

HW: Hello, CPPeers!

Or: Practicing with Classes and, finally, using `std::vector`

Objectives

1. Practice with classes
 - Constructors
 - Getters and Setters
 - Encapsulation
 - Public/Private
2. Additional practice with File I/O
3. Work with dynamic memory
 - Using vectors rather than arrays
4. More practice with throwing and catching exceptions

Overview

Many of you love Campuswire given the immediate help you can get by using the platform. Alas, Campuswire will no longer be available for free after the Spring semester (for real!). To help future students, you decide to use the skills you have gained through this course to come up with a new platform called CPPeers! As you are a big fan of social media, you decide to design this platform as a network of users and posts.

Example run:

```
Welcome to CPPeers
The options are:
1. load data file and add information
2. show posts by user
3. show posts with hashtags
4. show most popular hashtag
9. quit
-----> Enter your option: 1
Enter filename: TAMU-csce121-small.txt
Added User ajitjain
Added User michaelm
Added Post 42412212 by ajitjain
Added Post 58687888 by michaelm
Added Post 42353253 by michaelm
```

Now the user asks for option 2 (posts by user), option 3 (posts with tag), and option 4 (most popular hashtag):

```
Welcome to CPPeers
```

```
The options are:
```

1. load data file and add information
2. show posts by user
3. show posts with hashtag
4. show most popular hashtag
9. quit

```
-----> Enter your option: 2
```

```
Enter username: michaelm
```

```
When an #array is passed to a #function, it decays to a pointer.  
You can use pass-by-reference, if you need multiple updated  
values in the calling #function. It also helps save memory and  
CPU cycles.
```

```
Welcome to CPPeers
```

```
The options are:
```

1. load data file and add information
2. show posts by user
3. show posts with hashtag
4. show most popular hashtag
9. quit

```
-----> Enter your option: 3
```

```
Enter tagname: #dynamic
```

```
When deallocating #dynamic arrays, use delete[].
```

```
Welcome to CPPeers
```

```
The options are:
```

1. load data file and add information
2. show posts by user
3. show posts with hashtags
4. show most popular hashtag
9. quit

```
-----> Enter your option: 4
```

```
#function
```

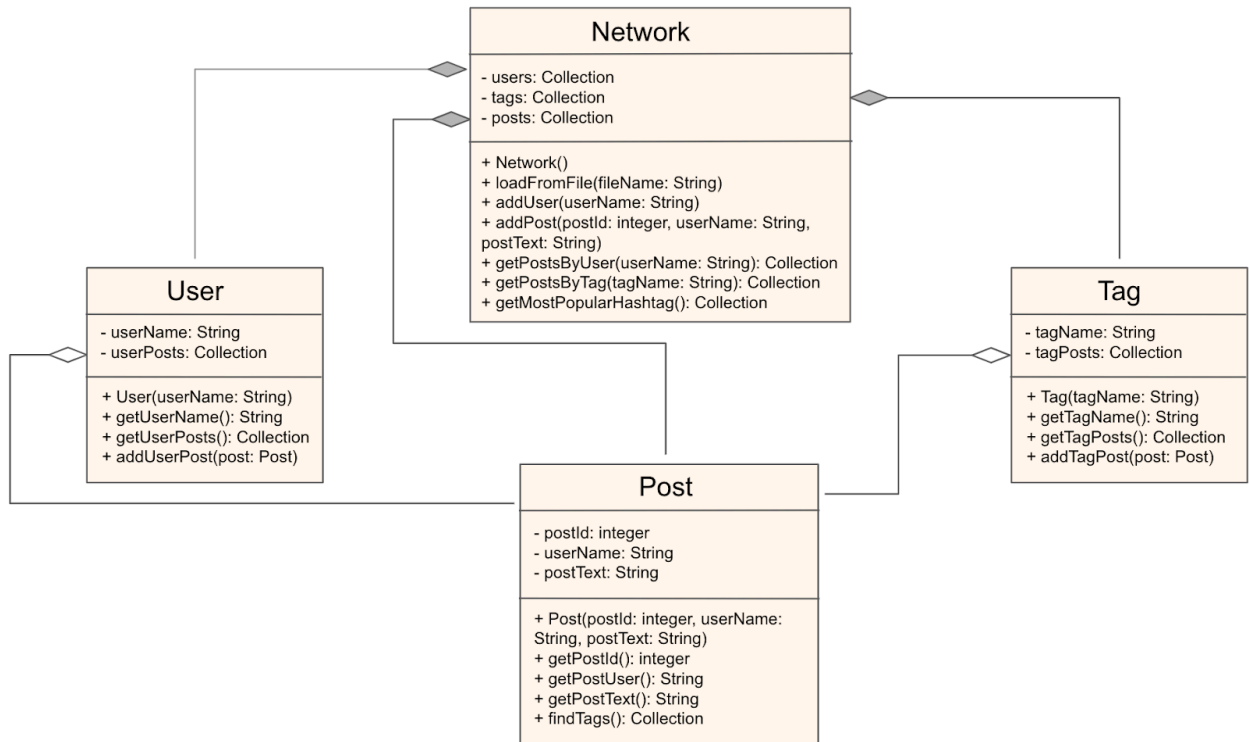
Roadmap

This is a guide on how to approach this homework. Specific requirements are in the Requirements section.

- Get the start code and test files on Mimir.
 - Code Files
 - `Network.h`: the class definition for the fabulous cppeers network; an interface to all users, posts, and hashtags
 - `Network.cpp`: definitions for the `Network` member functions
 - `User.h`: the class definition for the users of the cppeers network
 - `User.cpp`: definitions for the `User` member functions
 - `Post.h`: the class definition for the posts on the network
 - `Post.cpp`: definitions for the `Post` member functions
 - `Tag.h`: the class definition for the hashtags included within posts
 - `Tag.cpp`: definitions for the `Tag` member functions
 - `Cppeers-main.cpp`: this file is useful for testing your code locally; also, contains the choice menu
 - Data Files
 - See section on Data File Format below.
 - Files
 1. `TAMU-csce121-small.txt`
 2. `TAMU-csce121-large.txt`
 3. `unknown_entry.txt`
 4. `invalid_user_format1.txt`
 5. `invalid_user_format2.txt`
 6. `invalid_post_format1.txt`
 7. `invalid_post_format2.txt`
 8. `invalid_post_format3.txt`
 9. `Invalid_post_format4.txt`
- Review the code.
 - Read header and source files.
 - Compile and run the initial state of the starter code with the following command line.


```
g++ -std=c++17 -Wall -Wextra -pedantic -g -fsanitize=address,undefined Network.cpp User.cpp Post.cpp Tag.cpp Cppeers-main.cpp
```

- Confirm that the initial state of the starter code executes without errors and with only warnings related to unused variables/parameters and functions not returning values. Go ahead and submit the starter code to Mimir so you can take a look at the test cases.
- Review the organization of the classes. This UML class diagram shows relationships among Network, User, Tag, and Post classes used for this program.



- Submit code to Mimir before doing any development.
 - Review the test cases so you know what to expect.
- Look at the dependencies in classes to determine order to implement.
 - Network depends on User, Post, and Tag
 - Implement User, Post, and Tag prior to implementing Network
 - Post depends on Tag
 - Implement Tag prior to implementing Post
 - User depends on Post
 - Implement Post prior to implementing User
 - Tag has no dependencies
- Look at the dependencies within each class to determine the best order to implement methods (i.e. functions) of each class.
- Implement the classes and remaining helper functions in Cpppeers_main.cpp.

Hints and Recommendations

- As you develop your code, you only need to implement the methods/functions you need at the moment. You don't have to get everything done at once. It is not a good strategy to do so.
 - You can utilize `Cppeers-main.cpp` to test the functions that you have implemented so far. Alternatively, you can create a separate tester file with its own main function.
 - Plan and think before you code.
 - Think before you code.
 - Pick small problems to solve first, write test cases first, and develop incrementally.
 - Use descriptive (long) naming conventions for variables and functions.
 - Add comments to the code to describe anything which is not obvious from the code.
 - Use whitespace (indentation, newlines) to visually organize code.
 - Use and add functions to reduce code duplication and increase abstraction.
 - It is the end of the semester. You are busy. Start and finish early.
 - Recompile and rerun (`run == test`).
 - Check for errors.
 - If no errors, move on
 - Else, start debugging
- 
- When changing your code to fix errors, before executing or uploading to Mimir, stop and explain to yourself (or to a [rubber duck](#)) why your bug fix will work.
- When you get stuck, ask for help on Campuswire and attend office hours. The TAs and the instructors will help you.
 - Read the prior posts before posting a new question (which may be a duplicate).

Requirements

- You are required to implement classes `Network`, `User`, `Post`, and `Tag`. You are required to implement any of the class member functions not already provided for you in the starter code.
 - When implementing constructors, if necessary, assign default values to data members that do not have a corresponding constructor parameter.
 - When throwing exceptions, you are not required to put any specific message. However, putting meaningful messages will help you understand the source of the error.
- You are required to implement the user menu in `Cppeers-main.cpp`.
- Use of an unapproved header file will result in a score of 0.
- The program must compile without warnings or errors.

```
g++ -std=c++17 -Wall -Wextra -pedantic -g -fsanitize=address,undefined
Network.cpp User.cpp Post.cpp Tag.cpp Cppeers-main.cpp
```

- The program must run without errors or unhandled exceptions.

Allowed Includes

- `<iostream>`
- `<fstream>`
- `<sstream>`
- `<vector>`
- `<string>`
- `<cctype>`
- `<stdexcept>`
- `"Post.h"`
- `"User.h"`
- `"Tag.h"`
- `"Network.h"`

class Post

We provide

- Post class definition (Post.h)
- Constructor (Post.cpp)
 - `Post::Post(unsigned int postId, string userName, string postText)`
 - Throw `std::invalid_argument` if any of the following are true:
 - `postId` is zero
 - `userName` is empty
 - `postText` is empty
- Getters (Post.cpp)
 - `unsigned int Post::getPostId()`
 - `string Post::getPostUser()`
 - `string Post::getPostText()`

You implement in Post.cpp

- Methods / Member functions
 - `vector<string> Post::findTags()`

- Return a vector containing unique candidate tags extracted from postText
 - Tags are words within postText that begin with a '#'.
 - Tags are not case-sensitive.
 - Remove punctuations ('!', ',', '.', '?') at the end of each extracted tag, if any.
 - As an example, the following need to be treated as the same, and need to be added as the lowercase version (#happy) to the vector:
 - #happy
 - #Happy
 - #HaPpY
 - #HAPPY
 - #happy!
 - #happy!!
 - and so on..
- When finding tags, this function returns all candidate strings beginning with '#'.
 - Note: The function does NOT filter out invalid Tags, such as #, #1a23, or #!happy. Such checks are performed in Tag's constructor (see Tag). Relatedly, the reason for removing punctuation above is to identify unique candidate strings, not validation.

class User

We provide

- User class definition (User.h)

You implement in User.cpp

- Constructor
 - User::User(string userName)
 - Throw std::invalid_argument if any of the following are true:
 - userName is empty
 - userName does not start with a letter 'a' - 'z'
 - userName contains uppercase letters 'A' - 'Z'
 - In other words it should be all lowercase letters
- Getters
 - string User::getUserName()
 - vector<Post*>& User::getUserPosts()

- Methods / Member Functions
 - `User::addUserPost(Post* post)`
 - If post is nullptr, throw `std::invalid_argument`

class Tag

We provide

- class definition (Tag.h)

You implement in Tag.cpp

- Constructor
 - `Tag::Tag(string tagName)`
 - Throw `std::invalid_argument` if any of the following are true:
 - tagName length is less than 2
 - the first character of tagName is not '#'
 - the second character in tagName is not from 'a' - 'z'
 - tagName contains uppercase letters 'A' - 'Z'
 - In other words it should be all lowercase letters
 - there is one or more consecutive punctuations at the end of the tagName (e.g., #happy!, #happy!?, etc.), where a punctuation is one of the following: ('!', ',', '.', '?')
- Getters
 - `string Tag::getTagName()`
 - `vector<Post*>& Tag::getTagPosts()`
- Methods / Member Functions
 - `Tag::addTagPost(Post* post)`
 - If post is nullptr, throw `std::invalid_argument`

class Network

We provide

- Network class definition (Network.h)

You implement in Network.cpp

- Methods / Member Functions
 - `Network::addUser(string userName)`
 - If a user with this name already exists, throw `std::invalid_argument`
 - User names are not case sensitive. For example, the following are treated as the same and added as the lowercase version (ajitjain):

- ajitjain
 - AjitJain
 - AJITJAIN
 - AjltJain
 - and so on.
- Let any exceptions from the User constructor go through, i.e., do not catch exceptions from the User constructor
- When creating a new User object, remember that we are working with dynamic memory (Use the 'new' keyword to allocate memory on the heap)
- If no exception, add the user to the Network data member users, and at the end of the function, output "Added User " followed by the User Name to standard output
- Network::addPost(unsigned int postId, string userName, string postText)
 - If a post with this id already exists or if no user with this name exists, throw std::invalid_argument
 - Adding a post to the network is a multi-step process and includes message passing among various components to accomplish the following:
 - Creation of the post (see Post; remember we are working with dynamic memory)
 - Addition of the post to the Network data member posts
 - Addition of the post information to the corresponding user (see User)
 - Extraction of candidate hashtags contained within the post (see Post)
 - For each candidate hashtag: search (within the collection of tags stored in the Network) OR create (see Tag; only if the tag does not exist and remember we are working with dynamic memory)
 - Note: When attempting to create a Tag, if an exception is thrown (by Tag constructor), catch the exception and resume processing for the next candidate hashtag
 - Remember to add a newly created tag to the Network data member tags
 - addition of the post information to each tag (see Tag)
 - Let any exceptions from the Post constructor go through, i.e., do not catch exceptions from the Post constructor

- If no exception, at the end of the function, output "Added Post " followed by the Post Id, " by ", and User Name to standard output
- `Network::loadFromFile(string fileName)`
 - If file could not be opened, throw `std::invalid_argument`
 - Refer to the [Data Files](#) section, which specifies the format of User and Post information included in the file. Add users and posts as you read the lines from the file by calling `addUser()` and `addPost()`.
 - If the file content does not match the specified format, throw `std::runtime_error`. This includes:
 - User or Post entry not following the specified format
 - Unknown entry (neither User nor Post)
 - Note: You throw `std::invalid_argument` only when the file could not be opened. If there are `std::invalid_argument` thrown by add user and post operations, you catch them here and throw `std::runtime_error` instead.
- `vector<Post*> Network::getPostsByUser(string userName)`
 - If `userName` is empty or not found, throw `std::invalid_argument`
 - Note: Assuming you followed the multi-step process for `addPost()` above, you now do not need to traverse all the posts in the Network
- `vector<Post*> Network::getPostsWithTag(string tagName)`
 - If `tagName` is empty or not found, throw `std::invalid_argument`
 - Note: Assuming you followed the multi-step process for `addPost()` above, you now do not need to traverse all the posts in the Network
- `vector<string> Network::getMostPopularHashtag()`
 - Return the tag which occurs in the maximum number of posts. In case of a tie, return all such hashtags. If the network has no hashtag, return an empty vector.
 - Note: A hashtag should be counted only once per post. Assuming you followed the multi-step process for `addPost()` above, you now do not need to traverse all the posts in the Network and extract tags again from each post.

Helper functions in `Cppeers-main.cpp`

Use the menu example in the Overview above to fill in the gaps when implementing the following helper functions.

- `void processPostsByUser(Network& cppeers)`
 - Helper function for menu choice 2
- `void processPostsWithHashtags(Network& cppeers)`
 - Helper function for menu choice 3

- `void processMostPopularHashtag(Network& cpeers)`
 - Helper function for menu choice 4

Submission

The source files to submit are named:

1. `Cppers-main.cpp`
2. `Network.h` and `Network.cpp`
3. `User.h` and `User.cpp`
4. `Post.h` and `Post.cpp`
5. `Tag.h` and `Tag.cpp` Background Information: `std::vector`

Supporting Information

Data File Format

The information format is given below. Each line starts with the string **User** or **Post**, which identifies the type of information. Based on the type of information, one or more values follow. In the case of User, the value that follows is a `UserName`. In the case of Post, values that follow are `PostId`, `UserName`, and `PostText`. Implementing the functions listed in the previous section will allow you to check whether the read information is valid or not.

User Information Format

User *UserName*

Example

User ajitjain

Post Information Format

Post *PostId UserName PostText*

Example

Post 42412212 ajitjain When deallocating dynamic arrays, use `delete[]`.

Example valid file:

```
User ajitjain
User michaelm
Post 42412212 ajitjain When deallocating #dynamic arrays, use delete[].
Post 58687888 michaelm When an #array is passed to a #function, it
decays to a pointer.
```

Post 42353253 michaelm You can use pass-by-reference, if you need multiple updated values in the calling #function. It also helps save memory and CPU cycles.

Useful functions and classes

You may use any of the functionality provided by C++ strings and C++ vectors in your solution. Most students may find the following operations useful:

- `getline`
 - An example of how to use this is provided in `Cppeers-main.cpp`
- `vector`
 - `push_back(elem)`
Adds a new element `elem` at the end of the vector
 - `size()`
Returns the number of elements in the vector
 - More information about vectors is available in the [Background Information section](#)
- `istreamstream`
 - You already used this in a labwork. If you need to refresh your memory, an example of how to use this is provided in `Cppeers-main.cpp`

Hints and Recommendations

Hints

1. Read the header and source files.
2. Think before you code.
3. Pick small problems to solve first, write test cases first, and develop incrementally.
4. Use descriptive (long) naming conventions for variables and functions.
5. Add comments to the code to describe anything which is not obvious from the code.
6. Use whitespace (indentation, newlines) to visually organize code.
7. Use functions to reduce code duplication and increase abstraction.

Vector class Overview

`std::vector` is a C++ standard class that provides the functionality of dynamically allocated arrays. You can learn more about C++ vectors in the documentation ([link to std::vector documentation](#)) or in the Zybook Chapter 8.2.

A simple example of using `std::vector` to store a list of consecutive int values is below:

```

#include <iostream>
#include <vector>

int main() {
    // n is not fixed (providing the same capability as dynamic arrays)
    size_t n = 0;

    std::cout << "Enter value of n: ";
    std::cin >> n;

    std::vector<int> v;
    for (size_t i = 0; i < n; i++) {
        v.push_back(i); // add i to the last position in v
    }

    // let's print the elements in a vector
    // we can use a for loop as before
    for (size_t i = 0; i < n; i++) {
        std::cout << v.at(i) << " ";
    }
    std::cout << std::endl;
    return 0;
}

```

Have Fun!

It is much more fun for you and you get better support from instructors and TAs when you do not start working on the homework just before it is due.