CMPT 225 - Assignment 1 - FriendsBook

Please, read the entire assignment first before starting it!

Please, read the Submission part of this assignment to determine what constitutes Part 1 and Part 2 of this assignment!

This assignment is to be done on your own!

Objectives

The objective of this assignment is for you to practice ...

- Designing and implementing a data collection as an abstract data type (ADT) class, while satisfying a set of given requirements.
- Applying the four steps of the software development process while creating an object-oriented (OO) solution to a problem.

Step 1 - Problem Statement and Requirements

In this assignment, we are asked to develop a FriendsBook application (a simple social network), maintaining the profile of its members.

Each member of this social network must have a profile that contains the member's username, name (first and last name), email address and birthday.

The FriendsBook application must allow a member (or member-want-to-be) to ...

- Join the social network (by creating a profile for this new member).
- Search for a particular member (profile).
- Modify the member's profile: modify the name and/or the email address and/or the birthday of the user, **but not the member's username**.
- Leave the social network (by removing the member's profile from the network).
- Print all the current members in ascending order of username. Here (SamplePrintResult.txt) is an example of how the output of the "print" option must look like (i.e., its format).

NOTE: Of course, the above behaviour does not quite match how real social network applications work (for example, no members of a real social network application have access to the username of other members), but this is good enough for us to achieve the objectives of this Assignment 1.

Step 2 - Design and Step 3 - Implementation

In this assignment, the design step, in which you establish the classes you shall need in your software solution, has been done for you. Indeed, you shall need two C++ classes and a file containing the **main** function:

1. The class that models each member of this simple social network is called Profile class. It is an ADT class. This class has already been created. Download Profile.h and Profile.cpp, open both files and carefully read their content. You must understand how this Profile ADT class works in order to complete MyADT class (see below) and to use this Profile class in your solution to the FriendsBook problem.

   Important: You do not have to submit this class, which signifies that you cannot modify it.

2. The data collection class that manages all the Profile objects: MyADT class.

   - Download the partially completed MyADT.h and MyADT.cpp and carefully read their content. The documentation these two files contain will help you complete the design and implementation of this class.

     Important: As you complete this class, you cannot remove/modify the code that is already in MyADT.h and MyADT.cpp.

   - To complete the design and implementation of this class, you will also need to satisfy the following requirements. But, first, let's define a few terms:
     - The letter **n** represents the total number of Profile objects in the data collection MyADT.
     - The letter **m** represents the number of Profile objects in MyADT that have a username starting with the same first letter.
     - For example, if we store Profile objects with the following usernames (listed here in no particular order): "zythba", "abcdef", "tttottt", "anotheruser", "trotrutra", "againanotheruser", then **n** would be 6, **m** for the letter "a" would be 3, **m** for the letter "t" would be 2 and **m** for the letter "z" would be 1.

   - Here are the requirements you need to satisfy when completing the design of this class and implementing it:
     - The **insert** method of MyADT class must have a time efficiency of O(m).
     - The **search** method of MyADT class must have a time efficiency of O(m) or O($\log_2$ m).
     - The **remove** method of MyADT class must have a time efficiency of O(m).
     - In terms of modifying a Profile object, one needs to first search for this object (time efficiency of O(m) or O($\log_2$ m)) then one performs the required modification in O(1). Note that there is no need to create a **modify** method for MyADT class.
     - The **print** method of the MyADT class must have a time efficiency of O(n).

       Question: What does this imply in terms of the **insert** method of MyADT class? In other words, how is the **insert** method of MyADT class to behave in order for the **print** method of this class to have a time efficiency of O(n)?

   - You can only use arrays as the concrete data structure (CDT) of MyADT class. You cannot use linked lists.
   - You cannot use 26 "if" statements in your implementation of the methods of MyADT class. This would qualify for "repeated code".
   - You cannot use a switch statement with 26 switches in your implementation of the methods of MyADT class. This would also qualify for "repeated code". Why is repeating our code a bad thing?
   - Your default constructor cannot call the **new** operator MAX_ALPHA times. Unfortunately, this may be perceived as wasteful! Only call the **new** operator when you need heap memory.
   - If the description of a method does not say anything about printing, then make sure your implementation of this method does not print anything.
   - You must submit your completed **MyADT.h** and **MyADT.cpp**.

3. And the social network FriendsBook application file that contains the **main** function and other functions: FriendsBook.cpp. Download this file and read its content. As you will see, this program has already been implemented and is ready for you to use in order to solve the FriendsBook problem. You do not have to submit this file, so, feel free to use/modify it as you wish.

Make sure that each of your files ...

- Contains a comment header block composed of filename, class description, class invariant (if any), author, date of creation/last modification.
- And that all the methods (public and private) of your classes have appropriate documentation: a description, a precondition (if any) and a postcondition (if any). Most of this documentation has already been provided for you. Make sure you understand its purpose.

  Note: The same documentation must appear in the class header file (.h) and in the class implementation file (.cpp) of each of your classes.

Finally, your solution must ...

- Not make use of C++ STL, i.e., Standard Template Library. More specifically, your solution cannot make use of data collections (or containers), such as the vector class, defined in STL as well as algorithms, such as the sort(...) function, defined in STL.
- Not make use of struct.
- Satisfies the *Good Programming Style* described on the Good Programming Style web page of our course web site.

Step 4 - Compilation and Testing

In this assignment ...

- You will need to create test cases for your MyADT class and implement them in a test driver. Your test cases and test driver must conform to the description of test case and test driver given in our lectures and labs.
- Once you have written this test driver, test your MyADT class separately from the FriendsBook application by using your test driver. This type of testing (testing a class *in isolation* using a test driver) allows you to more easily verify that your implementation of the MyADT class is working and satisfies the requirements described above.
- Once you are satisfied with your class implementation, you can then expand your testing to the FriendsBook application, i.e., test all your classes together and ensure they solve the problem stated in the Problem Statement in Step 1 above.
- In order to create the necessary executables to perform the testing described above, you must use this makefile. This makefile compiles the files you create in this assignment on our ***target machine*** and creates two executables:
  - **td**, which is short for **test driver**: allows you to test your MyADT class.
  - **fb**, which is short for **FriendsBook**: allows you to test your friendsBook application.

  You must not modify the content of this makefile. If this makefile does not work for you, make sure you are using it on our ***target machine***. If it still does not work for you, please, see the instructor.

Marking Scheme

When marking Assignment 1, we shall consider some or all of the following criteria:

- Compiling: Does your code compile on the ***target machine*** using the supplied makefile?
- Execution: Does your code solve the problem stated in this assignment?
- Important: In this Assignment 1, 50% of the marks is given to the above two criteria. This signifies that if your code does not compile, you cannot get more than 50% for Assignment 1.
- Requirements: Does your solution satisfy the requirements described in this assignment?
- Coding Style: Has *Good Programming Style*, described on the Good Programming Style web page of our course web site, been used?
- Documentation: Does your solution satisfy the documentation requirements described in this assignment?
- Note that you cannot use code that has not been written by you in the context of this course (this semester) or provided by the instructor.

Marking

When marking Assignment 1, the TAs will follow the testing process described in our Lab 0, Lab 1 and Lab 2, namely, they will use a script, like the **test.py** of Lab 0 and Lab 1, along with input files (*.in), if appropriate, and expected result files (*.er). The script will compare the results/output of your code against the expected results. The success of this comparison will determine whether your code passes or fails the test cases.

For this reason, you must implement your code such that it produces results/output that exactly matches the requirements listed in this Assignment 1. For example, if four members were to join your FriendsBook application, i.e., the four members illustrated in the file **SamplePrintResult.txt** described above, and you were to print all the members of your FriendsBook application afterwards, the output your FriendsBook application (which makes use of your MyADT class), i.e., the listing of its members, would be expected to exactly match the content of the file **SamplePrintResult.txt** in order to pass the test case (since the content of this file represents the expected result of the **print** operation).

Finally, you may wonder "why can I not remove/modify the code already found in the provided files?"

The answer is that, when we mark Assignment 1, we will test your submitted class using a test driver program (client code) that will be implemented using the public interface of the Profile class and the MyADT class. If you change the provided public interfaces (for example, if you change the type of a parameter from **float** to **int**), the test driver program used for marking your assignment will be expecting a **float** and will no longer compile with your submitted files, which are expecting an **int**, and marks will be deducted.

Remember our *target machine*?

You can use any C++ IDE you wish to do your assignments in this course as long as the code you submit compiles and executes on our ***target machine*** (Ubuntu Linux, C++ and g++ versions running on the CSIL workstations).

Why? Because your assignments will be marked on the ***target machine***.

Submission

- **Assignment 1 Part 1** is due on CourSys on Friday, May 24 at 23:59:59. For Part 1, you need to submit **MyADT.h**, **MyADT.cpp**.

  Hopefully, we will be able to mark and give you feedback on your MyADT class before the due date of Assignment 1 Part 2.

  Whether or not we are able to do this, you will be able to modify (if modifications are needed) your MyADT class after the first submission (Assignment 1 Part 1 submission) and resubmit it as part of your second submission (Assignment 1 Part 2 submission).

- **Assignment 1 Part 2** is due on CourSys on Friday, May 31 at 23:59:59. For Part 2, you need to submit your modified (if modifications were needed) **MyADT.h** and **MyADT.cpp** as well as the test driver you wrote to test your MyADT class. This file must be named **MyADTTestDriver.cpp**.
- Late assignments will receive a grade of 0.