CSE 332 Studio Session on Overloading C++ Functions and Operators

These studio exercises are intended to introduce concepts and techniques for overloading C++ functions and operators, 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 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 "Overloading 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 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, and 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. Declare and define a class that has one integer member variable and a constructor that takes an integer and initializes the member variable with it. Declare a couple objects of this class in your main function, using different values to initialize them. After including the `<iostream>` header file and opening up the standard namespace, observe what happens when you try to print out one of those objects directly (i.e., using the object's name, `cout` and its << operator).

Then, outside the class (and with main still trying to print the objects using `cout` and its << operator) declare and define a stream insertion operator (**hint:** the name you will need to use to do that is `operator<<`) that takes a reference to an `ostream` and a reference to a const object of the same class, and uses the `ostream` object's << operator to print out the object's integer member variable. Try doing that both with and without making the operator you have added a friend of the class, and as the answer to this exercise describe what happened (1) without defining the stream insertion operator, (2) when defining the stream insertion operator but not making it a friend of the class, and (3) when defining the stream insertion operator and making it a friend of the class.

3. With the stream insertion operator from exercise 2 declared as a friend of the class, and your main program using the operator to chain together insertions into a single line (such as `cout << i << j << endl;` where `i` and `j` are objects of your class), experiment with the function returning nothing or returning its `ostream` parameter as appropriate, and also updating the friend declaration's return type to match (e.g., with the return type of the operator being `void` vs. `ostream &` vs. `const ostream &`): as the answer to this exercise explain what happens (and why) in each of those cases.

4. Modify your class constructor to that its integer parameter is defaulted to be 0 if no argument is supplied. Declare an object in main without giving it a constructor parameter (as in `MyClass j;`) and add a line that prints out the value of that object. After your program builds and runs correctly, try adding a default constructor to your class. As the answer to this exercise, describe what happened when you did that, and why it did.

5. After removing the default constructor, declare and define appropriate public less than (`operator<`), equivalence (`operator==`), member assignment (`operator=`), and addition (`operator+`) operators in your class. All of these operators should take a reference to a const object of the same class as their only parameter. The less than and equivalence operators should be const methods with a `bool` return type. The addition operator should be a const method that returns an object of the class by value. The assignment operator should return a reference to the object on which it was called. Compare different combinations of objects for equivalence and less than, and print out the results (**hint:** due to operator precedence, when printing out some expressions you may need to surround them with parentheses). Add different combinations of objects and print out the results. Try different combinations of assignment, including chained assignment (as in `a = b = c;`). As the answer to this exercise for each of these tests say what you tried, what the output was, whether or not the correct output is being produced, and why you think the output is correct or incorrect.

6. Include the `<vector>` and `<algorithm>` header files (above the using declaration you added previously) and in your main function declare a vector that contains **`const`** objects of your class. Push a couple of your class objects into the vector, use the sort algorithm to sort the entire vector (using the results of calls to the vector's begin and end methods as the parameters to call to the sort algorithm) and then print out all the locations in the vector. After your program builds and runs correctly, try making your less than, equivalence, and addition operators const vs. non-const. As the answer for this exercise, say (based on those observations) whether or not any of these operators **must** be const to be used with sort over a vector of const objects in that programming environment.

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

7. After making the less than, equivalence, and addition operators const once again in your class, change the access permission of its member variable from private to protected and then derive another class from it using public inheritance. That other class should have its own protected integer member variable in addition to the one it inherits from your original class. Declare and define a similar constructor and set of operators in the derived class, which take parameters of the derived class type rather than the base class type where appropriate. In your main function, declare objects of the derived class, and then try using operators involving different combinations of base and derived objects. Pay particular attention to symmetry: for example, what happens if you try `b == d` vs. `d == b` or `b + d` vs. `d + b` where `b` is an object of the base class and `d` is an object of the derived class. As the answer to this exercise, say which of the operators do not support symmetry when mixing base and derived objects and why you think that occurs.

8. Take the operators out of both the base and derived classes, and instead declare and define them as stand-alone operators that take two parameters of the appropriate types. As the answer to this exercise, say how many different operator signatures were needed and what they were, in order to achieve symmetry of the operators with all possible permutations of derived and base objects.