CSE 332 Studio Session on Specialized Library Facilities

These studio exercises are intended to introduce basic features of C++ tuples, bitsets, regular expressions and random number generation, and to give you experience using those concepts and techniques within the Visual C++ environment.

In this studio you will again work in self-selected small groups of 2 or 3 people. 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 assistants (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 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 "Specialized Library Facilities". 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. Find/choose your team members in the studio area, 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 used in the lecture slides on C++ program structure, which are available on the course web page. Write down the names of the team members who are present (if a team member arrives late please catch them up on the work, and add their name) as the answer to this exercise.

2. Declare a typedef with the type name **`student`** for a tuple that takes one unsigned integer and one string that will be initialized with a student's id and name respectively. In your main function declare two student tuples with different names and ids (**hint:** you may need to use the **`make_tuple`** function). Print out the contents of the student tuples (**hint:** you may need to use the **`get`** function template), and as the answer to this exercise please show the output of your program.

3. In your main function, declare a **`vector`** of **`student`** tuples, and push back different tuples with different ids and names into the **`vector`**. Print out the contents of the vector, then sort the vector by student ID (which should be the default behavior if you use the **sort** algorithm directly) and then print out the contents of the vector again. As the answer to this exercise, please (1) show the output of your program and (2) explain why that output was produced.

4. Add a bitset variable to the student tuple for classification of each student's year in school (Freshman, Sophomore, Junior, or Senior), and update your code from the previous exercise so that student tuples are initialized with different values for each student's year, and so that a string indicating each student's year in school is printed out according to that classification (**hint:** you can reduce the number of bits needed and also make the code simpler if you use an enumeration to distinguish between the bits and also use the bitset functions **`all`** and **`none`**). As the answer to this exercise please show the output of your program and please say how many bits were needed to represent the classification.

5. Repeat the previous exercise, but in your main function please use regular expressions to ensure that only the students whose names start with letters A through M will be printed. As the answer to this exercise please show the regular expression you wrote to do that, and please also show the output of your program.

6. Modify your code from the previous exercise so that the student ids are generated using a random number engine. Try seeding the random number generator with the same number every time a random number is needed, seeding it only once at the beginning with a particular number, and seeding it only once at the beginning with the result of calling **`time(0)`**. Run your program repeatedly for each of those cases (and especially for the version that is seeded with the call to **`time(0)`**, try running your program with almost no time in between runs versus waiting a second or two between each of them – running your program repeatedly from a script is one way to do this or you can just hit the uparrow and enter keys in rapid succession or with longer pauses) and observe its output each time. As the answer to this question please explain what happens in each of these cases.

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

7. Modify your solution to the previous exercise so that it will print out any name that contains exactly 6 alphabetic letters with an arbitrary number of spaces before, in between or after those letters, and as the answer to this exercise please show the regular expression that you used to do that.

8. Repeat the previous exercise, using the special I/O utilities (I/O manipulators, etc.) to format the output into neatly arranged columns. As the answer to this exercise, please describe how you used those utilities and show the output of your program.

9. Declare, define, and use your own **`operator<`** or another appropriate callable operator, function or object for imposing a strict weak order over **`student`** tuples, so that the students are printed out in sorted order by name instead of by student id. As the answer to this exercise please explain how you did that, and show the output of your program with the students sorted in that order.