Implementing two stacks using one array

The task is to create a data structure *twoStacks* that represents two stacks. Implementation of *twoStacks* should use only one array, i.e., both stacks should use the same array for storing elements. Following functions must be supported by *twoStacks*.

- push1(int x) --> pushes x to first stack.
- push2(int x) --> pushes x to second stack.
- pop1() --> pops an element from the first stack and return the popped element.
- pop2() --> pops an element from the second stack and return the popped element.

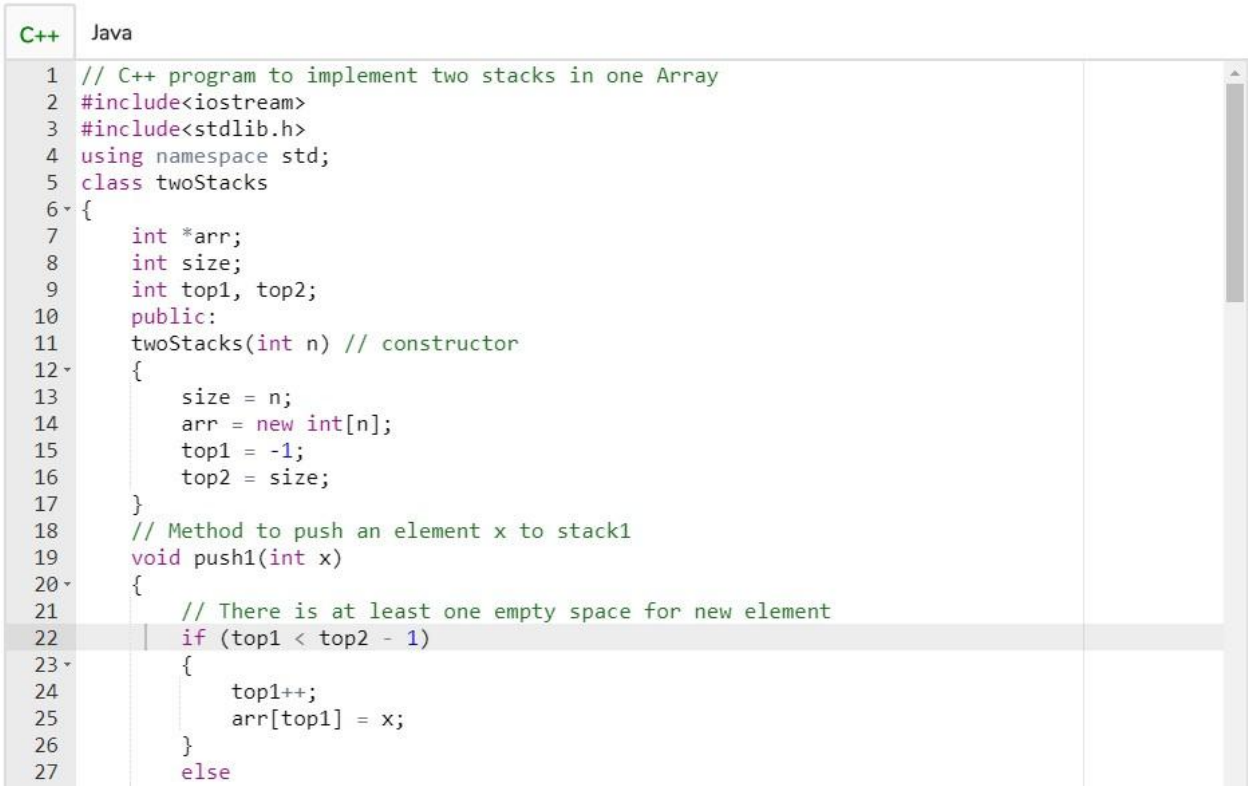
Note: Implementation of *twoStack* should be space efficient.

Method 1 (Divide the space in two halves): A simple way to implement two stacks is to divide the array into two halves and assign the half space to the first stack and the other half to the second stack, i.e., use arr[0] to arr[n/2] for stack1, and arr[(n/2) + 1] to arr[n-1] for stack2 where arr[] is the array to be used to implement two stacks and size of array be **n**.

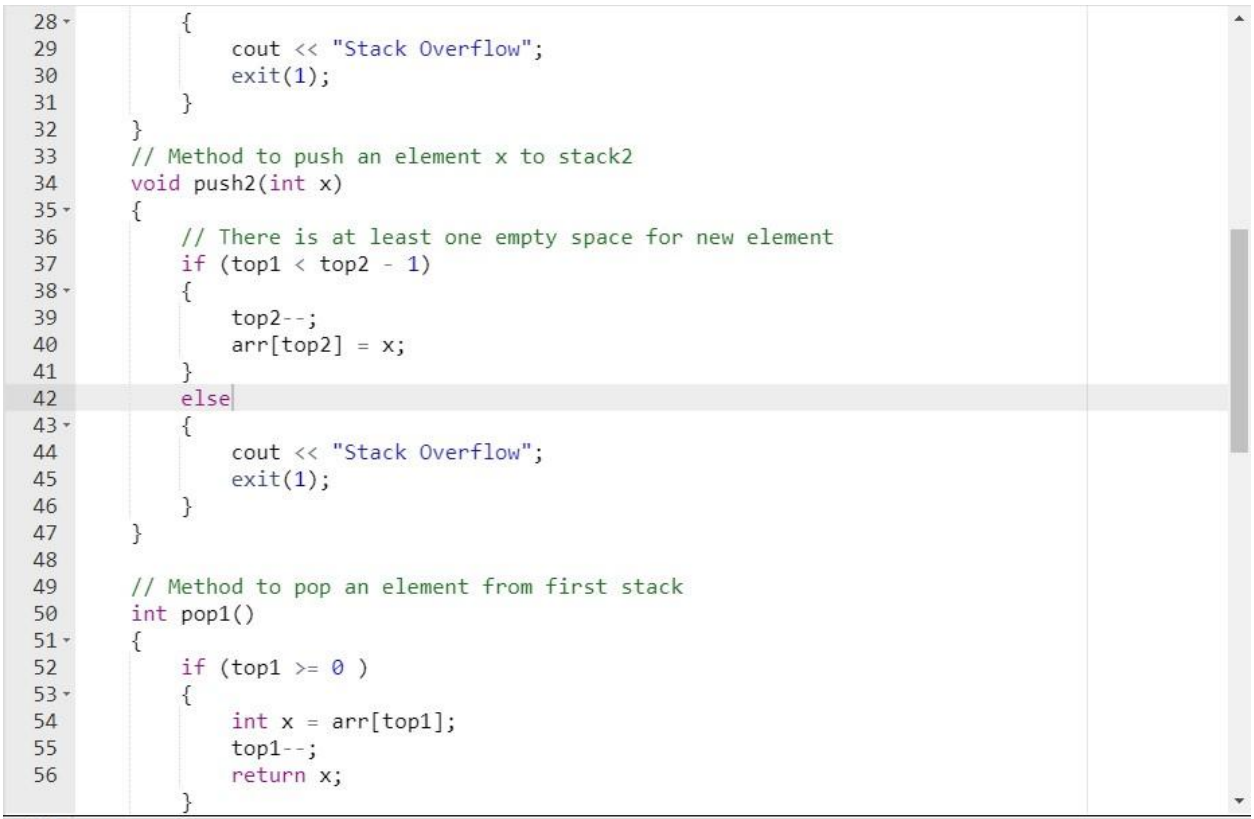
The problem with this method is an inefficient use of array space. A stack push operation may result in stack overflow even if there is space available in arr[]. For example, say the array size is 6 and we push 3 elements to stack1 and do not push anything to the second stack2. When we push the 4th element to stack1, there will be overflow even if we have space for 3 more elements in the array.

Method 2 (A space-efficient implementation): This method efficiently utilizes the available space. It doesn't cause an overflow if there is space available in arr[]. The idea is to start two stacks from two extreme corners of arr[]. The first stack, *stack1* starts from the leftmost element, the first element in *stack1* is pushed at index 0. The second stack, *stack2* starts from the rightmost corner, the first element in stack2 is pushed at index (n-1). Both stacks grow (or shrink) in opposite direction. To check for overflow, all we need to check is for space between top elements of both stacks.

Below the space-efficient implementation(Method 2) of the above task:

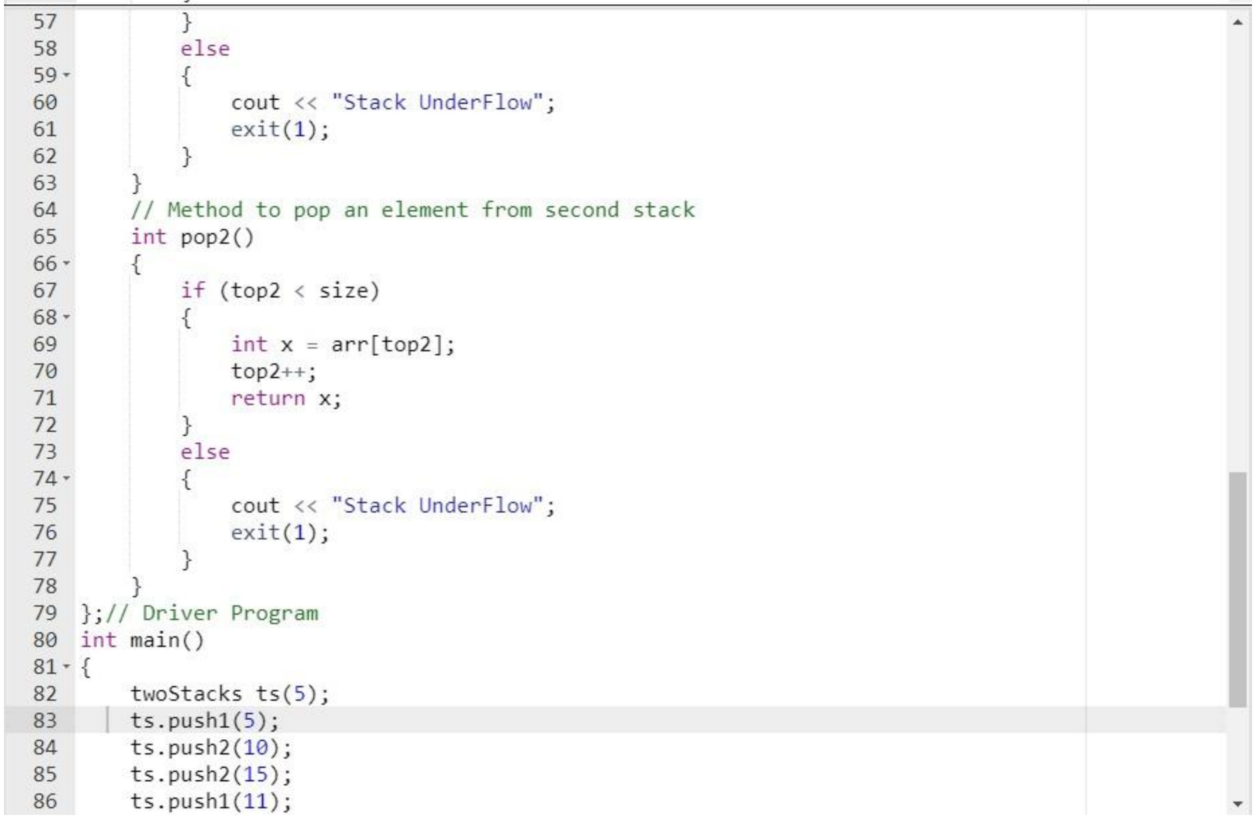


```
// C++ program to implement two stacks in one Array
#include<iostream>
#include<stdlib.h>
using namespace std;
class twoStacks
{
    int *arr;
    int size;
    int top1, top2;
    public:
    twoStacks(int n) // constructor
    {
        size = n;
        arr = new int[n];
        top1 = -1;
        top2 = size;
    }
    // Method to push an element x to stack1
    void push1(int x)
    {
        // There is at least one empty space for new element
        if (top1 < top2 - 1)
        {
            top1++;
            arr[top1] = x;
        }
        else
```

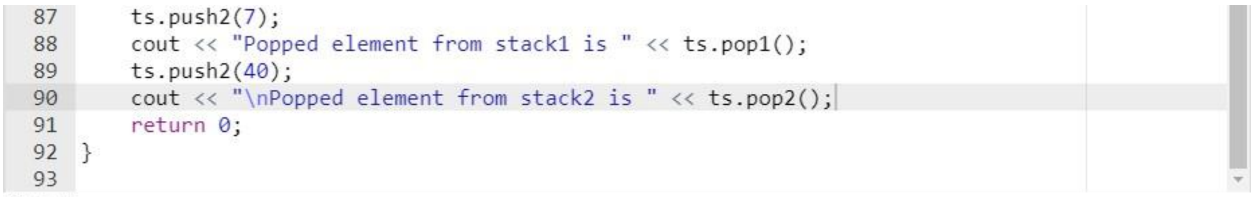


```cpp
        {
            cout << "Stack Overflow";
            exit(1);
        }
    }
    // Method to push an element x to stack2
    void push2(int x)
    {
        // There is at least one empty space for new element
        if (top1 < top2 - 1)
        {
            top2--;
            arr[top2] = x;
        }
        else
        {
            cout << "Stack Overflow";
            exit(1);
        }
    }

    // Method to pop an element from first stack
    int pop1()
    {
        if (top1 >= 0 )
        {
            int x = arr[top1];
            top1--;
            return x;
```



```cpp
        }
        else
        {
            cout << "Stack UnderFlow";
            exit(1);
        }
    }
    // Method to pop an element from second stack
    int pop2()
    {
        if (top2 < size)
        {
            int x = arr[top2];
            top2++;
            return x;
        }
        else
        {
            cout << "Stack UnderFlow";
            exit(1);
        }
    }
};// Driver Program
int main()
{
    twoStacks ts(5);
    ts.push1(5);
    ts.push2(10);
    ts.push2(15);
    ts.push1(11);
```



```cpp
    ts.push2(7);
    cout << "Popped element from stack1 is " << ts.pop1();
    ts.push2(40);
    cout << "\nPopped element from stack2 is " << ts.pop2();
    return 0;
}

```

Output:

```
Popped element from stack1 is 11
Popped element from stack2 is 40
```

*The **time complexity** of all of the four push and pop operations of both stacks is O(1).*