2143 - OOP **Name:** ____________________

Fall 2020

Exam 1

10/1/2020

READ THE INSTRUCTIONS

- Use pencil only
- Write your name at the top of all pages turned in.
- Do not remove the staple from your test.
- Handwriting that is illegible (messy, small, not straight) will lose points.
- Indentation matters. Keep code aligned correctly.
- Answer all questions in the provided space directly on the test.
- **Failure to comply will result in loss of letter grade.**

This exam is 7 pages (without cover page) and 8 questions. Total of points is 149.

Grade Table (don't write on it)

Question	Points	Score
1	24	
2	10	
3	15	
4	25	
5	15	
6	10	
7	40	
8	10	
Total:	149	

1. (24 points) Given the list of definitions,match it with the correct word. The same word can be used more than once. All words don't have to be use.

#	Answer	Definition
1.	**E** :: Constructor	It is a special type of subroutine called to initialize an object.
2.	**F** :: Destructor	Cleans up allocated memory.
3.	**I** :: Inheritance	Defining class A by using the components of class B.
4.	**A** :: Abstraction	Hiding the details of the implementation from the user.
5.	**G** :: Encapsulation	Packaging data and methods together.
6.	**H** :: Friend	Used to allow access to private members in a class.
7.	**P** :: Polymorphism	Overloading methods.
8.	**L** :: Method	A function, except its in a class.
9.	**C** :: Class-Variable	A variable that is shared by all instances of a class.
10.	**D** :: Composition	Using classes A,B,C within the definition of class D.
11.	**O** :: Overloading	Same function name, different parameters.
12.	**Q** :: Private	Variables in this section cannot be read by sub classes.

Word Choices

A	Abstraction	F	Destructor	K	Member-Variable	P	Polymorphism
B	Class	G	Encapsulation	L	Method	Q	Private
C	Class-Variable	H	Friends	M	Multiple-Inheritance	R	Protected
D	Composition	I	Inheritance	N	Object	S	Public
E	Constructor	J	Instance-Variable	O	Overloading	T	Virtual

2. 10 On your answer sheet, write A-J and label each with abstraction or encapsulation. I know in class I downplayed the differences. But I'm curious, and that is why they are only worth 1 point each.
   (a) (1 point) **Abstraction** shows only useful data by providing the most necessary details.
   (b) (1 point) **Encapsulation** hides internal working.
   (c) (1 point) **Encapsulation** solves problem at implementation level.
   (d) (1 point) **Encapsulation** wraps code and data together.
   (e) (1 point) **Abstraction** is focused mainly on what should be done.
   (f) (1 point) **Encapsulation** is focused on how it should be done.
   (g) (1 point) **Encapsulation** helps developers to organize code easily.
   (h) (1 point) **Abstraction** hides complexity.
   (i) (1 point) **Abstraction** solves problem at design level.
   (j) (1 point) **Abstraction** hides the irrelevant details found in the code.

3. (15 points) There are 3 major concepts when we think about OOP. What are they?

1.	**Encapsulation**	2.	**Inheritance**	3.	**Polymorphism**

01-Concepts.md

4. (25 points) Write a **Point3D** class **definition** that will represent a 3D point. Assume all values to be integers. Do not add any setters or getters.

   Include:

   (a) (5 points) Include a default constructor that sets each data member to zero.

   (b) (10 points) Include an overloaded constructor to init each data member.

   (c) (10 points) Include a copy constructor.

```cpp
class Point3D{
    int x;
    int y;
    int z;
public:
    Point3D(): x{0}, y{0}, z{0}{}   // default constructor using init lists
    Point3D(){                      // default written old way
        x = y = z =0;
    }

    Point3D(int x , int y , int z): x{x}, y{y}, z{z}{}  // overloaded using init lists
    Point3D(int _x , int _y , int _z){                  // overloaded written old way
        x = _x;
        y = _y;
        z = _z;
    }

    Point3D(const Point3D &rhs){
        this->x = rhs.x;
        this->y = rhs.y;
        this->z = rhs.z;
    }

};
```

Reference: 04-OperatorOverloading

5. (15 points) Overload **ostream** for our 3D class so it prints the values like so: $[x, y, z]$ where x,y, and z would be integers (obviously).

```
friend ostream& operator<<(ostream &os,const Point3D &rhs){
    return os << "[" << rhs.x <<","<<rhs.y<<","<<rhs.z<<"]";
}
```

Reference: 04-OperatorOverloading

6. (10 points) Overload the multiplication operator for our 3D point class (just multiply each value with its equivalent in each instance).

```
  Point3D& Point3D::operator*(const Point3D &rhs) {

    this->x = this->x * rhs.x;
    this->y = this->y * rhs.y;
    this->z = this->z * rhs.z;

    return *this;
  }
```

Reference: 04-OperatorOverloading

7. (20 points) Assignment operator
   (a) (10 points) Overload the assignment operator for our 3D point class.
   (b) (10 points) What must we check for when overloading the assignment operator and why?

(a)

```
Point3D& Point3D::operator=(const Point3D &rhs) {
  // Check for self-assignment!
  if (this == &rhs)      // Same object?
    return *this;        // Yes, so skip assignment, and just return
  this->x = rhs.x;
  this->y = rhs.y;
  this->z = rhs.z;

  return *this;
}
```

Reference: 04-OperatorOverloading

(b)

We need to check for ***self assignment*** so that we do not destroy the values in "**this**" instance of the object. Put another way: *Self assignment fails because the memory associated with the current value of the left-hand-side is deallocated before the assignment, which would invalidate using it from the right-hand-side.*

Reference: 04-OverloadAssign.md

8. (10 points) We can overload pretty much any operator for any class. Operators, when overloaded, are classified as *destructive* and *non-destructive*. Here is a list of common operators: $+$, $-$, $*$, $/$, $\%$, $=$, $++$, $--$, $+=$, $-=$, $/=$, $\%=$. Can you explain to me which of these operators would be in either category and why?

   Convenience operators (aka compound assignment operators $+=$, $-=$, $/=$, $\%=$, etc.) are considered destructive because they overwrite the data on the left hand side of the operator. The assignment operator is NOT destructive because of the programmers choice to place whichever object on the left hand side of the operator.

```
class MyClass{
    //...
};

//...

MyClass A;
MyClass B;
MyClass C;

A += B;     // overwrites values in A no matter what (destructive).

C = A + B;  // returns a new instance of MyClass and assigns it to C

A = A + B;  // returns a new instance of MyClass and chooses
            // to overwrite A. But only be choice.
```

Reference: 04-OperatorOverloading