ETEC2101 **Lab2: Templates and Operators** Points: **100**

1. (**20 points**) Make the optimization we discussed in class (rather than just growing by one, double our *capacity* when we run out of space; shrink capacity by half if we remove enough elements). Have a static const member that determines the minimum capacity of the array (I used 5)
2. (**20 points**) Convert our ArrayList class into a general-purpose templated class. Remember: this means that array_list.cpp is no longer necessary (unfortunately)!
3. (**20 points**) Create / modify these operators / operations:
   a. Add a capacity getter (similar to the size method)
   b. Make all methods const-safe that can be
   c. Replace our get method with a [] operator. Return a *reference* to the object (not a copy)
   d. Make the stream output operator (<<) work for all ostream-derived output objects (ofstream, stringstream, cout, etc.). Do this with an inline non-member friend function.
   e. Overload the = operator and also make a copy-constructor that will make a "deep-copy" of our ArrayList. Make sure to use const-references for the “other” ArrayList here
4. (**20 points**) Create a “normal” Foo class in the “testing” namespace. This Foo class should
   a. Be split across a .h and .cpp file (since it’s not templated)
      i. Note: it is recommened to define bodies like this in a cpp file like this
      `void testing::Foo::get_value() { /* … */ }`
      …As opposed to wrapping the entire cpp file in a namespace curly-brace
   b. Have an int and string attribute
   c. Have a default constructor (set the attributes to 0 and “”) *and* a constructor that takes initial values for both attributes.
   d. [No destructor necessary since we’re not dynamically allocating anything]
   e. Have getters and setters for both attributes
   f. Overload the stream operator and product output of this form:
      `<FOO:7:Bob>`
   As a programming exercise, I’d like you to do this using “Method 1” from the slides (and just put a declaration in the .h file, body in the .cpp file)
5. (**20 points**) Revise and expand our basic tests[1] from Lab1 (the file I/O part is not necessary in this lab and can be removed). Test our code with something besides strings: here I want you to do a float test and also a test that uses a custom type (just make a super-simple Foo class with 2 data members [I used an integer and a string]). Make sure to test all new functionality.
6. (**+20 points**) Add the ability to manage a “database” of Foo objects using a menu like we did in Lab1.

[1] In this class (and most of the software world), even if the code is completely correct, if you don't have a test for it, it's always considered only partially complete.