Lab 6A

References and Copy Constructor

Introduction

The goal of these exercises is to investigate the use of references, copy constructors, and **const** modifiers. The first exercise involves doing a hand trace of several definitions and assignments involving references and pointers. The remaining exercises involve studying the use of a copy constructor in the **String** class, exploring using **const**, and previewing issues involving assignment of objects.

Suggested Time: 45 minutes.

Instructions

1. Answer the following question: What will be stored in the variable **a** after the following statements have been executed?

```
int   a;
int   *p = &a;
int   &b = a;
int   *&q = p;
a = 5;
b = 7;
*q = 100 + a;
```

2. In the lab directory there is a version of the **String** class and the demo program **DemoStrn.cpp** . Build and run**.** What is the sequence of constructors and destructors that are invoked when the program is run?

3. Replace the initialization **String a("Alpha")** in **main()** by

```
String a = "Alpha";
```

Rebuild and run the program. What is the difference with the previous result?

4. The parameters in the **PrintStrings()** function are passed by value, resulting in a lot of overhead as copy constructors are invoked. Change the function so that the parameters are passed by reference. Since **PrintStrings** does not change the parameters, they should also be declared as **const**, resulting in the following:

```
void PrintStrings(const String&a, const String& x)
{
   cout << "a = " << a.GetString() << endl;
   cout << "x = " << x.GetString() << endl;
}
```

Build the program. Fix any compile errors, build and run again.

Lab 6B

Passing a Stack Object

Introduction

In this lab you will implement a **MoveStack** function that will perform the operation of popping elements from the first stack and pushing them onto the second stack, until the first stack is empty. To demonstrate the effects of call by value versus call by reference, you will pass the first stack as an object and the second stack as a pointer to an object.

Suggested Time: 20 minutes.

Instructions

1. Build and run the starter program. You will see outputs from the **IntStack** methods and trace statements from constructors and destructors.

2. Before your **main()** function define a function **MoveStack()** whose first parameter is an **IntStack** object and whose second parameter is an **IntStack** pointer. Your function should return **void**.

3. Implement the body of your function by moving the loop code from **main()** to **MoveStack()**. Adjust the code to account for the fact that the second parameter is a pointer to **IntStack**.

4. In **main()** place a call to **MoveStack()** where the loop used to be.

5. Build and run your program. Do you have exactly the same behavior as before? Explain any differences.

6. You don't have the same behavior, because you made a copy of the first stack in the **MoveStack()** function, and the elements were popped from the copy, not from the original stack. Fix this by passing both stacks via pointers.