This is a take-home final exam. You will have a week to work on your solution and turn it in electronically. You may not consult with other members in the class for this exam, though you may use your copy of the containers developed as a team over the semester. You should document the source of files you use from others. The work you do to solve this problem MUST be your own.

This exam consists of one multipart problem on 3 numbered pages, including this cover sheet.

Your complete solution files are due to be sent in using the grade command given on the next page by the end of the Final Exam period for this class on Monday, 6 May 2013 at 11:45 am.
1. Sets

For the final, you will be implementing a new class, similar to, but not exactly the same as, the set container in the STL. Our sets will be more similar to the mathematical definition of set; a collection of unique data items, which we will keep in sorted order.

You will build the Set class using as much of what you have built over the semester as possible; i.e. your Container and SortedContainer (slightly modified). Note: you may not use the STL to implement your Set. You may begin with some shared code for Containers and SortedContainers from your previous rotation teams. All modifications required below and all code for Set must be your own. Be sure to document what code you send is “shared”.

You will turn in all files needed to implement and test your Set, a makefile to compile the code, and your analysis file described below. You may build as many test drivers as you like, but I will only look at your final driver which should convince me that your version of the classes meet as much of the spec as you can. Make sure your makefile has the version of your program you want me to look at as the primary target (my script will compile your code with the command “make” To turn in your code, make sure all files needed to compile and execute your driver are submitted. Use a grade command similar to:

```
grade -final,ee205 *.cc *.h makefile analysis
```

(a) Dynamic Array Container (20 Points).

You will begin with your Array based Container, but modify it to dynamically allocate storage for the data. When initially created, the Array based Container will dynamically allocate an array of size determined by a const int MAX_CONTAINER_SIZE. Your Container will also dynamically resize the array when the container gets full (i.e. isfull() will always return false, like for the Link List Container). To simplify the implementation, you may simply allocate a new array twice the current size whenever the Container array gets full and copy the data from the old array before deleting it. For purposes of this exam, make MAX_CONTAINER_SIZE a small value, 5, so that the dynamic sizing can be tested.

Your Container should provide all of the operations described in the original Container as well as overloaded operators for equality comparison and output and provide a linear search operation called find(). You may add, but not delete, other operations you think would be useful, provided you document them in Analysis (see below).

You may use any version of the Array based Container from any of your previous rotation teams as a starting point; just be sure to identify where you got your initial Container in the heading comment of your files and in your Analysis (see below).

(b) Iterators (10 Points).

For credit on this section, your Container must provide an iterator cooperating class for your Container.
(c) **Sorted Container** (10 Points).
Each individual should have a version of SortedContainer from Lab 5 derived from Container. Verify your SortedContainer works with the modified Container above. (Using Insertion sort when placing data in a SortedContainer is sufficient; you do not need the selection sort constructor).
If you had shared code with any of your team members, be sure to document that in the files and Analysis.

(d) **Set** (25 Points).
A Set “is a” SortedContainer with the added property that all entries are unique (trying to add a duplicate data item should silently do nothing). Your Set should provide the following additional operations (overloaded operators). For set objects s and t:

\[
s[\text{Entry}] - \text{evaluates to a bool; true if Entry is a member of set } s
\]
\[
s \cup t - \text{evaluates to a new set that is the union of } s \text{ and } t
\]
\[
s \cap t - \text{evaluates to a new set that is the intersection of } s \text{ and } t
\]
\[
s \subseteq t - \text{evaluates to a bool; true if } s \text{ is a subset of } t
\]
\[
\text{(every element of } s \text{ is in } t)\]

(e) **Template** (25 Points).
Finally, for full credit on this section, modify your above classes to be templatized for the element type, Entry.

(f) **Analysis** (10 Points).
In this section, provide a text file describing the status of your final code, including how far in the development you were able to complete (what works and what does not work). Also include a description of what code you submitted is “shared” with what other members of the class.
Finally, critique your implementation of Set. Is this a good implementation approach? Are there better choices? Explain why or why not.