

## CSE 332 Studio Session on C++ Memory Management with Classes

These studio exercises are intended to introduce memory management concepts and techniques involving C++ classes, and to give you experience using them in the Visual C++ environment.

In this studio you will again work in self-selected small groups, and as before, students who are more familiar with the material are encouraged to help those for whom it is less familiar, and asking questions during the studio sessions is highly encouraged as well.

Please record your answers as you work through the following exercises. After you have finished please post your answers to the required exercises, and to any of the enrichment exercises you completed, to the course message board as a reply to my posting titled “Memory Management with Classes”. 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 as you work through these exercises that each team member has a chance to participate actively – e.g., take turns coding, looking up details, debugging, etc., and please also refer to the slides and the posted code examples as you work.**

### 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. Change the signature of the main function in the source file that Visual C++ generated to match the one that was specified for the previous studios. 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. Add the header file and source file for the class you developed in the previous studio session on memory management into your project. You will use objects of this class to detect when things are created and destroyed in this studio (we will refer to this class as the “detector class”). Add a private member variable that is a pointer to an object of the detector class, and in each of the constructors initialize that pointer to zero. In your main function, declare a local object of the detector 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. Try to make the member variable an object of the detector class type instead of a pointer or reference to detector class type. As the answer to this exercise, please say what happens when you do that, and explain briefly why that happens.

Before proceeding, please change the member variable back to being a pointer to the detector class type.

4. Modify your detector class destructor so that it calls **delete** on the pointer member variable. Repeat exercise 2 but with that change, and as the answer to this exercise, explain what happens when delete is called on a pointer whose value is 0.

5. Add a constructor to your class that takes an unsigned integer as its only parameter and initializes the pointer member variable to 0. If the unsigned integer is 0 the constructor should do nothing, and otherwise should allocate a chain of n additional objects of the detector class type where n is the value of the unsigned integer. To do this you should repeatedly use new and a call to this same constructor (passing one less than the value received) to create the next object in the chain, and assign its address to the pointer member variable. In your main function try creating different length chains of objects using this constructor, and as the answer to this question show the output for creating a chain of length 3 this way.

6. Modify your copy constructor so that it does a deep copy of the chain of objects, starting with the object from which it was constructed. In your main function, add another local object that is copy constructed from the original object 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, do ones that interest you).

7. Modify your assignment operator so that (a) it does a check for self-reference and if so does nothing, otherwise (b) checks whether it or any of the objects in its chain are in the chain of the object whose reference is being passed to it and if so throws an exception, and otherwise (c) makes a deep copy of the passed chain of objects (except for the object that was passed to it), destroys its current chain of objects (except for itself), and replaces its chain with the deep copy. As the answer to this exercise, explain what happens when you do self assignment, vs. assignment from a completely separate object chain.

8. Modify your copy constructor so that it does a shallow copy of the chain of objects, starting with the object from which it was constructed (hints: to make this work correctly, you'll likely need to add a Boolean ownership flag and set it correctly in each constructor and handle deletion in the destructor based on that flag). As the answer to this exercise, compare how many objects are created and destroyed with this approach, vs. the deep copy approach.