CSE 332 First Studio Session on C++ Algorithms

These studio exercises are intended to give additional coverage of the C++ Algorithms Library, and to give you experience using it 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 "Algorithms 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. One useful feature of the STL algorithms is that they're designed to work as well (or even better) with native pointers as with other iterator types declared for the various STL containers. In your main function, declare a (plain-old-C++-style) array of integers, initialized with an odd number of unsorted values in which each value appears one or more times. For example:

   ```
   int arr[] = {-2, 19, 80, -47, 80, 80, -2};
   ```

   Declare a variable of type **`int *`** that is initialized to point just past the end of the array (hint: use pointer arithmetic with the starting address of the array, and the number of positions in the array). Include **`<algorithm>`** and **`<iterator>`** and **`<iostream>`** and use the **`copy`** algorithm to print out the contents of the array by passing it the starting address of the array, the pointer that points just past the end of the array, and a variable of type **`ostream_iterator<int>`** (an output iterator for an **`ostream`**) that is initialized with **`cout`** (optionally you can also pass the iterator's constructor a string like **`" "`** or **`"\t"`** to print between each of the integer values in the array). Build your program and give its output as the answer to this exercise.

3. Extending your code from the previous exercise, declare a container variable of type **`vector<int>`** and an iterator variable of type **`back_insert_iterator< vector<int> >`** (a special kind of iterator that uses the vector's **`push_back()`** method to "output" values to a vector). Use the **`copy`** algorithm to copy the contents of the array into the vector, and then use it again to print the vector using the output iterator for an **`ostream`** that you used in the previous exercise. As the answer to this exercise, give the output your program now produces.

4. Use the **`sort`** algorithm to sort the integers in your array and vector from the previous exercises into non-decreasing order (smallest to largest), and then use the **`copy`** algorithm to have your program again print out the contents of the array and vector after they are sorted. As the answer to this exercise, again give the output your program now produces.

5. Using the sorted array and vector from the previous exercise, use the **`adjacent_find`** algorithm to locate and print out each range of repeated elements on a separate line, as in:

   ```
   80 80 80
   -2 -2
   ```

   As the answer to this exercise, show the code that you wrote to do this.

6. Using the sorted array and vector from the previous exercise, use the **`count`** and **`accumulate`** algorithms to compute the median (middle value), mean (average), and mode (most frequent value) of the integers they contain. **Hints:** (1) with an odd number of positions the median is easy to calculate, using pointer arithmetic; (2) to avoid round-off errors use a float initialized with the result of running the accumulate algorithm to calculate the mean; (3) one way to calculate the mode is to iterate through one of the sorted containers and each time a new value is found call the count algorithm on the remaining sorted range to see how many of that value there are and then skip over that many repeated elements (using pointer arithmetic). As the answer to this exercise, give the median, mean, and mode values that were calculated.

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

7. Declare a struct that represents a playing card. In the struct declare an enumerated type for a card's rank (with values for two through ten, jack, queen, king, and ace in increasing order), and an enumerated type for a card's suit (with values clubs, diamonds, hearts, and spades) in increasing order.

   In the struct declare member variables for the rank and suit, along with operators **<** (based on rank and then suit) and **==** and a constructor, and outside the struct declare an insertion operator (**<<**) that takes (and returns) a reference to an **`ostream`** and also takes a reference to an object of the struct type.

   In your main function, use two nested for loops (one to iterate through the rank values and one to iterate through the suit values) to push all 52 possible combinations of rank and suit into a vector of cards, sort the vector, and print out its contents. Build and run your program, and give its output as the answer to this exercise.

8. Move 5 randomly chosen cards at a time from the vector from the previous exercise, into several different vectors that represent hands. Pass each of the hands to a function that sorts a local copy of the hand, and prints out a message indicating the highest kind of hand it represents:

    High card (lowest kind of hand)
    One pair (two cards of the same rank)
    Two pair (two cards of one rank and two cards of another rank)
    Three of a kind (three cards of the same rank)
    Straight (all cards are in consecutive sequence by rank)
    Flush (all cards are of the same suit)
    Full house (two cards of one rank, and three cards of another rank)
    Four of a kind (four cards of the same rank)
    Straight flush (highest kind of hand - all cards are in sequence by rank and are of the same suit)

As the answer to this exercise, please give the output your program produced for several repeated runs.