); } void push(int val) { allData.push(val); if (minData.size() == 0 || val <= minData.peek()) { minData.push(val); } } int pop() { if (size() == 0) { System.out.println("Stack underflow"); return -1; } else { int val = allData.pop(); if (val == minData.peek()) { minData.pop(); } return val; } } int top() { if (size() == 0) { System.out.println("Stack underflow"); return -1; } else { return allData.peek(); } } int min() { if (size() == 0) { System.out.println("Stack underflow"); return -1; } else { return minData.peek(); } } } public static void main(String[] args) throws Exception { BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); MinStack st = new MinStack(); String str = br.readLine(); while (str.equals("quit") == false) { if (str.startsWith("push")) { int val = Integer.parseInt(str.split(" ")[1]); st.push(val); } else if (str.startsWith("pop")) { int val = st.pop(); if (val != -1) { System.out.println(val); } } else if (str.startsWith("top")) { int val = st.top(); if (val != -1) { System.out.println(val); } } else if (str.startsWith("size")) { System.out.println(st.size()); } else if (str.startsWith("min")) { int val = st.min(); if (val != -1) { System.out.println(val); } } str = br.readLine(); } } }

**4. Analysis:**

After looking at the code, let's analyse each function and code snippet individually.

****

The above code is already available to us. The other functions are coded using it. SIZE:

****

The size( ) function simply returns the size of all the elements i.e. the size of allData stack. PUSH:

****

We have already discussed the push function. The value is pushed in "allData" everytime and in "minData" only when it's empty or the given value is less than the peek of "minData". MIN:

****

If the stack is empty then we return -1 else the peek value of "minData" stack is returned. TOP:

****

Similar to the min function we just return the peek of the "allData" stack. POP:

****

We have already discussed this function in the strategy section. Refer to that.

Time Complexity :

O(1) According to the question, we were required to write all the functions in linear time which we have.

SPACE COMPLEXITY :

O(n) The space complexity is of order n because stacks of n spaces are used. Voila! You are finished with this question. In the next question we'll see a variation to this question only. So don't forget to check that out too. For now, Goodbye.