Jordan Stein
CSCI 4761
Lab 2

The purpose of this program is to create an online personal weekly schedule manager that communicates through TCP sockets. It is a concurrent application server that handles different scheduler clients simultaneously.

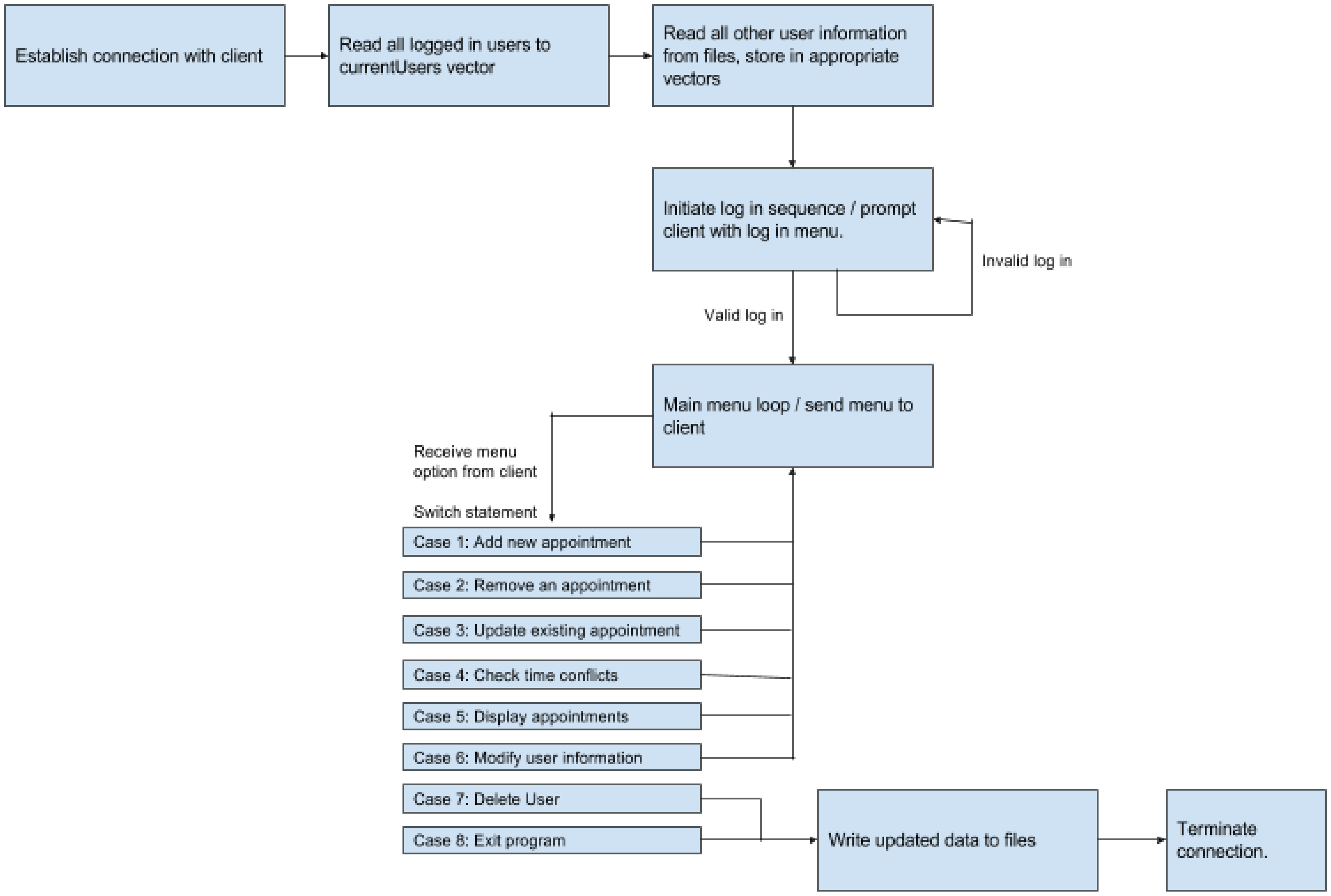
The required functionalities are:

- Adding / Deleting users
- Modify user information
- Login authentication
- Adding/removing/updating appointments
- Checking appointment time conflicts
- Displaying appointments for specific times / time ranges.

I first started by creating the user database offline, then added online functionality to them as they were functional. To begin, I made the User class with functionalities to create users and appointments. I then created separate functions that would operate the control flow of the program by integrating the User class. Once the database was operating, I plotted a program flow to establish how I want the server connection to operate with the client.

Program flow:
1. establish connection with client
2. read all logged in users to currentUsers vector
3. read all user information from files, store in approporiate vectors.
4. initiate log in sequence or create account
5. once logged in, keep looping a main menu with options.
6. switch statement that calls functions for each menu option the client sends the server.
7. loop menu until client logs out (exits menu, deletes user)
8. write all new data to files.
9. close connection

A flow chart of the program is demonstrated on the next page.


Establish connection with client
Read all logged in users to currentUsers vector
Read all other user information from files, store in appropriate vectors
Initiate log in sequence / prompt client with log in menu.
Invalid log in
Valid log in
Main menu loop / send menu to client
Receive menu option from client
Switch statement
Case 1: Add new appointment
Case 2: Remove an appointment
Case 3: Update existing appointment
Case 4: Check time conflicts
Case 5: Display appointments
Case 6: Modify user information
Case 7: Delete User
Case 8: Exit program
Write updated data to files
Terminate connection.

Once the program flow was created, I begun writing the server functionality inline with the flow chart. A lot of editing was required at this point because my database was initially written completely on the client side. I had begun to replace all of my cout statements with send() to send the server the menu options. Fortunately, I have been able to establish this connection for all functionalities over time.
When designing the program, I had created many functions to operate each of the menu cases. These functions allowed me to add and delete users, print information, read and write data from files, create/remove/update appointments, modify user information, check time conflicts, and display appointment information. I separated the functions into two different files. The functions.cpp file holds all functions to control the flow of the program. The user.cpp functions allow the other functions to operate with the user class.

All functions from both files are listed on the following pages.

===== functions.h / functions.cpp =====
Includes all functionality for menu options.
Functions included:
// Adds a unique user to the users vector
void addUser();
// Removes a user from the users vector
void deleteUser();
// Prints all users to the console
void printUsers();
// Reads all user information from csv file into users vector. Returns false if file fails to read
bool readUsers();
// Writes all user information from vector into text file
void writeUsers();
// recieves a date/time from client with correct formatting.
std::string receiveDateTime();
// converts a date/time string to a value
long dateToValue();
// Creates an appointment for a user, along with a description for the appointment
void createAppointment();
// Removes an appointment for a user, along with the description for the appointmentvoid removeAppointment();
// Updates information about an appointment for a user
void updateAppointment();
// Displays all appointments for the user
void displayAppointments();
// Compares username and password for given user.
bool validateLogin();
// Modifies user information
void modifyUser();
// Checks all user appointments for time conflicts, sends them to client.
void checkTimeConflicts();
// Deletes the current user that is logged in. Returns true if user agrees deletion.
bool deleteUser();
// reads all users who are logged in from a file.
void readLogIn();
// writes new logged out person to file. boolean determines log in or log out
void writeLogIn();

===== users.h / users.cpp =====
user class that holds information for all users
private variables:
username
name
password
phone
email
vector<string> appointments
vector<string> descriptions
functions:
User(std::string, std::string, std::string, std::string, std::string); // constructor
User(); // default constructor
~User(); // destructor
// Getters / Setters
void setUsername(std::string);
std::string getUsername();
void setName(std::string);
std::string getName();
void setPassword(std::string);
std::string getPassword();
void setPhone(std::string);
std::string getPhone();
void setEmail(std::string);
std::string getEmail();
void setAppointments(std::vector<std::string>);
std::vector<std::string> getAppointments();
void setDescriptions(std::vector<std::string>);
std::vector<std::string> getDescriptions();
// Adds a created appointment to the users appointment and description lists
void addAppointment(std::string);
void addDescription(std::string);
// sorts all appointments chronologically
void sortAppointments();
// deletes all appointments / descriptions for the user
void deleteAllAppointments();
// returns a string of all appointments/descriptions for the user
std::string returnAppointments();
// returns a string of all user data

```
std::string returnUserData();
// deletes appointment at the index given.
void deleteAppointment(int apptIndex);
// updates appointment. first entry is the appointment number, second is new data, third
boolean is date/time(true) or description(false) update
void updateAppointment(int, std::string, bool);
// overload == operator to compare two user objects
bool operator==(const User& user);
```