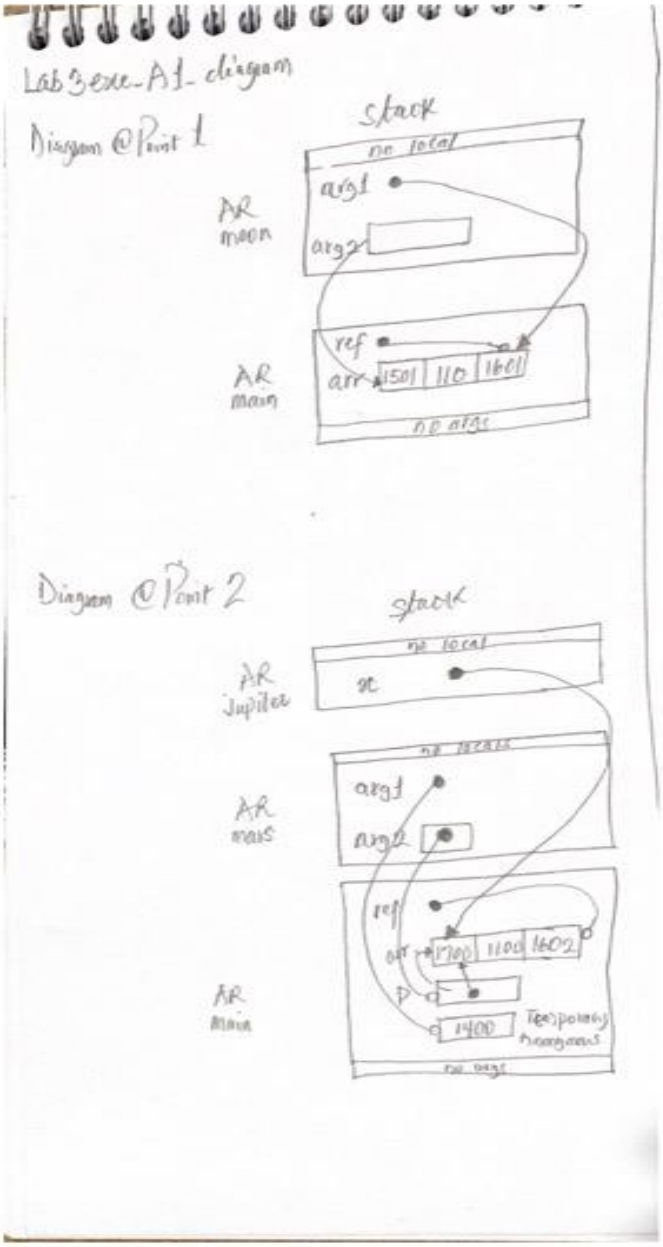
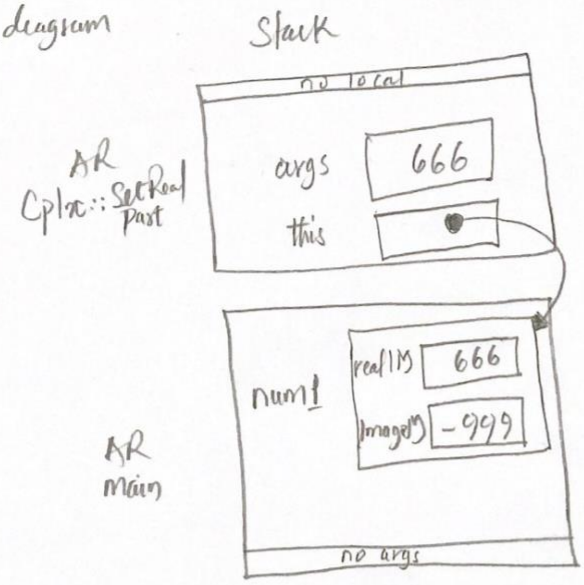
Course: ENSF 614 – Fall 2023
Lab 1:
Instructor: M. Moussavi
Student Name: Emmanuel Alafonye
Submission Date: October 13, 2023

```
/*
* File Name: lab3exe_A.cpp
* Assignment: ENSF 614 Lab 3, exercise A
* Created by Mahmood Moussavi
* Completed by: Emmanuel Alafonye
* Submission Date: October 13, 2023.
*/
```

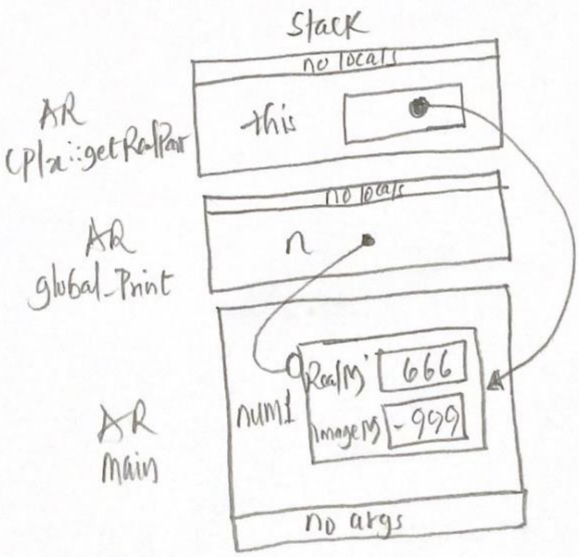


Lab3exe-B1_diagram

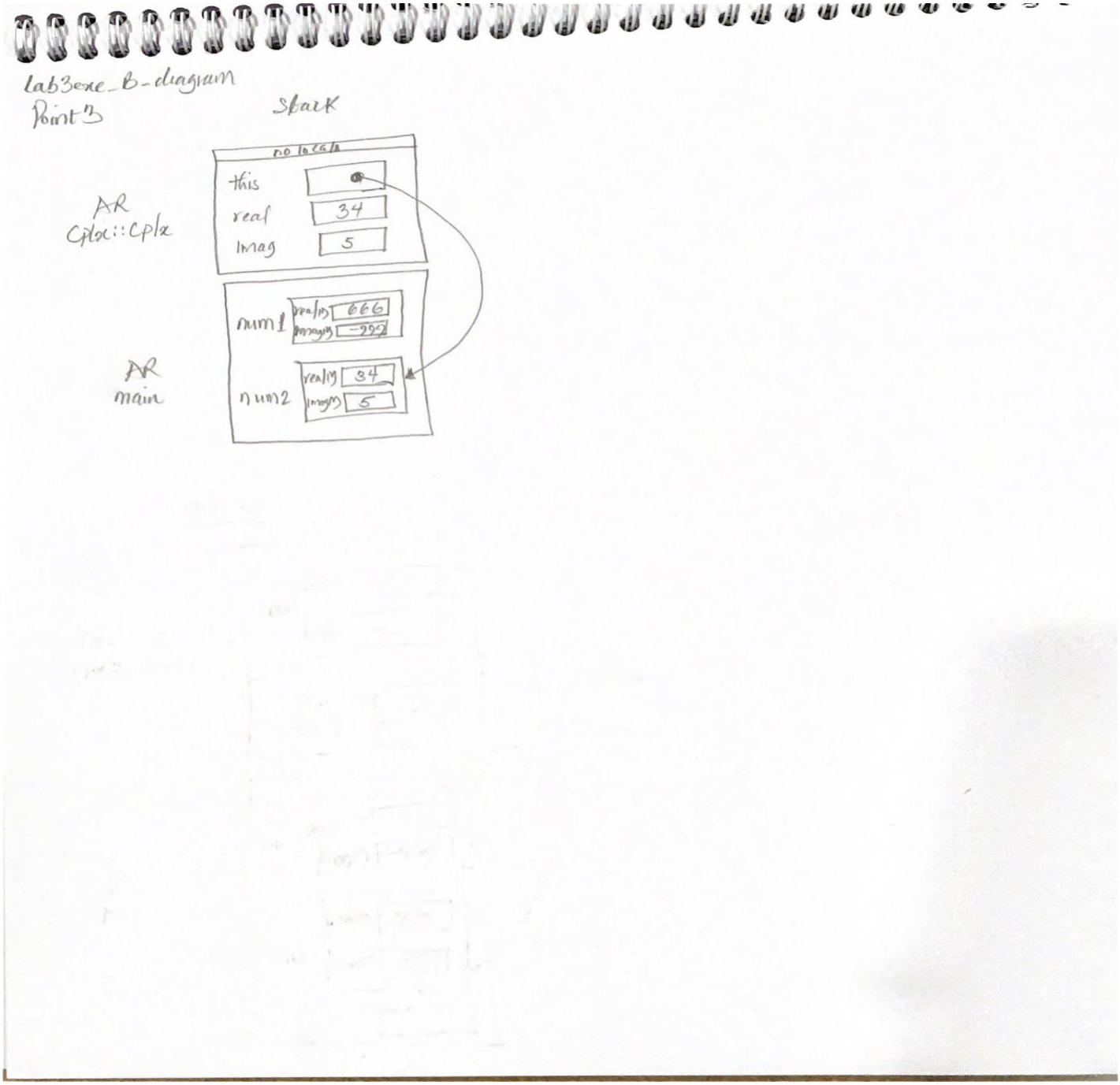
Point 1.



Point 2.



```
/*
* File Name: lab3exe_B.cpp
* Assignment: ENSF 614 Lab 3, exercise B
* Created by Mahmood Moussavi
* Completed by: Emmanuel Alafonye
* Submission Date: October 13, 2023.
*/
```



Lab3Clock.h

```
/*
* File Name: lab3exe_C.cpp
* Assignment: ENSF 614 Lab 3, exercise C
* Created by Mahmood Moussavi
* Completed by: Emmanuel Alafonye
* Submission Date: October 13, 2023.
*/

#ifndef LAB3CLOCK_H
#define LAB3CLOCK_H


// Preprocessor directive that prevents multiple inclusions of this header file.
// If LAB3CLOCK_H is not defined, the code between #ifndef and #endif will be included.

class Clock {
public: // Public member function to get components
   Clock(); // Default constructor
   Clock(int seconds); // Constructor with seconds argument
   Clock(int hours, int minutes, int seconds); // Constructor with hours, minutes, and seconds

   int get_hour() const;
   int get_minute() const;
   int get_second() const;

   void set_hour(int hours);
   void set_minute(int minutes);
   void set_second(int seconds);

   void increment();
   void decrement();
   void add_seconds(int seconds);

private: // Private data to store components
   int hour;
   int minute;
   int second;

   int hms_to_sec() const;
   void sec_to_hms(int total_seconds);
};

#endif
```

Lab3Clock.cpp

```
#include "lab3Clock.h"
```

```
// Default constructor
Clock::Clock() {
   hour = 0;
   minute = 0;
   second = 0;
}

// Constructor with seconds argument
Clock::Clock(int seconds) {
   sec_to_hms(seconds);
}

// Constructor with hours, minutes, and seconds
Clock::Clock(int hours, int minutes, int seconds) {
   if (hours >= 0 && hours <= 23 && minutes >= 0 && minutes <= 59 && seconds >= 0 && seconds <= 59) {
      hour = hours;
      minute = minutes;
      second = seconds;
   } else {
      hour = 0;
      minute = 0;
      second = 0;
   }
}

// Getter functions
int Clock::get_hour() const {
   return hour;
}

int Clock::get_minute() const {
   return minute;
}

int Clock::get_second() const {
   return second;
}

// Setter functions
void Clock::set_hour(int hours) {
   if (hours >= 0 && hours <= 23) {
      hour = hours;
   }
}

void Clock::set_minute(int minutes) {
   if (minutes >= 0 && minutes <= 59) {
      minute = minutes;
   }
```

```
}

void Clock::set_second(int seconds) {
    if (seconds >= 0 && seconds <= 59) {
        second = seconds;
    }
}

// Additional functionalities
void Clock::increment() {
    if (hour == 23 && minute == 59 && second == 59) {
        hour = 0;
        minute = 0;
        second = 0;
    } else {
        add_seconds(1);
    }
}

void Clock::decrement() {
    if (hour == 0 && minute == 0 && second == 0) {
        hour = 23;
        minute = 59;
        second = 59;
    } else {
        add_seconds(-1);
    }
}

void Clock::add_seconds(int seconds) {
    int total_seconds = hms_to_sec() + seconds;
    sec_to_hms(total_seconds);
}

// Helper functions
int Clock::hms_to_sec() const {
    return hour * 3600 + minute * 60 + second;
}

void Clock::sec_to_hms(int total_seconds) {
    if (total_seconds >= 0 && total_seconds <= 86399) {
        hour = total_seconds / 3600;
        minute = (total_seconds % 3600) / 60;
        second = total_seconds % 60;
    } else {
        hour = 0;
        minute = 0;
        second = 0;
    }
```

}

```
(base) Emmanuels-MacBook-Pro:lab3exe ataene$ ./clock_program
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
00:00:00
Object t2 decremented by 6 seconds. Expected time is: 23:59:59
23:59:54
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
23:20:54
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
00:00:00
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
```

```
/*
 * File Name: lab3exe_D.cpp
 * Assignment: ENSF 614 Lab 3, exercise D
 * Created by Mahmood Moussavi
 * Completed by: Emmanuel Alafonye
 * Submission Date: October 13, 2023.
 */

#include "MyArray.h"

MyArray::MyArray()
{

   sizeM = 0; // Creating the empty array.

   storageM = new EType[0]; // set the size
}

// Constructor for the object array.
```

```
MyArray::MyArray(const EType *builtin, int sizeA){
   sizeM = sizeA;
   storageM = new EType[sizeM];

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

      storageM[i] = builtin[i];
   }
}
// Copy constructor for the myArray class
MyArray::MyArray(const MyArray &source){
   sizeM = source.sizeM;
   storageM = new EType[sizeM];
   for (int i = 0; i < sizeM; i++)
   {
      storageM[i] = source.storageM[i];
   }
}

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

   if (this == &rhs){

      return *this; // Handle self-assignment
   }

   delete[] storageM;

   sizeM = rhs.sizeM;

   storageM = new EType[sizeM];

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

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

   return *this;
}

// Destructor
MyArray::~MyArray(){

   delete[] storageM;
}

int MyArray::size() const{
```

```
   return sizeM;
}
// Returns the element of specified index.
EType MyArray::at(int i) const{

   if (i >= 0 && i < sizeM)
   {
      return storageM[i];
   }
   else
   {
      return 0;
   }
}

void MyArray::set(int i, EType new_value)
{

   if (i >= 0 && i < sizeM)
   {
      storageM[i] = new_value;
   }
}

// Resizes the array to a new size
void MyArray::resize(int new_size){
   if (new_size < 0)
   {
      return;
   }

   EType *newStorage = new EType[new_size];

   int copySize = (new_size < sizeM) ? new_size : sizeM;

   for (int i = 0; i < copySize; i++)
   {
      newStorage[i] = storageM[i];
   }

   delete[] storageM;

   storageM = newStorage;

   sizeM = new_size;
}
```

```
(base) Emmanuels-MacBook-Pro:lab3exe_D ataene$ ./lab3exe_D
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)
```