**Course**: ENSF 614 – Fall 2023

**Lab #**: Lab 3

**Instructor**: Moussavi

**Student Name**: Yajur Vashisht, Balkarn Gill

**Submission Date**: October 13th, 2023

**Part A**

A graph paper with a diagram

Description automatically generated

**Part B**

A graph paper with a diagram

Description automatically generated

**Part C**

lab3Clock.h

#ifndef LAB3CLOCK\_H

#define LAB3CLOCK\_H

class Clock {

public:

Clock();

Clock(int s);

Clock(int h, int m, int s);

void set\_hour(int h);

void set\_minute(int m);

void set\_second(int s);

int get\_hour() const;

int get\_minute() const;

int get\_second() const;

void increment();

void decrement();

int hms\_to\_sec() const;

void sec\_to\_hms(int totalSeconds);

void add\_seconds(int secondsToAdd);

private:

int hour;

int minute;

int second;

};

#endif

lab3Clock.cpp

#include "lab3Clock.h"

*// Default Constructor*

Clock::Clock() {

hour = 0;

minute = 0;

second = 0;

}

*// Used for seconds*

Clock::Clock(int s) {

if (s < 0) {

hour = 0;

minute = 0;

second = 0;

} else {

hour = s / 3600;

minute = (s % 3600) / 60;

second = (s % 3600) % 60;

}

}

*// Used for all variables in clock*

Clock::Clock(int h, int m, int s) {

if (h < 0 || h > 23 || m < 0 || m > 59 || s < 0 || s > 59) {

hour = 0;

minute = 0;

second = 0;

} else {

hour = h;

minute = m;

second = s;

}

}

void Clock::set\_hour(int h) {

if (h >= 0 && h <= 23) {

hour = h;

}

}

void Clock::set\_minute(int m) {

if (m >= 0 && m <= 59) {

minute = m;

}

}

void Clock::set\_second(int s) {

if (s >= 0 && s <= 59) {

second = s;

}

}

int Clock::get\_hour() const {

return hour;

}

int Clock::get\_minute() const {

return minute;

}

int Clock::get\_second() const {

return second;

}

void Clock::increment() {

second++;

if (second >= 60) {

second = 0;

minute++;

if (minute >= 60) {

minute = 0;

hour++;

if (hour >= 24) {

hour = 0;

}

}

}

}

void Clock::decrement() {

second--;

if (second < 0) {

second = 59;

minute--;

if (minute < 0) {

minute = 59;

hour--;

if (hour < 0) {

hour = 23;

}

}

}

}

int Clock::hms\_to\_sec() const {

return hour \* 3600 + minute \* 60 + second;

}

void Clock::sec\_to\_hms(int totalSeconds) {

hour = totalSeconds / 3600;

minute = (totalSeconds % 3600) / 60;

second = totalSeconds % 60;

}

void Clock::add\_seconds(int secondsToAdd) {

if (secondsToAdd > 0) {

int totalSeconds = hms\_to\_sec();

totalSeconds += secondsToAdd;

sec\_to\_hms(totalSeconds);

}

}

Output

(base) yajurvashisht@yajurs-macbook Lab 3 % g++ -o lab3exe\_C lab3Clock.cpp lab3exe\_C.cpp

(base) yajurvashisht@yajurs-macbook Lab 3 % ./lab3exe\_C

Object t1 is created. Expected time is: 00:00:00

00:00:00

Object t1 incremented by 86400 seconds. Expected time is: 00:00:00

00:00:00

Object t2 is created. Expected time is: 00:00:05

24:00:05

Object t2 decremented by 6 seconds. Expected time is: 23:59:59

23:59:59

After setting t1's hour to 21. Expected time is: 21:00:00

21:00:00

Setting t1's hour to 60 (invalid value). Expected time is: 21:00:00

21:00:00

Setting t2's minute to 20. Expected time is: 23:20:59

23:20:59

Setting t2's second to 50. Expected time is 23:20:50

23:20:50

Adding 2350 seconds to t2. Expected time is: 00:00:00

24:00:00

Adding 72000 seconds to t2. Expected time is: 20:00:00

44:00:00

Adding 216000 seconds to t2. Expected time is: 08:00:00

104:00:00

Object t3 is created. Expected time is: 00:00:00

00:00:00

Adding 1 second to clock t3. Expected time is: 00:00:01

00:00:01

After calling decrement for t3. Expected time is: 00:00:00

00:00:00

After incrementing t3 by 86400 seconds. Expected time is: 00:00:00

00:00:00

After decrementing t3 by 86401 seconds. Expected time is: 23:59:59

23:59:59

After decrementing t3 by 864010 seconds. Expected time is: 23:59:49

23:59:49

t4 is created with invalid value (25 for hour). Expected to show: 00:00:00

00:00:00

t5 is created with invalid value (-8 for minute). Expected to show: 00:00:00

00:00:00

t6 is created with invalid value (61 for second). Expected to show: 00:00:00

00:00:00

t7 is created with invalid value (negative value). Expected to show: 00:00:00

00:00:00

**Part D**

MyArray.cpp:

#include <iostream>

using namespace std;

#include "MyArray.h"

#include <cassert>

#include <algorithm>

MyArray::MyArray() : sizeM(0), storageM(nullptr) {}

MyArray::MyArray(const EType \*builtin, int sizeA):sizeM(sizeA) {

if (sizeM > 0) {

storageM = new EType[sizeM];

for (int i = 0; i < sizeM; ++i) {

storageM[i] = builtin[i];

}

} else {

storageM = nullptr;

}

}

MyArray::MyArray(const MyArray& source) : sizeM(source.sizeM) {

if (sizeM > 0) {

storageM = new EType[sizeM];

for (int i = 0; i < sizeM; ++i) {

storageM[i] = source.storageM[i];

}

} else {

storageM = nullptr;

}

}

MyArray& MyArray::operator =(const MyArray& rhs) {

if (this != &rhs) {

delete[] storageM;

sizeM = rhs.sizeM;

if (sizeM > 0) {

storageM = new EType[sizeM];

for (int i = 0; i < sizeM; ++i) {

storageM[i] = rhs.storageM[i];

}

} else {

storageM = nullptr;

}

}

return \*this;

}

MyArray::~MyArray() {

delete[] storageM;

}

int MyArray::size() const {

return sizeM;

}

EType MyArray::at(int i) const {

if (i >= 0 && i < sizeM) {

return storageM[i];

} else {

cout << "Not valid." << endl;

}

}

void MyArray::set(int i, EType new\_value) {

if (i >= 0 && i < sizeM) {

storageM[i] = new\_value;

} else {

cout << "Not valid." << endl;

}

}

void MyArray::resize(int new\_size) {

if (new\_size == sizeM) {

return;

}

EType\* newStorage = new EType[new\_size];

int copySize = std::min(new\_size, sizeM);

for (int i = 0; i < copySize; ++i) {

newStorage[i] = storageM[i];

}

delete[] storageM;

sizeM = new\_size;

storageM = newStorage;

}

Output:

Elements of a: 0.5 1.5 2.5 3.5 4.5

(Expected: 0.5 1.5 2.5 3.5 4.5)

Elements of b after first resize: 10.5 11.5 12.5 13.5 14.5 15.5 16.5

(Expected: 10.5 11.5 12.5 13.5 14.5 15.5 16.5)

Elements of b after second resize: 10.5 11.5 12.5

(Expected: 10.5 11.5 12.5)

Elements of b after copy ctor check: 10.5 11.5 12.5

(Expected: 10.5 11.5 12.5)

Elements of c after copy ctor check: -1.5 11.5 12.5

(Expected: -1.5 11.5 12.5)

Elements of a after operator = check: -10.5 1.5 2.5 3.5 4.5

(Expected: -10.5 1.5 2.5 3.5 4.5)

Elements of b after operator = check: -11.5 1.5 2.5 3.5 4.5

(Expected: -11.5 1.5 2.5 3.5 4.5)

Elements of c after operator = check: 0.5 1.5 2.5 3.5 4.5

(Expected: 0.5 1.5 2.5 3.5 4.5)