

Two Stacks in C++

```
#include <iostream>
#include <vector>
using namespace std;

class TwoStack {
private:
    vector<int> data;
    int tos1; // Top of stack 1
    int tos2; // Top of stack 2
public:
    TwoStack(int cap) {
        // Constructor to initialize the two stacks
        data.resize(cap); // Resize the vector to given
        capacity
        tos1 = -1; // Initialize top of stack 1 to -1
        tos2 = cap; // Initialize top of stack 2 to cap (end
        of array)
    }

    int size1() {
        // Returns the size of stack 1
        return tos1 + 1;
    }

    int size2() {
        // Returns the size of stack 2
        return data.size() - tos2;
    }

    void push1(int val) {
        // Pushes an element onto stack 1
        if (tos2 == tos1 + 1) {
            cout << "Stack overflow\n";
        } else {
            tos1++;
            data[tos1] = val;
        }
    }


    void push2(int val) {
        // Pushes an element onto stack 2
        if (tos2 == tos1 + 1) {
            cout << "Stack overflow\n";
        } else {
            tos2--;
            data[tos2] = val;
        }
    }

    int pop1() {
        // Pops an element from stack 1
        if (size1() == 0) {
            cout << "Stack underflow\n";
            return -1;
        } else {
            int val = data[tos1];
            tos1--;
            return val;
        }
    }
}
```


Let's break down a **tabular dry run** of your TwoStack implementation with:

Capacity = 5

Operations = push1(10), push1(20), push2(30), push2(40), push2(50), push1(60)

 Initial State


- data: [_, _, _, _, _]
- tos1 = -1, tos2 = 5
- Stack 1 grows → from index 0
- Stack 2 grows ← from index 4

 Operation by Operation

Operation	tos1	tos2	data (0 to 4)	Note
push1(10)	0	5	[10, _, _, _, _]	10 pushed to stack 1
push1(20)	1	5	[10, 20, _, _, _]	20 pushed to stack 1
push2(30)	1	4	[10, 20, _, _, 30]	30 pushed to stack 2
push2(40)	1	3	[10, 20, _, 40, 30]	40 pushed to stack 2
push2(50)	1	2	[10, 20, 50, 40, 30]	50 pushed to stack 2
push1(60)	-	-	Overflow!	tos2 == tos1 + 1 (2 == 2)

 Stack Status

- Stack1 (left): [10, 20] → tos1 = 1
- Stack2 (right): [50, 40, 30] → tos2 = 2

 Output Operations

```
cout << "top1: " << st.top1() << "\n"; // 20
cout << "pop1: " << st.pop1() << "\n"; // 20
cout << "top1: " << st.top1() << "\n"; // 10
cout << "pop1: " << st.pop1() << "\n"; // 10
cout << "top2: " << st.top2() << "\n"; // 50
cout << "pop2: " << st.pop2() << "\n"; // 50
cout << "top2: " << st.top2() << "\n"; // 40
cout << "pop2: " << st.pop2() << "\n"; // 40
cout << "top1: " << st.top1() << "\n"; // Underflow
(-1)
cout << "pop1: " << st.pop1() << "\n"; // Underflow
(-1)
cout << "top2: " << st.top2() << "\n"; // 30
cout << "pop2: " << st.pop2() << "\n"; // 30
```

```

int pop2() {
    // Pops an element from stack 2
    if (size2() == 0) {
        cout << "Stack underflow\n";
        return -1;
    } else {
        int val = data[tos2];
        tos2++;
        return val;
    }
}

int top1() {
    // Returns the top element of stack 1
    if (size1() == 0) {
        cout << "Stack underflow\n";
        return -1;
    } else {
        return data[tos1];
    }
}

int top2() {
    // Returns the top element of stack 2
    if (size2() == 0) {
        cout << "Stack underflow\n";
        return -1;
    } else {
        return data[tos2];
    }
}
};

int main() {
    // Hardcoded example
    int capacity = 5;
    TwoStack st(capacity);

    // Perform operations
    st.push1(10);
    st.push1(20);
    st.push2(30);
    st.push2(40);
    st.push2(50);
    st.push1(60);

    cout << "top1: " << st.top1() << "\n";
    cout << "pop1: " << st.pop1() << "\n";
    cout << "top1: " << st.top1() << "\n";
    cout << "pop1: " << st.pop1() << "\n";
    cout << "top2: " << st.top2() << "\n";
    cout << "pop2: " << st.pop2() << "\n";
    cout << "top2: " << st.top2() << "\n";
    cout << "pop2: " << st.pop2() << "\n";
    cout << "top1: " << st.top1() << "\n";
    cout << "pop1: " << st.pop1() << "\n";
    cout << "top2: " << st.top2() << "\n";
    cout << "pop2: " << st.pop2() << "\n";

    return 0;
}

```

✓ Final Stack States

- Stack1: empty
- Stack2: empty
- tos1 = -1, tos2 = 5

Stack overflow

```
top1: 20
pop1: 20
top1: 10
pop1: 10
top2: 50
pop2: 50
top2: 40
pop2: 40
Stack underflow
top1: -1
Stack underflow
pop1: -1
top2: 30
pop2: 30
```