| ETEC2101 | **Lab1: Create the project and start ArrayList** | Points: **100** (there are 120 points possible) |
|---|---|---|

*Our first data structure will be made in several phases. This first phase is mainly meant to (re-) familiarize you with C++ class constructs and to introduce you to C++ string and stream operations. A note about my teaching style: I attempt to show you everything I think you'll need to successfully complete a lab. But if you don't understand something in a lab spec sheet, it's your responsibility to ask.*

1. (**10 points**) Create a new console application project, and do the project re-organization as discussed in sections 1b-1g of the "C++ guide" (on blackboard)
2. (**75 points**) Make an **ArrayList** class
    a. in the **ssuds** (SSU Data Structures 😊) namespace
    b. (for now) make only the class declaration in array_list.h (and put in the include folder) and the body in array_list.cpp (in the src folder). Once we switch to using templates (Lab2), the cpp file will unfortunately need to go away.
    c. The only (protected) attributes I want you to store are:
        i. **mData**: an array (not vector or any other array-like contstruct) of std::string's
        ii. **mSize**: the current size of the array (unsigned int)
    d. There should be one (public) constructor (a default constructor that takes no arguments) that initializes the attributes to appropriate starting values.
    e. There should be a destructor that frees up any memory the ArrayList currently has allocated.
    f. There should be these additional (public) methods:
        i. **append**: takes a string as an argument and appends it to the end of the internal array. Note: since we have no excess capacity (that will change in lab2), we'll have to re-allocate the array every time we do this (or insert)
        ii. **size**: a simple getter for mSize
        iii. **insert**: takes a string to insert as the first argument and an unsigned int index for where to insert that element in the array. If the index is 0, this should be the new first element. If it is the length-1, it should be added to the end. If it is anything outside that range, throw a std::out_of_range exception.
        iv. **clear**: similar to the destructor (which is called when the instance of ArrayList goes out of scope, or if a dynamically created instance is deleted), but meant to be called during the lifetime of the object. Should reset the ArrayList back to its default state.
        v. **get**: should take an index as an argument. Should raise an exception if that index is bad. Otherwise, return the string at that spot.
        vi. **find**: should take a string (that we're searching for) as an argument and an optional starting index to search for. Should return the index of the first occurrence after the starting index, or -1 if we cannot find that string.
        vii. **remove**: should take an index. Raise an exception if the index is bad. Otherwise, remove the given element and collapse the array so it just holds the remaining elements.
        viii. **remove_all**: should take a string as an argument and should use the find and remove methods to remove all occurrences of that string (if any). Should return the number of instances removed.
3. (**10 points**) for good style and documentation of your ArrayList class. Use doxygen-style comments in the .h file (class, attribute, and method docstrings)
4. (**10 points**) Design a simple main program that tests out the functionality in your ArrayList class. Print out (using cout) the output you're using for testing and put a comment beside it for the desired output. For example, this might be the start of your program:

```
ssuds::ArrayList tester;
tester.append("Bob");
tester.append("Carl");
std::cout << tester.size() << std::endl;       // 2
std::cout << tester.get(0) << std::endl;       // Bob
std::cout << tester.get(1) << std::endl;       // Carl
// …
```

5. (**15 points**) the Application:
    a. in main (it's OK if you want to put this code right after your #4 test code), open a text file (which is either not there, or a text file containing one string per line). Store all of the strings (if any) in an ArrayList.
    b. present the user a main menu that allows them to call the major ArrayList methods. Call that method.
    c. Save out the results to the same text file when the user indicates they want to exit the loop.
6. Zip your entire project folder and submit on blackboard (before the due date!)