

## CSE 332 Studio Session on Generic Programming in C++

These studio exercises are intended to give an overview of basic concepts and techniques for generic programming with the C++ Standard Template Library (STL), 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 “Generic Programming 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 struct for a point in 2-dimensional space with two private integer member variables representing x and y coordinates in 2-space, a public constructor that takes two integers and initializes the member variables with them, and public const < and == operators that compare objects of that class by their distance from the origin (0,0). Also declare an ostream insertion operator outside of your struct that takes a reference to an ostream and a reference to a const instance of the struct, prints out the values of the struct’s member variables, and returns a reference to the ostream. In your main function, declare a vector of that type, and push back several objects of the type into the vector. Use the vector’s index operator (**operator [ ]**) and the ostream insertion operator you wrote to print out each of the objects in the vector. Build your program, fix any errors or warnings, and run your program - give the output it produces as the answer to this exercise.
3. Modify your program from exercise 2, so that instead of using the indexing operator, it uses a for loop and moves an iterator through the positions of the vector, using the iterator to print out each element (**hint:** ranges in the STL are from the beginning position up to but not including the ending position given, so you should stop iterating (and not print out an element) when the iterator reaches the position given by the vector’s end() method). Again give your program’s output as the answer to this exercise.

4. In your main function declare a list container of the struct type, push back the same set of objects into the list as you did into the vector, and use an iterator to go through and print its elements as you did in exercise 3. Compare the output produced for the list to the output produced for the vector, and as the answer to this exercise say whether or not there was any difference between them.

5. Modify your code from the previous exercises so that it uses the copy algorithm and an ostream\_iterator template parameterized with the struct type to print out the elements in each container, as in the lecture slides. Again build and run your program. Try constructing the ostream\_iterator with only cout, with cout and “\n”, with cout and “ ” and with cout and any other separator you want to try. As the answer to this exercise, explain what happens if you don’t provide a separator, and what happens if you do provide one.

6. Modify your code from the previous exercise so that before printing out the contents of the containers it calls sort on the iterators returned by the begin() and end() methods of each container. Repeat the same thing with a third parameter to the sort algorithm, that is an instance of the **greater** template parameterized with the struct type, as in the lecture slides. Declare and define a (binary predicate functor) struct that has only a function call operator that takes two const references to your 2-dimensional point struct, and returns true if and only if the x coordinate of the first is less than that of the second, or if the x coordinates are the same returns true if and only if the y coordinate of the first is less than that of the second. Use an instance of this functor struct as the third parameter to the sort algorithm, and as the answer to this exercise explain what happened to the order of the elements in each of these cases.

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

7. Repeat exercise 5 with a set container. As the answer to this exercise explain whether or not you saw any differences in behavior with the set vs. the vector or list, and if there were any what they were.

8. Repeat exercise 6 with a set container. As the answer to this exercise explain whether or not you saw any differences in behavior with the set vs. the vector or list, and if there were any what they were.