CS 104 (Fall 2013) — Assignment 3
Due: 09/26/2013, 11:59am

BitBucket directory name for this homework (case sensitive): HW3

For this homework, and all other subsequent ones, you should be using Makefiles. This will be the last homework for which submission through Blackboard is allowed. Starting at Homework 4, all homeworks must be submitted through BitBucket.

This homework is the first part of your long-term Social Networking Site project. It will be graded as part of the project (rather than a regular homework). Since you will be reusing and improving and changing this code a lot, we strongly recommend that you design it cleanly and comment it well. You will thank yourself in a few weeks.

Also, we recommend that you read all of the questions before attempting any one. The questions build up toward a coherent whole, so how you solve one question may impact how you solve another one.

We strongly recommend that you create separate files for each of the classes you will write. Also, you should separate your header files from the code for each of them, and follow good practice in terms of `#include` (see our coding guide on the course web site).

(1) Review C++ Interludes 1,2,4 from the textbook.

(2) Implement your own `template` doubly linked list `class`. You are welcome to use your own code from Homework 2. Your class should at least have the following functions:

(a) a constructor (or multiple, if you want) and destructor.

(b) a function for adding an element.

(c) a function for removing an element.

(d) some way of going through all elements, e.g., by returning the `head` element.

(3) Create a `class WallPost` for a wall post. At the least, your class should contain the following fields (which should be `private`):

(a) a field for a string (what the post says)

(b) one more field of your choice (name of the author, date and time it was made, or something else you think of).

Your class should have at least the following functions:

(a) a constructor (or multiple, if you want) and a destructor.

(b) functions to read and change the values of the private fields.

(c) a function to return the entire wall post (string and the other field) as a `string` in a nicely formatted way, for printing.

(Later in the class, you will want to add things like responses to wall posts, visibility to other users, and more. We're mentioning this now to justify why you need an entire `class` here, rather than just some strings.)

(4) Implement a `class Wall` for a user's wall. Your class should internally contain a linked list (your own implementation) of `WallPost` elements. Your class should have at least the following functions:

(a) a constructor and a destructor.

(b) a function for adding a new wall post.

(c) a function for removing a wall post.

(d) a function for writing the entire wall to a properly formatted string.

(e) a function for reading a wall from a properly formatted string.

For now, you don't have to worry about misformatted walls.

(5) Implement a `class User` for a user of your social network. It should internally contain a `Wall`, and private variables for the user's name and password, and at least one more item of your choice (such as birthday, address, university, or others). At least, it should have the following functions:

(a) a constructor and a destructor.

(b) functions to edit the user's private data field, where it makes sense.

(c) a function for adding a wall post by the user on his/her own wall (for now, no one can post on anyone else's wall).

(d) a function for deleting a wall post by the user on his/her wall.

(e) a function for giving a string containing all of the user's information in a nice format. (Yes, that includes even the password for now — you'll fix that later.)

(f) a function for reading the user's data from a properly formatted string. If you want, that function can just be a constructor.

Notice that for now, users don't have friends. So our "social network" isn't very social, or much of a network, yet. We'll fix that in later assignments.

(6) Implement a `class UserList` for a database of users. Internally, it should be implemented as a linked list of `User` entries (using your own Linked List implementation). At least, it should have the following functions:

(a) a constructor and a destructor.

(b) a function to add a user.

(c) a function to delete a user.

(d) a function to write the entire formatted user list (including all data) to a file.

(e) a function to read an entire (correctly formatted) file and turn it into a user list.

(7) Implement a simple text-based (`cin, cout`) user interface for your social networking site. When you start the program, it should first read all the information about existing users from a file. It should allow you to:

(a) Log in as a user. For this, you will probably want to add functionality to your `UserList` class so that you can find a particular user. When the user logs in (entering a name and password), this means that the user will enter the menu described below, and have access to his/her posts, add to them, etc.

(b) Create a new user. When you do this, make sure not to create another user with the same name as an existing one. After the new user is created, you can either make the program exit, or return to this menu.

(c) Quit the program.

When a user logs in, your program should allow the user to:

(a) Display all his/her existing wall posts.

(b) Create a new wall post.

(c) Delete an existing wall post. Think about a good way in which the user can select the post to delete.

(d) Change any of the fields that you think should be changeable.

(e) Log out. When a user logs out, all the changes that were made should be stored in the file so they are available next time.

The user interface you write can be simple, but it should be safe. Your program should not crash just because someone enters garbage at the prompts. That is, make sure to check for legal inputs. (However, you don't have to guard — for now — against someone tampering with your data file.)

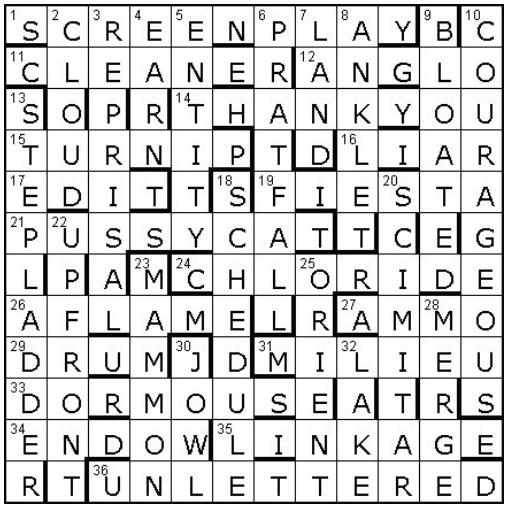
(8) Chocolate Problem: This problem does not count towards your grade, but if you solve it, you can earn up to 2 chocolate bars. Feel free to specify your preferred type of chocolate — your requests may be accommodated if they are reasonable. Submitting this problem a few days late is ok, but in that case, please submit it directly by e-mail to David.

Use backtracking to implement a crossword puzzle generator. The user should specify the shape and where the words go For examples, see Figure 1 below. The first example shows a shape with lines separating words, while the second one shows an input with blacked-out squares. The input should *not* contain the letters (or clues), just the shapes. It's the job of your program to fill in letters. Feel free to define any way to input the shapes that you like. The easiest is probably some text file format in which you encode the type of square (blocked, white, thick lines on which sides, ...) in different letters. But if you prefer to design an interactive user interface, or read JPGs, or anything else, that's fine by us.

Once you've read in the shape, use recursion/backtracking to find a solution with only legal words. You probably want to create a file containing the words to use. Hint: it's much easier to find legal crosswords when you have a big file of words. If your vocabulary is only 50–500 words, it's much harder. If you Google, you'll probably be able to find a bunch of dictionaries for English or other languages, in some useful text file format.

You don't need to provide clues for the words. Also, you don't need to write a user interface that lets people actually solve the crosswords you generate. Of course, if you find this entertaining, we won't stop you.

Document your program and user interface well enough that we can actually test it.



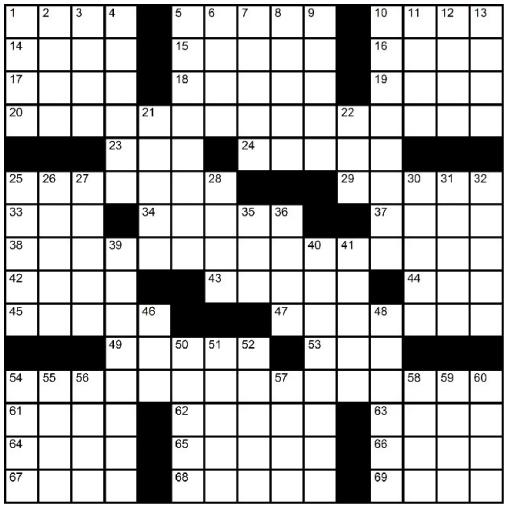


Figure 1: Example Crossword Shapes