Project 2: Array Bags and Sets

Background:

You have finally learned about your first ADT, the array bag. What you should have taken away from this topic is that the purpose of an abstract data type is that it is a logical description of how various users will be able to view the data and the operations that are allowed without regard to how they will be implemented. This means that a majority of people will be concerned only with what the data is representing and not with how it is to be constructed. By providing this level of abstraction, we are creating an **encapsulation** around the data. The idea is that by encapsulating the details of the implementation, we are hiding them from the user's view. This is called **information hiding.**

Project 2 will make you work with ***Templates and ADTs.*** After completion of this project you must be very comfortable with compiling multiple classes as well as template classes (i.e. you do not compile them!!!) into one program and instantiating objects of a template class for multiple data types. You will also be dealing with examples of operator **overloading**.

Implementation:

- ***You will also be given an hpp for `ArrayBag`, a template class which you must implement (you will write the cpp yourself)*** based on the comments given in the hpp for each function.
- Note that this ArrayBag class will be very similar to the one you learned in class, some of the functions will be identical so feel free to use the lectures as a guide on how to implement those functions. But you will also have new functions that were NOT covered in class. It is your job to use your knowledge of Array Bags to implement each of these functions how you best see fit.
- You will encounter operator overloading. This sounds complicated but all you are doing is making the **+=**, -= and /= work with your ADT you made.
  If we had 2 array bags: `ArrayBag<int> x` and `ArrayBag <int> y`, I will now be allowed to do `x+=y;`

Additional resources:

• ***Abstract Data Types:***
  ○ Geeks for Geeks
  ○ Neso Academy
	• ***Template Classes:***
	  ○ CPP Manual
	  ○ Geeks for Geeks
	  ○ Tutorials Point
		• ***Operator Overloading:***
		  ○ CPP Manual
		  ○ Geeks for Geeks
		  ○ Programiz

Task: Implementing the `ArrayBag` functions!

Note all the functions you will need to implement and explanations for each are provided for you in the hpp so we won't go too in depth into what you need to do for each function.

What is operator overloading:

C++ allows you to specify more than one definition for a **function name** or an **operator** in the same scope, which is called **function overloading** and **operator overloading** respectively.

Please make a note: This is different than ***overriding*** which you ran into in project 1. When you override you replace how the function works for a particular class. When you ***overload*** you are giving the function multiple different uses so the compiler can determine the most appropriate definition to use by using context clues and comparing the argument types you have used to call the function or operator with the parameter types specified in the definitions.

An example of the += operator being *already* overloaded is when you use it with an integer vs. when you use it with a string.

When I have 2 integers:

```
int x = 5;
int y = 3;

x += y;
```

This sets the value of x to 8 now because += used with integers lets you add numbers.

But what if I had 2 strings?

```
std::string x = "Hello ";
std::string y = "World";

x += y;
```

Now x will be set to the value "Hello World" because += used with strings lets you *concatenate* strings.

So what are we doing with +=, -= and /= for ArrayBags? We are doing some basic set theory. You didn't cheat during discrete mathematics… right?

- += is the union of 2 array bags (adding contents of bag B to the end of Bag A)
- /= is the intersection of 2 bags (Bag A will have contents that only appear in **both** bag A and bag B)
- -= is the set difference of the 2 array bags (Bag A will get rid of anything that's in Bag B)

Picturing a venn-diagram may help for some of these concepts, for example for += you can image Bag A will now have contents of the entire venn-diagram while with /= Bag A will now only contain what's in the overlapping middle section of the venn-diagram.

Note: We are using the terms bag *A* and bag *B* but here's what each refers to:

```
x += y;
```

Given that x and y are both array bags, x would be bag *A* and y would be bag *B*. So here we are adding contents of bag B (y) to that of bag A (x). Any further questions feel free to ask during office hours!

Testing

How to compile:

```
g++ <test main file> -std=c++17
```

You must always implement and test your programs **INCREMENTALLY!!!** What does this mean? Implement and test one method at a time.

- Implement one function/method and test it thoroughly (multiple test cases + edge cases if applicable).
- Implement the next function/method and test in the same fashion. **How do you do this?** Write your own `main()` function to test your class. In this course you will never submit your test program, but you must always write one to test your classes. Choose the order in which you implement your methods so that you can test incrementally: i.e. implement mutator functions before accessor functions. Sometimes functions depend on one another. If you need to use a function you have not yet implemented, you can use stubs: a dummy implementation that always returns a single value for testing. Don't forget to go back and implement the stub!!! If you put the word STUB in a comment, some editors will make it more visible.

Grading Rubric

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

Important: You must start working on the projects as soon as they are assigned to detect any problems with submitting your code and to address them with us **well before** the deadline so that we have time to get back to you **before** the deadline. This means that you must submit and resubmit your project code **early** and **often** in order to resolve any issues that might come up **before** the project deadline.

There will be no negotiation about project grades after the submission deadline.

Submission:

You will submit the following file(s)**:** `ArrayBag.cpp`

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 argument for a submission that does not compile. Once you have done all the above you submit it to Gradescope.