

Name: _____

Lab Section: _____

NET2006: Introduction to Computing

Lab 5: Working with Arrays

Overview

This is an *in-class, individual* lab assignment. There are several goals. You will:

- Identify and correct syntax errors.
- Trace program execution to understand how arrays are organized and how they behave.
- Learn about the **stack** as a tool of management.
- Implement minor modifications in the source code.

I have placed copies of this lab's source code on my website.

Details: Create a Project

Here is the program. It is also available on my website (so that you don't have to type it in).

File Containing Class Specification and *main()*

```
// 5-Stackarray-Improved-RangeChecking.cpp
// an improved stack class
#include <iostream>
using namespace std;
/////////////////////////////////////////////////////////////////
class Stack {
private:
    static const int MAX = 10; // better approach: static means one and
                               // only one instance for all stack objects

    int nTop; // Number of nTop of stack
    int anStack[MAX]; // Stack: array of integers
public:
    stack() // Constructor
    {
        nTop = -1;
    } // -1 means that the stack is empty
    bool isEmpty()
    { return (nTop <= -1); }
    void isEmpty()
    { return (nTop >= MAX-1); }
    void Push(int nVarIn) // Put number on stack
    {
        if (isEmpty()) { // Watch for array overrun
            cout << "Stack Full\n";
            return false;
        }
        else {
            anStack[++nTop] = nVarIn;
            return true;
        }
    }
    int Pop() // Take number off stack
    {
        if (isEmpty()) // Watch for array underrun
            cout << "Empty Stack\n";
        else
            return anStack[nTop--];
    }
}
```

```
/////////////////////////////////////////////////////////////////
void main()
{
    Stack s1; // Breakpoint
    s1.Push(11);
    s1.Push(12);

    Stack s2;
    s2.Push(21);
    s2.Push(22);

    cout << "s1.1: " << s1.Pop() << endl; //12
    cout << "s1.2: " << s1.Pop() << endl; //11

    s2.Push(23);
    s2.Push(24);

    cout << "s2.1: " << s2.Pop() << endl; //24
    cout << "s2.2: " << s2.Pop() << endl; //23

    s1.Push(13);
    s1.Push(14);
    s1.Push(15);
    s1.Push(16);

    cout << " s1.3: " << s1.Pop() << endl; //16
    cout << " s1.4: " << s1.Pop() << endl; //15
    cout << " s1.5: " << s1.Pop() << endl; //14
    cout << " s1.6: " << s1.Pop() << endl; //13

    if (s1.isEmpty())
        cout << "Stack 1 is empty.";
    else
        cout << "Stack 1 has remaining items.";

    if (s2.isEmpty())
        cout << "Stack 2 is empty.";
    else
        cout << "Stack 2 has remaining items.";
}
```

1: Show values in **s1**. (see page 2 diagrams)

2: Show values in **s1** and **s2**. (see page 2 diagrams)

3: Show values in **s1** and **s2**. (see page 2 diagrams)

4: Show values in **s1** and **s2**. (see page 2 diagrams)

5: Show values in **s1** and **s2**. (see page 3 diagrams)

What is displayed? (see page 3 diagrams)

Details: Dealing with Syntax Errors

Find and correct the syntax errors in the preceding code. If you can recognize them without compiling, that's great (you are internalizing more knowledge about the C++ language). If you need to use the compiler, that's okay too (you are gaining more skill in using the compiler to find errors).

(3 marks) When you successfully compile the corrected program, identify the syntax corrections directly in the program code in this document.

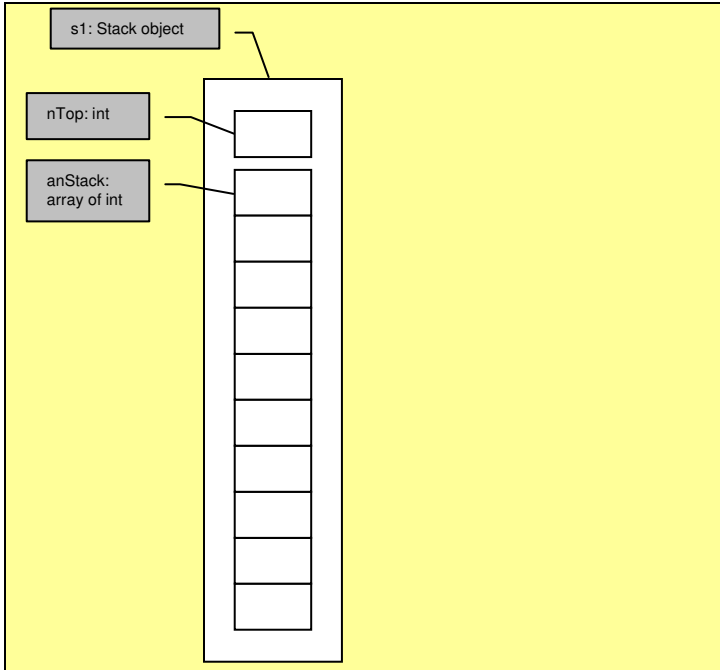
Details: Inspecting the stack Objects

There are five locations where I expect you to document the values stored in your two **Stack** objects, and I've highlighted those sections in the source code.

Location 1:

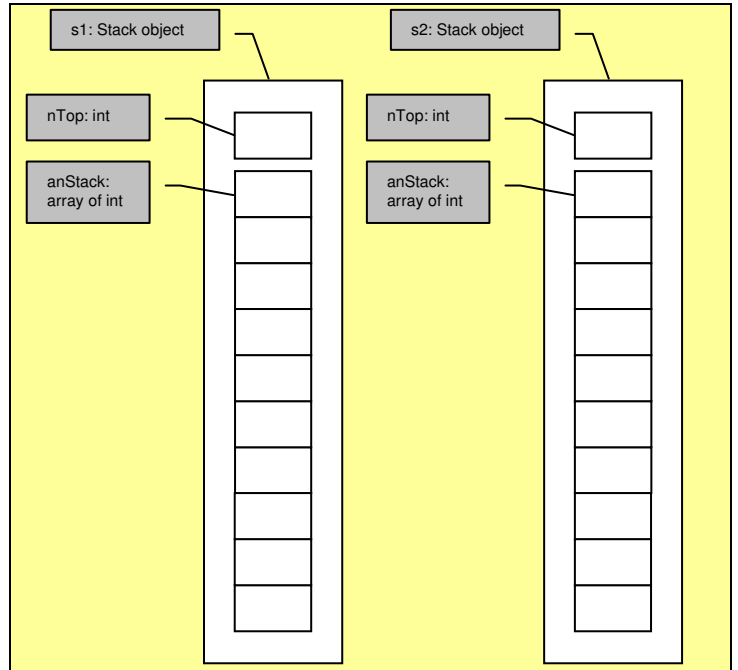
Set a breakpoint after the declaration / definition of the **Stack** object **s1**.¹ The **s1** object will have been built and its constructor will have done its work. When program execution reaches the breakpoint, open a *Watch* (or *Autos*) window to inspect the contents of the two **Stack** objects. What values do you see?

At each of the locations (1 through 5) identify which values are garbage.



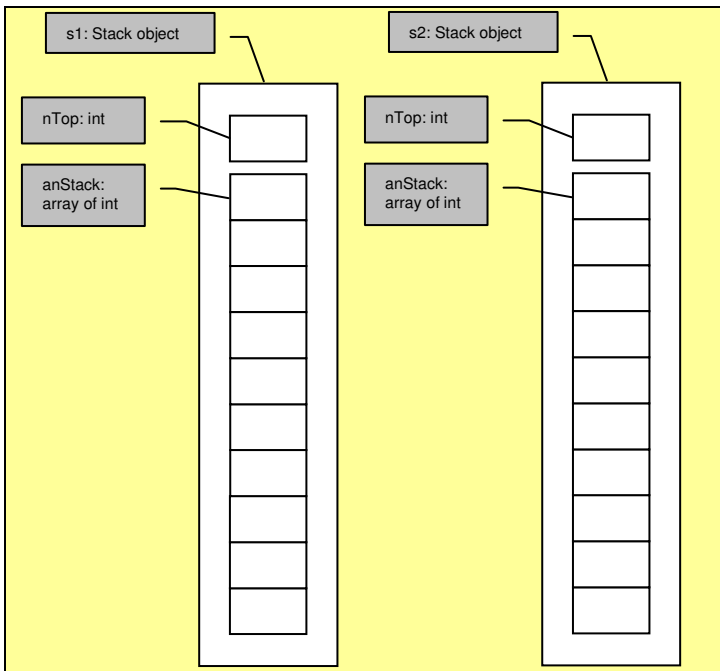
Location 3:

What is in your two **Stack** objects at the next location?



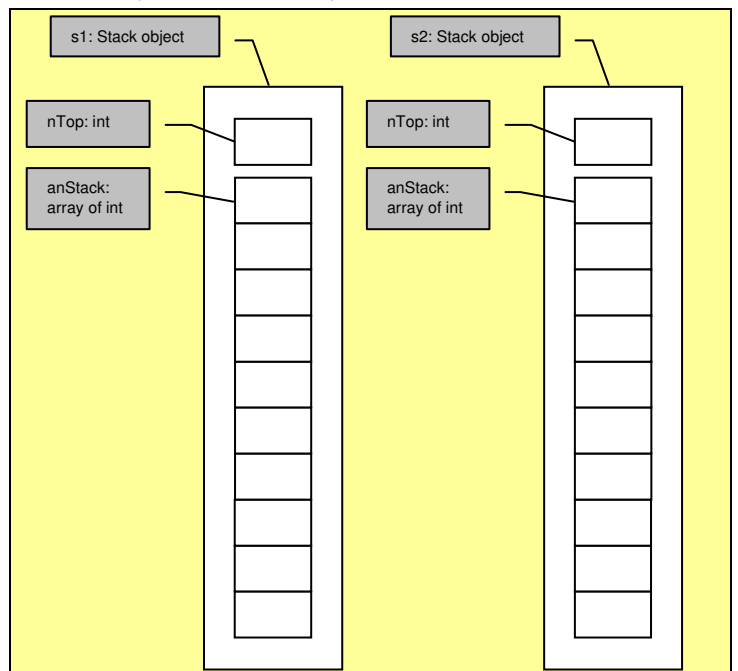
Location 2:

What is in your two **Stack** objects at the next location?



Location 4:

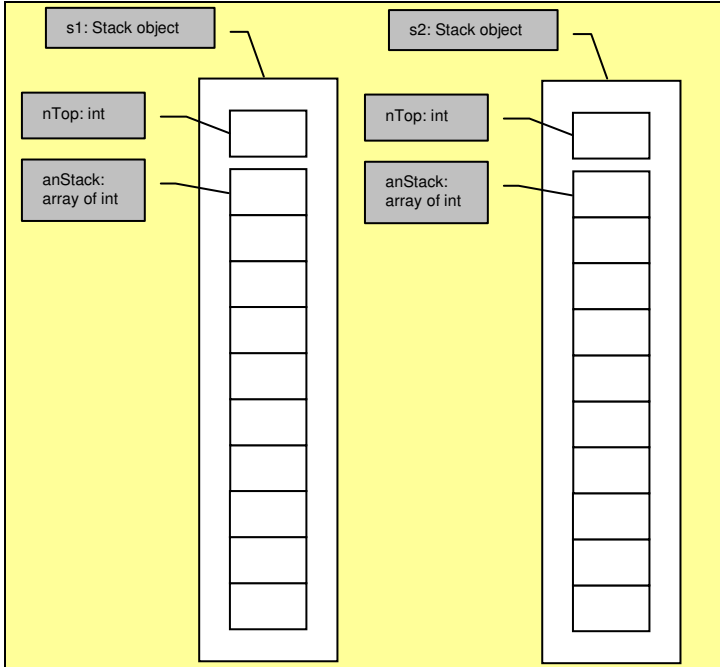
What is in your two **Stack** objects at the next location?



¹ Make sure you step past the declaration / definition of **s1** so that the constructor has been called. Thus, the break will be on **s1.Push(11)**;
Lab 5: Working with Arrays

Location 5:

What is in your two **Stack** objects at the next location?



Details: Implement and Test a Function to Clear the stack

Add a member function that will **clear** the stack.

Modify your main() to test and see if it behaves as you expect.

Submission Process

- Print the file *5-Stackarray-Improved-RangeChecking.cpp*. This revised source code will contain the source code syntax corrections and the implementation of **clear**.
- Staple this source code to this lab document.
- Submit this by the end of this lab period.

Location 5:

What is displayed on the screen?

