CSE 332 First Studio Session on C++ Templates and Generic Programming

These studio exercises are intended to introduce basic features of C++ templates 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 "Templates Studio I. 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. Add a new header file to your project and in that file define (in Visual C++ as in many other compilers, template definitions must be included by the code that uses them) a counting function template with a single ITERATOR type parameter (for the type of iterators the function will use) that takes two arguments of the parameterized iterator type and returns a size_t containing the number of elements in the range (from the position pointed to by the first iterator, up to but not including the position pointed to by the second iterator).

   In your main function, declare an array of integers and a vector of integers, and add different numbers of elements to them. Call (and print out the result of calling) the counting function using appropriate iterators (pointers to the start of the array and to just past the last element in the array, or iterators returned by the vector's begin() and end() methods). As the answer to this exercise, please give the results of calling the count method with each of those iterator ranges.

3. Add another header file, and in it declare and define a struct template that takes a single type parameter and has a single public member variable of that type and a public constructor that takes a const reference to an object of the parameterized type.

   In your main function, declare and initialize an object whose type is that template parameterized with int, and another object whose type is that template parameterized with char. Use the sizeof function to determine and print out the sizes of the two objects, and the sizes of the types with which they were parameterized, and give the results as the answer to this exercise.

4. In the header file where your struct is declared and defined, but outside the struct itself, define an ostream insertion operator template that takes a reference to an ostream, and a reference to const for an object whose type is the struct parameterized with the same type with which the operator template is parameterized. The operator template should stream the member variable of the struct template object into the ostream, and then return a reference to the ostream. In your main function, stream the different struct template objects you have instantiated to cout, and as the answer to this exercise, show what was printed as a result of doing that

5. Define equivalence (operator==) and less than (operator<) operator templates for the struct template from the previous exercise. In your main function, declare a few more objects of each of the types from the previous exercise, some with the same value and some with different values. Compare (and print out the results of comparing) different combinations (and for < different orders) of objects of the same type (some with the same values and some with different values) using those operators, and show the results of those comparisons as the answer to this exercise.

6. Try out the compiler-supplied copy construction and assignment behaviors for the struct template, using the objects from the previous exercises (and any new ones you'd like to add) and printing out the values of the resulting objects. As the first part of the answer to this exercise, summarize whether or not those operations behaved as you would expect, and explain why you think that.

   Again using the struct template, try constructing an object without supplying an initialization value. As the second part of this exercise, describe what happens and why it happens that way. For non-template code we could try using default values to constructors to work around this issue, but that does not work well with templates. As the third part of this exercise, please explain what the problem with doing that would be for templates (hint: would any single default value satisfy all possible cases for all possible parameterized types, including abstractions like dice, cards, letter tiles, etc.? Why or why not?).

   Add a default constructor that does nothing, and stream out the default constructed object to cout. Based on your thinking about the previous part of this exercise and what was printed in this part, for the fourth and last part of the exercise, please say whether or not it is always a good idea to provide default constructors for templates, and explain why you think that.

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

7. Change the struct template you added in exercise 3 into a class template, and make its member variable private. As the first part of the answer to this exercise, please describe what happens when you try to build your program.

   To fix this problem add an appropriate friend declaration to your class (hint: the keyword friend must be preceded by the template keyword and the type parameter list, e.g., template <typename T>) so that your program builds correctly once again. As the second part of the answer to this exercise, please describe what the problem was, and how adding the friend declaration fixed it.

8. Add another counting function template like the one you wrote in exercise 2, but have it take a second type parameter and takes a third argument of that second parameterized type by value. This overloaded version of the counting function should return the number of elements in the range (delimited by the first and second arguments) for which the element's value (obtained by dereferencing the iterator at that position) and the third parameter are equivalent (based on the == comparison).