Course: ENSF694 – Summer 2025

Lab #: Lab 3

Instructor: Mahmood Moussavi

Student Name: John Zhou

Submission Date: July 24, 2025

Exercise A  
The static area only contains string constant like "   " , "\nin mars: " etc.

## Point1

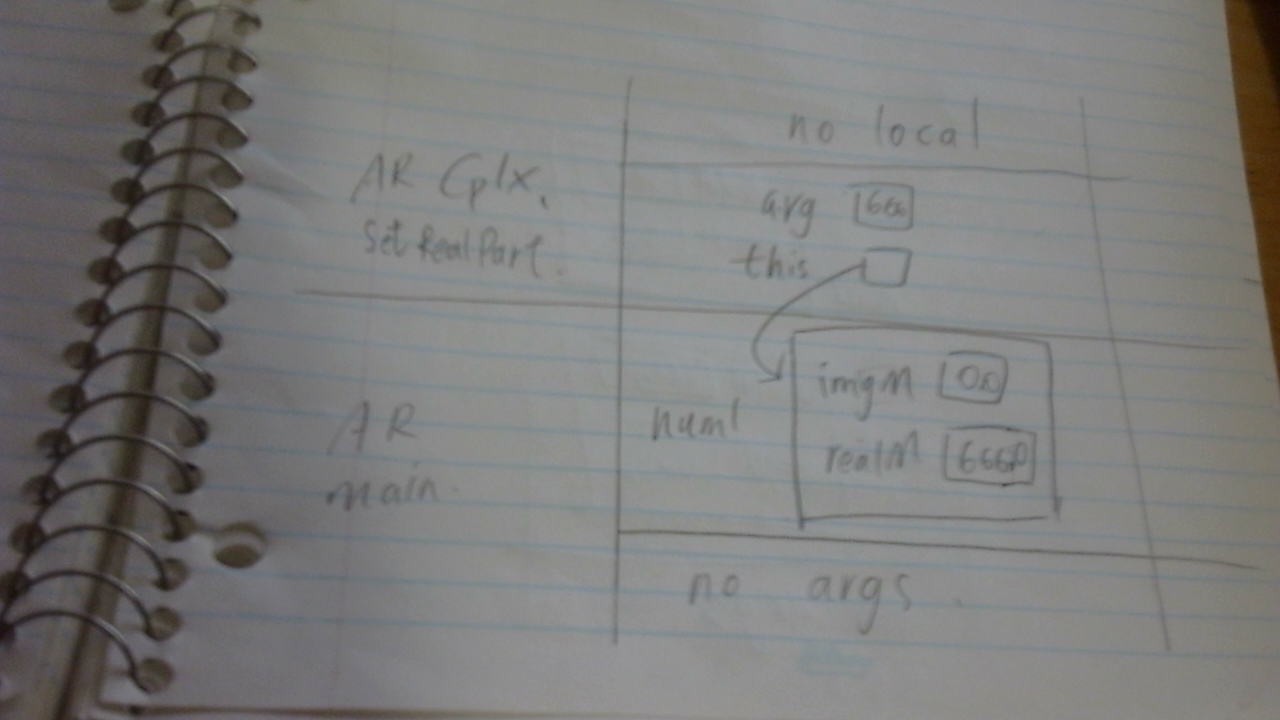
## 

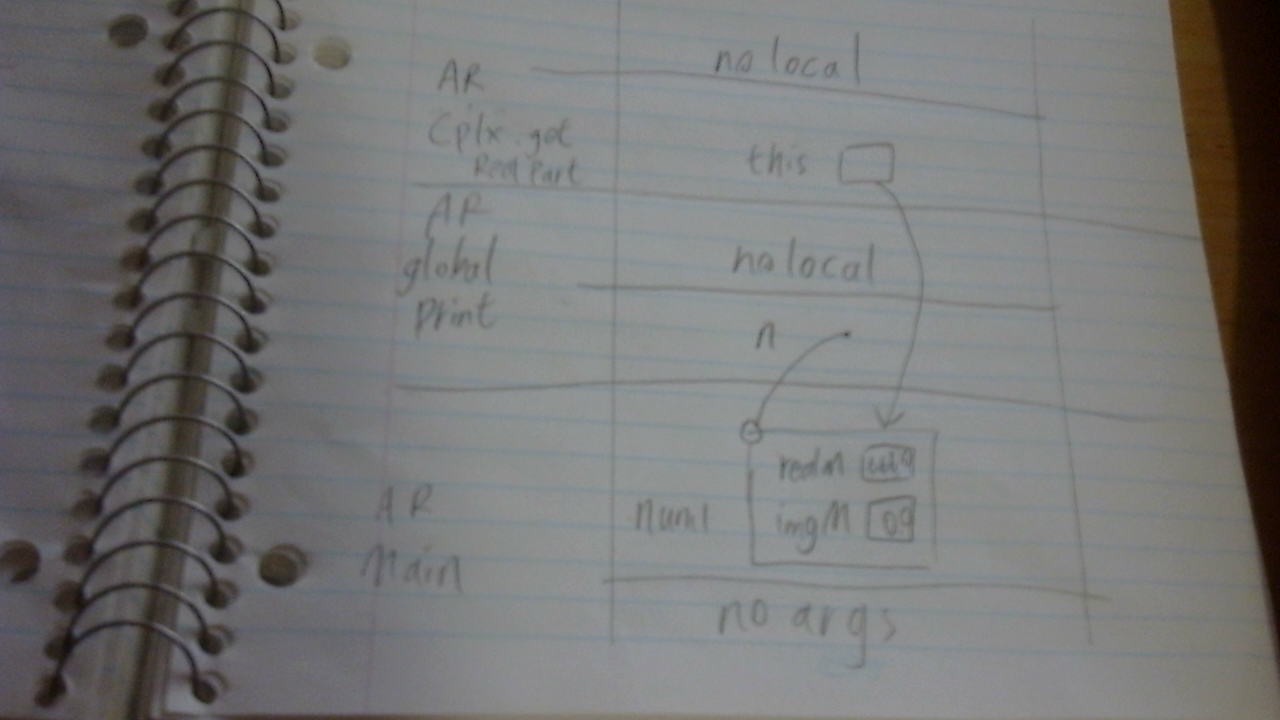
## Point2

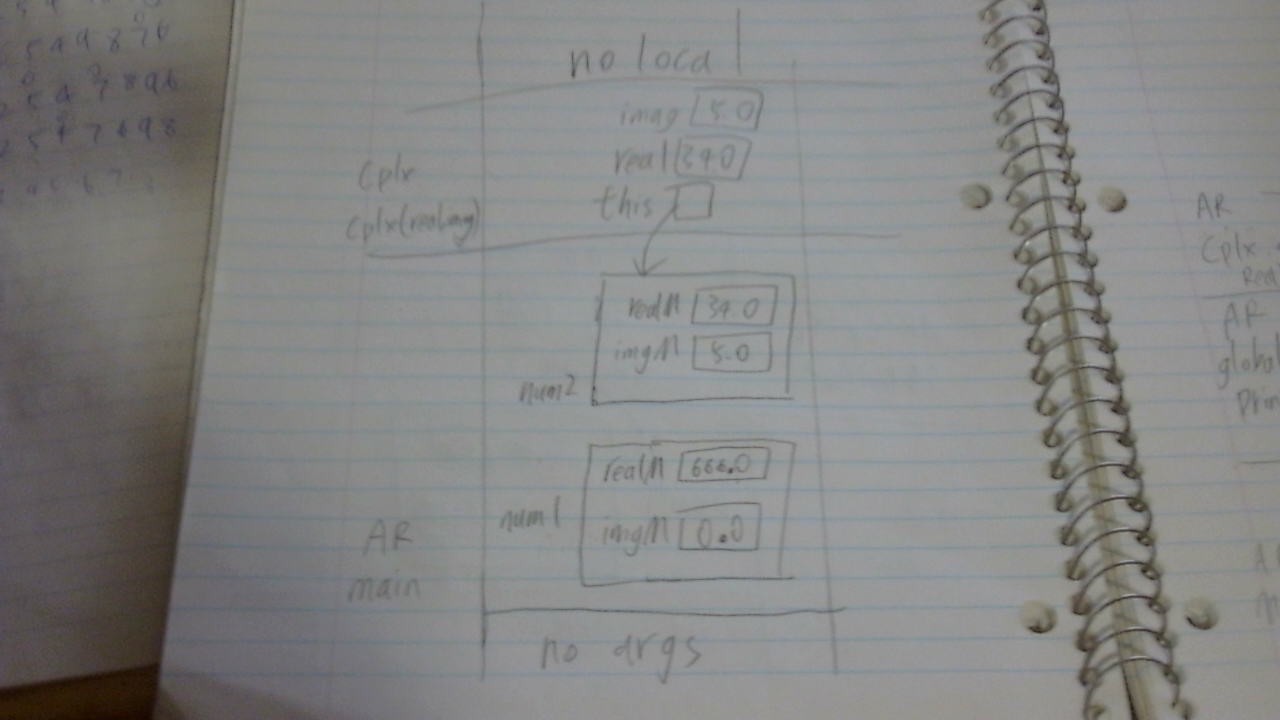
## 

Exercise B

The static area only contains string constant like "\nYour complex number is: ("

,  "\nTesting member functions add and subtract: \n" etc.  
  
Point1  
stack area  


Point2  
stack area  


Point3  
stack area  


Exercise C  
cpp  
// lab3Clock.cpp

// ENSF 694 Summer 2025 LAB 3 - EXERCISE C

// Created by: John Zhou

#include "lab3Clock.h"

#include <iostream>

using namespace std;

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

{

}

Clock::Clock(int s)

{

    if (s < 0)

    {

        \*this = Clock();

    }

    else

    {

        sec\_to\_hms(s);

    }

}

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

                                                    minute(m), second(s)

{

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

    {

        \*this = Clock();

    }

    else

    {

        this->hour = h;

        this->minute = m;

        this->second = s;

    }

}

int Clock::get\_hour() const

{

    return hour;

}

int Clock::get\_minute() const

{

    return minute;

}

int Clock::get\_second() const

{

    return second;

}

int Clock::get\_time\_in\_seconds() const

{

    return hms\_to\_sec();

}

void Clock::set\_hour(int h)

{

    if (h >= 0 && h < 24)

    {

        hour = h;

    }

}

void Clock::set\_minute(int m)

{

    if (m >= 0 && m < 60)

    {

        minute = m;

    }

}

void Clock::set\_second(int s)

{

    if (s >= 0 && s < 60)

    {

        second = s;

    }

}

void Clock::set\_time(int h, int m, int s)

{

    set\_hour(h);

    set\_minute(m);

    set\_second(s);

}

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

{

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

}

void Clock::sec\_to\_hms(int s)

{

    int hour = s / 3600; // Calculate hours

    hour = hour % 24;    // Ensure hours are within a 24-hour range

    this->hour = hour;

    this->minute = (s % 3600) / 60;

    this->second = s % 60;

}

void Clock::increment()

{

    second++;

    if (second == 60)

    {

        second = 0;

        minute++;

        if (minute == 60)

        {

            minute = 0;

            hour++;

            if (hour == 24)

            {

                hour = 0;

            }

        }

    }

}

void Clock::decrement()

{

    if (hms\_to\_sec() == 0)

    {

        set\_time(23, 59, 59);

    }

    else

    {

        sec\_to\_hms(hms\_to\_sec() - 1);

    }

}

void Clock::add\_seconds(int s)

{

    if (s < 0)

    {

        cout << "Error: Seconds must be a positive integer" << endl;

        return;

    }

    int total\_seconds = hms\_to\_sec() + s;

    total\_seconds = total\_seconds % (24 \* 60 \* 60);

    sec\_to\_hms(total\_seconds);

}

## h

// lab3Clock.h

// ENSF 694 Summer 2025 LAB 3 - EXERCISE C

// Created by: John Zhou

#ifndef lab3\_exe\_C\_Cplx

#define lab3\_exe\_C\_Cplx

/\* The following class definition represents a clock and contains three

 \* private data members called hour, minute, and second.

 \*/

class Clock

{

public:

    // Default constructor

    Clock();

    /\* PROMISES: Initializes the clock with default values (00:00:00). \*/

    Clock(int s);

    /\* PROMISES: Initializes the clock based on the total number of seconds (since 00:00:00).

       REQUIRES: s to be a non-negative integer representing the total seconds. \*/

    Clock(int hours, int minutes, int seconds);

    /\* PROMISES: Initializes the clock with the given hours, minutes, and seconds.

       REQUIRES: hours (0-23), minutes (0-59), and seconds (0-59). If out of range, resets to 00:00:00. \*/

    int get\_hour() const;

    /\* PROMISES: Returns the current hour of the clock (0-23). \*/

    int get\_minute() const;

    /\* PROMISES: Returns the current minute of the clock (0-59). \*/

    int get\_second() const;

    /\* PROMISES: Returns the current second of the clock (0-59). \*/

    int get\_time\_in\_seconds() const;

    /\* PROMISES: Returns the current time in seconds since 00:00:00. \*/

    void set\_hour(int h);

    /\* REQUIRES: h to be in the range of 0 to 23.

       PROMISES: Sets the hour of the clock to the specified value (0-23). \*/

    void set\_minute(int m);

    /\* REQUIRES: m to be in the range of 0 to 59.

       PROMISES: Sets the minute of the clock to the specified value (0-59). \*/

    void set\_second(int s);

    /\* REQUIRES: s to be in the range of 0 to 59.

       PROMISES: Sets the second of the clock to the specified value (0-59). \*/

    void increment();

    /\* PROMISES: Increments the clock by one second, updating the time accordingly. \*/

    void decrement();

    /\* PROMISES: Decrements the clock by one second, updating the time accordingly. \*/

    void add\_seconds(int s);

    /\* REQUIRES: s to be a non-negative integer. If negative, prints an error message.

       PROMISES: Adds the specified number of seconds to the clock. If the total exceeds 24 hours, it wraps around. \*/

    void set\_time(int h, int m, int s);

private:

    int hour;

    int minute;

    int second;

    // Converts the time to seconds

    int hms\_to\_sec() const;

    /\* PROMISES: Converts the current hour, minute, and second into total seconds. \*/

    // Converts seconds to hour, minute, second format

    void sec\_to\_hms(int s);

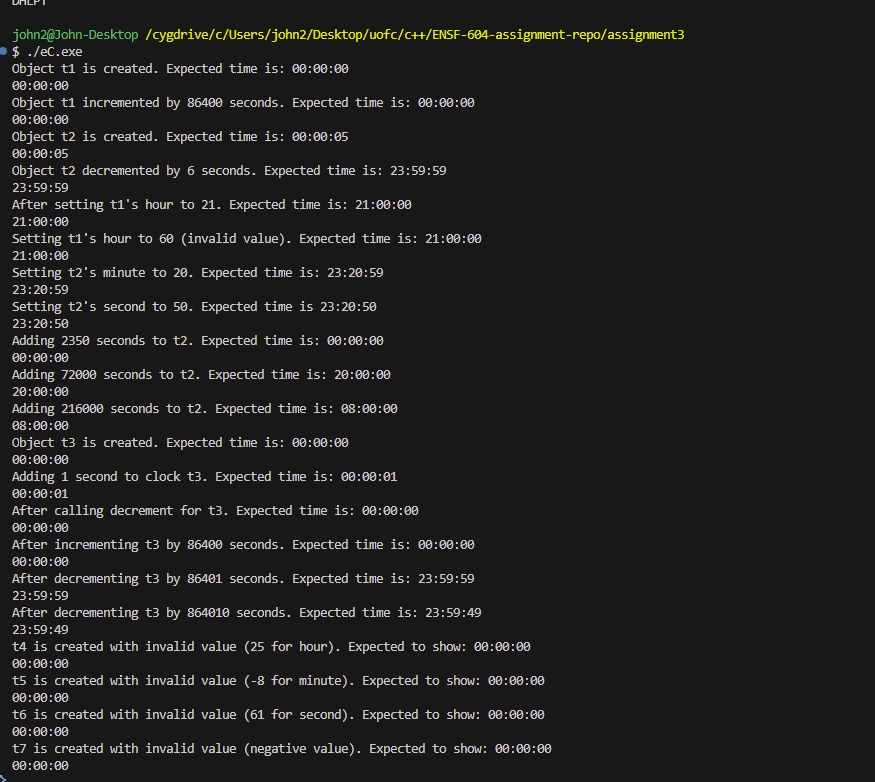
    /\* REQUIRES: s to be a non-negative integer representing total seconds.

       PROMISES: Converts the given total seconds to hour, minute, and second format. \*/

};

#endif

## Execution output



Exercise D

//

// CircularQueue.cpp

// Circular Queue

// ENSF 694 Summer 2025 LAB 3 - EXERCISE D

// Created by Mahmood Moussavi on 2024-04-09.

// implemented by: John Zhou

#include "CircularQueue.h"

#include <stdexcept>

CircularQueue::CircularQueue() : head(0), tail(0), count(0)

{

}

bool CircularQueue::isFull() const

{

return count == (SIZE-1);

}

bool CircularQueue::isEmpty() const

{

return count == 0;

}

int CircularQueue::enqueue(int v)

{

if (isFull())

{

throw std::overflow\_error("Queue is full. Cannot enqueue.");

}

arr[tail] = v;

int inserted\_index = tail;

tail = (tail + 1) % SIZE;

count++;

return inserted\_index;

}

int CircularQueue::dequeue()

{

if (isEmpty())

{

throw std::underflow\_error("Queue is empty. Cannot dequeue.");

}

int removed\_index = head;

head = (head + 1) % SIZE;

count--;

return removed\_index;

}

int CircularQueue::counter() const

{

return count;

}

const int \*CircularQueue::get\_arr() const

{

return arr;

}

void CircularQueue::displayQueue() const

{

if (isEmpty())

{

std::cout << "Queue is empty." << std::endl;

return;

}

int index = head;

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

{

std::cout << arr[index] << std::endl;

index = (index + 1) % SIZE;

}

}

# Exercise E

//

// DynamicStack.cpp

// Dynamic Stack

// ENSF 694 Summer 2025 LAB 3 - EXERCISE E

// Created by Mahmood Moussavi on 2024-04-09.

// implemented by: John Zhou

#include "DynamicStack.h"

#include <stdexcept>

DynamicStack::DynamicStack(int n)

{

entry = 0;

initial\_capacity = n;

current\_capacity = n;

array = new int[current\_capacity];

}

DynamicStack::DynamicStack(DynamicStack const &stack)

{

entry = stack.entry;

initial\_capacity = stack.initial\_capacity;

current\_capacity = stack.current\_capacity;

array = new int[current\_capacity];

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

{

array[i] = stack.array[i];

}

}

DynamicStack::~DynamicStack()

{

delete[] array;

}

int DynamicStack::top() const

{

if (entry == 0)

{

throw std::underflow\_error("stack empty. no top");

}

return array[entry - 1];

}

int DynamicStack::size() const

{

return entry;

}

bool DynamicStack::empty() const

{

return entry == 0;

}

int DynamicStack::capacity() const

{

return current\_capacity;

}

DynamicStack &DynamicStack::operator=(DynamicStack const &rhs)

{

if (this != &rhs)

{

delete[] array;

entry = rhs.entry;

initial\_capacity = rhs.initial\_capacity;

current\_capacity = rhs.current\_capacity;

array = new int[current\_capacity];

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

{

array[i] = rhs.array[i];

}

}

return \*this;

}

void DynamicStack::push(const int &obj)

{

if (entry == current\_capacity)

{

int new\_capacity = current\_capacity \* 2;

int \*new\_array = new int[new\_capacity];

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

{

new\_array[i] = array[i];

}

delete[] array;

array = new\_array;

current\_capacity = new\_capacity;

}

array[entry++] = obj;

}

int DynamicStack::pop()

{

if (entry == 0)

{

throw std::underflow\_error("stack empty. can't pop");

}

entry--;

return array[entry];

}

void DynamicStack::clear()

{

entry = 0;

if (current\_capacity != initial\_capacity)

{

current\_capacity = initial\_capacity;

}

delete[] array;

array = new int[initial\_capacity];

}

void DynamicStack::display()

{

if (entry == 0)

{

cout << "Stack is empty." << endl;

}

else

{

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

{

cout << array[i] << " ";

}

cout << endl;

}

}

# Exercise F

String\_Vector transpose(const String\_Vector &sv)

{

    String\_Vector vs;

    int rows = sv.size();

    int cols = sv[0].size();

    vs.resize(cols);

    for (int i = 0; i < cols; ++i)

    {

        for (int j = 0; j < rows; ++j)

        {

            vs[i].push\_back(sv[j][i]);

        }

    }

    return vs;

}

Program output  
