CSE 332 First Studio Session on C++ Copy Control

These studio exercises are intended to introduce basic C++ copy control operations and to give you experience using them 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 "Copy Control Studio I". 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. 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. Copy the header file and source file for the detector class you developed in the previous studio session on dynamic memory management into the appropriate directory under your project for this studio, and then add those files to this studio's project. You will use detector objects to identify when things are created and destroyed in this studio.

   Declare and define another class (we'll call this one the "wrapper" class to distinguish it from the detector class) with a private member variable that is an object of the detector class. In your main function, declare a local object of the wrapper class. Build and run your program, and as the answer to this exercise say (1) how many objects were created (2) how many objects were destroyed.

3. Change the member variable of the wrapper class to being a pointer to an object of the detector class type (instead of being an object of that type); add a default constructor that initializes first initializes that pointer to zero (in the base/member initialization list) and then (in the body of the constructor) dynamically allocates a detector object (from the heap using new) and stores the dynamically allocated object's address in the pointer member variable. Build and run your program, and as the answer to this exercise say (1) how many objects were created how many objects were destroyed in this version, and (2) briefly explain what differences you observed compared to the previous exercise, and why you think any such differences occurred.

4. Add a destructor to the wrapper class that calls delete on the pointer member variable. Repeat the previous exercise(s) with that change, and as the answer to this exercise, explain what happens when you do that.

5. Add a Boolean member variable to your wrapper class, and in the default constructor initialize it to true. Modify your wrapper class destructor so that it calls delete on the pointer member variable if and only if the Boolean member variable is true. Build and run your code and confirm that it behaves the same as it did (in terms of the construction and destruction of detector objects) as it did in the previous exercise. Then, add a copy constructor to your wrapper class that initializes the Boolean member variables to false, and does a shallow copy (copies just the pointer address) from the const object that was passed by reference to it. In your main function, declare a second wrapper object, that is copy constructed from the first one (which is default constructed). Build and run your program and as the answer to this exercise, please say (1) how many times were detector objects created and/or destroyed, (2) whether or not you saw any problems with allocation or deallocation with that scheme, and (3) which wrapper objects were responsible for creating and destroying which detector objects.

6. Modify your copy constructor so that it does a deep copy of the detector object, and modify your destructor so that it always calls delete on the pointer member variable (feel free to remove the Boolean member variable from the class since it only is needed for shallow copy). Build and run your code, and as the answer to this exercise say how many objects are created (and destroyed) in this case due to copy construction.

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

7. Repeat exercise 5, but instead of allocating wrapper objects on the stack, allocate them on the heap and store pointers to them. Use delete to destroy them in different orders, and as the answer to this exercise please explain what problems if any you saw (and if there were problems why they occurred).

8. Repeat exercise 5, but instead of using a Boolean member variable in the wrapper class to indicate ownership of the detector object, just always have the destructor call delete on the pointer member variable. Build and run your code, and as the answer to this exercise please explain what problems if any you saw (and if there were problems why they occurred).