Course: ENSF 614 – Fall 2023  
Lab #: Lab 3  
Instructor: Mahmood Moussavi  
Student Name: Satchytan Karalasingham, Romil Dhagat  
Submission Date: October 13, 2023

Lab 3 Exercise A:

AR DIAGRAMS:  
A blackboard with white text

Description automatically generated

Lab 3 Exercise B:

AR DIAGRAM:

A blackboard with white text and a diagram

Description automatically generated

Lab 3 Exercise C:

CODE:

// lab3Clock.h

// ENSF 614 Lab 3 Exercise C

// Instructor: M. Moussavi

// Submitted by: Satchytan Karalasingham, Romil Dhagat

// Development Date: October 13th, 2023// lab3CLock.h

// By: Romil Dhagat and Satchy Keralasingham

#ifndef lab3Clock\_H

#define lab3Clock\_H

**class** Clock{

**private**:

**int** hour;

**int** minute;

**int** second;

**int** hms\_to\_sec() **const**;

**void** sec\_to\_hms(**int** sec);

**public**:

Clock();

Clock(**int** seconds);

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

**int** get\_hour() **const**;

**int** get\_minute() **const**;

**int** get\_second() **const**;

**void** set\_hour(**int** h);

**void** set\_minute(**int** m);

**void** set\_second(**int** s);

**void** increment();

**void** decrement();

**void** add\_seconds(**int** seconds);

};

#endif

// lab3Clock.cpp

// ENSF 614 Lab 3 Exercise C

// Instructor: M. Moussavi

// Submitted by: Satchytan Karalasingham, Romil Dhagat

// Development Date: October 13th, 2023// lab3CLock.cpp

// By: Romil Dhagat and Satchy Keralasingham

#include <iostream>

**using** **namespace** std;

#include "lab3Clock.h"

Clock::Clock(): hour(0), minute(0), second(0){

}

Clock::Clock(**int** sec){

sec\_to\_hms(sec);

}

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

**if** (h >= 0 && h <= 23 && m >= 0 && m <= 59 && s >= 0 && s <= 59) {

hour = h;

minute = m;

second = s;

} **else** {

hour = 0;

minute = 0;

second = 0;

}

}

**int** Clock::get\_hour() **const** {

**return** hour;

}

**int** Clock::get\_minute() **const** {

**return** minute;

}

**int** Clock::get\_second() **const** {

**return** second;

}

**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;

}

}

**void** Clock::increment() {

**int** sec = hms\_to\_sec();

sec = (sec + 1) % 86400;

sec\_to\_hms(sec);

}

**void** Clock::decrement() {

**int** sec = hms\_to\_sec();

sec = (sec - 1 + 86400) % 86400;

sec\_to\_hms(sec);

}

**void** Clock::add\_seconds(**int** seconds) {

**int** sec = hms\_to\_sec() + seconds;

sec = (sec + 86400) % 86400;

sec\_to\_hms(sec);

}

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

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

}

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

hour = sec / 3600;

sec %= 3600;

minute = sec / 60;

second = sec % 60;

**if** (second < 0 || second > 59) {

second = 0;

}

**if** (hour < 0 || hour > 23) {

hour = 0;

}

**if** (minute < 0 || minute > 59) {

minute = 0;

}

}

OUTPUT:

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

00:00:05

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

00:00:00

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

20:00:00

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

08:00:00

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

Program ended with exit code: 0

SCREENSHOT:

A screenshot of a computer

Description automatically generated

Lab 3 Exercise D:

CODE:

// myArray.cpp

// ENSF 614 Lab 3 Exercise D

// Instructor: M. Moussavi

// Submitted by: Satchytan Karalasingham, Romil Dhagat

// Development Date: October 13th, 2023

// This program demonstrates the use of all the features

// of the MyArray class.

#include "MyArray.h"

MyArray::MyArray() : sizeM(0), storageM(**nullptr**)

{

// Constructor for an empty array

}

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

{

// Constructor to create an object by copying a built-in array

storageM = **new** EType[sizeA];

**for** (**int** i = 0; i < sizeA; i++)

{

storageM[i] = builtin[i];

}

}

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

{

// Copy constructor

storageM = **new** EType[sizeM];

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

{

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

}

}

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

{

// Assignment operator

**if** (**this** != &rhs)

{

// Check for self-assignment

**if** (storageM)

{

**delete**[] storageM;

}

sizeM = rhs.sizeM;

storageM = **new** EType[sizeM];

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

{

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

}

}

**return** \***this**;

}

MyArray::~MyArray()

{

// Destructor

**if** (storageM)

{

**delete**[] storageM;

}

}

**int** MyArray::size() **const**

{

**return** sizeM;

}

EType MyArray::at(**int** i) **const**

{

// Check for valid index

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

{

**return** storageM[i];

}

**else**

{

**return** 0;

}

}

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

{

// Check for valid index

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

{

storageM[i] = new\_value;

}

}

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

{

**if** (new\_size < 0)

{

}

**else** **if** (new\_size == 0)

{

// If the new size is zero, deallocate the storage

**if** (storageM)

{

**delete**[] storageM;

storageM = **nullptr**;

}

sizeM = 0;

}

**else**

{

// Allocate a new array and copy elements

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

**int** elementsToCopy = (new\_size < sizeM) ? new\_size : sizeM;

**for** (**int** i = 0; i < elementsToCopy; i++)

{

newStorage[i] = storageM[i];

}

// Deallocate the old storage and update the size and storage pointer

**if** (storageM)

{

**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)

Program ended with exit code: 0

SCREENSHOT:

A screenshot of a computer program

Description automatically generated