Lab Report

Course: ENSF 337 - Programming Fundamentals for Software and Computer

Lab #: 7

Instructor: Dr. Maan Khedr

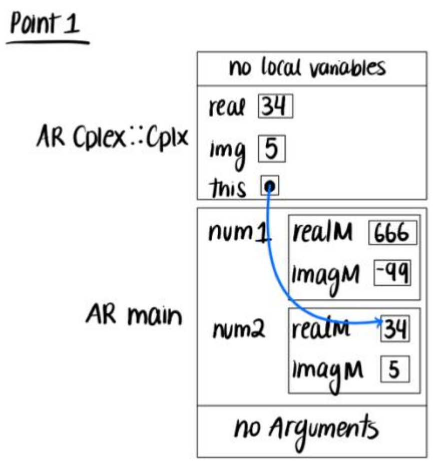
Student Name: Sadia Khandaker

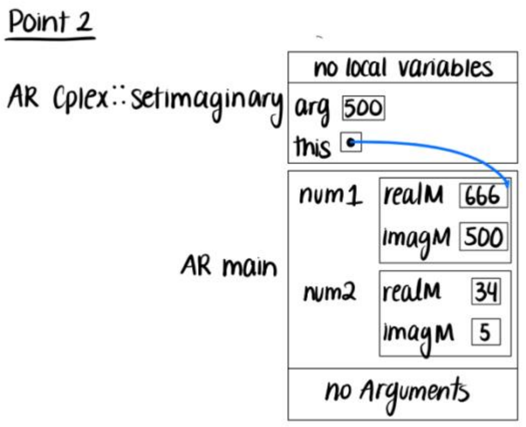
Lab Section: B04

Date Submitted: November 18, 2021

Exercise A: Memory Allocation of C++ Objects

AR Diagram at Point 1 and Point 2





Exercise C: Designing a C++ Class

Program:

lab7Clock.h

```
/* ENSF 337 - Lab 7 Exercise C
 * File name: lab7Clock.h
 * Completed by: Sadia Khandaker
 * Submission Date: Nov 18, 2021
 */

#ifndef LAB7EXEC_LAB7CLOCK_H
#define LAB7EXEC_LAB7CLOCK_H

class Clock {
private:
    int hour;
    int minute;
    int second;

    int hms_to_sec();
    void sec_to_hms(int n);

public:
    Clock();
    Clock(int sec);
    Clock(int hr,int min,int sec);

    void set_hour(int hr);
    void set_minute(int min);
    void set_second(int sec);
    int get_hour() const;
    int get_minute() const;
    int get_second() const;

    void increment();
    void decrement();

    void add_seconds(int sec);
};
#endif //LAB7EXEC_LAB7CLOCK_H
```

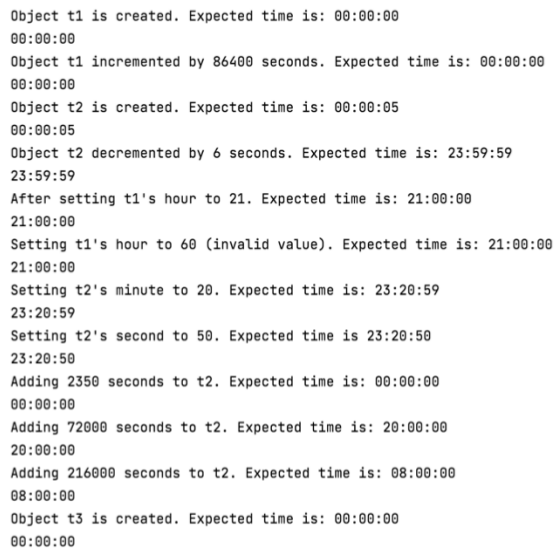
lab7Clock.cpp

```
/* ENSF 337 - Lab 7 Exercise C
 * File name: lab7Clock.cpp
 * Completed by: Sadia Khandaker
 * Submission Date: Nov 18, 2021
 */

#include <iostream>
#include <iomanip>
#include "lab7Clock.h"
using std::cout;
using std::endl;
using std::setw;
using std::setfill;

Clock::Clock() {
    hour = 0;
    minute = 0;
    second = 0;
}

Clock::Clock(int sec) {
    hour = 0;
    minute = 0;
    second = 0;
    sec_to_hms(sec);
}

Clock::Clock(int hr, int min, int sec) {
    hour = 0;
    minute = 0;
    second = 0;
    while(hr>=0 && hr<=23 && min>=0 && min <=59&& sec>=0 &&
sec<=59) {
        set_hour(hr);
        set_minute(min);
        set_second(sec);
        break;
    }
}

void Clock::set_hour(int hr) {
    while(hr>=0 && hr<=23) {
        hour = hr;
        break;
    }
}

void Clock::set_minute(int min) {
    while(min>=0 && min<=59) {
        minute = min;
        break;
    }
}

void Clock::set_second(int sec) {
    while(sec>=0 && sec<=59) {
        second = sec;
        break;
    }
}

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

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

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

void Clock::increment() {
    second++;
    if (second==60) {
        second = 0;
        minute++;
        if (minute==60) {
            minute = 0;
```

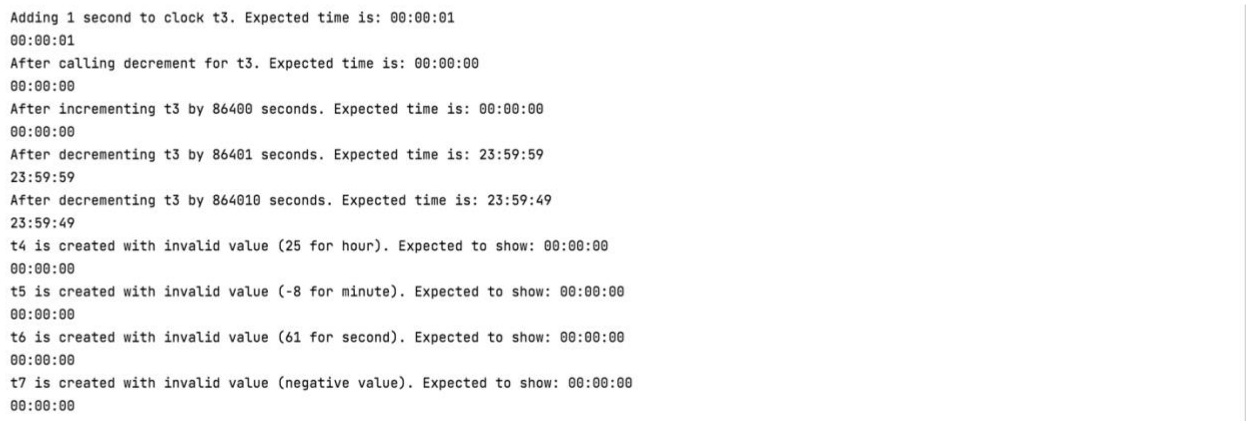
```
                hour++;
                if (hour==24) {
                    hour = 0;
                }
            }
        }
    }

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


    void Clock::add_seconds(int sec) {
        for(int i=1;i<=sec;i++) {
            increment();
        }
    }

    int Clock::hms_to_sec() {
        return(second+minute*60+hour*60*60);
    }

    void Clock::sec_to_hms(int n) {
        int hr=0;
        int min=0;
        int sec=0;
        if(n>0){
            hr=n/(3600);
            n=n-hr*(3600);

            if(n>=60){
                min=n/60;
                n=sec%60;
            }else
                sec=n;
            if(hr==24)
                set_hour(0);
            else
                set_hour(hr);
            set_minute(min);
            set_second(sec);
        }
    }


    void print(const Clock & t);
    // REQUIRES: t refers to an object of class Clock with valid
    values.
    // PROMISES: displays the time values in an object of class Clock:
    //            in the following hh:mm:ss.

    void print(const Clock & t) {
        cout << setw(2) << setfill('0') << t.get_hour() << ":" <<
    setw(2) << t.get_minute() << ":" << setw(2) << t.get_second() <<
    endl;

    }

    int main(void) {
        Clock t1;
        cout << "Object t1 is created. Expected time is: 00:00:00\n";
        print(t1);

        for (int i = 0; i < 86400; i++)
            t1.increment();

        cout << "Object t1 incremented by 86400 seconds. Expected time
    is: 00:00:00\n";
        print(t1);

        Clock t2(86405);
        cout << "Object t2 is created. Expected time is: 00:00:05\n";
        print(t2);

        for (int i = 0; i < 6; i++)
            t2.decrement();

        cout << "Object t2 decremented by 6 seconds. Expected time is:
    23:59:59\n";
        print(t2);

        t1.set_hour(21);
        cout << "After setting t1's hour to 21. Expected time is:
    21:00:00\n";
        print(t1);
        t1.set_hour(60);
        cout << "Setting t1's hour to 60 (invalid value). Expected time
    is: 21:00:00\n";
        print(t1);

        t2.set_minute(20);
        cout << "Setting t2's minute to 20. Expected time is:
    23:20:59\n";
        print(t2);

        t2.set_second(50);
        cout << "Setting t2's second to 50. Expected time is
    23:20:50\n";
        print(t2);

        t2.add_seconds(2350);
        cout << "Adding 2350 seconds to t2. Expected time is:
    00:00:00\n";
        print(t2);

        t2.add_seconds(72000);
        cout << "Adding 72000 seconds to t2. Expected time is:
    20:00:00\n";
        print(t2);

        t2.add_seconds(216000);
        cout << "Adding 216000 seconds to t2. Expected time is:
    08:00:00\n";
        print(t2);

        Clock t3(0, 0, 0);
        cout << "Object t3 is created. Expected time is: 00:00:00\n";
        print(t3);

        t3.increment();
        cout << "Adding 1 second to clock t3. Expected time is:
    00:00:01\n";
        print(t3);

        t3.decrement();
        cout << "After calling decrement for t3. Expected time is:
    00:00:00\n";
        print(t3);

        for (int i = 0; i < 86400; i++)
            t3.increment();

        cout << "After incrementing t3 by 86400 seconds. Expected time
```

```
is: 00:00:00\n";
    print(t3);

    for (int i = 0; i < 86401; i++)
        t3.decrement();

    cout << "After decrementing t3 by 86401 seconds. Expected time
is: 23:59:59\n";
    print(t3);

    for (int i = 0; i < 864010; i++)
        t3.decrement();

    cout << "After decrementing t3 by 864010 seconds. Expected time
is: 23:59:49\n";
    print(t3);

    // Object t4 is created with invalid value of hour (i.e. 25),
and valid values
    // for other members, minute and second respectively.
    Clock t4(25, 0, 0);
    cout << "t4 is created with invalid value (25 for hour).
Expected to show: 00:00:00\n";
    print(t4);

    // Object t5 is created with invalid value of minute (i.e. -8),
and valid values
    // for other member, hour and second.
    Clock t5(23, -8, 59);
    cout << "t5 is created with invalid value (-8 for minute).
Expected to show: 00:00:00\n";
    print(t5);

    // Object t6 is created with invalid value of second (i.e. 61),
and valid values
    // for other members.
    Clock t6(23, 59, 61);
    cout << "t6 is created with invalid value (61 for second).
Expected to show: 00:00:00\n";
    print(t6);

    Clock t7(-10);
    cout << "t7 is created with invalid value (negative value).
Expected to show: 00:00:00\n";
    print(t7);

    return 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
```

Exercise D: A Simple Class Vector and Copying Object

Program:

```
/* ENSF 337 - Lab 7 Exercise D
 * File Name: simpleVector.cpp
 * Completed By: Sadia Khandaker
 * Submission Date: November 18, 2021
 */

#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];
}
```

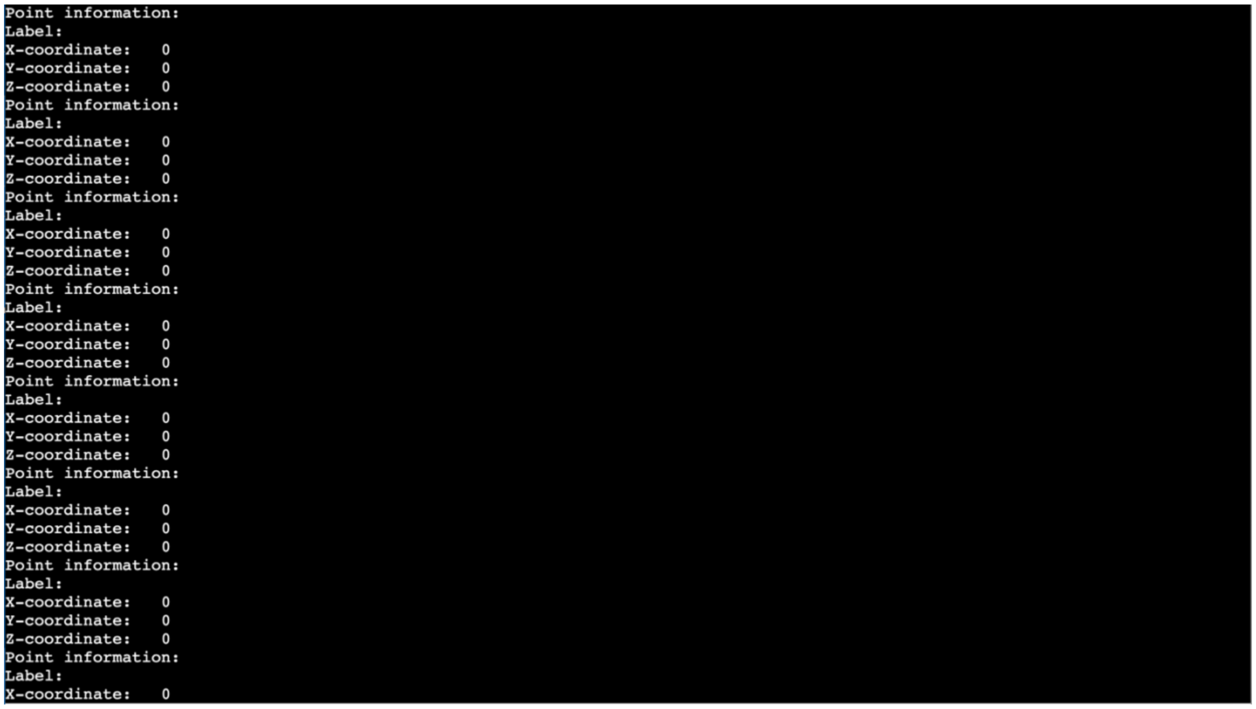
```
TYPE& SimpleVector::at(int i) {
    assert(i >= 0 && i < sizeM);
    return storageM[i];
}

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) {
    while (sizeM < capacityM) {
        if (sizeM == capacityM != 0) {
            capacityM *= 2;
        } else if (sizeM == capacityM == 0) {
            capacityM = 2;
        }
    }
    TYPE *n_arr = new TYPE[capacityM];
    for (int i = 0; i < sizeM; i++) {
        n_arr[i] = storageM[i];
    }
    n_arr[sizeM] = val;
    delete[] storageM;
    storageM = n_arr;
    sizeM++;
}

SimpleVector::SimpleVector(const SimpleVector& source) {
    sizeM = source.sizeM;
    storageM = new TYPE [sizeM];
    assert (storageM != 0);
    for (int i = 0; i<sizeM; i++){
        storageM[i] = source.storageM[i];
    }
}

SimpleVector& SimpleVector::operator= (const SimpleVector& rhs ){
    sizeM = rhs.sizeM;
    capacityM = rhs.capacityM;
    storageM = new TYPE[capacityM];
    for(int i = 0; i < sizeM; i++) {
        storageM[i] = rhs.storageM[i];
    }
    return *this;
}
```

Output:

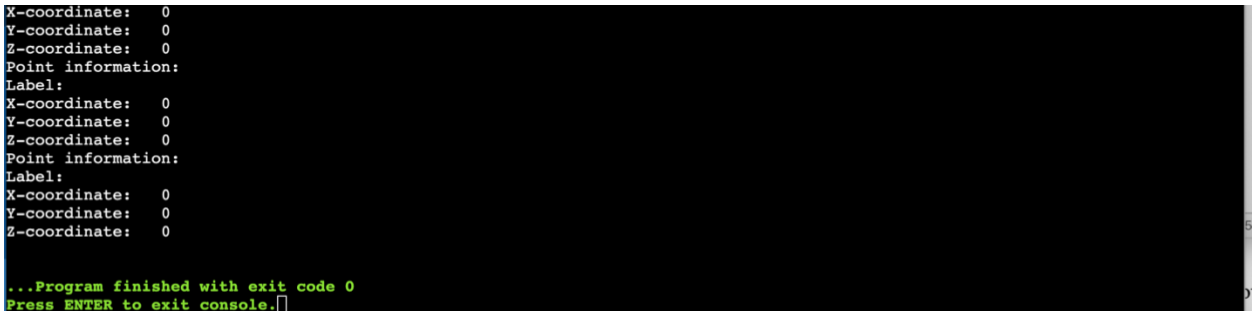
```
Object v1 is expected to display: 45 69 12
45 69 12
Object v2 is expected to diaplay: 3000 6000 7000 8000
3000 6000 7000 8000

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
```

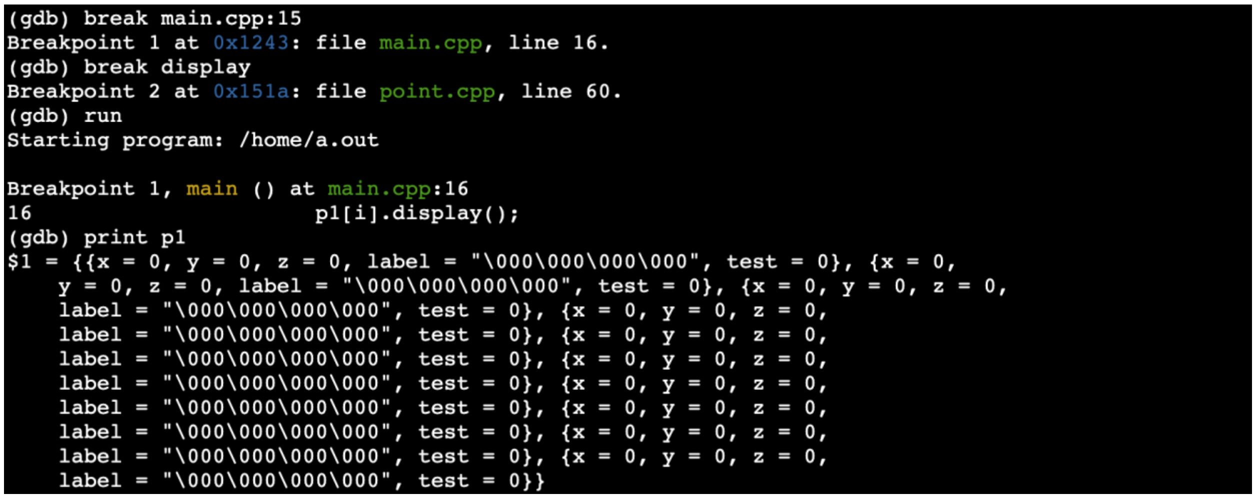
Exercise E: Code Debugging and Tracing

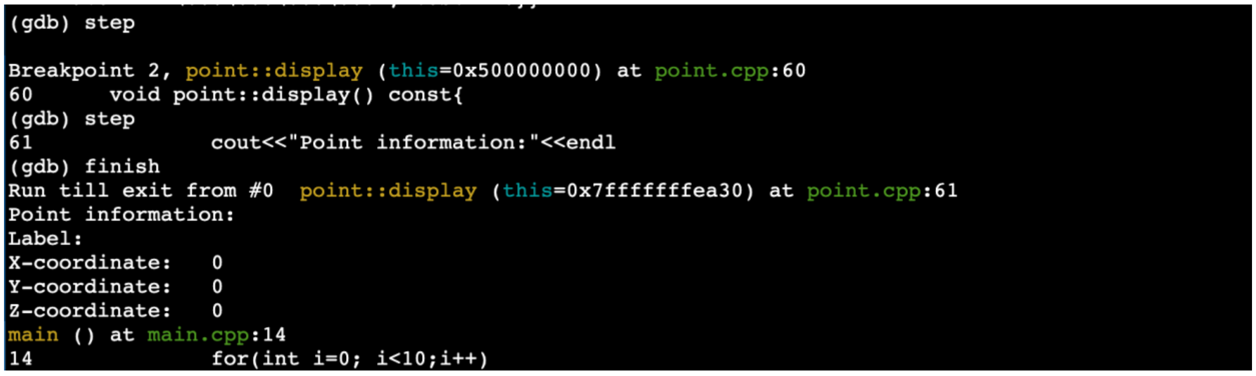
Screenshots of Replicating the Steps Provided:

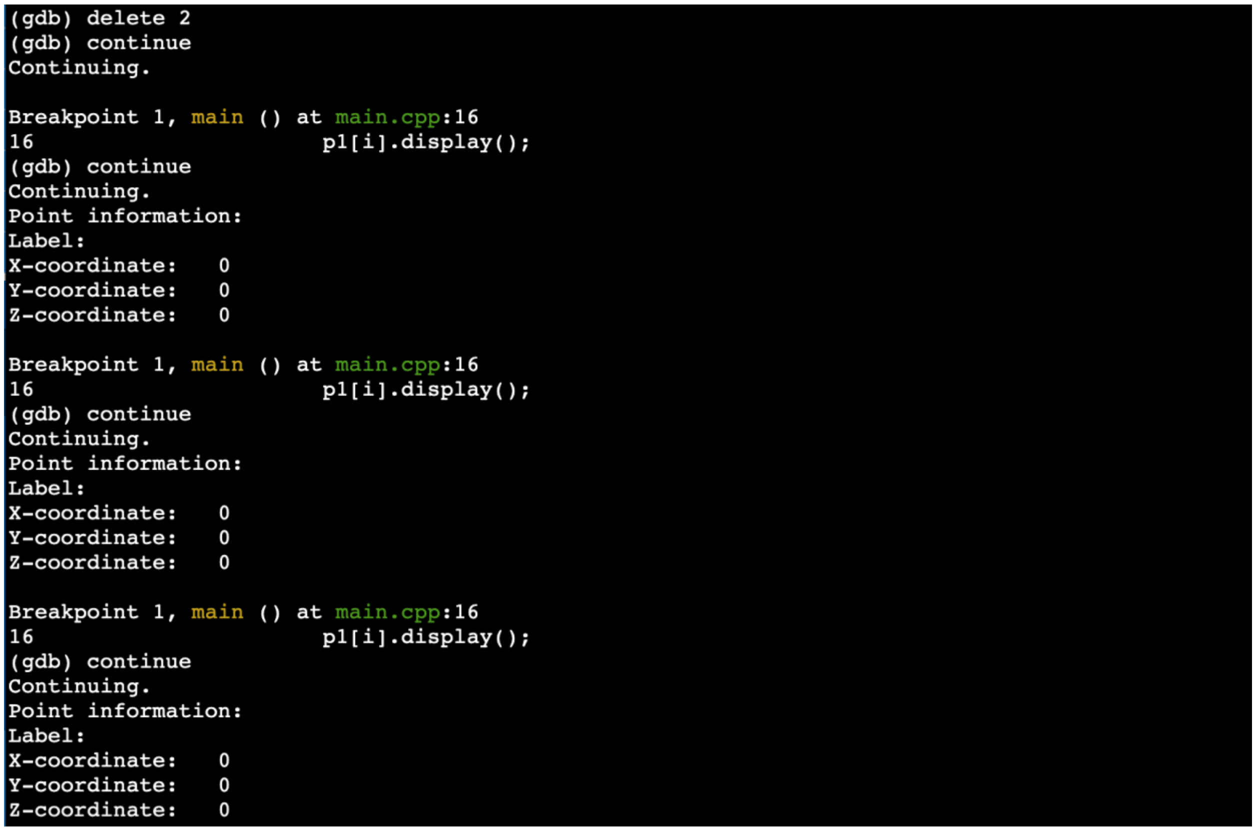


```
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0
Point information:
Label:
X-coordinate:   0
```

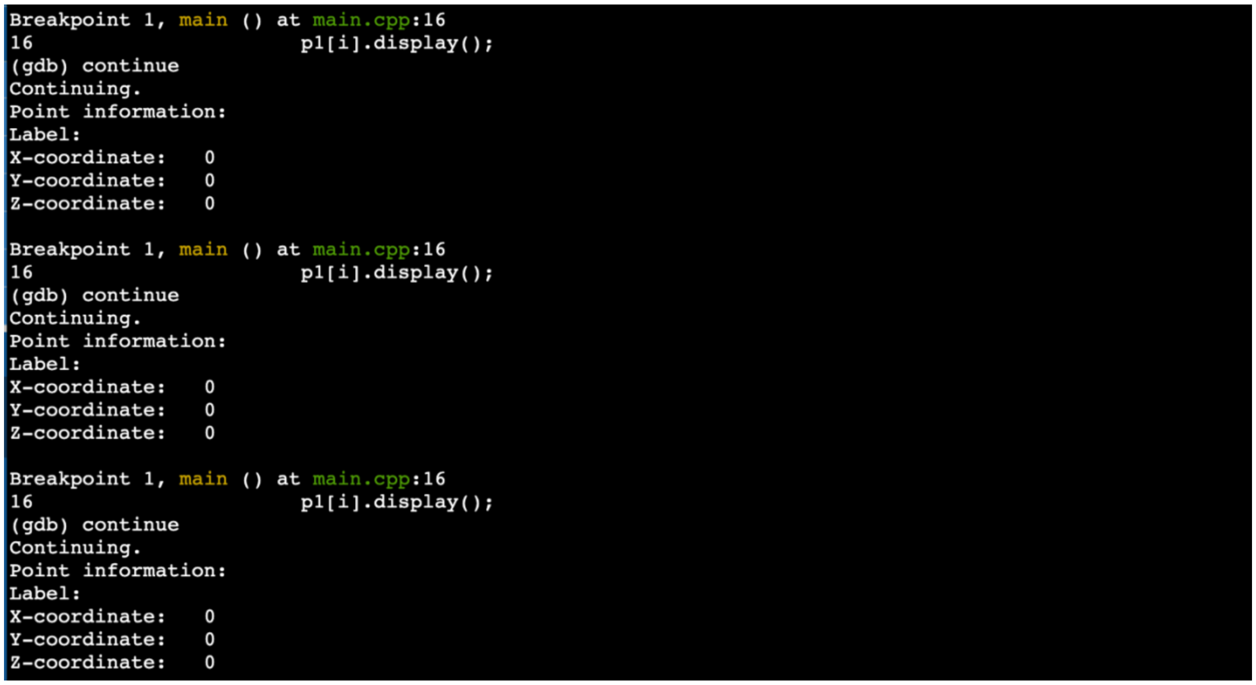

```
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0


...Program finished with exit code 0
Press ENTER to exit console.
```


```
(gdb) break main.cpp:15
Breakpoint 1 at 0x1243: file main.cpp, line 16.
(gdb) break display
Breakpoint 2 at 0x151a: file point.cpp, line 60.
(gdb) run
Starting program: /home/a.out

Breakpoint 1, main () at main.cpp:16
16                      p1[i].display();
(gdb) print p1
$1 = {{x = 0, y = 0, z = 0, label = "\000\000\000\000", test = 0}, {x = 0,
    y = 0, z = 0, label = "\000\000\000\000", test = 0}, {x = 0, y = 0, z = 0,
    label = "\000\000\000\000", test = 0}, {x = 0, y = 0, z = 0,
    label = "\000\000\000\000", test = 0}, {x = 0, y = 0, z = 0,
    label = "\000\000\000\000", test = 0}, {x = 0, y = 0, z = 0,
    label = "\000\000\000\000", test = 0}, {x = 0, y = 0, z = 0,
    label = "\000\000\000\000", test = 0}, {x = 0, y = 0, z = 0,
    label = "\000\000\000\000", test = 0}, {x = 0, y = 0, z = 0,
    label = "\000\000\000\000", test = 0}, {x = 0, y = 0, z = 0,
    label = "\000\000\000\000", test = 0}}
```

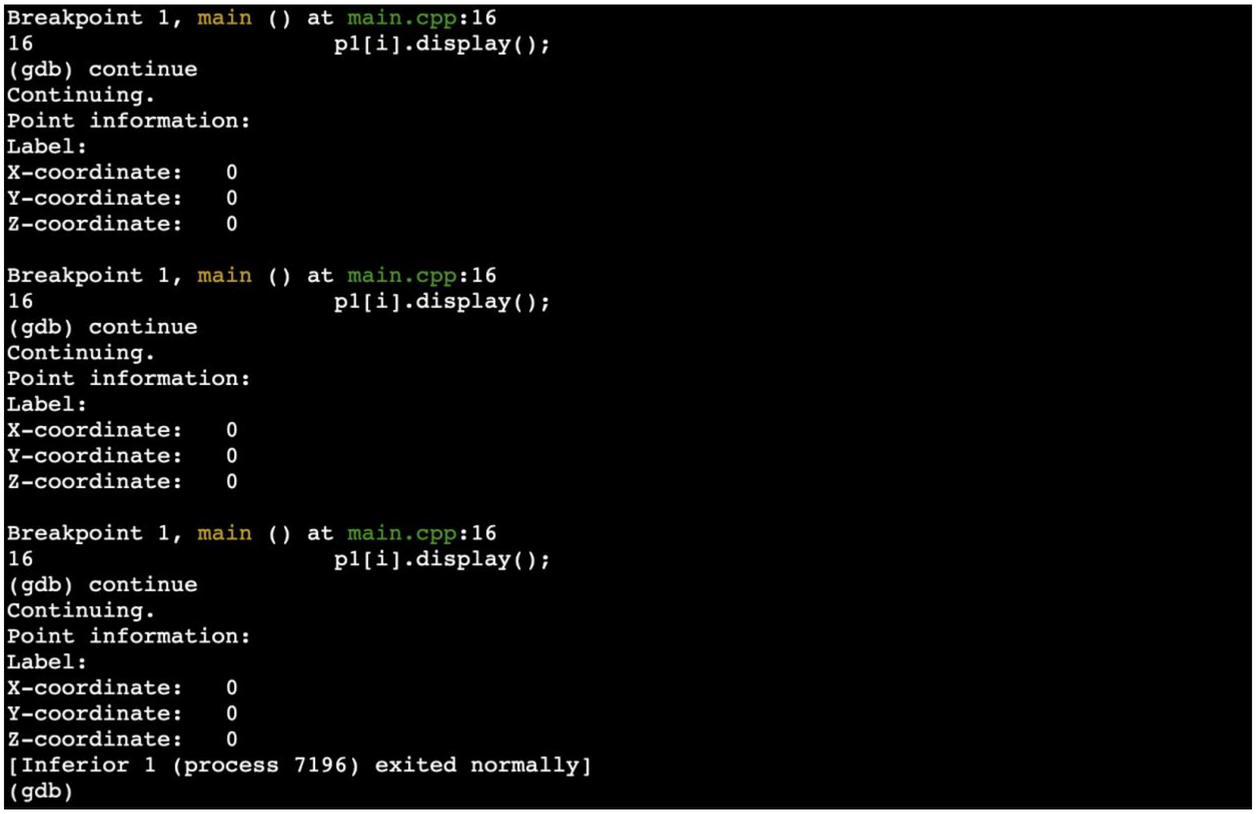

```
(gdb) step

Breakpoint 2, point::display (this=0x500000000) at point.cpp:60
60      void point::display() const{
(gdb) step
61              cout<<"Point information:"<<endl
(gdb) finish
Run till exit from #0  point::display (this=0x7fffffffea30) at point.cpp:61
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0
main () at main.cpp:14
14              for(int i=0; i<10;i++)
```



```
(gdb) delete 2
(gdb) continue
Continuing.

Breakpoint 1, main () at main.cpp:16
16                      p1[i].display();
(gdb) continue
Continuing.
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0

Breakpoint 1, main () at main.cpp:16
16                      p1[i].display();
(gdb) continue
Continuing.
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0

Breakpoint 1, main () at main.cpp:16
16                      p1[i].display();
(gdb) continue
Continuing.
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0
```



```
Breakpoint 1, main () at main.cpp:16
16                      p1[i].display();
(gdb) continue
Continuing.
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0

Breakpoint 1, main () at main.cpp:16
16                      p1[i].display();
(gdb) continue
Continuing.
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0

Breakpoint 1, main () at main.cpp:16
16                      p1[i].display();
(gdb) continue
Continuing.
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0
```



```
Breakpoint 1, main () at main.cpp:16
16                      p1[i].display();
(gdb) continue
Continuing.
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0

Breakpoint 1, main () at main.cpp:16
16                      p1[i].display();
(gdb) continue
Continuing.
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0

Breakpoint 1, main () at main.cpp:16
16                      p1[i].display();
(gdb) continue
Continuing.
Point information:
Label:
X-coordinate:   0
Y-coordinate:   0
Z-coordinate:   0
[Inferior 1 (process 7196) exited normally]
(gdb)
```

How many times do you need to use the continue command?

The continue command was used 10 times until the program ended.

Why is the default constructor getting called these many times?

The default class constructor is called each time whenever a new object is created. Because of the code point p1[10]; there’s an array of 10 objects, hence the default constructor will be called 10 times.