COMP 3512
Object Oriented Programming in C++

Final Exam SOLUTION
Thursday April 19 2018

Full Name	
Signature	
Student ID	
Set	

Instructions:

1. This is a closed-book exam. No cheat sheets. You may not use any notes, texts, manuals, etc., during the exam.
2. No phones. Turn off your phone and put it away. If your phone rings you will earn zero on this exam.
3. No electronic devices. No calculators. No computers. No pocket organizers or translators. No watches. No music players. No headphones.
4. No talking. Speaking during the exam is not allowed. If you speak, you will earn zero.
5. Keep your eyes on your own paper. Looking at another exam, or purposely exposing your exam to the view of other candidates will be considered cheating and you will earn zero.
6. This midterm is 12 pages and 90 minutes long.
7. You must write your answers on this exam. You may write on the back.
8. No candidate will be admitted after 30 minutes.
9. No candidate will be permitted to leave during the first 30 minutes.
10. NO QUESTIONS. Candidates are NOT permitted to ask questions during the exam except in the case of supposed typos.
11. Good luck. You've got this.

Final grade: PART A _____ / 18 + PART B _____ / 47 = _____ / 65

Part A: Definitions (Write your answers in the space provided)

1. (6) What are the names and headers for the six standard member functions every class should have in C++? For the headers, assume we are working with a class called C.

#	Name	Header
1	**Default constructor**	**C()**
2	**Destructor**	**~C()**
3	**Copy Constructor**	**C(const C&)**
4	**Copy assignment operator**	**C& operator=(const C&)**
5	**Move constructor**	**C(C&&)**
6	**Move assignment operator**	**C& operator=(C&&)**

2. (2) What are "move semantics"?

Using the values stored in a temporary or rvalue to initialize another object or assign an object's value. The source object loses the content which is taken over or "moved to" the destination object. This only happens when the source of the value is an unnamed object, or when we treat it as an rvalue using std::move.

3. (2) What is an rvalue reference? How does an rvalue reference differ from a reference?

A reference to a temporary. An rvalue reference uses the && operator. A reference uses &, and is an alias to an existing object; it must be assigned a value when declared.

4. (3) What is the difference between an enumeration and a scoped enumeration? Provide an example.

Enumeration: enum
Scoped enumeration: enum class
The latter cannot be referenced without the scope operator ::
Values can be converted to integral types with unscoped enumerations, not scoped.

5. (3) What is a lambda? What is the general form of a lambda? What is an advantage of using a lambda instead of a function?

A lambda is an unnamed functor (a form of nested function)
[capture clause] (parameters) specifiers -> return type
{ body }.
We can define it where we want it, it may be inlined, it's easy and quick to write and understand.

6. (2) What does std::bind from the <functional> header file do? Why does it exist in the C++ library?

Returns a function object that is bound to a specific argument. Shorter code, polymorphism.

Part B: Code (Write your answers in the space provided)

7. Suppose we are going to design a class called Point3D. A Point3D must contain three integer values representing a geometric point's position on the x, y, and z axes respectively in 3D Cartesian space.

   Let's consider two different ways of implementing Point3D, and the consequences of these choices:

Implementation 1:	Implementation 2:
Store the three integer values in three variables of type int called x, y, and z	Store the three integer values in three variables of type int * called x, y, and z.

1. (2) What does the constructor look like for each implementation?

```
Point3D::Point3D(int x, int y, int z) : x{x}, y{y}, z{z} { }
Point3D(int xx, int yy, int zz) : x{new int(xx)}, y{new int(yy)}, z{new int(zz)}
{ //x = new int(xx); y = new int(yy); z = new int(zz); }
```

2. (2) What does the destructor look like for each implementation?

```
~Point3D() { }
~Point3D() { delete x, delete y, delete z; }
```

3. (2) What does the move constructor look like for each implementation?

```
Point3D(Point3D&& other) : x{other.x}, y{other.y}, z{other.z}
{ other.x = other.y = other.z = 0;}
Point3D(Point3D&& other)
{       delete x; delete y; delete z;
        x = other.x; y = other.y; z = other.z;
        other.x = other.y = other.z = nullptr;
}
```

4. (2) What does the move assignment operator look like for each implementation?

```
Point3D& operator=(Point3D&& other)
{       x = other.x; y = other.y; z = other.z;
        other.x = other.y = other.z = 0;
        return *this;}
Point3D& operator=(Point3D&& other)
{       delete x, delete y, delete z;
        x = other.x; y = other.y; z = other.z;
        other.x = other.y = other.z = nullptr;
        return *this;
}
```

8. (3) Write an insertion operator for the Point3D class that uses implementation 2. The insertion operator is a friend function to the Point3D class, and it has the following signature:

```
ostream& operator<<(ostream& os, const Point3D& point) {

     os << *(point.x) << " " << *(point.y) << " " << *(point.z)
        << endl;
     return os;

}
```

9. (3) Write an extraction operator for the Point3D class that uses implementation 2. The extraction operator is a friend function to the Point3D class, and it has the following signature:

```
istream& operator>>(istream& is, Point3D& point) {

     int x, y, z;
     if ( is >> x >> y >> z )
     {
          *(point.x) = x;
          *(point.y) = y;
          *(point.z) = z;
     }
     else {
          is.setstate(ios_base::failbit);
     }
     return is;
}
```

10. (6) Write a template that is parameterized over two typenames, a typename called Sequence and a typename called UnaryFunction. For this purpose, a Sequence is a sequence container from the STL and a UnaryFunction is a function that accepts a parameter p and returns something of the same type.

    The template is for a single function called my_map. The my_map function accepts two parameters, a Sequence called s and a UnaryFunction called uf. The return type of my_map is Sequence. The my_map function create a new Sequence, apply the unary function uf to each element p in the Sequence passed as a parameter, push the result to the back of the new Sequence, and then return the new Sequence.

    Hint: we can push an element e to the back of an STL sequence container using the push_back(e) function.

```
template<class Sequence, class UnaryFunction>
Sequence my_map(Sequence s, UnaryFunction uf)
{
        Sequence newSequence;
        for (typename Sequence::iterator it = s.begin(); it != s.end(); ++it) {
                newSequence.push_back(uf(*it));
        }
        return newSequence;
}
```

11. (3) This code will not compile. Fix the errors (Hint: there are 3).

```
class A
{
public:
    virtual ~A() { }
    virtual void act() = 0;
};

class B
{
public:
    ~B();
};

int main()
{
    A a;
    B b = nullptr;
    a = dynamic_cast<A *>(b);
}
```

```
class A
{
public:
    virtual ~A() { }
    virtual void act() = 0;
};

class B : public A
{
public:
    ~B();
};
int main()
{
    A * a;
    B * b = nullptr;
    a = dynamic_cast<A *>(b);
}
```

12. (2) Will this compile? If so, what is printed to standard output by this code? If not, why not?

```
void divide(const double * dividend, double divisor);

int main()
{
    double value = 100.0;
    cout << "value: " << value << endl;
    divide(&value, -10.0);
    cout << "value: " << value << endl;
    return 0;
}

void divide(const double * dividend, double divisor)
{
    double * localpointer;
    localpointer = const_cast<double *>(dividend);
    *localpointer /= divisor;

}
```

Yes
Value: 100
Value: -10

13. (2) Will this compile? If so, what is printed to standard output by this code? If not, why not?

```
void divide(const double * dividend, double divisor);

int main()
{
    const double value = 100.0;
    cout << "value: " << value << endl;
    divide(&value, -10.0);
    cout << "value: " << value << endl;
    return 0;
}

void divide(const double * dividend, double divisor)
{
    double * localpointer;
    localpointer = const_cast<double *>(dividend);
    *localpointer /= divisor;
}
```

Yes
Value: 100
Value: 100

14. (10) Note – this question spans 2 pages. For each of the statements labeled 1 through 10 in the main method of the following program, indicate if it compiles and, if so, what it prints to standard output.

```
#include <iostream>

using namespace std;

class A
{
public:
     virtual ~A() {}
     virtual void f1(){ cout << "A::f1" << endl; }
     virtual void f2() { cout << "A::f2" << endl; }
     virtual void f3(int n = -1) { cout << "A::f3(" << n
                                        << ")" << endl; }
     virtual void f4() { cout << "A::f4" << endl; }
     virtual void f5() { cout << "A::f5" << endl; }
};

class B : public A
{
public:
     virtual ~B() {}
     virtual void f1() { cout << "B::f1" << endl; }
     virtual void f2(int n = -1) { cout << "B::f2(" << n
                                        << ")" << endl; }
     virtual void f3() { cout << "B::f3" << endl; }
     virtual void f(int n) { cout << "B::f3(" << n << ")"
                                  << endl; }
     virtual void f5() { cout << "B::f5" << endl; }
};

class C : public B
{
public:
     virtual ~C() {}
     virtual void f2() { cout << "C::f1" << endl; }
     virtual void f2(int n = -1) { cout << "C::f2(" << n
                                        << ")" << endl; }
     virtual void f5() { cout << "C::f5" << endl; }
};

int main()
{
     A a;
     B b;
     C c;
```

```
    A& referenceAb = b;
    A& referenceAc = c;
    B& referenceBc = c;

    referenceAb.f1(); // 1
    referenceAc.f1(); // 2

    referenceAb.f2(); // 3
    referenceBc.f2(); // 4

    referenceAb.f3(); // 5
    referenceBc.f3(); // 6

    referenceAc.f4(); // 7
    referenceBc.f4(); // 8

    referenceBc.f5(); // 9
    referenceAb.f5(); // 10
}
```

1. **B::f1**
2. **B::f1**
3. **A::f2**
4. **C::f2(-1)**
5. **A::f3(-1)**
6. **B::f3**
7. **A::f4**
8. **A::F4**
9. **C::f5**
10. **B::f5**

15. (4) Implement a function object (functor) called Line. Line stores two doubles, slope and y_intercept. The Line constructor accepts two doubles, one for the slope and one for the y_intercept. The constructor should assign default values of 1 to each member variable if no parameters are provided.

    (4) Recall that a functor is any class or struct that overloads operator( ). The operator( ) for Line accepts a double called x. The operator( ) must return the value of y for the given x and the Line's slope and y_intercept using the formula for a line y = mx + b, where m is the slope and b is the y_intercept.

    (2) Suppose we have a std::vector called x_values. Apply the Line functor to each element in the vector and print the **y_intercept** that is generated. You may wish to use the for_each function defined in <algorithm>, or an iterator.

Corrected typo during exam from y_intercept to y value.

```
#include <iostream>
#include <vector>

class Line {
     double slope;
     double y_intercept;

public:
     Line(double slope = 1, double y_intercept = 1):
          slope{slope}, y_intercept{y_intercept} {}
     double operator()(double x)
     {
          return slope * x + y_intercept;
     }
};

int main()
{
     Line line;
     int x_array[] = {0, 1, 2, 3, 4, 5};
     std::vector<int> x_values {x_array,
               x_array + sizeof(x_array) / sizeof(int)};
     for (std::vector<int>::iterator it = x_values.begin();
          it != x_values.end(); ++it)
     {
          std::cout << line(*it) << std::endl;
     }

}
```

END OF EXAM
CHECK YOUR WORK
I HOPE YOU ENJOYED THE COURSE