CSE 332 Studio Session on C++ Classes

These studio exercises are intended to give a more complete coverage of the structure of C++ structs and classes and the major kinds of methods and features associated with them, and to give you experience using those concepts within the Visual C++ environment. In this studio you will again work in small groups. As before, students who are more familiar with the material are encouraged to help those for whom it is less familiar. Asking questions of your instructors and teaching assistants (as well as of each other) during studio sessions is highly encouraged as well.

Please record your answers you work through the following exercises. After you have finished please send your answers to the required exercises, and to any of the enrichment exercises you completed, in an e-mail to the cse332@seas.wustl.edu course e-mail account, with the subject line "Classes Studio". The enrichment exercises are optional but are a good way to dig into the material a little deeper, especially if you breeze through the required ones.

Please make sure that as you work through these exercises that each member of your team has a chance to participate actively – one way is to take turns coding, looking up details, debugging, etc.

PART I: REQUIRED EXERCISES

1. Find your team members in the studio area, sit down at/around and log in to one of the Windows machines, open up Visual Studio, and create a new Visual C++ Win32 Console Application project for this studio. Change the signature of the main function in the source file that Visual C++ generated to match the one that was specified in the lab assignments and in the lecture slides. Add a new header file and source file to your project, in which you will (respectively) declare and define classes and structs throughout this studio. Write down the names of the team members who are present (please catch up anyone arriving late on the work, and also add their name) as the answer to this exercise.

2. In the header file, declare a struct with two integer member variables. Do not add private, public, or protected specifications or any constructors, methods, or operators, or initialize the member variables – the idea here is to understand the basic behaviors of a struct or class and then work through understanding additional features (**hint:** don't forget the semicolon after the closing brace, which C++ expects). In your main program, declare an object of that type, adding header file includes etc. as needed. Print out the object's member variables to **`cout`**, and try to build and run your program. Then declare and define a default constructor that initializes the member variables to 0 using the base class / member initialization list (not the constructor body). Try to build and run your program, and as the answer to this exercise, please describe say what happened before and after you added the constructor, and if either of them worked what values were printed out and where they came from.

3. What happens if you change the declaration in the header file from a struct to a class, still without adding any private, public, or protected declarations? Try putting public vs. private declarations before the member variables within a struct vs. a class declaration: when can you access the member variables from the main function? As the answer to this exercise, please explain the difference between a struct vs. a class and describe how that difference explains any variations in the program behavior that you saw.

4. Change the declaration from the previous exercises to be a class declaration with private member variables, and add public accessor methods (which take no parameters and return an int) and mutator methods (which take an int, set the variable's value to the passed value, and return a reference to the object) for each of the member variables. Please declare the accessor methods to be const (by putting the keyword **`const`** after the closing parenthesis). Try calling the accessor and mutator methods on const and non-const objects of the class, in your main function, and comment out any that do not work (the accessors should work on both, but the mutators should only work on non-const objects). Use those methods to print out the member variables' original values, and for non-const objects to set those values to something different (and then print out the new values of the member variables). As the answer to this exercise, please show what the program printed out.

5. In your main function, try out the following operations (using the class name and accessors and mutators from your declaration, rather than mine), printing out the values of the variables after each line.

```
MyClass m; // default construction
cout << "m.x = " << m.x() << " and m.y = " << m.y() << endl;
m.x(7).y(3); // chained use of mutator functions
cout << "m.x = " << m.x() << " and m.y = " << m.y() << endl;
MyClass n(m); // copy construction
cout << "n.x = " << n.x() << " and n.y = " << n.y() << endl;
```

As the answer to this exercise, please explain what each of the compiler-supplied (i.e., it provides them if you don't) default constructor and copy constructor is doing.

6. Declare and define (in your header and source files respectively) a copy constructor for the class that takes another object of the same type by *const reference* and *initializes* the member variables (again in the base-member initialization list) to have the values of the member variables of the object from which the current object is being constructed; make sure to use the base-member initialization list for *initializing* member variables in the copy constructor (instead of *assigning* them in the body of the constructor). As the answer to this exercise, please explain what happens compared to the compiler-supplied version of the copy constructor that you saw in the previous exercise, when you run the same code statements as above.

PART II: ENRICHMENT EXERCISES (optional, feel free to skip some and do ones that interest you).

7. Add statements to your default constructor and copy constructor that print out the name of each of these methods and the memory address and member variable values of the object to **`cout`**, when they run. Add a function that takes an object of your class by value and returns that object by value (i.e., the function's return type is the class), and another function that takes an object by reference and returns that object by reference (i.e., the function's return type is a reference to the class), and call both of those functions from your main function. Run your program, and as the answer to this exercise, please say which of the constructor methods are called, and in what order, when objects are created in main, passed by value, returned by value, passed by reference, and returned by reference.

8. As the answer to this exercise, please try out the following using the code you developed for exercise 7, and answer each question. What happens if you declare your default constructor private but do not define it? Can you still create variables of that type? Does the compiler still supply a default constructor that can be used by statements in your main function?