Lab3(A) and (B): The Chirper App

Objective: Write a C++ program that implements a pre-alpha version of an app called Chirper – a microblogging interactive system. Users may post chirps, scroll through their friend's chirps, rechirp and like posts. The program will be implemented using doubly-linked lists. See video posted for an example.

I) App Features:

The Chirper app is a microblogging software system that allows a user to post and interact with short messages called chirps. Chirps may span up to **180 characters maximum**. User's can "like" their own posts, or a friend's post. They may also re-chirp a friend's post. A re-chirp implies that the content of the friend's chirp post (message text) is copied to the user's account as a new post. For the pre-alpha version, the Chirper app can sustain the user's account, and a maximum of **one friend**.

II) Program Flow:

Account Setup: Upon launching the app, the program will prompt the user to enter a username and display name in **setup_account()**. Note: the user may change their **display name** later by using the **change_dn(**) feature, however their username can not be changed.

Friend setup: Once the user's information has been setup, the program reads a file containing their one friend's username, display name, and all chirp posts. The function **add_friend()** provides this setup feature, using **setUser** and **set_dn**, while invoking **add_chirp_post()** to create a linked list of the chirp posts included in the file, with the specified user name and display name.

Menu: Once the user and friend have been setup by the chirper app, the user is presented with a menu containing 5 options:

i) **Chirp**: calls **add_chirp_post(…)**, which prompts the user to enter a message they would like to post, and adds the post to their feed (i.e. a node in their doubly-linked list)

ii) **Scroll through your posts/chirps:** calls **scroll_my_posts()**, which navigates through the user's chirps, starting from head of the list (oldest post), and may select to navigate to the **previous** or **next** chirp. If the node does not exist (i.e. user asks for previous node at head node), then the post remains the same, with an appropriate message written out to the terminal.

- **Rechirp** much like Chirper's predecessor, Twitter, allows a user to **copy a friend's** post on their own feed. **A user can not rechirp their own chirp**. Once the user rechirps a post, an arrow symbol is placed on the original post, indicating that the message has been re-chirped. When the user rechirp's a post, they may be brought back to the main menu.
- Users may also "**like**" their own chirp or a friend's chirp post, in which a heart symbol is placed on the original post. The number of likes is maintained and displayed per post.
- A user may also **delete** one of their posts using **delete_post()**. The option to delete may be presented for the friend, but is unusable.

iii) **Scroll through your friend's chirps:** your friend's posts have been previously setup and read from a file. The function allows the user to navigate through all the friend's posts. In this case,

the user may select previous posts, next posts, while liking or re-chirping posts as well. In the case that a rechirp is selected, the message is added to the user's feed as a new post, and the user may be redirected to the main menu(). **scroll_thru_friend_posts()** implements this functionality, calling **scroll_my_posts()** to navigate.

iv) **Change display name:** user may change the display name of their account

v) **Exit app**: exits the program. Note that this menu is looped until option 5 is exercised by the user i.e. the "exit" feature below.

Accordingly the **menu options** presented to the user are as follows:

1) Add a post
2) Scroll through your posts
3) Scroll through friend's posts
4) Change your display name
5) Exit App

2) Class Specifications:

This lab should consist of at least two to three C++ files: 1) header (**chirper.h**) and 2) an implementation (**chirper.cpp**) file, and **optionally** 3) **main.cpp**. The following hints provided in the lab are only suggestions. Feel free to use another data/return type if necessary. **Doubly-Linked list structure paired with a class implementation must be used** in your program. A list of suggested Chirp class variable members are presented on the next page.

Again, data types, return types and parameters passed are flexible and up to the programmer, as long the program works as intended. Ensure you provide **precondition** and **postconditions** for each function.

3) Application flow:

You will require a main function:

```
using namespace std;

int main(){
  Chirp ch;
  ch.setup_account();
  ch.menu();

  cout << "See you next time\n";
  return 0;
}
```

The following is a possible function flow call and struct/class definitions. This may help guide you through the coding and implementation process: (Note: one last time, this is simply a suggestion, and totally up to you. As long as all functionality is implemented accordingly).

```
  struct post{
    std::string message; //must not exceed 180 characters
    bool like;
    int num_likes;
    bool reChirp;
    post *previous;
    post *next;
  };
  typedef post* postPtr;


class Chirp; //define chirp, as chirp contains an instance of itself
class Chirp{
private:
  std::string user;
  std::string display_name;
  postPtr head; //head


  bool isEmpty();
  postPtr previous_chirp(…);
  postPtr next_chirp(…);
  void display_chirp(…);
  void add_like(…);
  void add_chirp_post(…);
  void delete_chirp(int);

  char* scroll_thru_posts();
  bool add_like(int msg);
  std::string scroll_my_posts(…….);
  void scroll_thru_friend_posts();
  void add_friends();
  void reChirp(…….);
  void change_dn();


public:
   Chirp(); //constructor
   Chirp(const Chirp &o);//copy constructor
   Chirp *friends; //one friend
   ~Chirp(); //destructor
   void setup_account();
   void menu();

   void setUser(std::string);
   std::string getUser();
   void setDN(std::string);
   std::string getDN();
};
```

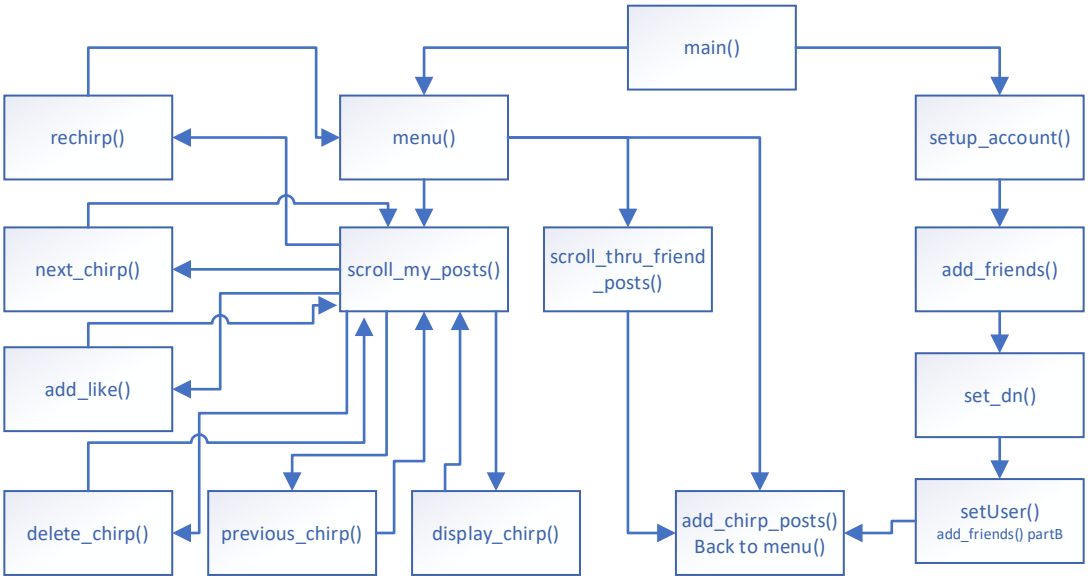

Fig. 1: Chirper App Function Call Flow

4) More Hints:

chirper.h

Include the following libraries and macro definition MSG_LENGTH at the top of your header file. Ensure to include the member variables and functions specified for the Chirper class. The heart char is used to display that the post has received likes (heart symbol), and the arrow char as a symbol for re-chirps (re-chirps are not tallied). Both UTF-8 encodings are provided to you below.

```
#include <iostream>
#include <string>
#include <sstream>
#include <iomanip>
#include <fstream>

#define MSG 180
const char heart[] = "\xe2\x99\xa5";
const char arrow[] = "\xe2\x86\x94";
```

chirper.cpp

The file “friends_list.dat” has been provided to you, containing a sample friend’s information, read in by the add_friends() function. The code for reading in this file, add_friends(), is provided to you. Feel free to add your own friend messages, posts and name as you wish. Ensure the number of friend posts is at least 3-5 as provided.

Once you have your doubly-linked list chirps and user profiles setup, the display_tweet() function, also provided to you, may be used to display chirps. Feel free to make any amendments to this code as you wish. Accordingly, your focus will be mostly dedicated to organizing the doubly linked lists, and providing all functionality for the member functions listed in chirper.h

5) Compiling your program & sample run:

Compile your program using the following g++ commands:

g++ -o <executable_name> chirper.cpp <main.cpp>

If you include a main.cpp, you will also have to include <the file> in the above compilation command.

A video of a sample run and expected output has been uploaded for your viewing. Please take this as an example. Your implementation is unique to you as a programmer. Apply your own touch to your version of chirper. Marks will be awarded for creativity, coding style, and outstanding effort.

Lab3A Deliverables: Implement Menu options 1,2,4,5

- **Launching app:** setup_account() functionality, which sets your username and display name. setUser(), setDN()
- **Option 1:** Adding posts to your Chirp wall – will require functions add_chirp_post(), isEmpty(), where every post is inserted as a node of a doubly-linked list. Ensure you consider all cases (head node insertion, last node insertion).
- **Option 2:** Scroll_my_posts() – calls scroll_thru_posts(), also called by scroll_thru_friends_posts(). Navigate (up and down) your posts, interface with **display_chirp()**. Implement "**Like"** post functionality, placing a heart symbol on the post, with the number of total likes .
- **Option 4:** Change display name – calls set_dn() to change your display name
- **Option 5:** Exit app - implement functionality to loop within menu until exit call by user.

Lab3b Deliverables: Launching App revisited + Option 3 + additional app functionality

- **Launching app revisited:** setup_account() must now also setup your friend and their respective profile and posts. Specifically this function should read the friend file (friends_list.dat), call add_friends(), setUser(), and setDN() to setup the friend information while reading from the file. This function must also make use of add_chirp_post(..) for every post inserted within your friend's feed. Specifically, a new post (node in the doubly-linked list) should be created for every post contained in the .dat file, considering one message post per line in the file.
- **Option 3: Scroll_thru_friends_post()** - once your friend's feed and posts have been setup, enable the scroll feature for your friend's post.
- **Copy constructor & rechirp()** – implement the functionality for rechirp-ing a friend's post. This will copy their post to your wall, adding the appropriate arrow symbol to the friend's post indicating that the post was rechirped. Ensure **like** functionality also works.
- **Delete_chirp()** – implement delete post functionality, where the rest of the posts and feeds should remain intact.
- **Destructor –** implement the destructor for the class and ensure that no memory leaks or dangling pointers exist once the program is terminated.

For all functionality above, ensure you implement object-oriented programming methods, do not copy and paste code unless absolutely necessary, and enforce encapsulation.

Deliverables:

Compress and include the following files as one zip, tar or tar.gz file named (where xxxx is the last 4 digits or your student id): FirstLastNameXXXX_Lab3A(or B).<zip/tar/tar.gz>

- Fill out, sign, and include the academic honesty cover page as a pdf
- chirper.h
- chirper.cpp (and optionally main.cpp)

Lab3a) : Upload the file to canvas by **Monday, June 22nd at 11:59pm**
Lab3b) : Upload the file to canvas by **Monday, June 29th at 11:59pm**
Only one demo held for Lab 3 – Friday, July 3rd (please sign up)