Course: Programming Fundamental – ENSF 337

Lab #: Lab 7

Instructor: Mansouri Habibabadi

Student Name: Drew Hengehold

Lab Section: B01

Date submitted: Nov 17, 2022

UCID: 30151823

Exercise A:

Diagram

Description automatically generated

Exercise C:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Source Code lab7Clock.h:*

//

// lab7Clock.h

// Clock

//

// Created by Drew Hengehold on 11/9/22.

//

#ifndef lab7Clock\_h

#define lab7Clock\_h

class Clock {

public:

// default consturctor

Clock(): secM(0), minM(0), hourM(0) {}

// seconds constructor

Clock(double secM);

// Full Initialization

Clock(double secM, double minM, double hourM);

double get\_second() const;

double get\_minute() const;

double get\_hour() const;

void set\_second(double n);

void set\_minute(double n);

void set\_hour(double n);

void increment();

void decrement();

void add\_seconds(double n);

private:

double hms\_to\_second();

void sec\_to\_hms(double n);

double convert\_seconds(double seconds);

double convert\_minutes(double seconds);

double convert\_hours(double seconds);

void check\_time();

double secM;

double minM;

double hourM;

};

#endif /\* lab7Clock\_h \*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Source Code lab7Clock.cpp:*

//

// lab7Clock.cpp

// Clock

//

// Created by Drew Hengehold on 11/9/22.

//

#include <stdio.h>

#include "lab7Clock.h"

Clock::Clock(double hour, double min, double seconds): hourM(hour), minM(min), secM(seconds){

check\_time();

}

Clock::Clock(double seconds):secM(convert\_seconds(seconds)), minM(convert\_minutes(seconds)), hourM(convert\_hours(seconds)){

check\_time();

}

double Clock::get\_second() const{

return secM;

}

double Clock::get\_minute() const{

return minM;

}

double Clock::get\_hour() const{

return hourM;

}

void Clock::set\_second(double n){

if(n< 60)

secM = n;

}

void Clock::set\_minute(double n){

if(n < 60)

minM = n;

}

void Clock::set\_hour(double n){

if(n < 24)

hourM = n;

}

void Clock::increment(){

double seconds = hms\_to\_second();

seconds++;

sec\_to\_hms(seconds);

}

void Clock::decrement(){

double seconds = hms\_to\_second();

seconds--;

if(seconds<0){

seconds = 86400-(((int)seconds%(-86400))\*(-1));

}

sec\_to\_hms(seconds);

}

void Clock::add\_seconds(double n){

double seconds = hms\_to\_second();

seconds += n;

sec\_to\_hms(seconds);

}

//PRIVATE FUNCTIONS START

double Clock::hms\_to\_second(){

return secM + (minM\*60) + (hourM\*3600);

}

void Clock::sec\_to\_hms(double n){

secM = convert\_seconds(n);

minM = convert\_minutes(n);

hourM = convert\_hours(n);

}

double Clock::convert\_hours(double seconds){

double hours;

hours = (double)(((int)seconds/3600)%24);

return hours;

}

double Clock::convert\_minutes(double seconds){

double minutes;

minutes = ((int)seconds%3600)/60;

return (double)minutes;

}

double Clock::convert\_seconds(double seconds){

return (double)(((int)seconds%3600)%60);

}

void Clock::check\_time(){

if((secM < 0) or (secM >= 60) or (hourM < 0) or (hourM >= 24) or (minM < 0) or (minM >= 60)){

secM = 0;

minM = 0;

hourM = 0;

}

}

Output Screenshot:

Text

Description automatically generated

Kept giving me clang errors I could figure out so had to run through terminal with XCode

Exercise D:

***PART 1:***

*AR Diagram for Point 1*

Diagram, schematic

Description automatically generated

*Question Answers:*

Q1: At point 2, the constructor was called three times, and the destructor was called once.

Q2: At the end of the main, the constructor was called three times, and the destructor has been called once.

***PART 2:***

*Source Code append Function:*

void DynString::append(const DynString& tail)

{

int big\_length = tail.lengthM + lengthM;

char \*big\_memory = new char[big\_length+1];

assert(big\_memory != 0);

for(int i = 0; i <lengthM; i++)

big\_memory[i] = storageM[i];

for(int i = 0; i <tail.lengthM; i++)

big\_memory[i + lengthM] = tail.storageM[i];

big\_memory[big\_length] = '\0';

delete[] storageM;

storageM = big\_memory;

lengthM = big\_length;

}

Program output:

Text

Description automatically generated

Exercise E:

*Source Code:*

// ENSF 337- lab 7 - Exercise D

// simpleVector.cpp

// Student functions written by Drew Hengehold

/\*

NOTES ON MEMORY ALLOCATION POLICIES FOR SimpleVector OBJECT:

- If vector objects are supposed to be empty storageM will be NULL and the values

of sizeM and capacityM must be set to zero.

- If the objects of vector are supposed to be initialize with supplied values of

an array, the size a and values in the dynamically memory space for storageM

must be identical to the size and values in the supplied array. And, the

values of sizeM and capacityM should be both set to the exact size of array.

- If any member function need to resize the vector, it should check the values of

sizeM and capacityM:

- If sizeM is equal to capacityM and vector is empty (i.e. capaictyM is equal to

zero), capcityM should be changed to 2.

- Otherwise, if sizeM is equal to capacityM, and capacityM is not zero (i.e.

vector is NOT empty), the value of capacityM should be doubled up

EXAMPLE: if capacityM is 5 and sizeM is also 5:

1. the value of capacityM should be changed to 10.

2. the dynamically allocated memory space for storageM should be reallocated

to 10.

3. The current values in the vector should be preserved and any unnecessary

dynamically allocated memory must to deallocated.

\*/

#include "simpleVector.h"

#include <cassert>

using namespace std;

SimpleVector::SimpleVector(const TYPE \*arr, int n) {

storageM = new TYPE[n];

sizeM = n;

capacityM = n;

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

storageM[i] = arr[i];

}

TYPE& SimpleVector::at(int i) {

assert(i >= 0 && i < sizeM);

return storageM[i];

}

const TYPE& SimpleVector::at(int i)const {

assert(i >= 0 && i < sizeM);

return storageM[i];

}

// The followng member function should follow the above-mentioned memory

// management policy to resize the vector, if necessary. More specifically:

// - If sizeM < capacityM it doesn't need to make any changes to the size of

// allocated memory for vector

// - Otherwise it follows the above-mentioned memory policy to create additionl

// memory space and adds the new value, val, to the end of the current vector

// and increments the value of sizeM by 1

void SimpleVector::push\_back(TYPE val) {

if(sizeM == capacityM){

int new\_capacity = (capacityM == 0) ? 2:2 \*capacityM;

TYPE \*new\_storage = new TYPE[new\_capacity];

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

new\_storage[i] = storageM[i];

delete[] storageM;

storageM = new\_storage;

}

storageM[sizeM] = val;

sizeM++;

}

SimpleVector::SimpleVector(const SimpleVector& source) {

sizeM= 0;

capacityM = 0;

storageM = 0;

TYPE \*storage = new TYPE[source.size()];

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

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

storageM = storage;

sizeM = capacityM = source.size();

}

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

if(this != &rhs){

TYPE \*storage = new TYPE[rhs.size()];

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

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

delete[] storageM;

storageM = storage;

sizeM = capacityM = rhs.size();

}

return \*this;

}

*Output:*

Text

Description automatically generated