Time Logger

CSC-17B

Dr. Mark Lehr

Royce Nguyen

INTRODUCTION:

The time logger is based on the time logger for the computer lab at Riverside City College. The time logger process at the RCC Computer Lab would go as follows: You login to a computer using your ID number. The timer starts. When you end your session, the timer stops. Then the program will track the amount of time you have put into the computer lab. The admin/teachers can check the logs and assign points to the lab portion of your grade accordingly.

THOUGHT PROCESS ON C++ IMPLEMENTATION:

The time logger is a very straight forward project. There is less data in the project compared to the survey engine and this allows for better storage of the data. I chose to represent time as the number of seconds since the epoch time. This is easy to implement with the <ctime> library. I chose to implement the program as such that the time are recorded in session classes which carry a date class for start and end time, as well as the name of the class session the student is currently attending. There is no class for admins, rather there are just boolean values that are set that give permission for the admin to do certain administrative actions.

ADMIN and USER:

The user is only able to join, leave, and view previous sessions/classes. The admin is able to create and delete both classes, and students. There is currently no sorting/filtering option available.

DIFFICULTIES IN CODING:

The most difficult part of programming the time logger is managing the time. Date and time are an infamous problem in programming. It seems very easy to do, but there are some smaller intricacies one must do. Some problems I ran into are time zones, leap years, daylight savings, 0 days, 60 seconds, etc. Most can be solved with just math though.

PSEUDOCODE:

output main menu

Take user input

Validate user input

If (login)

Attempt login

If (login successful)

If (user)

Model handle user

If (admin)

Model handle admin

Else

Output menu again

If (exit)

Save data

Exit program

Actual Code:

#include <bits/stdc++.h>

#include <ctime>

#include "Session.hpp"

#include "Date.hpp"

#include "View.hpp"

#include "Controller.hpp"

#include "Model.hpp"

using namespace std;

int main()

{

Model sys;

View mainmenu("mainmenu.txt");

Controller mainMenuCont(mainmenu.getSize());

//do while menu

do

{

mainmenu.display();

mainMenuCont.TakeInput();

//name and id holders

string str, str2;

switch(mainMenuCont.ReturnInput())

{

case 1:

//getline clear buffer

cin.ignore();

cout << "Enter first and last name: ";

getline(cin, str);

cout << "Enter ID: ";

cin >> str2;

//pass to model class

if (sys.handleLogin(str, str2))

{

break;

}

//handle admin functions or student function based on the login

sys.getAdmin() ? sys.handleAdmin() : sys.handleStudent();

break;

case 2:

//terminate.

//all database updates are written within the student/admin menu after the respective edits.

cout << "Program Terminated";

exit(1);

default:

break;

}

}

while(mainMenuCont.ReturnInput() != 2);

//2 is for exit

}

#include "View.hpp"

View::View(string filename)

{

//constructor with filename variables. reads in all the lines for menu based output

ifstream in;

in.open(filename);

string str;

while (getline(in, str))

{

choices.push\_back(str);

}

}

void View::display()

{

cout << endl;

//number is within the class.

//makes it easier to edit

for (int i = 0; i < choices.size(); i++)

{

cout << i+1 << ": " << choices[i] << endl;

}

}

int View::getSize()

{

//for controller class for input validation.

return (int)this->choices.size();

}

#include "Student.hpp"

#include "Session.hpp"

Student::Student(string sname, string sid)

{

//name and id constructor

name = sname;

id = sid;

};

void Student::pushSession(Session s)

{

//adding a session after a student has logged out

sessions.push\_back(s);

};

void Student::displayAllInfo()

{

//output name and id

cout << "Name: " << this->name << endl;

cout << "ID: " << this->id << endl;

for (int i = 0; i < this->sessions.size(); i++)

{

//output class name and the time range of the respective sessions

cout << sessions[i].getClassName() << endl;

sessions[i].getStart().output();

sessions[i].getEnd().output();

}

};

//all basic getter functions

int Student::getSessionStartUnix(int i)

{

return sessions[i].getStartUnixTime();

};

int Student::getSessionEndUnix(int i)

{

return sessions[i].getEndUnixTime();

};

string Student::getSessionClassName(int i)

{

return sessions[i].getClassName();

};

#include "Session.hpp"

#include <bits/stdc++.h>

using namespace std;

//session constructor with aggregate initialization

Session::Session(string sclass\_name, int sstart, int send) : start(sstart), end(send)

{

class\_name = sclass\_name;

};

#include "Model.hpp"

#include "View.hpp"

#include "Controller.hpp"

#include "Student.hpp"

#include "Date.hpp"

Model::Model()

{

//basic init

this->loggedin = false;

this->admin = false;

this->currentid = "";

this->currentname = "";

this->currentsession = "";

this->insession = false;

//read in data from files

ifstream in;

in.open("classes.txt");

string str, str2;

while (in >> str)

{

classes.push\_back(str);

}

in.close();

in.ignore();

in.open("valid\_student\_ids.txt");

while (getline(in, str))

{

in >> str2;

valid\_student\_ids.push\_back({str, str2});

in.ignore();

}

in.close();

in.open("admin\_ids.txt");

while (getline(in, str))

{

in >> str2;

valid\_admin\_ids.push\_back({str, str2});

in.ignore();

}

in.close();

in.open("studentdata.txt");

string id, name, class\_name;

int start, end;

in >> id;

//read in survey data from students

while (id != "-1")

{

in.ignore();

getline(in, name);

Student a(name, id);

in >> class\_name;

while (class\_name != "-1")

{

in >> start >> end;

Session b(class\_name, start, end);

a.pushSession(b);

in >> class\_name;

}

students.push\_back(a);

in >> id;

}

};

bool Model::handleLogin(string name, string id)

{

if (id != "" && name != "")

{

//go through each valid id

for (int i = 0; i < valid\_student\_ids.size(); i++)

{

//student login success

if (valid\_student\_ids[i] == make\_pair(name, id))

{

this->loggedin = true;

this->currentid = id;

this->currentname = name;

this->admin = false;

}

}

for (int i = 0; i < valid\_admin\_ids.size(); i++)

{

//admin login success

if (valid\_admin\_ids[i] == make\_pair(name, id))

{

this->loggedin = true;

this->currentid = id;

this->admin = true;

}

}

}

if (this->loggedin)

{

//outputting login result to the user

cout << "Login Successful" << endl;

cout << (this->admin ? "Admin" : "Student") << endl;

return false;

}

else

{

cout << "Login Unsuccessful" << endl;

}

//return that the login was successful, otherwise, continue login loop

return true;

}

void Model::handleStudent()

{

View studentMenu("studentmenu.txt");

Controller studentMenuCont(studentMenu.getSize());

//tell the user what class they're in

if (this->insession)

{

cout << "Current Session: " << this->currentsession << endl;

}

studentMenu.display();

studentMenuCont.TakeInput();

while(this->studentSwitch(studentMenuCont.ReturnInput()))

{

//tell the user if they are in a class or not

if (this->insession)

{

cout << "Current Session: " << this->currentsession << endl;

}

else

{

cout << "You are currently not in a session" << endl;

}

studentMenu.display();

studentMenuCont.TakeInput();

}

};

void Model::handleAdmin()

{

View adminMenu("adminmenu.txt");

Controller adminMenuCont(adminMenu.getSize());

adminMenu.display();

adminMenuCont.TakeInput();

//take admin menu input

while (this->adminSwitch(adminMenuCont.ReturnInput()))

{

adminMenu.display();

adminMenuCont.TakeInput();

}

};

bool Model::adminSwitch(int choice)

{

//switch branching. function names pretty self explanatory.

switch(choice)

{

case 1:

this->displayStudents();

return true;

case 2:

this->displayClasses();

return true;

case 3:

this->createClass();

this->updateClass();

return true;

case 4:

this->deleteClass();

this->updateClass();

return true;

case 5:

this->addStudent();

this->updateStudent();

return true;

case 6:

this->deleteStudent();

this->updateStudent();

return true;

default:

this->admin = false;

this->loggedin = false;

this->currentid = "";

return false;

break;

}

};

bool Model::studentSwitch(int choice)

{

//switch for student. also self explanatory

switch(choice)

{

case 1:

this->joinClass();

return true;

case 2:

this->leaveClass();

this->updateHours();

return true;

case 3:

this->checkHours();

return true;

case 4:

if (this->insession)

{

this->leaveClass();

this->updateHours();

}

this->loggedin = false;

this->currentid = "";

this->currentname = "";

this->currentsession = "";

return false;

default:

cout << "Error";

return false;

}

};

void Model::displayStudents()

{

//output all students.

for (int i = 0; i < students.size(); i++)

{

students[i].displayAllInfo();

}

}

void Model::displayClasses()

{

//output the name of all classes.

for (int i = 0; i < classes.size(); i++)

{

cout << classes[i] << endl;

}

};

void Model::createClass()

{

string str;

cout << "Enter class: ";

cin >> str;

classes.push\_back(str);

};

void Model::deleteClass()

{

//delete string from vector. Does not affect previous logs.

int choice;

View delClass("classes.txt");

Controller delClassCont(delClass.getSize());

delClass.display();

delClassCont.TakeInput();

classes.erase(classes.begin() + delClassCont.ReturnInput()-1);

}

void Model::updateClass()

{

//rewrite classes into database.

ofstream out;

out.open("classes.txt");

for (int i = 0; i < classes.size(); i++)

{

out << classes[i] << endl;

}

out.close();

};

void Model::updateHours()

{

ofstream out;

out.open("studentdata.txt");

for (int i = 0; i < students.size(); i++)

{

//output name and id

out << students[i].getID() << endl;

out << students[i].getName() << endl;

for (int j = 0; j < students[i].getSize(); j++)

{

//output the class name, and their start/end times

out << students[i].getSessionClassName(j) << endl;

out << students[i].getSessionStartUnix(j) << endl;

out << students[i].getSessionEndUnix(j) << endl;

}

out << -1 << endl;

}

out << -1 << endl;

out.close();

};

void Model::joinClass()

{

if (this->insession)

{

//check if the user is already in a session

cout << "Error, you are already in a session" << endl;

return;

}

View joinClassMenu("classes.txt");

Controller joinClassCont(joinClassMenu.getSize());

joinClassMenu.display();

joinClassCont.TakeInput();

//successful login

this->insession = true;

this->currentsession = classes[joinClassCont.ReturnInput()-1];

//init timer

start = time(0);

};

void Model::leaveClass()

{

if (!this->insession)

{

cout << "Error, you are not currently in a session" << endl;

return;

}

this->insession = false;

//end timer;

end = time(0);

//create the session and push it in

Session a(currentsession, start, end);

this->currentsession = "";

for (int i = 0; i < students.size(); i++)

{

//validate id, ids should be unique

if (students[i].getID() == currentid)

{

students[i].pushSession(a);

}

}

};

void Model::checkHours()

{

//create a map, key is class name, value is time spent total

map<string, int> times;

//iterate through all classes, add up all the times.

for (int i = 0; i < students.size(); i++)

{

if (students[i].getName() == this->currentname && students[i].getID() == this->currentid)

{

for (int j = 0; j < students[i].getSize(); j++)

{

times[students[i].getSessionClassName(j)] += students[i].getSessionEndUnix(j) - students[i].getSessionStartUnix(j);

}

}

}

//output results

for (auto it : times)

{

cout << it.first << ": " << it.second << endl;

}

};

void Model::addStudent()

{

cin.ignore();

cout << "Enter a student's first and last name: ";

string str, str2;

getline(cin, str);

cout << "Enter the student's ID: ";

cin >> str2;

//basic student creation

this->pushStudent(make\_pair(str, str2));

};

void Model::pushStudent(pair<string, string> a)

{

//push a pair value to valid student id's

valid\_student\_ids.push\_back(a);

};

void Model::deleteStudent()

{

View delStudentMenu("valid\_student\_ids.txt");

Controller delStudentCont(delStudentMenu.getSize());

delStudentMenu.display();

delStudentCont.TakeInput();

//using vector erase to destroy a pair based erase.

//input can take either the value for the id or the name.

//math magic :P

valid\_student\_ids.erase(valid\_student\_ids.begin() + (delStudentCont.ReturnInput() % 2 == 0 ? delStudentCont.ReturnInput()-1 : delStudentCont.ReturnInput()) / 2);

};

void Model::updateStudent()

{

ofstream out;

//rewrite student data to the database

out.open("valid\_student\_ids.txt");

for (int i = 0; i < valid\_student\_ids.size(); i++)

{

out << valid\_student\_ids[i].first << endl;

out << valid\_student\_ids[i].second << endl;

}

};

#include "Date.hpp"

#include <bits/stdc++.h>

using namespace std;

Date::Date(int secs)

{

//set the unit time

this->unixtime = secs;

//days per month

vector<int> days\_in\_month {31,28,31,30,31,30,31,31,30,31,30,31};

//start year of unix time

this->year = 1970;

//subtracting seconds per year respective to leap year

while (secs - (this->isLeap() ? 366 : 365) \* 24 \* 60 \* 60 > 0)

{

secs -= (this->isLeap() ? 366 : 365) \* 24 \* 60 \* 60;

this->year++;

}

//add day to february if leap year

if (this->isLeap())

{

days\_in\_month[1]++;

}

//subtracting seconds per month

this->month = 0;

while (secs - days\_in\_month[this->month]\*60\*60\*24 > 0)

{

secs -= days\_in\_month[this->month]\*60\*60\*24;

this->month++;

}

//subtracting seconds per day

this->day = 0;

for (int i = 0; i < days\_in\_month[this->month] && secs - 60\*60\*24 > 0; i++)

{

this->day++;

secs -= 60\*60\*24;

}

//subtracting seconds per hour

this->hour = 0;

while (secs - 60\*60 > 0)

{

this->hour++;

secs -= 60\*60;

}

//adding to adjust to timezone and daylight savings

this->hour += 17;

if (this->hour > 23)

{

hour -= 24;

this->day++;

}

//per minute

this->minute = 0;

while (secs - 60 >= 0)

{

secs -= 60;

this->minute++;

}

//remainder is seconds;

this->seconds = secs;

};

bool Date::isLeap()

{

return isLeap(this->year);

};

bool Date::isLeap(int year)

{

//basic leap year validator

if (year % 4 == 0)

{

if (year % 100 == 0 && year % 400 == 0)

{

return true;

}

else if (year % 400 != 0 && year % 100 == 0)

{

return false;

}

else

{

return true;

}

}

return false;

};

void Date::output()

{

//output format that nice for the eyes

cout << this->month+1 << "/" << this->day << "/" << this->year << endl;

cout << this->hour << ":";

cout << (this->minute < 10 ? "0" : "") << this->minute << ":";

cout << (this->seconds < 10 ? "0" : "") << this->seconds << endl;

};

#include "Controller.hpp"

Controller::Controller(int end)

{

//setting max for input validation

max = end;

}

void Controller::TakeInput()

{

cin >> choice;

//check if the input is within range

while (choice < 1 || choice > max)

{

cout << "Try again: ";

cin >> choice;

}

}

int Controller::ReturnInput()

{

return this->choice;

}