CSCI 104 – Summer 2014

Data Structures and Object Oriented Design

- Home
- Updates
- Syllabus
- Grading
- Lectures
- Labs
- **Assignments**
- Calendar
- Resources

Project, Part 1

- Due: Tues. July 1, 2014, 11:59pm (PST)
- Project-based assignments will be using their own repository that is distinct from the `dslib` one.
- Notice that because this is part of the project, the grade cannot be dropped.
- Remember to provide a Makefile that lets the course staff simply compile all questions correctly.
- Warning: while this project does not really rely on any recently added material, it is quite substantial. Start early!

Overview

Our project for the semester will be to model a microblog site such as Twitter. As you will see, it will require using quite a lot of the data structures you are learning about. At a high level, a microblog site is based on the following components:

1. Users that follow other users (this forms a graph with users as vertices and follow relationships as edges)
2. A timeline for each user (user x) that shows their posts as well as others posts from those that user x follows and eventually posts with appropriate

@ mentions
3. Quick lookup/indexing of tagged keywords (hashtags)

For this (first) part of the project, we focus on parsing users and their feeds. Notice, however, that you want to keep an eye on making your code well documented and easy to expand, as you will be adding to it later. (That said, you will also be allowed to rewrite your code later.)

The Project Code Repository

In an effort to better organize your code, the project will have its own separate repository that is also hosted on the course GitHub page. This repository will have the prefix `project_` followed by your USC username. At the end of the semester, this repository will reflect how much you learned and how far you grew as a developer. It is your obligation to organize it well and keep it professional.

When you visit the course home on GitHub, you will see two private repositories:

1. `hw_username` – for all **non-project** homework assignments
2. `proj_username` – which is to be used *only* for project-based assignments

You should be mindful of the distinction between the two repositories and be clear on which one you should use at any given time. It is your responsibility to use the proper repository for the proper assignment. Consider this practice for working on different projects within the same working environment.

Managing Multiple Repositories

You are used to organizing your work using directories. You may have a directory for each course you are taking, and in each one, you have a directory for notes, slides, assignments, etc. You don't usually place assignment files in the directory you use for your slides. The same applies to code repositories.

In git, the working copy of a repository is represented as a directory on your storage device. Lets say you have a directory on your Desktop called `cs104` which you use to keep all your code repositories. When you list the contents of this directory:

```
cd ~/Desktop/cs104
ls
```

You should get something like:

```
hw_ttrojan        Labs       SampleRepo
```

It is in this directory that you clone your new repository:

```
cd ~/Desktop/cs104
#Clone *your* private repository
git clone git@github.com:usc-csci104-summer2014/proj_ttrojan.git
#List the contents of the cs104 directory
ls
```

Now, when you are listing the contents of `~/Desktop/cs104`, you will see

```
hw_ttrojan        proj_ttrojan        Labs       SampleRepo
```

To work on the project, you change directory to your `proj_ttrojan` repository

```
cd proj_ttrojan
#Check the status of the repository
git status
```

Finally, you should keep in mind the following notes:

- Although this project will be built over multiple assignments, it is one coherent project, and you are building over the same code-base every time. This means that you will not be creating separate directories per assignment. Instead, you start each new assignment where you left off in the last one.
- You **do not** clone repositories inside other repositories for the same reason you don't mix Cheerios with Fruit Loops.
- You are expected to:
    - Update the `README.md` file
    - Create a `Makefile` for your project
    - Update the `.gitignore` file to be compatible with your project

Step 1 (Get your data types from HW3, 10%)

Create directories `src`, `include`, and `bin` under your `proj_` repository. Set up your `Makefile` so that all executables are generated in the `bin` directory, and set up your `.gitignore` so that the `bin` directory is not stored.

From HW3, copy your `AList.h` and `Map.h` header into the `include` directory. If there are any lingering bugs from HW3, please fix them. To your `AList` class, add the following public functions and implement them:

```
AList (const AList<myType> & other);
  // A copy constructor which performs a deep copy.

AList<myType> & operator= (const AList<myType> & other);
  // An assignment operators.
```

Step 2 (Implement sets, 25%)

In your `include` directory, create and implement a template *Set* data type. The header file should look as follows:

```
// necessary includes can go here

template <class T>
class Set {
  public:
    Set ();                          // constructor for an empty set
    Set (const Set<T> & other); // copy constructor, making a deep copy
    ~Set ();                         // destructor

    void add (const T & item);
      /* Adds the item to the set.
         Throws an exception if the set already contains the item. */

    void remove (const T & item);
      /* Removes the item from the set.
         Throws an exception if the set does not contain the item. */

    bool contains (const T & item) const;
      /* Returns true if the set contains the item, and false otherwise. *,

    int size () const;
      /* Returns the number of elements in the set. */

    bool isEmpty () const;
      /* Returns true if the set is empty, and false otherwise. */

    Set<T> setIntersection (const Set<T> & other) const;
```

```
      /* Returns the intersection of the current set with other.
         That is, returns the set of all items that are both in this
         and in other. */

    Set<T> setUnion (const Set<T> & other) const;
      /* Returns the union of the current set with other.
         That is, returns the set of all items that are in this set
         or in other (or both).
         The resulting set should not contain duplicates. */

    /* The next two functions together implement a suboptimal version
       of what is called an "iterator".
       Together, they should give you a way to loop through all elements
       of the set. The function "first" starts the loop, and the function
       "next" moves on to the next element.
       You will want to keep the state of the loop inside a private variabl
       We will learn the correct way to implement iterators soon, at
       which point you will replace this.
       For now, we want to keep it simple. */

    T* first ();
      /* Returns the pointer to some element of the set,
         which you may consider the "first" element.
         Should return NULL if the set is empty. */

    T* next ();
      /* Returns the pointer to an element of the set different from all
         the ones that "first" and "next" have returned so far.
         Should return NULL if there are no more element. */

  private:
    List <T> internalStorage;
    // other private variables you think you need.
};
```

Step 3 (Twitter, 65%)

You will begin your quest to model Twitter by building support to handle Users and their tweets.

** You may not use STL data structures here...only your own **

Users contain:

- a username
- a list of other Users whom they follow
- a list of other Users following them
- a list of tweets that user has posted

Each tweet contains:

- a timestamp in format: YYYY-MM-DD HH:MM:SS
- the username of the User who posted the tweet
- the actual text of the tweet (we won't impose the 140 character limit in this project)

For now, while texts may contain @usernames and #hashtags, you don't have to any special processing of these and can simply show that as text in the tweet

Your program will be supplied a file that contains all the information about users and tweets in a single place. The format of the file and an example that illustrates the format are shown below.

File Format

```
number_of_users
```

```
username following_username ... following_username
...
username following_username ... following_username
timestamp username tweet_text
timestamp username tweet_text
...
timestamp username tweet_text
```

Illustrative Example (`twitter.dat`)

```
4
Mark Tommy Jill
Tommy Jill Sam
Jill Sam
Sam Mark Tommy
2014-05-20 12:35:14 Mark What's up friends
2014-05-19 12:35:15 Jill Can't believe Johhny Manziel went to the Browns
2014-05-20 00:56:34 Jill Why did Chanos change their name #backwardsdriveth
2014-05-21 10:30:27 Sam @alghanmi where do I get more github stickers?
```

Your job is to produce one output file per user with the filename `username.feed` (e.g. `Mark.feed`, `Tommy.feed`, etc.). The feed should list the username on the first line and then all the tweets from the user and any users being followed sorted based on timestamp. Thus, the file above should produce:

So when you run the program as:

```
$ ./twitter twitter.dat
```

It should produce the following files

Mark.feed

```
Mark
2014-05-19 12:35:15 Jill Can't believe Johhny Manziel went to the Browns
2014-05-20 00:56:34 Jill Why did Chanos change their name #backwardsdriveth
2014-05-20 12:35:14 Mark What's up friends
```

Tommy.feed

```
Tommy
2014-05-19 12:35:15 Jill Can't believe Johhny Manziel went to the Browns
2014-05-20 00:56:34 Jill Why did Chanos change their name #backwardsdriveth
2014-05-21 10:30:27 Sam @alghanmi where do I get more github stickers?
```

Jill.feed

```
Jill
2014-05-19 12:35:15 Jill Can't believe Johhny Manziel went to the Browns
2014-05-20 00:56:34 Jill Why did Chanos change their name #backwardsdriveth
2014-05-21 10:30:27 Sam @alghanmi where do I get more github stickers?
```

Sam.feed

```
Sam
2014-05-20 12:35:14 Mark What's up friends
2014-05-21 10:30:27 Sam @alghanmi where do I get more github stickers?
```

Your program must meet this output format or you will lose significant points as this helps our grading scripts!

To help you get started we require you to have a few of the following classes.

user.h

```
#ifndef USER_H
#define USER_H

#include <string>
/* Add appropriate includes for your data structures here */

/* Forward Declaration to avoid #include dependencies */
class Tweet;

class User {
 public:
  /**
   * Constructor
   */
  User(std::string name);

  /**
   * Destructor
   */
  ~User();

  /**
   * Gets the name of the user
   *
   * @return name of the user
   */
  std::string name();

  /**
   * Gets all the followers of this user
   *
   * @return Set of Users who follow this user
   */
  Set<User*> followers();

  /**
   * Gets all the users whom this user follows
   *
   * @return Set of Users whom this user follows
   */
  Set<User*> following();

  /**
   * Gets all the tweets this user has posted
   *
   * @return List of tweets this user has posted
   */
  AList<Tweet*> tweets();

  /**
   * Adds a follower to this users set of followers
   *
   * @param u User to add as a follower
   */
  void addFollower(User* u);

  /**
   * Adds another user to the set whom this User follows
   *
   * @param u User that the user will now follow
   */
  void addFollowing(User* u);
```

```
  /**
   * Adds the given tweet as a post from this user
   *
   * @param t new Tweet posted by this user
   */
  void addTweet(Tweet* t);

  /**
   * Produces the list of Tweets that represent this users feed/timeline
   *  It should contain in timestamp order all the tweets from
   *  this user and all the tweets from all the users whom this user follov
   *
   * @return vector of pointers to all the tweets from this user
   *         and those they follow in timestamp order
   */
  AList<Tweet*> getFeed();

 private:

 /* Add appropriate data members here */

};

#endif
```

tweet.h

```
#ifndef TWEET_H
#define TWEET_H
#include <iostream>
```

```
#include <string>
#include "datetime.h"

/* Forward declaration */
class User;

/**
 * Models a tweet and provide comparison and output operators
 */
class Tweet
{
 public:
  /**
   * Default constructor
   */
  Tweet();

  /**
   * Constructor
   */
  Tweet(User* user, DateTime& time, std::string& text);

  /**
   * Gets the timestamp of this tweet
   *
   * @return timestamp of the tweet
   */
  DateTime const & time() const;

  /**
   * Gets the actual text of this tweet
   *
   * @return text of the tweet
   */
  std::string const & text() const;

  /**
   * Return true if this Tweet's timestamp is less-than other's
   *
   * @return result of less-than comparison of tweet's timestamp
   */
  bool operator<(const Tweet& other){
    return _time < other._time;
  }

  /**
   * Return true if this Tweet's timestamp is greater-than other's
   *
   * @return result of greater-than comparison of tweet's timestamp
   */
  bool operator>(const Tweet& other){
    return _time > other._time;
  }

  /**
   * Outputs the tweet to the given ostream in format:
   *   YYYY-MM-DD HH::MM::SS username tweet_text
   *
   * @return the ostream passed in as an argument
   */
  friend std::ostream& operator<<(std::ostream& os, const Tweet& t);
```

```
  /* Create any other public or private helper functions you deem
     necessary */


 private:
  DateTime _time;
  std::string _text;

  /* Add any other data members you need here */


};

/* Leave this alone */
struct TweetComp
{
  bool operator()(Tweet* t1, Tweet* t2)
  {
    return (*t1 > *t2);
  }
};
#endif
```

datetime.h

```
#ifndef DATETIME_H
#define DATETIME_H
#include <iostream>

/**
 * Models a timestamp in format YYYY-MM-DD HH:MM:SS
 */
struct DateTime
{
  /**
   * Constructor
   */
  DateTime();

  /**
   * Another constructor
   */
  DateTime(int hh, int mm, int ss, int year, int month, int day);

  /**
   * Return true if the timestamp is less-than other's
   *
   * @return result of less-than comparison of timestamp
   */
  bool operator<(const DateTime& other);

  /**
   * Return true if the timestamp is greater-than other's
   *
   * @return result of greater-than comparison of timestamp
   */
  bool operator>(const DateTime& other);

  /**
   * Outputs the timestamp to the given ostream in format:
   *   YYYY-MM-DD HH::MM::SS
   *
```

```
    * @return the ostream passed in as an argument
    */
   friend std::ostream& operator<<(std::ostream& os, const DateTime& othe

   /* Add data members here -- they can all be public
    * which is why we've made this a struct */



};

#endif
```