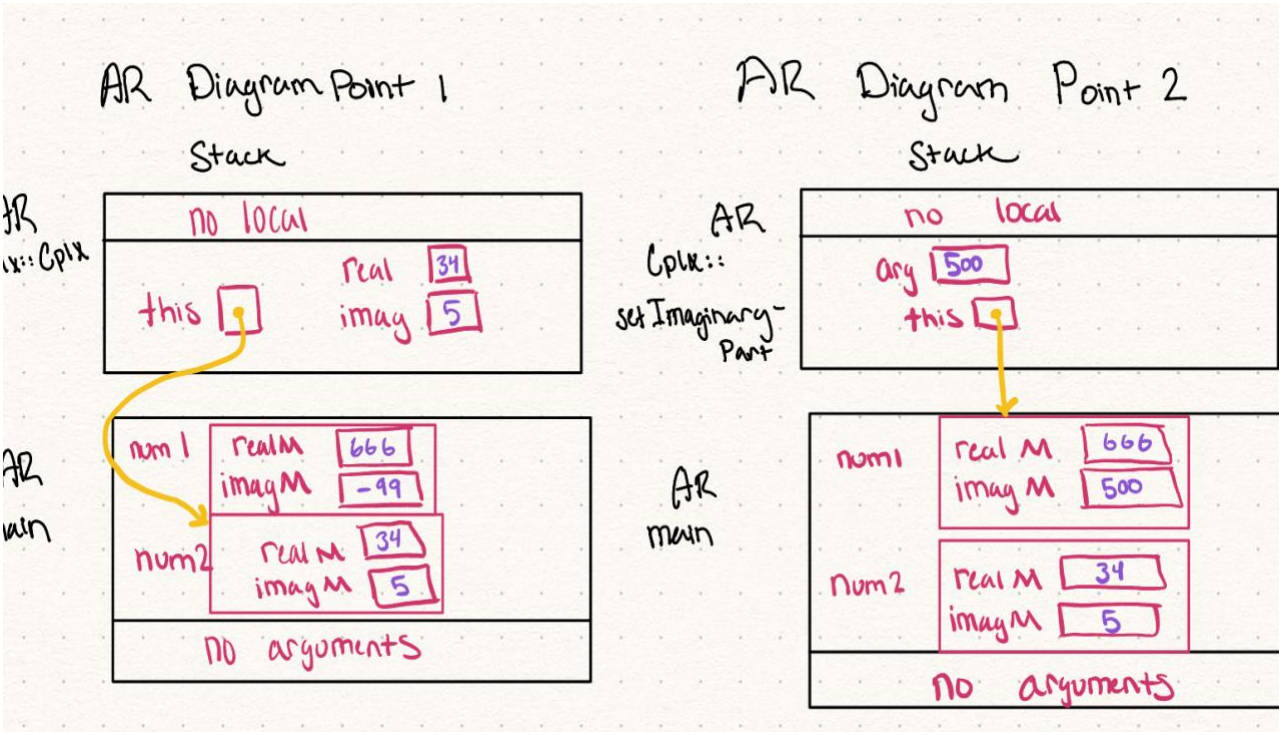
Course: Programming Fundamental – ENSF 337
Lab #: Lab 7
Instructor: Mansouri Habibabadi
Student Name: Drew Hengehold
Lab Section: B01
Date submitted: Nov 17, 2022
UCID: 30151823

Exercise A:


AR Diagram Point 1
Stack
AR
Cplx::Cplx
no local
this
real 34
imag 5
AR
main
num1
realM 666
imagM -99
num2
realM 34
imagM 5
no arguments
AR Diagram Point 2
Stack
AR
Cplx::
setImaginary-
Part
no local
arg 500
this
AR
main
num1
real M 666
imag M 500
num2
real M 34
imag M 5
no arguments

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);
}
```

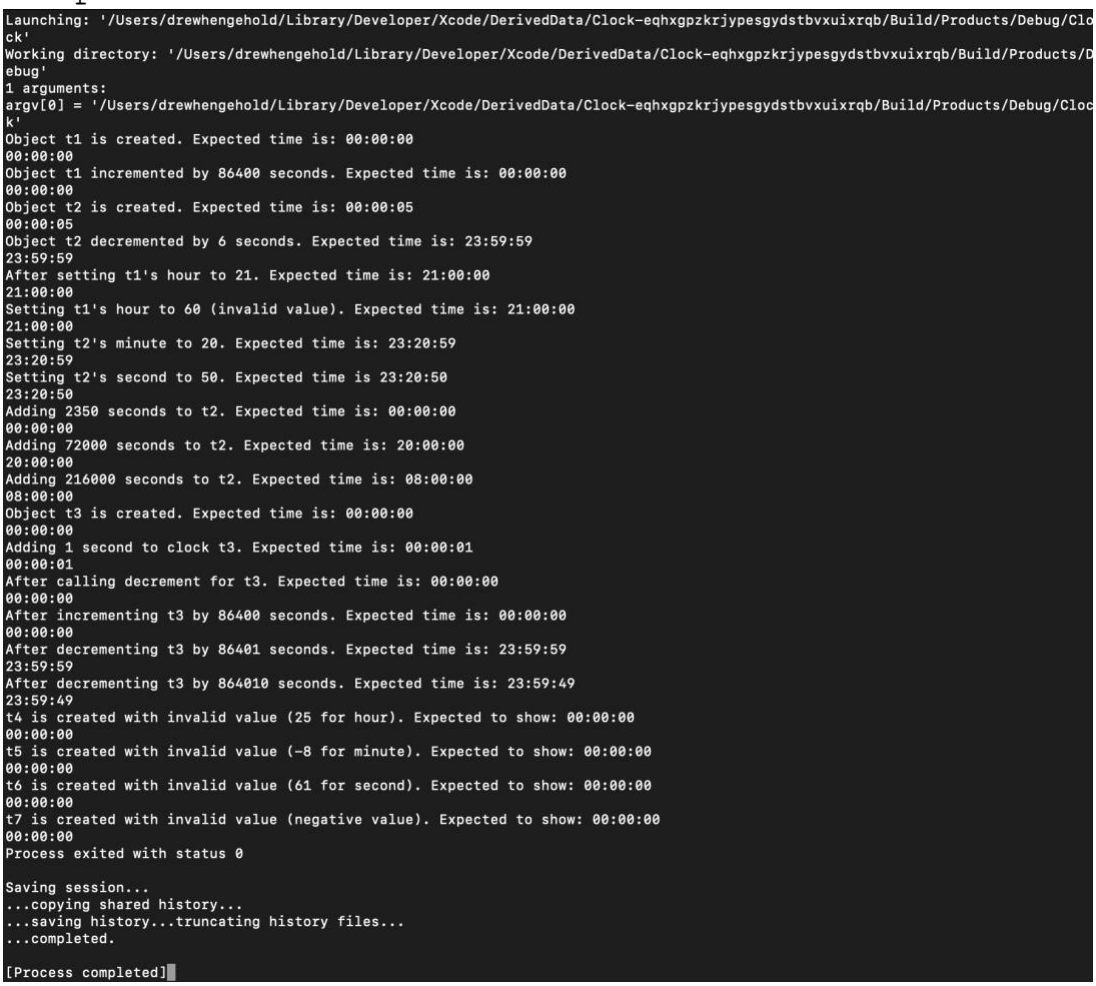
```
double Clock::convert_hours(double seconds){
    double hours;
    hours = (double)(((int)seconds/3600)%24);
    return hours;
}

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:

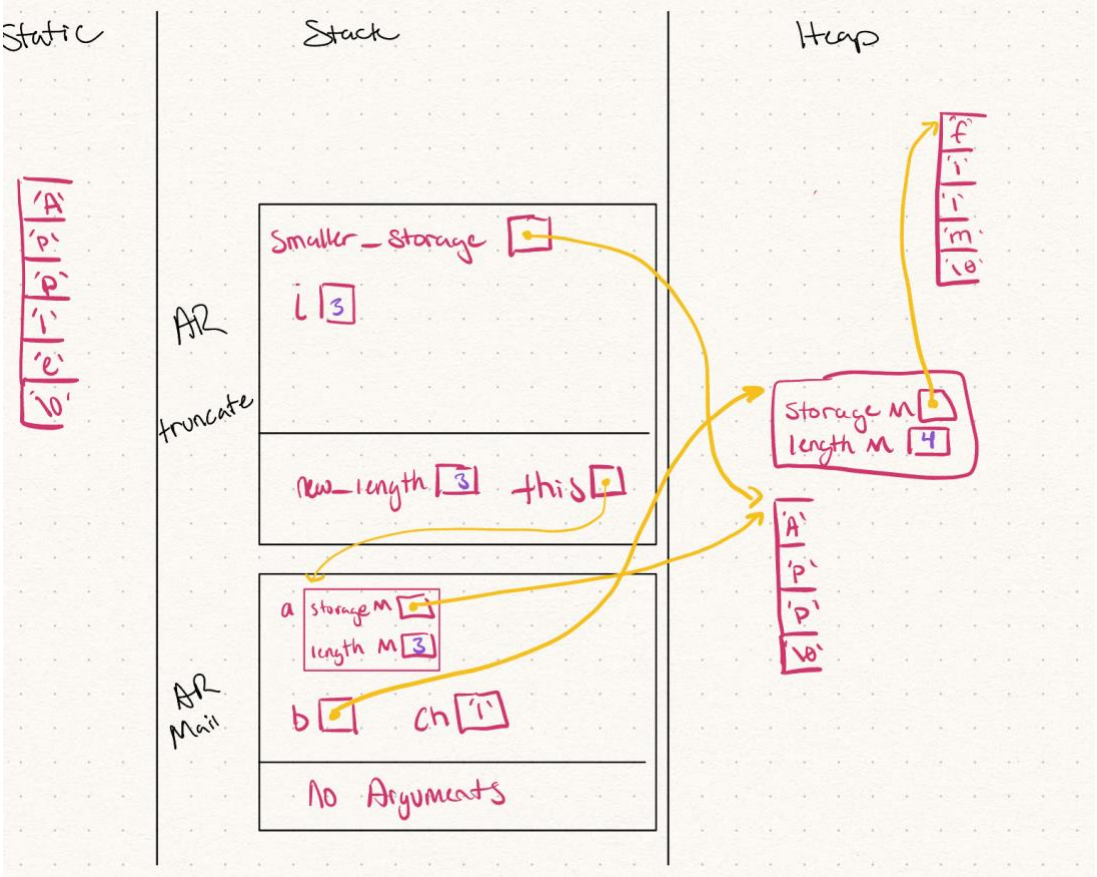



Kept giving me clang errors I could figure out so had to run through terminal with XCode

Exercise D:

***PART 1:***

*AR Diagram for Point 1*




*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:

```
Length of x: 3 (expected 3).

Contents of x: "" (expected "").
Length of x: 0 (expected 0).

Contents of x: "foot" (expected "foot").
Length of x: 4 (expected 4).

Contents of x: "foot" (expected "foot").
Length of x: 4 (expected 4).

Contents of x: "football" (expected "football").
Length of x: 8 (expected 8).
```

## 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:*

```
Object v1 is expected to display: 45 69 12
45 69 12
Object v2 is expected to diaplay: 3000 6000 7000 8000
3000 6000 7000 8000

After two calls to at v1 is expected to display: 1000 2000 12:
1000 2000 12

v2 expected to display: 3000 6000 7000 8000 21 28
3000 6000 7000 8000 21 28
```