Homework 3: Posts and Inheritance

CSCI 62, Fall 2024

1 Introduction

In this assignment, you will augment your social network by adding posts to your social network. You will:

1. Create two new classes, one of which inherits from the other
2. Use file I/O to read in a set of posts

There will be two kinds of posts displayed on someone's page: posts they have written on their own page (owner posts), and posts that others have written on their page (incoming posts).

1.1 Deliverables

Upload the following to Gradescope:

1. Your code: `user.h`, `user.cpp`, `network.h`, `network.cpp`, `post.h`, `post.cpp`, and `social_network.cpp`
2. Your `Makefile`
3. Your screenshot for Section 8.1.
4. You short answers `.txt` file

1.2 Autograder

Here is a link to the autograder tests: https://github.com/abacadaea/csci62/tree/main/hw3/autograder_tests

1.3 Requirements

- For this assignment, you will add two classes, `Post` and `IncomingPost`, which are both declared in `post.h` and both implemented in `post.cpp`.
- Declare *both* of `Post` and `IncomingPost` in `post.h` and implement *both* in `post.cpp`. Do not use a separate `incoming_post.{h,cpp}` for `IncomingPost`, or else the autograder will not detect it.

- Please implement the methods and fields exactly as described.
- You may include additional private methods if you desire.
- You may NOT use global variables.

2 Update your Makefile

Start by creating empty `post.h` and `post.cpp` files. Update your makefile: (1) add a target `post.o` that partially compiles `post.cpp`, and (2) update your executable target to link with `post.o`. Running `make` should compile successfully with the empty `post.h` and `post.cpp` files.

3 Post Class.

This represents a post on the profile page of some user (the "owner"). By default, the author of a post is the owner (`IncomingPost` allows for the author to be someone else). An overview is given in the diagram below, where + denotes public methods and – denotes private fields and the type or return type is give after the colon.

```
Post
+ Post()
+ Post(messageId: int, ownerId: int, message: string, likes: int)
+ toString(): string
+ getMessageId(): int
+ getOwnerId(): int
+ getMessage(): string
+ getLikes(): int
+ getAuthor(): string
+ getIsPublic(): bool
- messageId_: int
- ownerId_: int
- message_: string
- likes_: int
```

This object represents a Post made by User `ownerId_` on their own page.

- The `messageId_` is the id of the message.
- The string `message_` is the content of the post.
- The `likes_` counts the number of users who have liked the post.
- The parameterized constructor should take four parameters, in the order specified: `messageId`, ...
- The `toString()` should return a string "`[message_]` Liked by `[likes_]` people." where `[message_]` and `[likes_]` are replaced by their respective values.

- The methods `getMessageId`, `getOwnerId`, `getMessage`, `getLikes` are all getters.
- The methods `getAuthor` and `getIsPublic` should be virtual functions. `getAuthor` should return "", and `getIsPublic` should return true.

4 IncomingPost Class.

This represents a post by someone else on the owner's page. IncomingPost should inherit from Post. An overview is given in the diagram below, where + denotes public methods and – denotes private fields and the type or return type is give after the :

```
IncomingPost
+ IncomingPost()
+ IncomingPost(messageId: int, ownerId: int, message: string,
               likes: int, isPublic: bool, author: string)
+ toString(): string
+ getAuthor(): string
+ getIsPublic(): bool
- author_: string
- isPublic_: bool
```

- The primary constructor should take 6 parameters; one for each of the fields.
- `getAuthor` and `getIsPublic` are getters.
- `toString()` should return a string "`[author_]` wrote`[private]`: `[toString]`" where `[author_]` is the value of `author_`, `[toString]` is the value obtained by calling the Post `toString` method, and `[private]` is the empty string if `isPublic_` is true, and " (private)" if `isPublic_` is false.

5 Add to User class

A new private field `messages_` that is a vector of pointers to Posts. You may assume that the Posts are in the vector in chronological order. I suggest allocating memory for your Posts on the heap. Note that both the incoming posts AND the owner posts will be in the vector. Add to User:

```
+addPost(Post*):void
+getPosts(): vector<Post*>
+getPostsString(howMany:int, showOnlyPublic:bool):string
```

- getPostsStrings returns a string that holds the most recent howMany posts (or all posts, if there are less than howMany), according to the following instructions.
  - Your return string will consist of the concatenation of the correct number of Posts' strings, where the individual Post strings are separated by two newline characters.

  - To generate each Post's string, call each object's `toString()` method. For Post objects you should call the Post `toString` method, and for IncomingPosts you should call the IncomingPost `toString` method (you need to change one of the methods for this to be possible. What keyword should you use?).
  - You should put the most recent posts first (the ones with the highest messageIds). You may assume that the Posts for each user are stored in increasing order of messageId.
  - Depending on the value of the `showOnlyPublic` parameter, you may or may not include private IncomingPosts in your result (assume ordinary Posts are always public).

- You may NOT use `dynamic_cast` anywhere in this assignment.

6 Add to Network class

```
+addPost(ownerId:int, message:string, likes:int,
         isIncoming:bool, authorName:string, isPublic:bool): void
+getPostsString(ownerId:int, howMany:int, showOnlyPublic:bool): string
```

- addPost should add the new post to the messages vector of the user whose id is ownerId. Note that addPost does not have a messageId parameter. You may assume that the existing messages are numbered 0,1,2,3,..., and assign the next number as the messageId to the new post.

- getPostsString should make the corresponding call to `User::getPostsString` and return the result.

7 File I/O

Add the following additional methods to your Network class.

```
+readPosts(char* fname): int
+writePosts(char* fname): int
```

- readPosts (writePosts) should read (write) the posts from (to) a file whose format is described below.

- readPosts and writePosts should return -1 if the file cannot be opened.

- To implement writePosts, you should load all of the posts from all the users into a single vector of Post pointers, sort the Posts by their messageId using the STL sort method, and then write the posts in that order to a file. To call sort function, you should implement a comparison function for comparing two Post pointers.

Here is the file format.

```
1: single number representing how many posts are in the file
2: messageId_0
3: <TAB>message text
4: <TAB>ownerId
5: <TAB>likes
6: <TAB>an empty line if the message is an owner Post
        OR the string "public" or "private" if the message is an IncomingPost
7: <TAB>an empty line if the message is an owner Post
        OR an author if the message is an IncomingPost
8: messageId_1
...
```

We further require that in our file, the posts are numbered 0,1,2,...,n-1, where n is the number of posts, and the posts listed in increasing order of messageId. So `messageId_0` is always 0, `messageId_1` is always 1, etc.
Here is an example file posts.txt. Note the mixture of owner posts and incoming posts in the file.

8 Add to the main function

In addition to reading in the users from a file, update your `social_network.cpp` to now also read in the posts from a file whose name is provided by the user at the command line. Specifically, you should run your code now with
`./myExecutable users.txt posts.txt`
To read in the posts, you should call the `Network::readPosts` function that you wrote. You are not required to write the messages out to a file at this point. Add an option (option 5) to view the most recent posts for a particular person.
`5 Aled Montes 4`
This should show the 4 most recent posts on Aled Montes's page. For now you should show all Posts and all public AND private `IncomingPosts`. You will use the `Network::getPostsString` method. In the next homework, we will see that sometimes we want to view private posts, and sometimes we don't.

8.1 Take screenshot(s)

From your terminal, run your social network. Enter the options `5 Aled Montes 4` and `5 Sandhya Krish 4`. Take screenshot(s) of the terminal output. Make sure you include (1) running the executable with command line arguments (`./output users.txt posts.txt`), (2) entering the options `5 Aled Montes 4` (and `5 Sandhya Krish 4`), and (3) the output for each option.

9 Short Answer Questions

Answer these questions in a .txt file. Write at most two sentences for each question.

1. Why did we store the posts as a vector of Post pointers, and not as a vector of Posts? There are two main reasons

2. In `IncomingPost::toString()`, why can you not access the message field directly?

3. You want call the `Post::toString` method for Posts and `IncomingPost::toString` method for IncomingPost in `User::getPostsString`. What keyword makes this possible, and where does that keyword go?

4. In hours, (approximately) how much time did you spend on this assignment? (there are no wrong answers)