Project 2: Roster "is-a" Bag of Students

The classes in Project1 were pretty trivial. Project1 was mainly meant to provide a baseline to get you all on the same page with **separate compilation**, **compiling and running remotely with g++ on the Linux machines at Hunter (as a baseline to ensure correct compilation before submitting to Gradescope), working with multiple files/classes, dividing interface from implementation, understanding #includes and working with basic inheritance. IF YOU ARE NOT ABSOLUTELY COMFORTABLE WITH ALL OF THIS PLEASE SEEK HELP FROM THE UTAs IN LAB IMMEDIATELY!!!**

In Project2 we will work with the `ArrayBag` class discussed during lecture. This project consists of two parts:
1. Modify the `ArrayBag` class
2. Implement a class `Roster` which inherits from `ArrayBag` and stores `Student` objects (you will also make a small modification to the `Person` class to support `Roster`)

You will find the `ArrayBag` class (as discussed in lecture) on Blackboard under Course Materials / Project2.
First you **must read the `ArrayBag` interface** and understand how it works. You will need to know how to use `ArrayBag` objects by understanding its interface.
Note: Reading interfaces will be the way you learn to use language libraries, so this is great practice for that!

Implementation - 2 parts:

Work incrementally! Start from Part 1 (implement and test), when that runs correctly then move on to Part2.

Part1- ArrayBag modifications:

1. Modify the `add` method so that it will not allow duplicate items to be added to the ArrayBag (conceptually the Bag becomes a Set).

2. Implement <u>public</u> method `display()` to display the contents of the bag to standard output in the form "`item1, item2, … , itemN\n`"

   ```
   /**@post prints the contents of items_ to the standard output
          separated by commas and followed by a new line.**/
   void display() const;
   ```

3. Overload <u>public</u> `operator+=` to implement Set Union (**Hint:** you can use other ArrayBag operations to implement this)

   ```
   /** implements Set Union
    The union of two sets A and B is the set of elements which are in A,
    in B, or in both A and B.
    @param a_bag to be combined with the contents of this (the calling) bag
    @post adds as many items from a_bag as space allows
   */
   void operator+=(const ArrayBag<T>& a_bag);
   ```

Note: Because ArrayBag is of <u>fixed size</u>, += will only copy as many items from a_bag as there is space available without deleting its original contents. NOTICE HOW FIXED SIZE CAN BE AN ISSUE AND FORCE UNINTUITIVE IMPLEMENTATIONS! We will address the fixed-size problem soon.

4. Overload <u>public</u> `operator-=` to implement Set Difference (**Hint:** you can use other ArrayBag operations to implement this)

   ```
   /** implements Set Difference
    The (set) difference between two sets A and B is the set that
    consists of the elements of A which are not elements of  B
    @param a_bag to be subtracted from this (the calling) bag
    @post removes all data from items_ that is also found in a_bag
   */
   void operator-=(const ArrayBag<T>& a_bag);
   ```

IMPORTANT: Please remember that you DO NOT compile (or include in your project) the implementation (.cpp) of a Template class. Please look at slide 38 from Lecture 3 and make sure you

understand separate compilation with templates (it will probably help if, as suggested on slide 40, you first run a dummy test and make sure you can compile a simple/trivial template class)

Part2 - Roster class:

Write a class, `Roster`, that **inherits from `ArrayBag`** and stores `Student` objects. The roster class should have <u>at least</u> the following <u>public methods</u>:

1. **`Roster(); //default constructor for empty roster`**

2. `/**parameterized constructor`
      `@pre the input file is expected to be in CSV`
          `(comma separated value) where each line has format:`
          `"id,first_,name_,last_name\n"`
      `@param input_file_name the name of the input csv file`
      `@post Student objects are added to roster as per the data`
            `in the input file`
      `**/`
     `Roster(std::string input_file_name);`

3. `/**@post displays all students in roster, one per line`
            `in the form "first_name_ last_name_\n"`
      `**/`
     `void display();`

4. You need to also overload `operator==` for the `Person` class, otherwise you will have a problem when the `add()` methods tries to compare two students to add them to the `Roster`. *This statement is not trivial, if you don't understand please ask!*

Testing:

Testing your modification to ArrayBag:

Before you move to part2, YOU MUST make sure that your modifications to ArrayBag work correctly. To do so write your own main function (not for submission) that does the following:
- Instantiate two ArrayBag objects that stores integers
- Add integers to the two bags (some integers should be common to the two bags)
- Call += on one of the bags, display its contents and make sure the operation worked correctly (i.e. bag1 ∪ bag2) and that your modification to add worked s.t. there are no duplicates. Also **be sure your `dispaly()`function displays the**

contents of the bag as specified in Part1 above (Gradescope will rely on `display()` to check that other functions worke correctly)
- Call `-=` on one of the bags, display its contents and make sure the operation worked correctly (i.e. bag1 – bag2).

Testing the Roster class:
Again in a main function (not for submission) do the following
- Instantiate a Roster object with the name of an input file
- Display the roster and make sure that all students in the file were added to the Roster. Again **be sure your `display()` function here has been overloaded to display Students EXACTLY as specified above** in Part2 above.

The input file will be in **csv** (comma separated value) format, and each line corresponds to the information necessary to create a `CourseMember` object. Each line in the input csv has the following format:
`id,fist_name,last_name`
You can find a sample input file named `roster.csv` on Blackboard under Course Materials / Project2

Review – reading the input:
In C++ to read input from a file you need a file stream

`#include <fstream>`

Since we are only reading input you can use an `ifstream` object.
Since we are reading from a csv file, every student is on a line. On each line, each data item is separated by a comma.
You may use `string::getline()` to read lines from the `ifstream`.

`#include <string>`

You may find it useful to use a stringstream to then read each piece of data (id, first_name and last_name) from each line you read from the input file.

`#include <sstream>`

You may use `getline()` to read data from the `sstream` as well. Remember that `getline ()` may take a delimiter. The default delimiter is '`\n`', but if you are reading comma-separated values you can use '`,`' as the delimiter.

```
getline(stream, variable, delimiter);
```

Don't forget to:
- Open the stream before reading.
- Check that opening the stream did not fail before reading, and output (`cout`) an error message if it does fail.
- Close the stream after reading.

Note: Reading from input file should be familiar from CSci 135. If you need to review, lookup `ifstream, sstream` and `string::getline()`
References for each of these are easily found online:
http://www.cplusplus.com/reference/fstream/ifstream/
http://www.cplusplus.com/reference/sstream/stringstream/
http://www.cplusplus.com/reference/string/string/getline/
If you need help with this even after having reviewed the documentation, please don't hesitate to ask for help from the tutors in labB or make an appointment with me. After this project you will be expected to be able to read from csv files.

Grading Rubric:

- **Correctness 80%** (distributed across unit testing of your submission)
- **Documentation 10%**
- **Style and Design 10%** (proper naming, modularity and organization)

Notes:
- I reserve the right to detract points given by Gradescope if your submission does not comply in some way with this specification.
- A submission that implements all   required classes and functions but <u>does not compile</u> will receive <u>40 points total (including documentation and design)</u>.

Submission:

For this project you   will submit **4 files: ArrayBag.hpp, ArrayBag.cpp, Roster.hpp, Roster.cpp**
You must submit all files. For partial credit, even if you only submit the ArrayBag class, you must also submit the Roster files, even if only a "dummy" or empty class.

Your project must be submitted on Gradescope.

Although Gradescope allows multiple submissions, it is not a platform for testing and/or debugging and it should not be used for that. You MUST test and debug your program locally.

Before submitting to Gradescope you MUST ensure that your program compiles (with g++) and runs correctly on the Linux machines in the labs at Hunter (see detailed instructions on how to upload, compile and run your files in the "Programming Rules" document). That is your baseline, if it runs correctly there it will run correctly on Gradescope, and if it does not, you will have the necessary feedback (compiler error messages, debugger or program output) to guide you in debugging, which you don't have through Gradescope.

"But it ran on my machine" is not a valid excuse to get credit.

Once you have done all the above you submit it to Gradescope.

The due date is Thursday June 13 by 6pm. No late submissions will be accepted.

Have Fun!!!!!