Implement two Stacks in an Array

Create a data structure **twoStacks** that represent 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 first stack and return the popped element
- pop2() --> pops an element from second stack and return the popped element

Implementation of *twoStack* should be space efficient.

Implement two stacks in an array by Dividing the space into two halves:

The idea to implement two stacks is to divide the array into two halves and assign two halves to two stacks, 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.

Follow the steps below to solve the problem:

- To implement **push1():**
  - First, check whether the **top1** is greater than 0
    - If it is then add an element at the top1 index and decrement top1 by 1
    - Else return Stack Overflow
- To implement **push2():**
  - First, check whether **top2** is less than n - 1
    - If it is then add an element at the top2 index and increment the top2 by 1
    - Else return Stack Overflow
- To implement **pop1():**
  - First, check whether the **top1** is less than or equal to n / 2
    - If it is then increment the top1 by 1 and return that element.
    - Else return Stack Underflow
- To implement **pop2():**
  - First, check whether the **top2** is greater than or equal to (n + 1) / 2
    - If it is then decrement the top2 by 1 and return that element.
    - Else return Stack Underflow

Below is the implementation of the above approach.

```
#include <bits/stdc++.h>
using namespace std;

class twoStacks {
    int* arr;
    int size;
    int top1, top2;

public:
    // Constructor
    twoStacks(int n)
    {
        size = n;
        arr = new int[n];
        top1 = n / 2 + 1;
        top2 = n / 2;
    }

    // Method to push an element x to stack1
    void push1(int x)
    {
        // There is at least one empty
        // space for new element
        if (top1 > 0) {
            top1--;
            arr[top1] = x;
        }
        else {
            cout << "Stack Overflow"
                 << " By element : " << x << endl;
            return;
        }
    }

    // Method to push an element
    // x to stack2
```

```
void push2(int x)
{

    // There is at least one empty
    // space for new element
    if (top2 < size - 1) {
        top2++;
        arr[top2] = x;
    }
    else {
        cout << "Stack Overflow"
             << " By element : " << x << endl;
        return;
    }
}

// Method to pop an element from first stack
int pop1()
{
    if (top1 <= size / 2) {
        int x = arr[top1];
        top1++;
        return x;
    }
    else {
        cout << "Stack UnderFlow";
        exit(1);
    }
}

// Method to pop an element
// from second stack
int pop2()
{
    if (top2 >= size / 2 + 1) {
        int x = arr[top2];
        top2--;
        return x;
        }
        else {
            cout << "Stack UnderFlow" << endl;
            exit(1);
        }
    }
};

/* Driver program to test twoStacks class */
int main()
{
    twoStacks ts(5);
    ts.push1(5);
    ts.push2(10);
    ts.push2(15);
    ts.push1(11);
    ts.push2(7);
    cout << "Popped element from stack1 is "
         << ": " << ts.pop1() << endl;
    ts.push2(40);
    cout << "Popped element from stack2 is "
         << ": " << ts.pop2() << endl;
    return 0;
}
```

Output

```
Stack Overflow By element : 7
Popped element from stack1 is : 11
Stack Overflow By element : 40
Popped element from stack2 is : 15
```

Time Complexity:

- **Both Push operation:** O(1)
- **Both Pop operation:** O(1)

Auxiliary Space: O(N), Use of array to implement stack.

Problem in the above implementation:

The problem in the above implementation is that as we reserve half of the array for a stack and another half for the another stack. So, let if 1st half is full means first stack already have n/2 numbers of elements and 2nd half is not full means it doesn't have n/2 numbers of elements. So, if we look into the array, there are free spaces inside array(eg. in the next half) but we cannot push elements for stack 1(because first half is reserved for stack 1 and it's already full). It means this implementation show stack overflow although the array is not full. The solution for this answer is the below implementation.

Implement two stacks in an array by Starting from endpoints:

The idea is to start two stacks from two extreme corners of arr[].

Follow the steps below to solve the problem:

- Stack1 starts from the leftmost corner of the array, the first element in stack1 is pushed at index 0 of the array.
- Stack2 starts from the rightmost corner of the array, the first element in stack2 is pushed at index (n-1) of the array.
- Both stacks grow (or shrink) in opposite directions.
- To check for overflow, all we need to check is for availability of space between top elements of both stacks.
- To check for underflow, all we need to check is if the value of the top of the both stacks is between 0 to (n-1) or not.

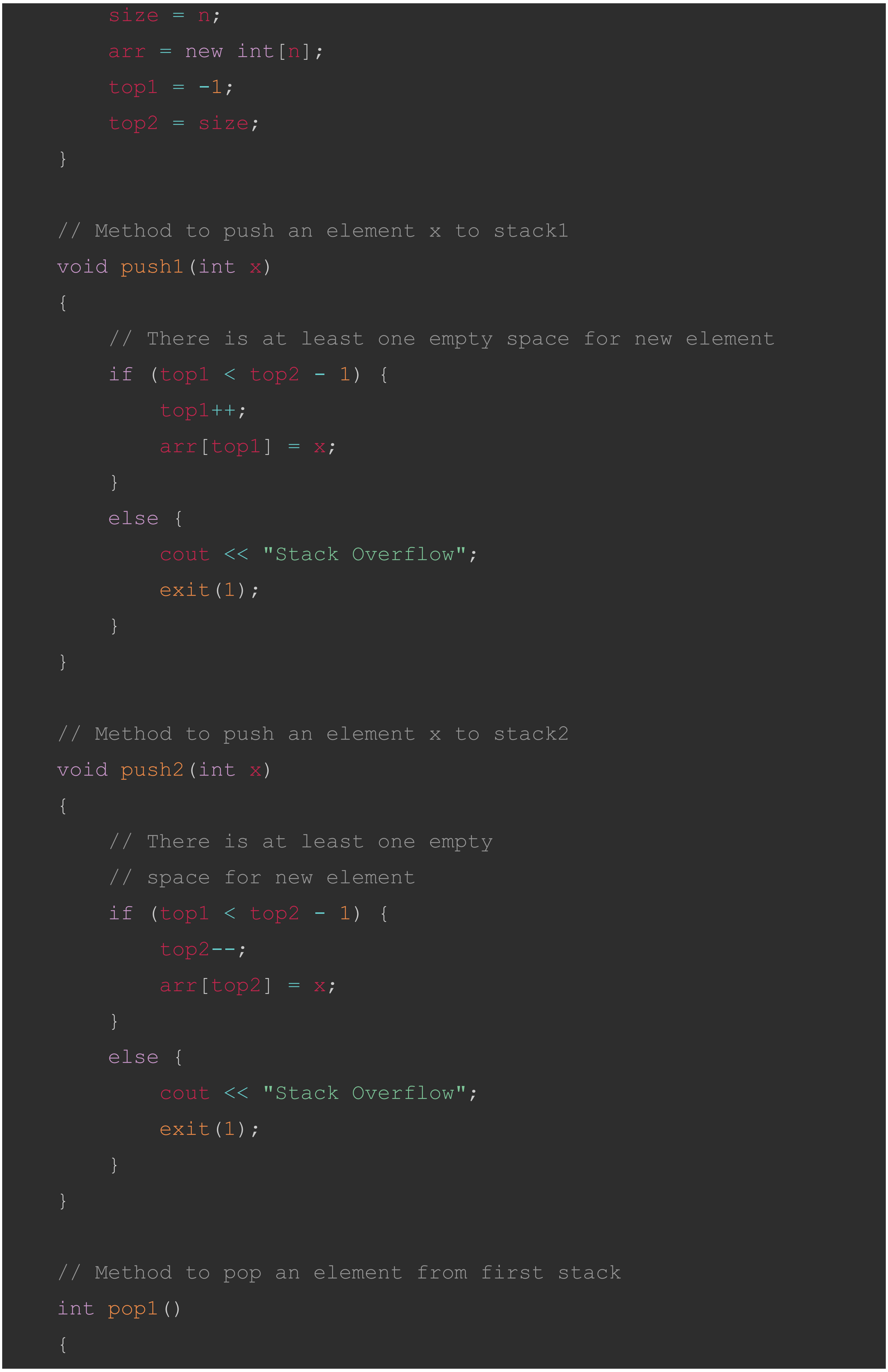
Below is the implementation of above approach:

```
#include <iostream>
#include <stdlib.h>

using namespace std;

class twoStacks {
    int* arr;
    int size;
    int top1, top2;

public:
    twoStacks(int n) // constructor
    {
```

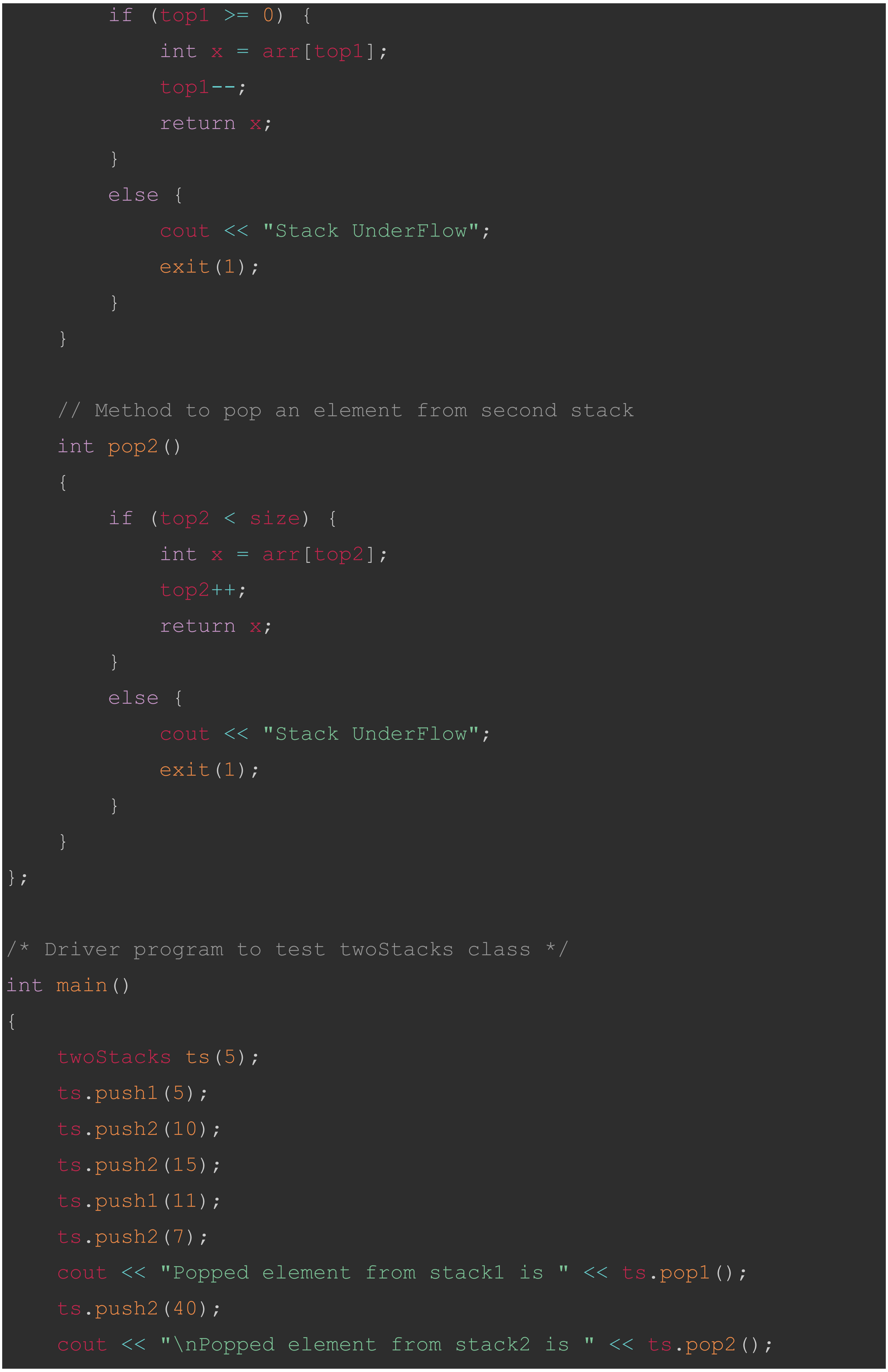


```
        size = n;
        arr = new int[n];
        top1 = -1;
        top2 = size;
    }

    // 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 {
            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;
        }
        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 to test twoStacks class */
int main()
{
    twoStacks ts(5);
    ts.push1(5);
    ts.push2(10);
    ts.push2(15);
    ts.push1(11);
    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
```

Time Complexity:

- **Both Push operation:** O(1)
- **Both Pop operation:** O(1)

Auxiliary Space: O(N), Use of the array to implement stack.

Mark as Read

Report An Issue