Project 1C: A Bag of ?

Part A and B of Project1 were pretty trivial, these were 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!!!**

With Part C things will get more interesting. Please expect the projects to become increasingly more involved and requiring more work/time. The goal of Part C is to get you comfortable working (and compiling) with templates and ADTs while reviewing some old concepts such as obtaining input from a file or from the user.

We will work here with the `ArrayBag` class discussed during lecture. You will first modify this class with additional member functions and then you will test it by adding "stuff" to the bag and testing that your member functions behave correctly.

You will find the `ArrayBag` class (as discussed in lecture) on Blackboard under Course Materials / Project1.
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.

Implementation:

You will modify the `ArrayBag` class to add the following **public** member functions (note that you can implement the following functions by calling other `ArrayBag` member functions):

```
/**
@param a_bag to be combinded with the contents of this bag
@return a new ArrayBag that contains all elements from this bag (items_)
and as many elements from the argument a_bag as space requires. Note
that it may contain duplicates
*/
ArrayBag<T> bagUnion(const ArrayBag<T>& a_bag) const;
```

Note: Because ArrayBag is of fixed size, bagUnion will copy the contents of the calling bag first and then whatever fits from the argument a_bag

```
/**
@param a_bag to be intersected with the contents of this bag (items_)
@return a new ArrayBag that contains the intersection of the contents
of this bag and those of the argument a_bag. This intersection does
not contain duplicates (e.g. every element occurring in BOTH bags will
be found only once in the intersection, no matter how many
occurrences in the original bags) as in set intersection A ∩ B
*/
ArrayBag<T> bagIntersectionNoDuplicates(const ArrayBag<T>& a_bag) const;

/**
@param a_bag to be subtracted from this bag
@return a new ArrayBag that contains only those items that occur in this
bag or in a_bag but not in both,
*/
ArrayBag<T> bagDifference(const ArrayBag<T>& a_bag) const;
```

Extra Credit:

```
/**
@param a_bag to be intersected with the contents of this bag
@return a new ArrayBag that contains the intersection of the contents
of this bag and those of the argument a_bag. This intersection
may contain duplicate items (e.g. if object x occurs 5 times in
one bag and 3 times in the other, the intersection contains 3
occurrences of that item)
*/
ArrayBag<T> bagIntersection(const ArrayBag<T>& a_bag) const;
```

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)

Testing:

Testing your modification to ArrayBag:
Although `BagTest` (described next) does not test your additional member functions to the `ArrayBag` class, you must do so thoroughly (Gradescope will!!!).
For testing union, intersection and difference, one possible (simple) way is to test them in main with integer bags.
Make sure you have different test cases as well as edge cases (e.g. empty intersection or empty difference, etc.)

Further testing of ArrayBag:
To further test the `ArrayBag` class you will write another class `BagTest (BagTest.hpp, BagTest.cpp)`. The purpose of this class is to instantiate an `ArrayBag` that contains `CourseMembers`. It will read an input file, create `CourseMember` objects and add them to the bag. It will also ask the user for a last name and will remove the first occurrence of `CourseMember` with that last name that it finds in the bag. (This class could be extended to test an `ArrayBag` with other types, but we won't do that here)

This class has one **private** data member:

```
ArrayBag<CourseMember> bag_;
```

This class has the following **public member functions**:

```
/**
 @pre the input file is in csv format as follows:
 "id,first_name,last_name,title\n"
 @post this function asks the user for an input file name.
 It extracts the information necessary to create a CourseMember object
 from each line in the input file, and it adds the corresponding
 CourseMember object to bag_.
 @return returns the populated bag_
 **/
ArrayBag<CourseMember> testCourseMemberBag();


/**
 @post asks the user for a last name and removes ONE CourseMember
 with that last name if it finds one
 @return returns the bag_ after removal if any
 **/
ArrayBag<CourseMember> removeCourseMemberFromBag();
```

```
/**
 @post prints to the standard output all CourseMeber
 objects found in bag_, one pre line if the format:
 id first_name last_name
 **/
void displayCourseMemberBag(const ArrayBag<CourseMember>& a_bag);
```

To break up the work, a suggested **private** member function:

```
ArrayBag<CourseMember> createBagFromInput(std::string input_file);
```

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,title`
You can find a sample input file named `roster.csv` on Blackboard under Course Materials / Project1
You can ignore the `title` entry for now, and use `id`, `first_name` and `last_name` to create `CourseMember` objects.

For this project you must also **add a default constructor to the CourseMember class.**

Extra Credit:

- overload `operator==` for the `CourseMember` class
- Implement the overloaded member function for the `BagTest` class:

```
/**
 @param member to be removed from  bag_
 @post removes one occurrence of member if found in bag_
 @return returns the bag_ after removal if any
 **/
ArrayBag<CourseMember> removeCourseMemberFromBag(const CourseMember&
member);
```

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 piece of information you will need is separated by a comma. One possible way to obtain each piece of information (you are welcome to explore alternatives) is to use the string function (`include <string>`)

```
istream& getline (istream&  is, string& str, char delim);
```

Where `is` is your ifstream, `str` is the string you are reading into (e.g. `first_name`) and you can use `','` as delimiter.
Notice that you have an extra piece of information on each line (title) that you can ignore for this project.

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.

Submission:

For the regular submission you will submit **3 files: ArrayBag.cpp, BagTest.hpp, BagTest.cpp**

There will be a separate submission if you choose to do the extra credit. You must still submit to the regular submission to obtain the project credit and submit again for the extra credit. The extra credit submission will require **5 files: ArrayBag.cpp, BagTest.hpp, BagTest.cpp, CourseMember.hpp, CourseMember.cpp**

Your project must be submitted on Gradescope. Submission will open 48 hours before the due date. The due date is Friday February 22 by 6pm. No late submissions will be accepted.

Have Fun!!!!!