COEN 79: Object-Oriented Programming
Homework #2

Tamir Enkhjargal

January 27, 2020

Question #1

What are the effects of using *inline member functions*? Write a function that uses at least two inline member functions.

Inline member functions are when you complete the implementation inside your header file, rather than in the implementation file. What this does is when you run a program, the Program Counter will run through your program just as is line-by-line. This leads to faster execution time, but uses up more memory if you call the same function multiple times. You also should use this method when the implementation is short.

```
In code.h file:
#ifndef COEN79_HW2
#define COEN79_HW2
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

class myProgram {
    public:
        void myTime() { cout << time() << endl; }
        void onePlusOne() { cout << 2 }
}
#endif COEN79_HW2

In main.cpp file:
#include "code.h"
using namespace std;
main() {
    cout << "What is the system time?" << endl;
    myTime();
    cout << "What is one plus one?"
    onePlusOne();
}
```

Question #2

What are the differences between *references* and *pointers*?

References are just another alias to the actual object itself, while pointers are new objects created that store the memory address of a variable. References do not create any new objects, while pointers are usually 4 or 8 bytes (depending on 32-bit or 64-bit processor.

Question #3

What is the *value semantics* of a class?

Value semantics is how values can be copied from one object to the other. These are done through assignment operators and copy constructors.

Write a class for which you can *rely on the compiler* to provide you with valid value semantics.

```
class Angle {
public:
    // CONSTRUCTOR
    angle(int initial_d = 0, int initial_m = 0);

private:
    int degrees;
    int magnitude;
}
```

Question #4

Where should we include the *invariants* of a class? Why?

We include the invariants of a class at the top of the implementation file of that class. This is because in the header file, there is information hiding; people looking into your header file do not care about how your functions are implemented. However, people looking into your implementation file and wishing to change code or include something in, they must follow the rules of the invariants.

Question #5

The header of the `point` class is as follows:

```
class point {
public:
    // CONSTRUCTOR
    point (double initial_x = 0.0, double initial_y = 0.0);

    // MODIFICATION MEMBER FUNCTION
    void set_x (double& value);
    void set_y (double& value);

    // CONST MEMBER FUNCTIONS
    point operator+ (double& in) const;

private:
    double x; // x coordinate of this point
    double y; // y coordinate of this point
}
```

Which line results in error? Explain why.

```
1. main() {
2.     point myPoint1, myPoint2, myPoint3;
3.     double shift = 8.5;
4.     myPoint1 = shift + myPoint2;
5.     myPoint 3 = myPoint1.operator+(shift);
6.     myPoint1 = myPoint 1 + shift;
7. }
```

The error is on line 4: `myPoint1 = shift + myPoint2;`.

What is the solution?

The solution is to just swap the two around to be `myPoint1 = myPoint2 + shift;` or create another operator overloaded function that takes in a `point& rhs` object.

Question #6

Why the following code does not compile? What is the solution? Note: The notation '<>' is used for *template* classes. You will learn template classes in future classes.

```
#include < iostream >
#include < list >

namespace coen79 {
    template < typename T >
    class list {
        private:
            int array[20];
    };
}

using namespace std;
using namespace coen79;

int main(int argc, const char * argv[]) {
    using namespace std;
    list < int > v1;
    list < int > v2;
    return 0;
}
```

This code does not compile because you are using both statements:

```
using namespace std;
using namespace coen79;
```

The compiler does not know which `list` class you are referring to. To make it not ambiguous you need to use `coen79::list < int > v1;` to make it specific.

Question #7

What is the output of this code? Discuss your answer.

```
#include < iostream >
using namespace std;

class CMyClass {
    public:
        static int m_i;
};

int CMyClass::m_i = 0;

CMyClass myObject1;
CMyClass myObject2;
CMyClass myObject3;

int main() {
    CMyClass::m_i = 2;
    myObject1.m_i = 1;

    cout << myObject1.m_i << endl;
    cout << myObject2.m_i << endl;

    myObject2.m_i = 3;
    myObject3.m_i = 4;

    cout << myObject1.m_i << endl;
    cout << myObject2.m_i << endl;
}
```

The output will be:

```
1
1
4
4
```

Question #8

I wrote the following code to print 11 “Ouch!”. Why is it not working as expected?

```
#include < iostream >
using namespace std;

int main(int argc, const char * argv[]) {
    std::size_t i;
    for(i = 10; i >= 0; --i)
        cout << "Ouch!" << endl;
    return 0;
}
```

The code runs infinitely through the loop. When i hits 0, and is decremented, it wraps around because it must always refer to an unsigned integer type, so it hits your computer's `INT_MAX` value.

Question #9

In the following code, indicate if the selected lines are legal or illegal:

```
#include <iostream>

class small {
public:
    small() {size = 0;};
    void k() const;
    void h(int i);
    friend void f(small z);

private:
    int size;
};

void small::k() const {
    small x, y;
    x = y;              // ILLEGAL
    x.size = y.size;    // ILLEGAL
    x.size = 3;         // ILLEGAL
};

void small::h(int i) {

};

void f(small z) {
    small x, y;
    x = y;              // LEGAL
    x.size = y.size;    // LEGAL
    x.size = 3;         // LEGAL
    x.h(42);            // LEGAL
};

int main() {
    small x, y;
    x = y;              // LEGAL
    x.size = y.size;    // ILLEGAL
    x.size = 3;         // ILLEGAL
    x.h(42);            // LEGAL

    return 0;
}
```

Question #10

Modify the following code to generate the given output. Do not modify the `main` function.

```
#include < iostream >
using namespace std;

class box {
public:
    // GLOBAL VARIABLE
    static int count;

    // CONSTRUCTOR
    box(double l = 2.0, double b = 2.0, double h = 2.0) {
        length = l;
        breadth = b;
        height = h;
        cout << "Number of box objects created so far: " << ++count << endl;
    }

    double volume() {
        return length * breadth * height;
    }

private:
    double length;
    double breadth;
    double height;
};

box::count = 0;

int main(void) {
    box Box1(3.3, 1.2, 1.5);

    box Box2(8.5, 6.0, 2.0);

    return 0;
}
```

Output:
Number of box objects created so far: 1
Number of box objects created so far: 2