2143 - OOP
Fall 2022
Test III
December 3, 2022

Name: _____

READ THESE INSTRUCTIONS
• Use pencil only • Do not crumple, fold, or mutilate your exam. • Handwriting that is illegible (messy, small, not straight) will lose points. • Answer ALL questions on the paper or answer sheet provided but NOT directly on the test.
Failure to comply will result in loss of letter grade

Question	Points	Score
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
7	10	
8	10	
9	10	
10	10	
11	10	
12	10	
13	10	
Total:	130	

1. (10 points) Explain the difference between a class and an object.

Solution:
A class is the definition of an Abstract Data Type. It's the part that defines how data is stored and what methods work on that data. An object is an instance of that same class. This means that a copy of the class has been invoked and resides in memory with its own state.

2. (10 points) Explain what polymorphism is and give some examples.

Solution:
- Polymorphism is a"thing" (function / object) that acts differently based on context or situation.
- It could be as simple as overloading a function with a different parameter list so that the "function" behaves differently when called with different parameters.
- Overloading operators is another prime example. The "plus" sign (+) adds integers but concatenates strings. This is a great example.
- Overriding is another form of polymorphism, where functions in sub-classes redefine a parent method so it acts accordingly with the sub-class. - There are also two types of polymorphism: static and dynamic which I won't define here.

3. (10 points) Given the snippet below. Write a c++ snippet that uses the smallest amount of code and least number of variables possible to invoke all the **show()** methods in each class definition. One from Base, Derived, and Child. As there is some leeway in my question, remember what topics we are covering on this exam, and choose your solution wisely.

```
1  class Base{
2  public:
3      virtual void show() { cout<<" In Base! n"; }
4  };
5
6  class Derived: public Base{
7  public:
8      void show() { cout<<"In Derived Class! n"; }
9  };
10
11 class Child: public Base{
12 public:
13     void show() { cout<<"In Child Class!  n"; }
14 };
```

Solution:
The goal of this question was for you to use your knowledge of dynamic polymorphism

to invoke each of the three methods. Using the same pointer for all three uses the least amount of code.

```
Base *b;
b = new Base;
b.show();
b = new Derived;
b.show();
b = new Child;
b.show();
```

4. (10 points) We spent a lot of time in class discussing these two topics. I'll give you a hint they both involve pointers. Can you think of two major topics that we discussed that only matter when pointers are used?

Solution:

1.) Deep Copy
2.) Dynamic Polymorphism

5. (10 points) Describe why pointers effect the two topics in the previous question so much?

Solution:

Deep Copy:
A shallow copy of an object will only copy values at the highest level. This means if there are pointers involved, only addresses get copied between objects. If that pointer is the "root" of a large binary search tree, none of the tree data gets copied in a shallow copy. Therefore, we must implement the code to manually copy that tree into the new object: aka deep copy.

Dynamic Polymorphism:
Dynamic Polymorphism is another topic where pointers are a must. If pointers are not used to point to objects, there is no runtime polymorphism. All choices would be made statically at compilation time. Using pointers, along with virtual methods allows choices to be made while the process is executing.

6. (10 points) Will this snippet run? Depending on which side you pick, **explain** your case.

```
class Graphics{
public:
```

```
3     virtual void sprite() = 0;
4     virtual void move() = 0;
5     virtual void rotate() = 0;
6  };
7
8  int main(void){
9     Graphics * g = new Graphics;
10
11    return 0;
12 }
```

Solution:

It will NOT run. Graphics is an interface (or at least contains pure virtual methods) and cannot be instantiated.

7. (10 points) What is the difference between an **abstract class** and an **interface** ?.

Solution:

Abstract Class:
An abstract class is a class that contains at least one pure virtual function. It can contain more. Remember, any class with a pure virtual function cannot be instantiated and must be extended (inherited from) and implement that method in the child class. Having said that, an abstract class can have other methods that are implemented, but has 1 or more pure virtual methods. **Interface:**
An interface is similar to an abstract class as it must contain one or more pure virtual functions, however, it cannot contain ANY methods that are implemented. All methods are pure virtual and must be implemented in a sub class.

8. (10 points) Pick **one** of the design patterns below, and tell me about it.
Facade , **Singleton** , **Factory** , **Observer**

Solution:

Facade is a structural design pattern that provides a simplified (but limited) interface to a complex system of classes, library or framework. While Facade decreases the overall complexity of the application, it also helps to move unwanted dependencies to one place.

Singleton is a creational design pattern, which ensures that only one object of its kind exists and provides a single point of access to it for any other code. In short if we create a hundred copies of a singleton, they are all the same!

Factory Method is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.
Observer is a behavioral design pattern that allows some objects to notify other objects about changes in their state. The Observer pattern provides a way to subscribe and unsubscribe to and from these events for any object that implements a subscriber interface.

9. (10 points) Look at the code snippet below and determine what **protection** level X, Y, Z are in each derived class. Sometimes depending on the protection level, a data member may not be accessible by the sub class. Clearly print **public** , **private** , or **protected** in the first column of blanks when appropriate, but in the second column only write if a data member is **inaccessible**.

```
1  class Base {
2    public:
3      int X;
4    protected:
5      int Y;
6    private:
7      int Z;
8  };
9
10 class PublicDerived: public Base {
11
12   // 1.) X _____          |    _____
13
14   // 2.) Y _____          |    _____
15
16   // 3.) Z _____          |    _____
17 };
18
19 class ProtectedDerived: protected Base {
20
21   // 4.) X _____          |    _____
22
23   // 5.) Y _____          |    _____
24
25   // 6.) Z _____          |    _____
26 };
27
28 class PrivateDerived: private Base {
29
30   // 7.) X _____          |    _____
31
32   // 8.) Y _____          |    _____
33
34   // 9.) Z _____          |    _____
35
36 };
```

Solution:

Public inheritance makes *public* members of the base class *public* in the derived class, and the protected members of the base class remain protected in the derived class.

Protected inheritance makes the *public* and *protected* members of the base class *protected* in the derived class.

Private inheritance makes the *public* and *protected* members of the base class *private* in the derived class.

Note: private members of the base class are inaccessible to the derived class.

1.) X is public
2.) Y is protected
3.) Z is inaccessible

4.) X is protected
5.) Y is protected
6.) Z is inaccessible

7.) X is private
8.) Y is private
9.) Z is inaccessible

10. (10 points) Given the code below. What would you add so that both Rectangle AND Triangle classes are forced to implement the area method.

```
class Polygon {
protected:
  double width, height;

public:
  void set_values(double a, double b) {
    width = a;
    height = b;
  }
};

class Rectangle : public Polygon {
public:
  double area() { return width * height; }
};

class Triangle : public Polygon {
public:
```

```
  double area() { return width * height / 2; }
};
```

Solution:
Simply add a pure virtual function to class Polygon:

```
class Polygon {
protected:
  double width, height;

public:

  virtual double area() = 0;  //<=====================================HERE

  void set_values(double a, double b) {
    width = a;
    height = b;
  }
};
```

11. (10 points) Is overloading the assignment operator a replacement for a copy constructor? Are they really just the same thing? Please state your case and give examples.

Solution:

Copy constructor	Assignment operator
It is called when a new object is created from an existing object, as a copy of the existing object This operator is called when an already initialized object is assigned a new value from another existing object.	It creates a separate memory block for the new object. It does not create a separate memory block or new memory space.
It is an overloaded constructor.	It is a bitwise operator.
C++ compiler implicitly provides a copy constructor, if no copy constructor is defined in the class.	A bitwise copy gets created, if the Assignment operator is not overloaded.

Use the code snippet below to answer the next two questions following the snippet:

```
class Thingy {
public:
  static int thingyCount;

  // Constructor definition
  Thingy(string n) {
    name = n;

    // Increase every time object is created
```

```
    thingyCount++;
  }
  double GetName() { return name; }

private:
  string name;
};
```

12. (10 points) Write the necessary code in the space provided to initialize the variable **thingyCount** to an appropriate value. Write another line of code that would print *thingyCount* to *stdout*.

Solution:
By declaring a function member as static, you make it independent of any particular object of the class. A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::.

Actual Answer:

```
int Thingy::thingyCount = 0;

cout<<Thingy::thingyCount<<endl;
```

13. (10 points) Write a member function for **Thingy** to allow access to **thingyCount** . You can write your method as if you were writing it inline if you wish.

Solution:
A static member function can only access static data member, other static member functions and any other functions from outside the class.

Static member functions have a class scope and they do not have access to the this pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.

Actual Answer:

```
static int getThingyCount() {
   return thingyCount;
}
```