Studio Session on C++ Associative Containers

These studio exercises are intended to give an initial coverage of the structure of C++ Ordered Associative Containers and the major kinds of features associated with them, and to give you experience using those features within the Visual C++ environment.

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 professor and teaching assistant (as well as of each other) during the studio sessions is highly encouraged as well.

Please record your answers you work through the following exercises. After you have finished, please submit your answers to the required exercises and to any of the enrichment exercises you completed to Blackboard or our OJ system (if applicable).

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.

PART I: REQUIRED EXERCISES

1. Open Visual Studio and create a new Visual C++ Empty Project for this studio. In your main function, declare a set container that holds C++ style strings (e.g., a container of type **`set<string>`**). Use the **`copy`** algorithm to transfer all the C-style strings that were passed to the main function (including the program name) into the set container, and then use the **`copy`** algorithm to print out all the strings in the set to **`cout`** (**hint:** for the first **`copy`** you can use pointers to the beginning and just past the end of **`argv`** as the source range, and an **`inserter`** for the set container as the destination; for the second **`copy`** you can use iterators returned by set container's **`begin()`** and **`end()`** methods as the source range, and an **`ostream_iterator`** for **`cout`** as the destination). Build and run your program with different command line arguments (including some trials where the same argument is passed more than once on the command line) and as the answer to this exercise please (1) explain whether or not the container enforced uniqueness of the keys in the container and (2) show output from your program that supports that conclusion.

2. Repeat the previous exercise using a set container that holds C style strings (e.g., a container of type **`set<char *>`**). As the answer to this exercise please describe any differences you observed compared to the previous exercise, and if there were differences, please explain why you think they occurred (**hint:** think about what operator might be used for the comparison, and how it is defined over **`char *`**).

3. Repeat exercise 1 with a set container that holds C++ style strings (e.g., a container of type **`set<string>`**) and pass the program the same set of command line arguments in different orders. As the answer to this question please describe whether or not the set container is enforcing an ordering of the keys it contains, if so how it is distinguishing the keys and show examples of the output you saw that support those conclusions.

4. Repeat exercise 2 with a set container that holds C style strings (e.g., using a container of type **`set<char *>`**) and pass the program the same set command line arguments in different orders. As the answer to this question please describe whether or not the set container is enforcing an ordering of the keys it contains, if so how it is distinguishing the keys and show examples of the output you saw that support those conclusions (**hint:** think about what operator might be used for the comparison, and how it is defined over **`char *`**).

5. Write a function that takes two references to const C++ style strings and returns a **`bool`** that is true if and only if the first is greater than the second. In your main function declare a set container to hold C++ style strings that orders them according to the function you wrote (**hint:** using **`decltype`** can help with this as the example in the textbook illustrates). Repeat exercise 3 with that container and as the answer to this exercise (1) show the code you wrote and (2) the output your program produced for one run that illustrates how the keys in the set are being ordered.

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

6. Repeat the previous exercise for a container of C style strings and as the answer to this question again show your code and the output from an illustrative run of your program.

7. Repeat any of the previous exercises using a **`multiset`** instead of a set, and as the answer to this exercise please describe any differences you saw compared to the version of the exercise that used a set.

8. Use a map from strings to unsigned integers, a callable object you will write (a lambda or a struct with an overloaded function call operator would work well for this) that uses the map and it its subscript operator, and the **`for_each`** algorithm to count how many times each of the strings passed to the main function appeared in **argv**. **Hint**: as in the example in the textbook, the first time a string is used as a key to the map's subscript operator an entry will be created for that key, with the associated unsigned integer value-initialized to 0, so you can just use the ++ operator on the reference returned by the subscript operator each time. As the answer to this exercise please show the code that you used to do this.