CSE 332 Second Studio Session on C++ Copy Control

These studio exercises are intended to introduce additional 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 II". 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. In your main function, declare and define the following variables: two variables of type **`int`** that are initialized with different integer values, two variables that are l-value references to type **`int`** (which are initialized to refer to the two **`int`** variables, respectively), and two variables that are r-value references to type **`int`**, which are initialized to refer to two different expression the two **`int`** variables, respectively. Try printing out the values of the variables, and then for those for which you can output values try assigning them to each other in different combinations, and printing out the value that was assigned in each case. As the answer to this exercise please indicate which values could and could not be assigned, what happened in each case when values were successfully assigned, and why you think that succeeded or failed in each case.

3. Use the **`std::move`** function to initialize r-value references to the two **`int`** variables in the code from the previous exercise, and then try printing out the values of the moved **`int`** variables to the standard output stream, using them to initialize other **`int`** variables, and assigning their values to **`int`** variables. Try to build and run your code, and as the answer to this exercise please describe what happens when you do that.

4. Copy the header and source files for the detector and wrappers classes from the previous studio session on basic copy control, into the appropriate directory under your project for this studio, and then add those files to this studio's project. Design and implement a copy-assignment operator for the wrapper class that appropriately checks for self assignment, uses the wrapper class' (deep) copy constructor to make a local copy of the passed object, and then uses swap to trade the copied implementation for the old implementation (which then should be destroyed when the object returns).

   Modify your detector class so that it (1) keeps a static counter of how many detector objects have been allocated, and (2) numbers each detector object by storing (and then incrementing) the count's value in a member variable of the detector object when it's constructed. Have your wrapper class copy constructor and copy-assignment operator print out the numbers of the detector objects in them, when they first start and just before they end (you may need to declare friendship in order to do that).

   In your main function, try different combinations of default constructing, copy constructing, and copy-assigning wrapper class objects. Build and run your code, and based on the output of your program, as the answer to this exercise please say (1) whether or not detector objects are being allocated and de-allocated correctly by the copy construction and copy-assignment operations, and (2) why you think that.

5. Declare and define a correct move constructor (in addition to the existing copy constructor) for your wrapper class (e.g. along the lines illustrated in the example in the lecture slides), and then in your main function use wrapper objects in different ways so that at least one copy construction occurs and in another case one move construction occurs. Add output statements to your copy constructor and move constructor so that when they are called a uniquely identifying message is printed out, and build and run your code. As the answer to this exercise please (1) explain why you think the code is behaving correctly and (2) show the output from your program that demonstrates that is so.

6. Declare and define a correct move assignment operator (in addition to the existing copy assignment operator) for your wrapper class (e.g. along the lines illustrated in the example in the lecture slides), and then in your main function use wrapper objects in different ways so that at least one copy assignment occurs and in another case one move assignment occurs. Add output statements to your copy assignment and move assignment operators so that when they are called a uniquely identifying message is printed out, and build and run your code. As the answer to this exercise please (1) explain why you think the code is behaving correctly and (2) show the output from your program that demonstrates that is so.

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

7. Repeat exercise 5 or 6 but break the implicit promise made when producing the move operation, by modifying an object after it's been moved from. As the answer to this exercise please explain what happened when you did that, and why you think what you saw happened.

8. Repeat exercise 5 or 6 but break the implicit promise that the moved object is safely destructible by modifying the relevant move operation so it does not zero out the object that was moved from. As the answer to this exercise please say what happened when you did that, and why that happened.