2143 - OOP
Spring 2023
Exam 1
February 28, 2023

Name: _____

READ THESE INSTRUCTIONS
• Use pencil only • Name on every page of answer sheets. • Do not remove the staple from your exam. • Do not crumple or fold your exam. • Handwriting that is illegible (messy, small, not straight) will lose points. • Indentation matters. Keep code aligned correctly. • Keep your answers on the answer sheet in ORDER! • Help me ... help you! I will take points off for excessive sloppiness.
Failure to comply will result in loss of letter grade

This exam is 5 pages (without cover page) and 7 questions. Total of points is 100.

Grade Table (don't write on it)

Question	Points	Score
1	15	
2	10	
3	20	
4	10	
5	10	
6	20	
7	15	
Total:	100	

1. (15 points) There are 3 major concepts when we think about OOP. What are they? Use layman's terms to define each word, give examples if possible.

Answer:
PIE

Polymorphism
This is the ability of an object or function to take on multiple forms or behave in different ways based on the context in which it is used.

Inheritance
A mechanism in that allows a new class, called the "subclass" or "derived class," to be based on an existing class, called the "base class," "superclass," or "parent class." The subclass inherits all the properties of the base class and can add new properties or modify existing ones.

Encapsulation
It refers to the practice of hiding the internal details of an object from the outside world and providing a well-defined interface for interacting with the object. Basically packaging data and methods together forcing users to use "methods" to access or alter the "data".

---

2. (10 points) What is the difference between a class and an object?

Answer:
A class is a template or blueprint for creating objects, while an object is an instance of a class that has its own state and behavior. A class defines the properties and methods that an object of that type will have, while an object can be created based on that class definition and can have its own unique state and behavior. Classes are static entities that exist at compile-time, while objects are dynamic entities that exist at run-time.

---

3. (20 points) Write a class **definition** with no implementation called **Box**. Your box class represents a three dimensional construct that has **length**, **width**, and **height**.

Look at the usage below to help determine what specific methods / constructors are needed. At a minimum, you will need to implement *setters* and *getters* for each data member. If this term confuses you, it means there should be at minimum two methods per data member to *SET* (change it) or *GET* (return what it is) the data member. Naming setters and getters is akin to: *setDataMember* and *getDataMember* (e.g. setWidth, or getHeight, etc).

Usage:

```
1  Box B1;                          // set each dimension to 0.0
2  Box B2(12.4,24.6,30.0);          // set length, width, height with passed in values
3  Box B3(B2);                      // Another box passed in to construct data members from
4  B1.expand(0.11);                 // expands all dimensions by 11 percent
5  B1.expand(0.11,0.22,0.08);       // expands length, width, height by 11, 22, and 8 percent respectively
6  B2.shrink(0.08);                 // shrinks all dimensions by 8 percent
7  B2.shrink(0.04,0.06,0.22);       // shrinks length, width, height by 4, 6, and 22 percent respectively
8  float volume = B3.getVolume();   // gets the volume of the box (length*width*height)
```

Answer:

```
class Box{
private:  // not necessary since default is private
    double length;
    double width;
    double height;
public:
    Box();
    Box(double,double,double);
    Box(const Box&);
    double getWidth();
    double getLength();
    double getHeight();
    void setWidth(double);
    void setLength(double);
    void setHeight(double);
    void expand(double);
    void expand(double,double,double);
    void shrink(double);
    void shrink(double,double,double);
    double getVolume();
};
```

---

4. (10 points) Overload the addition **+** operator for our *Box* class so it adds the volumes of two Boxes together. Look at example below:

```
    Box B1(10,10,10);
    Box B2(20,20,20);
    B1 = B1 + B2; // B1 =[Length: 30.0, Width: 30.0, Height: 30.0]
```

Assume you are defining this within the class definition (inline).

Answer:

```
Box operator+(const Box& rhs){
    double length;
    double width;
    double height;

    length = this->length + rhs.length;
    width = this->width + rhs.width;
    height = this->height + rhs.height;

    Box box;
    box.length = length;
    box.width = width;
    box.height = height;
    return box;
}
```

OR

```
Box operator+(const Box& rhs){
    Box box;
    box.length = this->length + rhs.length;
    box.width = this->width + rhs.width;
    box.height = this->height + rhs.height;
    return box;
}
```

OR

```
Box operator+(const Box& rhs){
    return Box(this->length + rhs.length, this->width + rhs.width, this->height + rhs.height);
}
```

---

5. (10 points) Implement the **getVolume** method for our *Box* class. Assume you are defining this **outside** of the class definition.

   Answer:

```
double Box::getVolume(){
    return length * width * height;
}
```

---

6. (20 Points) Copy constructor and Assignment Operator questions.

   (a) (5 points) Do we need a copy constructor for every class? Explain.

   Answer:
   No we do not need a copy constructor for every class. The only time you need a copy constructor is when your class dynamically allocates memory (uses pointers). A copy constructor created by the compiler would not do a deep copy which is necessary with pointers. So, no pointers, no copy constructor needed.

   (b) (5 points) Does the Box class need a copy constructor? Explain.

   Answer:
   No. It does not have any pointers, so a copy constructor is not necessary.

   (c) (5 points) Do we need to overload the assignment operator (=) for every class? Explain.

   Answer:
   No. This is for the same reason we don't need a copy constructor for every class. The compiler can handle non pointer data members just fine.

(d) (5 points) What do we need to check for when overloading the assignment operator? Explain.

Answer:
Self assignment. If we don't check for self assignment, we risk deleting the values in our object and losing them completely.

---

7. (15 points) Rewrite the following struct and add an overload for **ostream**:

```
1  struct Grades{
2      string fname;
3      string lname;
4      int grades[10];
5  };
```

The output should look like:

First: firstNameValue
Last: lastNameValue
Grades:
1. 33
2. 88
3. 99
...
10. 87

Answer:

```
1  struct Grades{
2      string fname;
3      string lname;
4      int grades[10];
5      friend ostream& operator«(ostream&os, const Grades& other){
6        os «"First: "« other.fname«"\nLast: "«other.lname«"\nGrades:\n";
7        for(int i=0;i<10;i++){
8          os«"\t"«i«". "«other.grades[i]«endl;
9        }
10       return os;
11     }
12 };
```