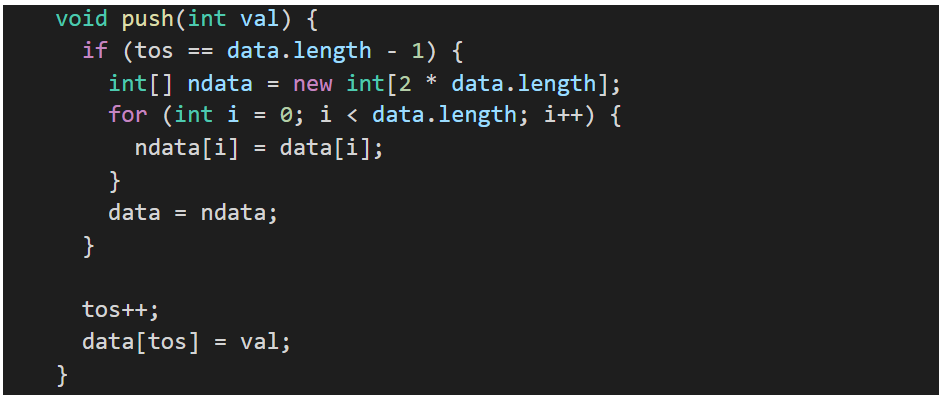
**1. Problem Discussion**

What is a Dynamic Stack? Dynamic stack is the one in which size of stack grows and shrinks as per requirement of push and pop functions. But here our only concern is for push function. We just need to make changes in this function as mentioned in the problem. You are required to change the body of the push function to accept the element even when the data array is completely full. In that case, you are required to reallocate the data array (to twice its size). You should not print "Stack overflow" ever. Your only job is to modify the above mentioned function as required. You can also check out our question video where our team has particularly explained the question and what needs to be done.

**2. Approach**

Push

In function, "push", as mentioned in the question that "Stack Overflow" should not be printed. For that whenever the data array is full, i.e. tos is pointing to the last index of data array (data.length - 1), you are required to reallocate the data array to twice its size. Now we run a for loop on the new data array and fill it exactly like the old one. Then change the pointer of the data array to a new data array. At last increment the tos by 1. And place new value at index tos of the new data array.

****

**3. Code**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static class CustomStack {

int[] data;

int tos;

public CustomStack(int cap) {

data = new int[cap];

tos = -1;

}

int size() {

return tos + 1;

}

void display() {

for (int i = tos; i >= 0; i--) {

System.out.print(data[i] + " ");

}

System.out.println();

}

void push(int val) {

if (tos == data.length - 1) {

int[] ndata = new int[2 \* data.length];

for (int i = 0; i < data.length; i++) {

ndata[i] = data[i];

}

data = ndata;

}

tos++;

data[tos] = val;

}

int pop() {

if (tos == -1) {

System.out.println("Stack underflow");

return -1;

} else {

int val = data[tos];

tos--;

return val;

}

}

int top() {

if (tos == -1) {

System.out.println("Stack underflow");

return -1;

} else {

return data[tos];

}

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

CustomStack st = new CustomStack(n);

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("display")) {

st.display();

}

str = br.readLine();

}

}

}

**4. Analysis**

Time Complexity: O(n)

When the stack becomes full, we double the size of the stack and copy all the elements, therefore the average time complexity becomes O(n).

Space Complexity: O(n)

When the stack will have space in that case the space complexity will be constant as the new element can be simply added to the same stack whereas if the stack is full then to add more elements, a new array of double the size of the previous stack will be defined making the space complexity 2n. And the average space complexity will become O(n).