2143 - OOP
Spring 2020
Exam 1

Name: _____

READ THE INSTRUCTIONS

- Use pencil only
- Write your name at the top of all pages turned in.
- Staple pages together at the top left corner.
- Make sure your pages are in order, with questions also in order.
- Handwriting that is illegible (messy, small, not straight) will lose points.
- Indentation matters. Keep code aligned correctly.
- All answers will be written on the paper provided, and not directly on the test.
- When listing answers on your answer sheet you should place answers **vertically** each on their own line.
- **Failure to comply will result in loss of letter grade.**

This exam is 6 pages (without cover page) and 10 questions. Total of points is 178.

Grade Table (don't write on it)

Question	Points	Score
1	12	
2	24	
3	50	
4	25	
5	10	
6	12	
7	15	
8	10	
9	10	
10	10	
Total:	178	

1. (12 points) There are 3 major concepts when we think about OOP. What are they?

   | **Encapsulation** | **Inheritance** | **Polymorphism** | Abstraction |

   Reference: 01-Concepts.md

---

2. (24 points) Given the list of definitions, find the correct word below and place the Letter+Word on your answer sheet.

Definitions:		
A	**Constructor**	It is a special type of subroutine called to create an object.
B	**Destructor**	Cleans up allocated memory.
C	**Member Variable**	Holds data associated with a class and its objects.
D	**Class**	A definition of an abstract data type.
E	**Encapsulation**	Packaging data and methods together.
F	**Instance Variable**	Created when the object is created, and destroyed when object is destroyed.
G	**Polymorphism**	Overloading methods.
H	**Method**	A function, except its in a class.
I	**Class Variable**	A variable that is shared by all instances of a class.
J	**Abstraction**	Hiding the details of the implementation from the user.
K	**Overloading**	Same function name, different parameters.
L	**Private**	Variables in this section cannot be read by sub classes.

Words:			
Abstraction	Destructor	Class	Class-Variable
Composition	Encapsulation	Friends	Inheritance
Instance-Variable	Member-Variable	Method	Multiple-Inheritance
Object	Overloading	Polymorphism	Public
Private	Protected	Virtual	

---

3. 30 On your answer sheet, write A-J and label each with abstraction or encapsulation:
   (a) (5 points) **Abstraction** shows only useful data by providing the most necessary details.
   (b) (5 points) **Encapsulation** hides internal working.
   (c) (5 points) **Encapsulation** solves problem at implementation level.
   (d) (5 points) **Encapsulation** wraps code and data together.
   (e) (5 points) **Abstraction** is focused mainly on what should be done.
   (f) (5 points) **Encapsulation** is focused on how it should be done.
   (g) (5 points) **Encapsulation** helps developers to organize code easily.
   (h) (5 points) **Abstraction** hides complexity.
   (i) (5 points) **Abstraction** solves problem at design level.
   (j) (5 points) **Encapsulation** hides the irrelevant details found in the code.

   Reference: 05-AbsVSEnc.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) Add a copy constructor.

```
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;
    }

};
```

5. (10 points) Whats the difference between copy constructor and an overloaded assignment operator? You should remember this since your genius professor had a temporary lapse in memory and then a sudden remembrance of this exact thing.

   A copy constructor is called when a new instance of a class is created using an object of the same type. Remember **object A** resides in memory and has values, so we construct our **new object B** with the values from **A**.
   An assignment operator assumes two objects, both already existing with no need to allocate memory or initialize variables (that's a constructors job).

   Also, from our study guide:

- A copy constructor is used to initialize a newly declared variable from an existing variable. This makes a deep copy like assignment, but it is somewhat simpler:
  - There is no need to test to see if it is being initialized from itself.
  - There is no need to clean up (e.g., delete) an existing value (there is none).
  - A reference to itself is not returned.

Reference: 03-CopyConstructor.md

---

6. (12 points) A class that requires deep copies **generally** needs 4 things. What are they?

   - A **constructor** to either make an initial allocation or set the pointer to NULL.
   - A **destructor** to delete the dynamically allocated memory.
   - A **copy constructor** to make a copy of the dynamically allocated memory.
   - An **overloaded assignment operator** to make a copy of the dynamically allocated memory.

   Reference: 02-ShallowVSDeep.md

---

7. (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: example.cpp

---

8. (10 points) There is a set of operators that are considered **destructive**. Which operators are they, and what does this mean?

   Discussed in study guide:

Destructive		
$+=$	$-=$	$*=$

   Similar convenience operators:

Also Destructive				
$/=$	$\%=$	$\hat{}=$	$\&=$	$\mid=$

   Assume we are working with **class MyClass**

```
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: 05-OperatorOverloading

9. (10 points) Overload the assignment operator for our 3D point class.

```
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: 05-OperatorOverloading

10. (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;

```

```
7       return *this;
8     }
```

Reference: 05-OperatorOverloading