

2143 - OOP
Spring 2021
Take Home Exam
April 25, 2021

Name: Linus Fackler

READ THESE INSTRUCTIONS

- D Create a digital document (PDF) that has zero handwriting on it. Print and bring to the final exam on *Tuesday April 27th from 8:00 am - 10:00am.*
- D Your presentation and thoroughness of answers is a large part of your grade. Presentation means: use examples when you can, graphics or images, and organize your answers!
- D I make every effort to create clear and understandable questions. You should do the same with your answers.
- D Questions should be answered in order and clearly marked.
- D Your name should be on each page, in the heading if possible.
- D Place your PDF on GitHub (after you take the actual final) and in your assignments folder.
- D Create a folder called **TakeHomeExam** and place your document in there. Name the actual document: **exam.pdf** within the folder.

Failure to comply with any of these rules will result in a NO grade. This is a courtesy exam to help you solidify your grade.

Grade Table (don't write on it)

Question	Points	Score
1	70	
2	15	
3	40	
4	10	
5	15	
6	10	
7	10	
8	20	
9	15	
10	20	
11	35	
12	10	
13	10	
Total:	280	

Warning: Support each and every answer with details. I do not care how mundane the question is ... justify your answer. Even for a question as innocuous or simple as "What is your name?", you should be very thorough when answering:

What is your name?: My name is Attila. This comes from the ancient figure "Attila the Hun". He was the leader of a tribal empire consisting of Huns, Ostrogoths, Alans and Bulgars, amongst others, in Central and Eastern Europe. My namesake almost conquered western Europe, but his brother died and he decided to go home. Lucky for us! We would all be speaking a mix of Asiatic dialects :)

Single word answers, and in fact single sentence answers will be scored with a zero. This is a take-home exam to help study for the final and boost your grade. Work on it accordingly.

1. This VS That. Not so simple answers Ç:

(a) (7 points) Explain the difference between a *struct* and a *class*. Can one do whatever the other does?

A struct is less secure than a class. Data members or methods of a struct are public by default and of a class private by default. That means you can access data members of a struct in a main directly, while you'd have to use public methods in a class to access the private data members outside of the class.

(b) (7 points) What is the difference between a *class* and an *object*?

A class is a definition of an abstract data type that works as a blueprint/template and consists of data members and methods. An object is an instance of a class that is declared. The class is "car" and the objects are "Miata", "Mustang", or "Skyline".

(c) (7 points) What is the difference between *inheritance* and *composition*? Which one should you lean towards when designing your solution to a problem?

A composition is a concept where it is possible to "be apart of something", rather than actually being it. A car **has** an engine, but **isn't** an engine. Inheritance on the other side is a concept where classes and object **inherit** properties from their parents' class. It's like a child that inherits money from their parents without actually having to do anything for it. This prevents them from having to do the same thing that their parents already did and therefore making code simpler and easier read. When designing a solution for a problem you should lean towards composition. It puts multiple things together and avoids complexity.

(d) (7 points) What is the difference between a *deep* vs a *shallow* copy? What can you do to make one or the other happen?

A **shallow copy** of an object copies all the member field values. It still points to the dynamically allocated memory of the original object. Changes in the original objects' data members also affects the copy.

A **deep copy** of an object is a fully independent copy of an object. Changes in the original won't affect the copy at all. It creates dynamically allocated memory for the copy.

An object “person” is composed of other objects, for example “Name” and “Address”.

Shallow copy of “person”: Only copies the main object and references the same inner objects as the original object. If you change the “Address” object of the original “Person” object will reflect as a change in obviously the original object, but also the copy.

```
Person P1;  
P1.setData( ... );  
Person P2 = P1;
```

Deep copy of “person”: Changes to “Address” on the original won’t affect the copy at all. For it, the class must have a copy constructor where the assignment operator is overloaded.

```
Person(Person &p)    // copy constructor  
{  
    address = new string;  
    *address = *(p.address);  
}
```

-
- (e) (7 points) What is the difference between a *constructor* and a *destructor*? Are they both mandatory or even necessary?

Constructors and destructors are member functions to initialize an object. The constructor is called when an instance of an object is created. The destructor is called when the object is deleted. It helps to deallocate memory. Like the words say, a constructor builds something, a destructor destroys something.

-
- (f) (7 points) What is *static* vs *dynamic* typing? Which does C++ employ and which does Python employ?

Static typing means data types are known and checked for correctness before the program is run. → `int x = 5` // x is declared as an int before the program is running

C++ is statically typed

Dynamic typing means data types are only known as the program is running.

Python is dynamically typed. → `x = 5` // x is declared as an int while program is running

The next line could be → `x = “Potato”` // x is now a string

-
- (g) (7 points) What is *encapsulation* vs *abstraction*? Please give some examples!

Encapsulation is the concept of bundling data and methods together. “Enclosing” something in a “capsule”. A capsule is mixed with several medicine.

Abstraction is the concept of hiding unnecessary details from the user to make it as user-friendly as possible. The user doesn’t know how a car works, he just needs to know how to operate it.

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

Abstract classes are classes that are abstract, but can't be initiated, rather used as subclasses. They have static fields and static methods.

Interface is just the declaration of methods of an object, not the actual implementation.

In an interface, we declare what an object can do, but not how.

The interface of class "car" has methods like "void changeGear(int newGear)" or "void turnAC(int status)", but doesn't show how. This is the job of the class "car".

(i) (7 points) What is the difference between a *virtual function* and a *pure virtual function*?

A virtual function is a member function that is declared within the base (parent) class. Derived (child) classes can override them. The object is dynamically typed.

A pure virtual function is a virtual function that doesn't have an implementation, rather a declaration that ends in =0.

Both are concepts of Run-time polymorphism.

(j) (7 points) What is the difference between *Function Overloading* and *Function Overriding*?

A virtual function is a member function that is declared within the base (parent) class. Derived (child) classes can override them. The object is dynamically typed.

A pure virtual function is a virtual function that doesn't have an implementation, rather a declaration that ends in =0.

Both are concepts of Run-time polymorphism.

2. Define the following and give examples of each:

(a) (5 points) Polymorphism

The ability of an object to take on many different forms. It can take on any form of its inherited (parent) class. A man can be a husband, father, employee, ..., just like a shape can be a triangle, circle, shape, ...

(b) (5 points) Encapsulation

The idea of bundling data and methods together. Hiding specific information from the outside → "enclosing" something in a "capsule"

A patient gets a capsule that consists of a mixture of different medicine. He doesn't see each medicine that's in there.

(c) (5 points) Abstraction

The concept of hiding unnecessary details from the user to make it as user-friendly as possible. The user doesn't need to know how a coffee machine works, he just needs to know what buttons to press, not what the machine behind the button does.

3. (a) (5 points) What is a default constructor?

A constructor that can be called with no arguments. In most cases it sets the private data members of the object.

(b) (5 points) What is an overloaded constructor? And is there a limit to the number of overloaded constructors you can have?

A constructor that can be called with no arguments. In most cases it sets the private data members of the object. There is no limit to the number of overloaded constructors. But each have to have a different parameter list.

(c) (5 points) What is a copy constructor? Do you need to create a copy constructor for every class you define?

A constructor is a constructor with the reference to an object of the same class as parameter. It is used to create a deep copy of an object.

No, it is not necessary for every class. Only if you want to create a deep copy of an object.

(d) (5 points) What is a deep copy, and when do you need to worry about it?

A deep copy is an independent copy of the original. It has its own dynamically allocated memory.

You need it if you want a copy that is not referenced to any of the original data members and therefore won't be affected if changes were made on the original.

(e) (5 points) Is there a relationship between copy constructors and deep copying?

You need a copy constructor to allocate new memory for the deep copy. If there is no copy constructor, then you can't allocate new memory for the copy just by assigning it to an existing object.

(f) (5 points) Is a copy constructor the same as overloading the assignment operator?

No. Both copy one object to another, but the copy constructor initializes a new object, while the assignment operator replaces the contents of an existing instance of an object.

(g) (10 points) Give one or more reason(s) why a class would need a destructor.

If a member variable is a pointer and therefore dynamically allocated memory is created which needs to be deleted before the program is terminated.

While the memory of the pointer is released, as it goes out of scope, the object that the pointer points to is not released.

4. (10 points) What is the difference between an abstract class and an interface?

Hint:

You should include in your discussion:

- Virtual Functions
- Pure Virtual Functions

An abstract class can have abstract and non-abstract methods, while an Interface can only have abstract methods.

If pure virtual functions/methods exist, the class is abstract and can't be initiated on its own.

They are mostly used to define abstract classes and interfaces.

Abstract classes are base classes where you have to derive from them and then implement the pure virtual functions.

Interfaces are empty classes where all functions are pure virtual and you have to derive and then implement all of the functions.

5. Describe the following (make sure you compare and contrast as well):

(a) (5 points) Public

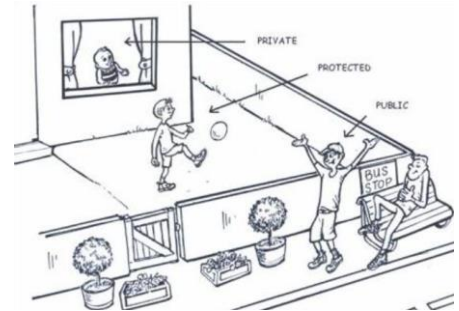
Data members or functions/methods that are accessible from outside of the function and are not protected.

(b) (5 points) Private

Data members or functions/methods that are only accessible from inside the class

(c) (5 points) Protected

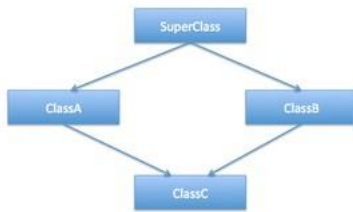
Like private but also accessible by inheriting the class



Hint:

- Make sure you define each item individually as well.
- Use examples.
- If your not sure, use examples to make your point.
- Ummm, example code is always welcome.

6. (10 points) What is the diamond problem?

**Hint:**

- This is a question about multiple inheritance and its potential problems.
- Use examples when possible, but explain thoroughly.

ClassC inherits from ClassA and ClassB, which both inherit from SuperClass.

This problem is about multiple inheritance, which is not allowed in some programming languages like Java.

7. (10 points) Discuss Early and Late binding.

Hint:

- These keywords should be in your answer: **static, dynamic, virtual, abstract, interface**.
- If you haven't figured it out use examples.

With early binding, the compiler associates directly an address to the function call. It replaces the machine call telling him to jump to the address. This can be achieved using the virtual keyword.

In late bindings, the compiler adds code that identifies the object at runtime then matches the call with the right function definition. This can be achieved using a virtual function.

8. (20 points) Using a **single** variable, execute the show method in *Base* and in *Derived*. Of course you can use other statements as well, but only one variable.

```
class Base{
    public:
    virtual void show() { cout<<" In Base n"; }
};

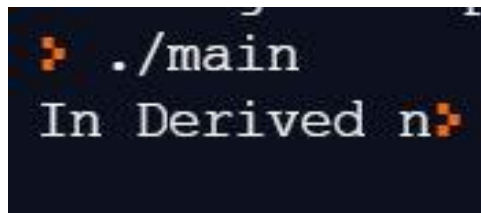
class Derived: public Base{
    public:
    void show() { cout<<"In Derived n"; }
};
```

Hint: This is implying that dynamic binding should be used. A pointer to the base class can be used to point to the derived as well.

```
int main()
{
    Base *B;
    Derived D;

    B = &D;
    B->show();

    return 0;
}
```



9. (15 points) Given the two class definitions below:

```
class Engine {} // The Engine class. class Automobile {} // Automobile class which is
parent to Car class.
```

You need to write a definition for a Car class using the above two classes. You need to extend one, and use the other as a data member. This question boils down to composition vs inheritance. Explain your reasoning after you write your Car definition (bare bones definition).

```
6  class Engine
7  {
8      int HP, torque, maxSpeed;
9  };
10
11  class Automobile
12  {
13      protected:
14          int year, numSeats, numWindows;
15          Automobile()
16          {
17              year = 1969;
18              numSeats = 5;
19              numWindows = 4;
20          }
21  };
22
23  class Car: public Automobile
24  {
25      protected:
26          Engine BigBlock;
27          string make, model;
28      public:
29          Car();
30          ~Car();
31  };
32
```

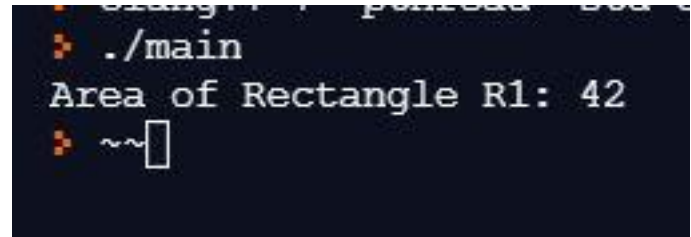

10. (20 points) Write a class that contains two class data members *numBorn* and *numLiving*. The value of *numBorn* should be equal to the number of objects of the class that have been instanced. The value of *numLiving* should be equal to the total number of objects in existence currently (i.e., the objects that have been constructed but not yet destructed.)

```
5
6  class Potatoes
7  {
8      private:
9          int numBorn = 0;
10         int numLiving = 0;
11
12     public:
13         Potatoes()
14         {
15             numBorn++; // number of instances of class
16             numLiving++;
17         }
18
19         ~Potatoes()
20         {
21             numLiving--; // only num of currently existing
22         }
23
24     };
```

11. (a) (10 points) Write a program that has an abstract base class named *Quad*. This class should have four member data variables representing side lengths and a *pure virtual function* called *Area*. It should also have methods for setting the data variables.
- (b) (15 points) Derive a class *Rectangle* from *Quad* and override the *Area* method so that it returns the area of the Rectangle. Write a main function that creates a Rectangle and sets the side lengths.
- (c) (10 points) Write a top-level function that will take a parameter of type *Quad* and return the value of the appropriate Area function.

Note: A **top-level function** is a function that is basically stand-alone. This means that they are functions you can call directly, without needing to create any object or call any class.

```
6 class Quad // a)
7 {
8     protected:
9         double length, height;
10
11     public:
12         virtual double Area() = 0;
13
14         void setSize(double l, double h)
15         {
16             length = l;
17             height = h;
18         }
19 };
20
21 class Rectangle :public Quad // b)
22 {
23     double Area()
24     {
25         return length * height;
26     }
27 };
28
29 double getArea(Quad *q) // c)
30 {
31     return q->Area();
32 }
33
34 int main()
35 {
36     Rectangle R1;
37     R1.setSize(6, 7);
38
39     cout << "Area of Rectangle R1: " << getArea(&R1) << endl;
40
41     return 0;
42 }
```



```
./main
Area of Rectangle R1: 42
~~
```

12. (10 points) What is the rule of three? You will have answered this question (in pieces) already, but in the OOP world, what does it mean?

The rule of three is a rule of thumb in C++ which claims that if a class defines either a **Destructor**, **Copy Constructor**, or **Copy Assignment Operator**, it should explicitly define all three. It can't just have 1.

13. (10 points) What are the limitations of OOP?

- Designing your program in OOP is harder than with other concepts
- Therefore, needs proper **Skill** to even start a program
- OOP requires proper planning before attempting a program
- The size of OOP programs is larger
- The Diamond Inheritance Problem
- Slower Execution of programs
- Not all problems can be reflected in OOP