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* 27*th 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:

hat 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?

There are a few differences between a structure and a class. The data that is inside of a structure is is public by default and can be accessed throughout the project. The data that is inside of a class is private by default meaning that it can only be accessed inside the class and not outside. A class hides the implementation from the user. The implementation of a structure is not secure and doesn't hide the complexty of the implementation from the user. A class uses private inheritance and the structure uses public inheritance within its implementation. While they are similar, a class is the building blocks of creating an object in C++. A structure is basic blueprint in a group of variables that creates an object and does not use a key word to be declared. When the user creates multiple structures, it does not change the value of the created structure instead creates a new instance (like creating a structure to hold a person name, grade,age, and height).

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

A class is the building blocks of creating an an object and all its component. This serves as the basic building block of the actual creation of an object(like building a car with a color,type and size) The object is the actual object that is created using that blueprint that is implemented by the class itself.

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

Iheritance in c++ is the process in which an object acquires all the properties and the stated behaviours that are associated with that. This is particulularly ueful when the user wants to modify certain things inside another class but still keep the original without modification. Composition of one or more of the objects of a certain class have a relationship with other classes.An example of this is a string object.(On thing is composed of another). Composition indicates that it has a "has-a" relationship between objects. Inheritance, indicates an "is-a" relationship that you lean toward using composition over inheritance.

(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 deep copy copies all the fields of an object and makes a copy of it to a dynamically allocated memory spot with a pointer pointing to them. To create a deep copy, one must write a copy constructor and an overloaded assignment operator or else, it will be self assigned and that is a very bad bad thing.

A Shallow copy is similar to deep copy in the assignment of each member of an object to that of another object, but it differs in the manner in which the field of reference type is copied.This is best to use if you are copying an object to another object and you aren't trying to constantly change

the object. This is accessed through the address referencing. The big difference between the two is that a deep copy is a pointer to type and the shallow copy is a pointer to address type. However, it is best to utilize the deep copy contructor if there is a lot of changes being made. Shallow copying is more efficient where object referencing allows the objects to be passed around by address and the entire object doesn't need to be copied.

(d) (7 points) What is the difference between a *constructor* and a *destructor*? Are they both mandatory or even necessary? There are many types of a constructor inside of a class. These include a copy constructor, a default constructor and overloaded consturcutor. The constructor is base level responsible for creating a things such as default value for a class variable or if the user wants to overload the constructor, then an overloaded constructor is used to assign the user defined value to be the new values of that inside of the class. A destructors main job is to destroy the object. This is mainly used in larger projects and not so much in small based projects where not a lot of overloading or assignments are being done. But the destructoys the instance of that variable and begins anew. The compiler automatically creates a default constructor if one is not provided but in the case of both of these, it is best to add these inside especially the constructor to create instance of the variable or object in question and assign it values or a default value.

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

Static typing refers to when certina data types need to be initialized and be static at tun time. This basically implies that they need to be setup to run at compilation of programs and cannot change thie value without specific instructions inside of the code blocks and program itself. These static typing types are usually cast in the program. A language is also known to be statically typed if variable types are known at ompile time Meaning that the type must be identified for each variable. This is the case in C++.

A language is considered to be dynamically typed if the types that are associated with the runtime value and variables . This basically means that a a programmer can write quicker when writing code because they do not have to announce what type their variables and other types are every single time. This is a huge component of the Python language.

(7 points) What is *encapsulation* vs *abstraction*? Please give some examples!
(Abstraction is in the design and encapsultation is in the implementation.) Abstraction hides the background details and emphasizes the essential data points for reducing the complexity and increase efficiency of the running of a program. An example of encapsulation is like taking a picture with a smartphone. The user does not need to understand how the smartphone process all the internal functions from taking the picture to creating the actual image by hiding all the unnecessary detail from the user. Encapsulation is also implemented during during the Implementation level and focuses mainly on how it should be done. Whereas the Abstraction is implemented at the design level focuses on what should be done.

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

An Interface can have only abstract methods. An abstract class can have abstract and non-abstract methods when utilized. An abstract class will allow other classes to inherit directly from it but it is not able to create and instance of itself. The only functionality of an abstract class is allow other subclasses to inherit from it.An interface is basically like an empty template. An interface t contains methods without a formal signature or body but it cannot do anything by itself It is basically an empty tempalet that you can copied to and filled. It is also used to provide hierarchies and guidelines for other subclasses.

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

A virtual function is a function that allows a program to call methods that don't really "exist upon compilation. This also refers to the inheritable and functions that are able to be overridden in which are dynamically dispatched. A pure virtual function is a function that acts as a sort of placeholder for other variabls and takes info from outside itself like other functions and other classes to fill in blank spaces. A huge advantage of using a pur virtual function is that information I very easily malleable and the different people utilizing it can change it according to their own needs. The only way that this will work it that there must be some class in which it is derived from where ethe data can be acquired from.By it standing alone, this is useless. An example of the use of a virtual function is one in which asks for certain information from the user. The program will then go through the program and change (name or variable value etc to fit that users needs)

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

_____

Function overloading is a feature in C++ where two or more functions can have the same name but different parameters. When a function name is overloaded with different tasks that it needs to accomplish, like reading a user input and changing the default values, it is called Function Overloading. When a function that is being used in c++, it can have the same names but one functions parameters cannot be identical to the next(i.e a default versus another function with the same name but with two parameters.)Function overloading can also be seen to have the same concept and take on the form of polymorphism.

Function overing is basically a feature that allows a child and the inherited class member to have the same function name at the same time without any overlaps. A child class will inherit the data member and the functions of the parent class. However, when the user would like to override the functionality inside of the child class, then that is where function overriding comes in to play. This essentially is like creating a newer version that is derived from the original inside of the child class.Below is an example of this feature.

To override a function you must have the same signature in child class. By signature I mean the data type and sequence of parameters. Here we don't have any parameter in the parent function so we didn't use any parameter in the child function.

```cpp
#include <iostream>
using namespace std;
class BaseClass {
public:
    void disp(){
        cout<<"Function of Parent Class";
    }
};
class DerivedClass: public BaseClass{
public:
    void disp() {
        cout<<"Function of Child Class";
    }
};
int main() {
    DerivedClass obj = DerivedClass();
    obj.disp();
    return 0;
}
```

Output:

```
Function of Child Class
```

2. Define the following and give examples of each :

(a) (5 points) Polymorphism

(b) (5 points) Encapsulation

(c) (5 points) Abstraction

Polymorphism is the the ability of an object to take several forms. This is a term that describes the basic principle in which different classes are able to be used withing the system and provide their own individual implementation. (many forms)

Encapsulation is a process that combines data member and functions togeter into a single item which is referred to more commonly as class, which in turn prevents direct access to this data Although, access to these items and data members are done through the functions of those classes.

abstraction in C++ is the creation of a functionality of a created class, but details that are unnecessary are not shown to the user. The details using abstraction are intentionally hidden from the user because some of them are unesseary for the user to see.

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

A default constructor is inside of a class declaration. A default constructors main purpose is to provide default values to the variables and parameters inside of the class. It is useful to have a default constructor instead of not having one. However, if the user does not define a default constructor, the system will automatically assign default values which more chances than not are not the ones the user wants.

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

Overloaded constructors are constructors inside of a declared class that can have the same name, but pass through them different number of parameters. The constructors cannot be the same or pass the same values like (constructor(int) and then declaring another constructor called(int). This will cause a compilation error due to the compiler not knowing which one to call. Also, depending on the number and the type of args that are passed inside of the program, the correspond constructor will be called and utilized.There is not a limit to the ammount of overloaded constructors that you can have inside of a class as long as no two constructors are identitcal.

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

A copy constructor is a constructor that is and overloaded constructor that copied all of the data from one object and then makes a copy of all those values into another address to be accessed. However, problem can arise from doing this is one is not careful and might develop a self assignment to itself which can be very problematic. This is where deep and shallow copy play a big role in copy constructors. And No you do not need a copy constructor for every class you define. The times that using a copy constructor is mainly when you are wanting to allocated nem memory dynamically from the heap. The compiler will automatically copy basic default values without having to use a copy constructor but one, it will move a lot slower and two, with default creating copy constructor for a class using pointers can create a wide ariety of issues. Thus, whenever the user is using new memory, it is best to use the copy constructor, if not, then it is probably best to just not use the copy constructor.

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

A deep copy is a copy that is created when the user wants to copy everything over using a pointer. By this when using pointer types in C++, when the deep copy is made, everything is copied over from the object over to new allocated memory and the pointer will create a new allocated memory spot that it points to each time a new object is created. The time that this should be used is when you are wanting to allocated new memory for the creation of new objects.

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

Yes there is. The reason for this being the case is if the user wants to create a deep copy of an object, then the contructor will copy everything over from the constructor, assign it new memory and utilize all the contents of the copy contructor. I.e if the user wants to create a deep copy of say a

class car object, the deep copy will gather all of that information using the copy constructor and create a new instance and thereby the relationship between the two is instantiated.

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

They are not the same. The one thing they do have in common are both are used to initializing one object to another object. The big difference between the two is that the copy constructor creates a separate memory space allocated for the new object. But the assignment operator does not create new space. It instead it uses the reference variable to point to the previous memory block.

Example of assignment operator   classname Obj1,Obj2;

Obj2=Obj1;//assigning obj1 to obj2

Example of copy constructor   classname (const classname & rhs){//

//body of copy constructor};

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

A class would need a destructor if if memory is allocated over and over again and a destructor is necessary if memory is allocated to the heap. This in term helps free up borrowed memory is memory is being allocated a lot throught the instance of the new key word when creating new instances. This in turns just helps out the memory of the program to run more efficiently. Especially if the contructor is creating a lot of objects and after it is done using the object and doesn't need them anymore, the code will destroy that object and then begin again.

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

5. An abstract class that is referred to in OOP has at least one abstract method. This method has no code in its base class, instead, the code will be derived from its derived class this method in the derived class must be implemented with the same access modifier, numbers, and type and have the same return type as that of the original bas class. The objects of the abstart class type are not created because the code itself is the one that creates the instance of an object from the abstart class would create an error upon runtime of a program.This also pertains to to virtual functions when you set a value of one class to say for instance, area equals 0, then you call that method function inside of another function, you can override that virtual function by user defined value to be overridden.

An interface main

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

   (a) (5 points) Public- The public keyword is used inside of a class which gives access to any functions or methods contained within with anyone inside of the current program that can be access. These methods contained within this block are then transferred over to the private section within the class that only the public of that class or friends have access to.

   (b) (5 points) Private- The keyword private means that the only one inside of the program that has access to any of the data is those only within the current class or others with the keyword "friend" have access to data within.

   (c) (5 points) Protected-Protected meands that only the current class and the subclassed of that specified class are able to access and of the protected fields or methods, unlike public which can be accessed anywhere throughout the program

   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.

---

7. (10 points) What is the diamond problem?
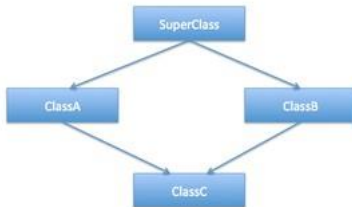
   Hint:

   - This is a question about multiple inheritance and its potential problems.

   - Use examples when possible, but explain thoroughly.

The diamond problem is when you have subclass inheriting from multiple parent classes also known as multiple inheritance. The problem arises when the an ambiguity is created from multiple parent classes that the child does not know which method to use or utilize. A solutions to fixing this problem is creating default methods and interfaces. This in turn allows the interfaces to utilizes the same default name with the same default name in two different interfaces to avoid issues with the multiple inheritance. Another solution that has been shown to solve this issue is Virtual inheritance By utilizing virtual inheritance, and ensurity can be made that ensures that the child class gets only a single instance of the common base class.Interestinly enough, I also found out that because of the issues that can arise from this, Java does not support multiple inheritance from more than one class to avoid issues that arise from this.

8. (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.

---

9. (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.

```cpp
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

Derived Obj; //create object from derived class named Obj
Obj.show();   // execute the show method inside of the Derived class
Obj.Base::show(); // execute the show method inside of base class this executes the show method in Base

---

(15 points) Given the two class definitions below:

```cpp
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 your write you Car definition (bare bones definition).

```
 1      class Engine
 2    ☐ {
 3    └ };
 4
 5    ☐ class Automobiles {
 6    └ };
 7
 8      //Derived class of the  Car
 9      class Car :public Automobiles
10    ☐ {
11      private:
12            //Member of Car
13            Engine engine;
14    └ };
15
16      //reasoning
17      //   Since car is an autombile, we inherit properties of automobiles
18      //   into the class car which has access to the public of auto
19      //   this shows inheritance
20
21      // every automobile contains an engine, so since this is the case, in the private section,
22      // the car is composed of all the components making up the car every car has an engine, showing
23      // an example of composition
```

the reasoning I wrote this down the way that I did is described inside of the code that I typed up in notepad plus plus which basically with this example, we are showing inheritance through the car class which has access to the public class of automobiles and it shows composition through accessing the engine class in the private section because all cars have engines.

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.) use static keyword

```
1    //class called living things
2    class LivingThings
3    {
4    private:
5            //number that born and lving
6            int numBorn;
7            int numLiving;
8
9    public:
10       //initialize static cariables for
11       //born objects and numliving
12       static int numOfBornObjects;
13       static int numOfLivingObjects;
14       //Constructor for class livingthings
15       LivingThings()//defaults
16       {
17           numBorn = numOfBornObjects;
18           numLiving = numOfLivingObjects;
19
20           numOfLivingObjects+= numOfBornObjects++;
21       }
22       //destructor for class
23       ~LivingThings()
24       {
25           //decrementing
26           numOfLivingObjects--;
27       }
28   };
29   //global static  incremented and decremented
30   int LivingThings::numOfBornObjects = 0;
31   int LivingThings::numOfLivingObjects = 0;
32
33   int main()
34   {
35       //creating objects from the class living things
36       LivingThings obj, obj1, obj2;
37       cout << "Number of Born Objects: " <<
38           obj2.numOfBornObjects << endl;
39       obj.~LivingThings();//call the destructor for obj
40       obj2.~LivingThings();//call the destructor of obj2
41       //print out the results of the number of living objects
42       cout << "Number of Living Objects: " <<
43           obj2.numOfLivingObjects << endl;
44   return 0;
45   }
```

Each time object is created and dies, the number count goes up and the numliving ins incremented and decremented through the class and the calls in main.

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 ca directly, wit need to create any object or call any class.

   SHOWN BELOW IN SCREENSHOT

```cpp
class Quad //create class named quad
{
protected:
        double height, width;//lengths
public:
        //methods
    //virtual default to area 0
    //will be overriden with rect class
    virtual double Area() = 0;
    //create a setter for comp area
    void setSides(double h, double w)
    {
      height = h;
      width = w;
    }
};
//below is the rectangle class implementing the quad class
//class rectangle has access to public of quad
class Rectangle :public Quad
{
//here we override the area methods
public:
    double Area() //default
    {
      return height * width;
    }
};

double getArea(Quad &quad)
{       //used in top level function in main
  return quad.Area();
}
//for testing implement in main
int main()
{
   //create obj rect derived from rectangle class
   Rectangle rect;
   //set sides
   rect.setSides(4, 5);
   //calling the  toplevel function
   cout << "Area of Rectangle: " <<
   getArea(rect) << endl;
   return 0;
}
```

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?

13. (10 points) What are the limitations of OOP?
There are pros and cons of OOP. Some of the pros of oop are a even though some of it may be tedious, implementation of OOP produces better productivity inside of the code itself. While some of the processes of using OOP may be tedious, in the long run it makes coding more efficient when dealing with the creation and implementation of code. It also helps in the cost of project implementation allow other users to easily share with one another.