CS 104 (Fall 2013) — Assignment 8
Due: 11/14/2013, 11:59am

BitBucket directory name for this homework (case sensitive): HW8

This homework is the third part of your long-term Social Networking Site project. It will be graded as part of the project (rather than a regular homework), and thus cannot be dropped. You should be building on top of your code for HW3/HW6, though this would also be a good time to fix some things you did back then that you would like to improve. (If you feel it is necessary, you can also start from scratch.)

You may now (if you want) replace your own home-grown `Bag`, `List` and similar classes with suitable C++ STL versions. If you do so, we strongly recommend that you replace one occurrence at a time. Replace one occurrence, then debug, and make sure that everything works again. Only then move on to the next occurrence. Otherwise, you may find so many problems at once that you feel overwhelmed.

In this and future homeworks, we will specify where you are allowed to use STL classes, and where you should use your own.

(1) Review C++ Interlude 7 and Chapters 11, 13, 17 from the textbook.

(2) Implement a way for a user to post on a friend's wall. More specifically, you should have the following functionality:

(a) A user can create a new post on a friend's wall.

(b) A user can delete his/her own post on a friend's wall.

(c) A user can delete a friend's post on his/her wall.

To implement this, you should change your `Wallpost` class so that each `Wallpost` now also stores who made the post, whose wall it was on, and when the post was made.

(3) Implement a way for users to comment on wall posts on their own and their friends' walls. Users should also be able to delete their own comments on posts (but not those of others). When a post is displayed, it should be displayed with comments, and the comments should be sorted by when they were made. You can achieve this either by storing them in this order, or using the sorting function you will implement for the next question. (In the latter case, you'll probably need to create a template version of your sorting function.) Comments should also be saved along with the original post, so they are available when the program is restarted.

You probably want to first create a class `WallpostComment` (or a similar name), with suitable fields and methods. Then, you should change your `Wallpost` class to also store a `list` (or `vector` or similar) of `WallpostComment`.

(Hint: you are welcome to learn about and utilize the functions in the `chrono` header of C++; they may save you some time for the task of sorting posts/comments by when they were made, and storing information about times.)

(4) Implement a way for users to see their friends' walls. A user should be able to select a friend from his/her list of friends, and view the friend's wall.

The user should be able to choose between (at least) two orders:

(a) Sort posts by when the post was originally made (most recent first).

(b) Sort posts by when the most recent activity was made on the post (most recent first). An activity is either making the post, or someone commenting on it.

(If you would like, you can also implement additional orderings.)

To do this, you will need to implement a sorting function, and you should implement your own instead of using STL for now. You will get full credit if you use one of Quick Sort, Merge Sort, or Heap Sort, and most of the credit for one of Selection Sort, Insertion Sort, or Bubble Sort.

Since you will be sorting by multiple different criteria, you will want to be able to pass a sorting criterion into your sorting function as a parameter. There are two ways to do this (which will also be briefly covered in lab this week):

- C++ allows you to pass functions as arguments (usually as pointers). You can declare your `sort` function to accept a function `bool isGreaterThan (const User & user1, const User & user2)`, and then pass different functions in as `isGreaterThen`, which do different types of sorting.
- If you are spooked by the idea of passing a function (although there's nothing too weird about this), you can instead put your function inside an object; such objects are often called *Comparators*. You can define a pure abstract class

```
class Comparator {
  public:
    static bool isGreaterThan (const User & user1, const User & user2) = 0;
}
```

  and then implement two versions of this pure abstract class. You can then pass the corresponding two objects (say, of type `SortByPostComparator` and `SortByCommentComparator`) as arguments into your function `sort`, which has as one of its argument a `const Comparator & mySortingRule`.

(5) Generate at least 10000 users with about 100 friends and some posts each for your social network. If you are truly masochistic, you may do this by hand. Otherwise, we recommend writing a little program (that uses some of the classes and functions from your project) to generate them. It is ok if your users are named "xiayf7eb hfjc8syb" and similar things. (Though if you prefer, you can also use lists of names to generate more realistic ones.) Make sure that your social network's functions (such as reading in all users, searching for friends, etc.) all still work at this scale.

(6) This problem is completely optional. It does not count towards your grade and doesn't earn chocolate. But we figure that some of you may enjoy it.

Add a nicer graphical user interface to your social network, using your newly acquired knowledge of QT.

(7) Chocolate Problem: This problem does not count towards your grade, but if you solve it, you can earn up to 3 chocolate bars. Feel free to specify your preferred type of chocolate — your requests may be accommodated if they are reasonable. Submitting this problem a few days late is ok, but in that case, please submit it directly by e-mail to David.

Write a QT-based testbed for learning about sorting algorithms and displaying their behavior visually. Here is an outline of the desired functionality and implementation approach:

(a) Implement a class `SortingArray`, which gives an array whose elements can only be accessed in the following ways:

- The entire array can be read from a given file.
- The array can be initialized to a random ordering of the numbers $\{1, \ldots, n\}$.
- Two entries can be compared via a function `bool lessThanOrEqual (int i, int j)`.
- Two entries can be swapped via a function `void swap (int i, int j)`.

In particular, there should be no `get` or `set` function (or an overloaded `[]` operator). The point of doing it this way is that the array can keep a count of how many comparisons and swaps have been executed. (And those can be reset when the array is read from a file or randomly ordered.) It may be a good choice to implement the first two as constructors.

(b) Implement the standard sorting algorithms (Insertion, Selection, Bubble, Merge, Quick, Heap) to operate just on the `SortingArray` type.

(c) Use QT to build a visual user interface. It should display the current array contents as dots, in the way we have been drawing arrays in class. Have the `SortingArray` and your user interface interact in such a way that whenever a `swap` is executed, the drawing is automatically updated. (The best way to do this may be to have the `SortingArray` get a pointer to the QT object that is in charge of drawing it, and have it call its `draw` function. For implementing `draw`, you probably want to build an iterator of your `SortingArray`.

(d) The user interface should at least support the following:

- Display the number of swaps and comparisons executed.
- Let the user choose the array size to sort, and whether to randomize or read from a file.
- Allow the user to speed up or slow down the execution, as well as pause it, so as to be able to see what is happening.