CSE 332 Studio Session on C++ Operator Overloading

These studio exercises are intended to introduce C++ operator overloading and to give you experience using those techniques 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 as you work through the following exercises. After you have finished please email your answers to the required exercises, and to any of the enrichment exercises you completed, to the course email account (cse332@seas.wustl.edu) with “Overloading Studio” in the subject line. 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. 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 the `<<` operator).

Then, outside the class (and with main still trying to print the objects using cout and the `<<` operator) declare and define an ostream 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 class, and uses the built in `<<` 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 ostream insertion operator, (2) when defining the ostream insertion operator but not making it a friend of the class, and (3) when defining the ostream insertion operator and making it a friend of the class.
3. With the ostream insertion operator from exercise 2 declared as a friend of the class, and your main program using the operator to chain together insertions (as in `cout << i << j << endl;` where `i` and `j` are objects of your class), experiment with the ostream insertion operator returning nothing (i.e., having a `void` return type) vs. returning a reference or reference-to-const for its `ostream` parameter, 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. If you have not done so already, modify the insertion operator from the previous exercise so that it returns a reference to a non-const `ostream`, and then declare and define a non-member addition operator (`operator+`) that takes references to two const objects of your class, initializes an object of the class with the sum of the objects’ member variables, and returns that object by value. In your main function output the results of adding various combinations of objects with different values, and as the answer to this exercise please show (1) your code, and (2) output from your program that indicates the operator you defined in this exercise is working correctly.

5. Declare and define copy-assignment and move-assignment operators that are members of your class. In your main function assign an l-value to one object and assign an r-value to another object. Run your code in the debugger, and set breakpoints in both the copy-assignment and move-assignment operators and then step through the code to see which operators are being called. As the answer to this exercise please show (1) your code, and (2) explain whether only copy-assignment is being used or both copy-assignment and move-assignment are being used, and why you think that.

6. Declare and define both prefix and postfix increment operators (`operator++()` and `operator++(int)`) respectively as members of your class. In your main function, use the ostream insertion operator to print out (1) the value of an object, (2) the result of an expression that uses prefix increment on the object, and (3) the result of an expression that uses postfix increment on the object. As the answer to this exercise, please show your code and the output of your program, and explain why (based on the output you saw) you think the prefix and postfix versions of the increment operator are implemented correctly.

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

7. Repeat exercise 6 using prefix and postfix decrement operators (`operator--()` and `operator--(int)`) respectively. As the answer to this exercise, please show your code and the output of your program, and explain why (based on the output you saw) you think the prefix and postfix versions of the decrement operator are implemented correctly.

8. Declare and define non-member equivalence and less than operators over const references to objects of your class, which return a `bool` value that is the result of applying the corresponding operation over the member variables of the objects. In your main function write different expressions involving these relational operators and output the results of those expressions. As the answer to this exercise please show your code and your program’s output, and say why (based on the output) you think those operators are implemented correctly.