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

Name: Paxton Proctor

READ THESE INSTRUCTIONS

- 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*.
- 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!
- I make every effort to create clear and understandable questions. You should do the same with your answers.
- Questions should be answered in order and clearly marked.
- Your name should be on each page, in the heading if possible.
- Place your PDF on GitHub (after you take the actual final) and in your assignments folder.
- 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	

Paxton Proctor

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:

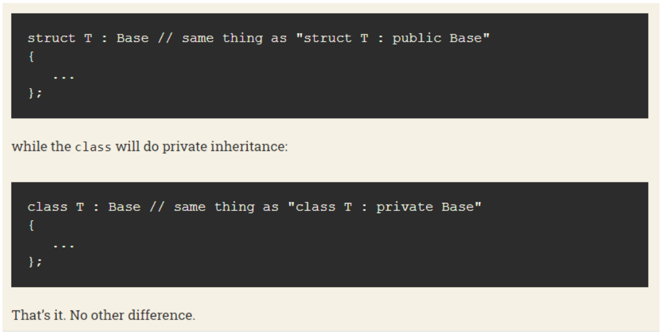
What is your name?: My name is Attila. This comes from the ancient figure "Attila the Hun". He was the leader of a tribal empire consisting of Huns, Ostrogoths, Alans and Bulgars, amongst others, in Central and Eastern Europe. My namesake almost conquered western Europe, but his brother died and he decided 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?

The differences in a class and a struct is the accessibility that each give. By default, member fields in **Classes** are *private*, whereas fields of a **Struct** are *public*. Technically structs can have constructors, methods, public, private, and protected members, use inheritance, and be templated just like a class.



```
struct T : Base // same thing as "struct T : public Base"
{
   ...
};
```

while the `class` will do private inheritance:

```
class T : Base // same thing as "class T : private Base"
{
   ...
};
```

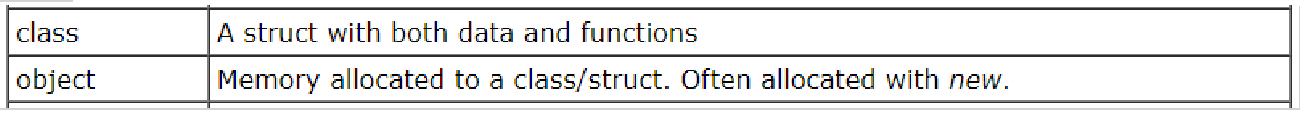
That's it. No other difference.

This article really helped clear things up: https://www.fluentcpp.com/2017/06/13/the-real-difference-between-struct-class/

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

A **Class** is an Abstract Data Type which includes methods and data members.

An **Object** is an instance of some Abstract Data Type. Which means it now resides in memory and has state.



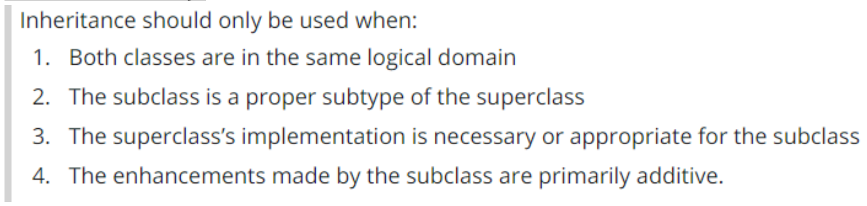
class	A struct with both data and functions
object	Memory allocated to a class/struct. Often allocated with *new*.

C++ notes: OOP Terminology: [C++ Notes: OOP Terminology (puc-rio.br)]

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

Inheritance is Defining a class A by basing it on class B

Composition is using instances of 1 or more classes as part of the definition of another class (like a data member).



Inheritance should only be used when:

1. Both classes are in the same logical domain
2. The subclass is a proper subtype of the superclass
3. The superclass's implementation is necessary or appropriate for the subclass
4. The enhancements made by the subclass are primarily additive.

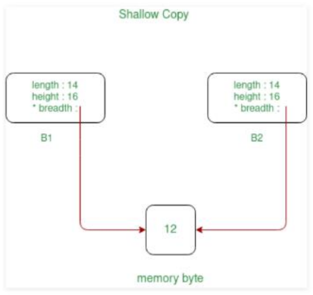
This article was a little annoying but definitely was able to clear up some things: https://www.thoughtworks.com/insights/blog/composition-vs-inheritance-how-choose

---

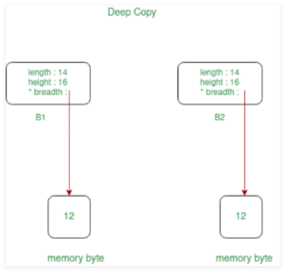
(d) (7 points) What is the difference between a *deep* vs a *shallow* copy? What can you do to make one or the other happen?

In shallow copy, an object is created by simply copying the data of all variables of the original object.

Shallow Copy of object if some variables are defined in heap memory, then:



In Deep copy, an object is created by copying data of all variables and it also allocates similar memory resources with the same value to the object.



To make one shallow or deep copy, deep copy is dynamically allocated memory.

Excellent Article: https://www.geeksforgeeks.org/shallow-copy-and-deep-copy-in-c/

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

Constructor is a special type of subroutine called to initialize an object whereas, **Destructor** cleans up allocated memory for a class.

You don't need to declare any of the special member functions explicitly, as long as you don't declare any special member functions that suppress the other ones you might want.

Article to read: https://www.geeksforgeeks.org/difference-between-constructor-and-destructor-in-c/

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

Static typed programming languages are those in which variables need not be defined before they're used. This implies that static typing has to do with the explicit declaration (or initialization) of variables before they're employed. *C++ is a Static type programming language*.
Dynamic typed programming languages are those languages in which variables must necessarily be defined before they are used. This implies that dynamic typed languages do not require the explicit declaration of the variables before they're used. *Python is an example of a dynamic typed programming language.*

An article Explaining typing: https://www.sitepoint.com/typing-versus-dynamic-typing/

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

Abstraction is the process or method of gaining the information.
While **encapsulation** is the process or method to contain the information.

The best example I can say really is Abstraction is the thought process, and Encapsulation is Real Implementation.

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

abstract class is used to define an implementation and is intended to be inherited from a class whereas, An **interface** class contains only a virtual destructor and pure virtual functions AND it has no implementation.

This Article seems okay but not great would definitely look for other examples and definitions: http://www.cplusplus.com/forum/beginner/157568/

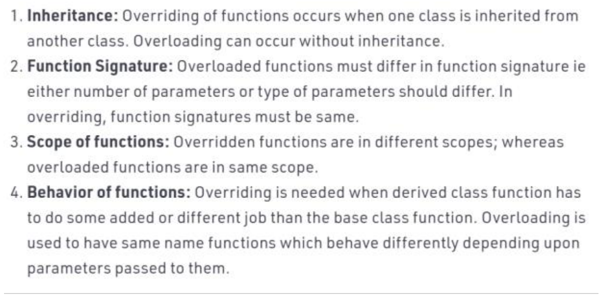
(i) (7 points) What is the difference between a *virtual function* and a *pure virtual function*?

A **virtual function** is a member function of base class which can be redefined by derived class.
A **pure virtual function** is a member function of base class whose only declaration is provided in base class and should be defined in derived class otherwise derived class also becomes abstract.

This Article is Excellent: https://www.geeksforgeeks.org/difference-between-virtual-function-and-pure-virtual-function-in-c/

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

The best way to show this would be in C++ code showing like the base class and the derived class



1. **Inheritance:** Overriding of functions occurs when one class is inherited from another class. Overloading can occur without inheritance.
2. **Function Signature:** Overloaded functions must differ in function signature ie either number of parameters or type of parameters should differ. In overriding, function signatures must be same.
3. **Scope of functions:** Overridden functions are in different scopes; whereas overloaded functions are in same scope.
4. **Behavior of functions:** Overriding is needed when derived class function has to do some added or different job than the base class function. Overloading is used to have same name functions which behave differently depending upon parameters passed to them.

The best article for examples and differences between overloading and overriding:

https://www.geeksforgeeks.org/function-overloading-vs-function-overriding-in-cpp/

---

2. Define the following and give examples of each:

(a) (5 points) Polymorphism

Has more than one form

Examples: Function overloading, operator overloading, function overriding, virtual functions.

(b) (5 points) Encapsulation

Packaging data and methods together in a single construct (ADT).

Example: The hierarchy of biological classification is something that would fit with the concept of inheritance. Top of the hierarchy are broad descriptions and classifications getting more detailed as you move down the ranks.

(c) (5 points) Abstraction

Abstraction is the process or method of gaining the information.

---

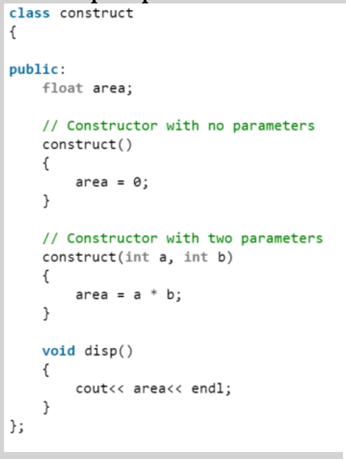
3. (a) (5 points) What is a default constructor?

A constructor without any arguments or with default value for every argument, is said to be ***default constructor***.

The compiler will implicitly *declare* default constructor if not provided by programmer, will *define* it when in need. Compiler defined default constructor is required to do certain initialization of class internals.

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

Very similar to function overloading, Overloaded constructors essentially have the same name (name of the class) and different number of arguments. Apparently there is but it shouldn't make a difference when people make there programs.

Examples: 

```
class construct
{

public:
    float area;

    // Constructor with no parameters
    construct()
    {
        area = 0;
    }

    // Constructor with two parameters
    construct(int a, int b)
    {
        area = a * b;
    }

    void disp()
    {
        cout<< area<< endl;
    }
};
```

This article has good definitions: https://www.geeksforgeeks.org/constructor-overloading-c/

(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 member function that initializes an object using another object of the same class. You do not need a copy constructor unless if it is dealing with deep copy which involves dynamic allocated memory.

(d) (5 points) What is a deep copy, and when do you need to worry about it?
In Deep copy, an object is created by copying data of all variables and it also allocates similar memory resources with the same value to the object.

You only need to worry about it when dealing with dynamic allocated memory.

(e) (5 points) Is there a relationship between copy constructors and deep copying?
The deep copying essentially uses copy constructors because deep copy is a copy constructor that creates dynamic memory for other constructors.

This Article: https://www.geeksforgeeks.org/shallow-copy-and-deep-copy-in-c/

(f) (5 points) Is a copy constructor the same as overloading the assignment operator?
Both the copy constructor and the overloading assignment operator are essentially the same in that they copy one object to another but the copy constructor creates new objects whereas, the assignment operator just replaces the contents of the existing objects.

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

Destructor cleans up allocated memory for a class. Which would mean that anything involving creating dynamic memory for a class will most likely need a destructor to prevent unnecessary memory from sitting in the heap.

---

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

abstract class is used to define an implementation and is intended to be inherited from a class whereas, An **interface** class contains only a virtual destructor and pure virtual functions AND it has no implementation.

This Article seems okay but not great would definitely look for other examples and definitions: http://www.cplusplus.com/forum/beginner/157568/

Hint:
You should include in your discussion:

- Virtual Functions
- Pure Virtual Functions

---

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

(a) (5 points) Public

If the class member declared as public then it can be accessed everywhere. This different than the Protected and Private in that Protected can be accessed only by inheriting or parenting and private that can only be accessed by the class that it is in.

(b) (5 points) Private

If the class members declared as private then it may only be accessed by the class that defines the member. Compared to public and protected in that they can be accessed from other classes through various means such as just being public or having inheriting or parent classes.

(c) (5 points) Protected

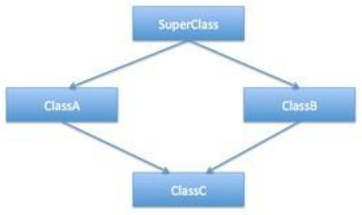
If the class members declared as protected then it can be accessed only within the class itself and by inheriting and parent classes. This is different compared to public for public can be accessed anywhere and private can only be accessed by the class that it is in.

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?
Multiple inheritance allows a child class to inherit from more than one parent class. The Problem arises when there is a ambiguity issue that essentially makes the child classes confused on which method it wants to use.
Virtual inheritance *solves* the classic “Diamond Problem”. It ensures that the child class gets only a single instance of the common base class.

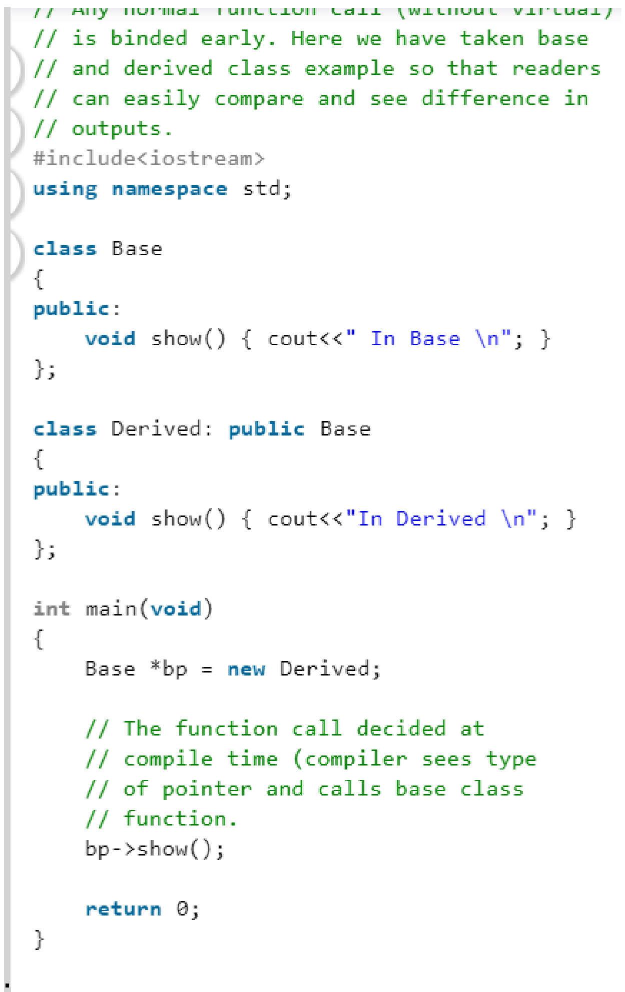
Article: https://www.freecodecamp.org/news/multiple-inheritance-in-c-and-the-diamond-problem-7c12a9ddbbec/



Hint:

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

7. (10 points) Discuss Early and Late binding.
Early Binding/static binding basically replaces the call with a machine language instruction that tells the mainframe to leap to the address of the function. This runs during the compile time and before the program runs



```
// Any normal function call (without virtual)
// is binded early. Here we have taken base
// and derived class example so that readers
// can easily compare and see difference in
// outputs.
#include<iostream>
using namespace std;

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

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

int main(void)
{
    Base *bp = new Derived;

    // The function call decided at
    // compile time (compiler sees type
    // of pointer and calls base class
    // function.
    bp->show();

    return 0;
}
```

Late binding/ dynamic binding is basically the compiler adds code that identifies the kind of object at runtime then matches the call with the right function. This can be achieved by declaring a virtual function.

```
// CPP Program to illustrate late binding
#include<iostream>
using namespace std;

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

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

int main(void)
{
    Base *bp = new Derived;
    bp->show();  // RUN-TIME POLYMORPHISM
    return 0;
}
```

Article: https://www.geeksforgeeks.org/early-binding-late-binding-c/

Hint:

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

---

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.

Derived Horriblework;
Horriblework.show();
Horriblework.Base::show();

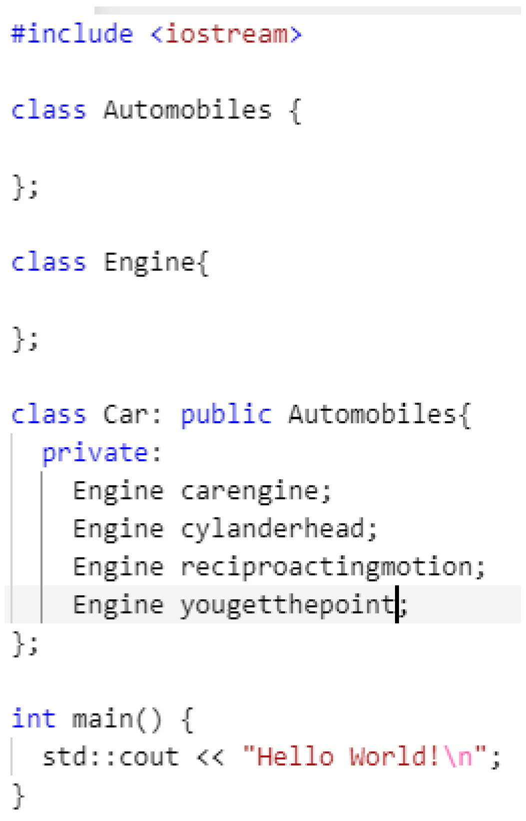
```
class Base{
        public:
        virtual void show() { cout<<" In Base n"; }
};

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.

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

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

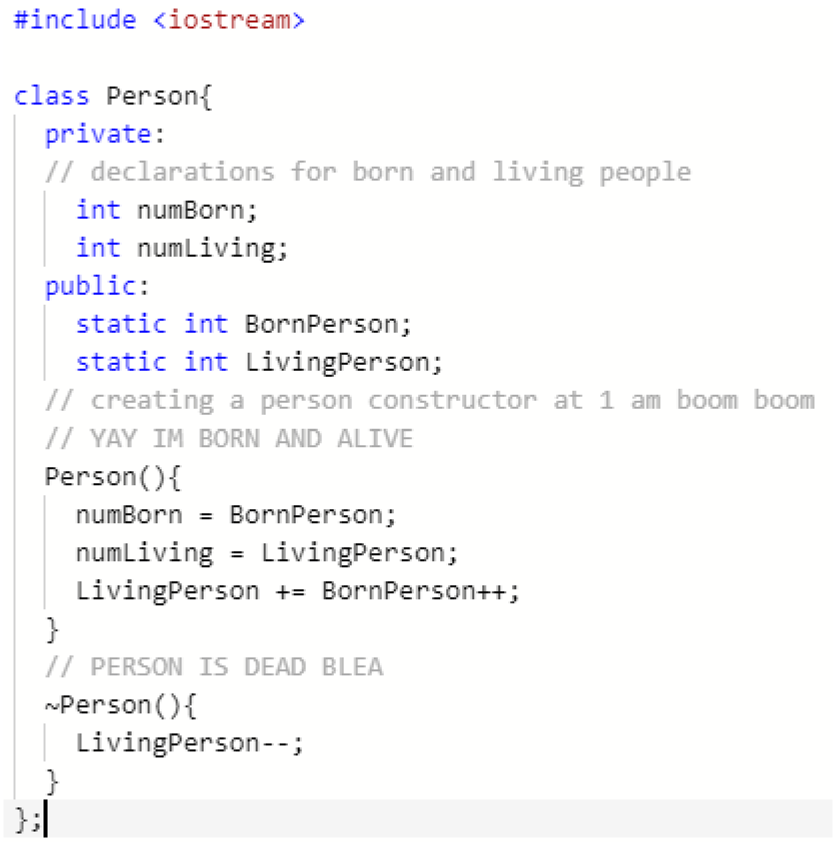
You need to write a definition for a Car class using the above two classes. You need to extend one, and use the other as a data member. This question boils down to composition vs inheritance. Explain your reasoning after your write you Car definition (bare bones definition).



```
#include <iostream>

class Automobiles {

};

class Engine{

};

class Car: public Automobiles{
  private:
    Engine carengine;
    Engine cylanderhead;
    Engine reciproactingmotion;
    Engine yougetthepoint;
};

int main() {
  std::cout << "Hello World!\n";
}
```

Since all cars are automobiles and we are using the public portion of automobiles then the that will illustrate inheritance since the car has public access to anything within automobile. All automobiles contain engines, brakes, wheels etc. which will show us a form of composition when using them as members.

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.)



```
#include <iostream>

class Person{
  private:
  // declarations for born and living people
    int numBorn;
    int numLiving;
  public:
    static int BornPerson;
    static int LivingPerson;
  // creating a person constructor at 1 am boom boom
  // YAY IM BORN AND ALIVE
  Person(){
    numBorn = BornPerson;
    numLiving = LivingPerson;
    LivingPerson += BornPerson++;
  }
  // PERSON IS DEAD BLEA
  ~Person(){
    LivingPerson--;
  }
};
```

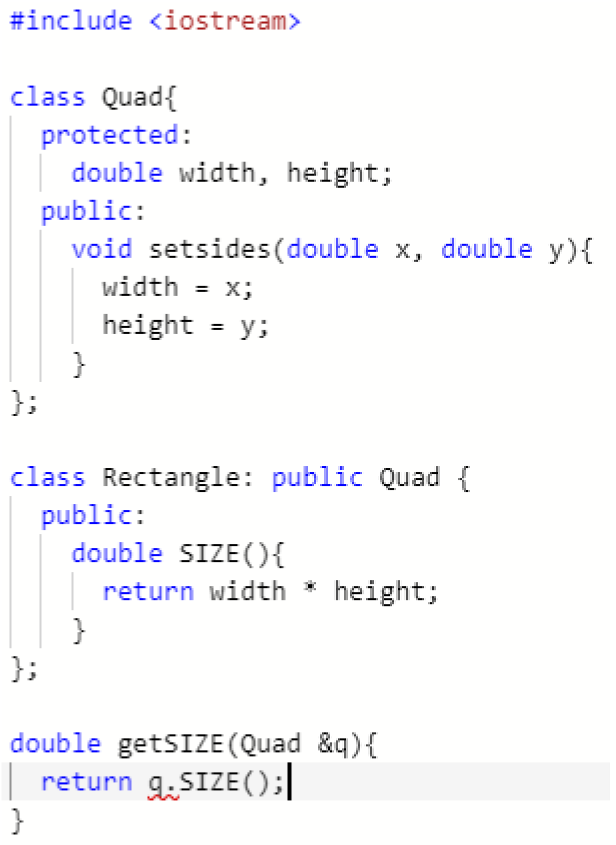
So, to sum things up each time a person otherwise known as object is created the count will go up and as the person dies or gets destroyed the count will go down which will influence the numlivings (P.S. Wasn't getting my main to work and I honestly don't know if my class is corrected so yeah…

---

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.



```
#include <iostream>

class Quad{
  protected:
    double width, height;
  public:
    void setsides(double x, double y){
      width = x;
      height = y;
    }
};

class Rectangle: public Quad {
  public:
    double SIZE(){
      return width * height;
    }
};

double getSIZE(Quad &q){
  return q.SIZE();
}
```

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.

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 requires, **Destructor** cleans up allocated memory for a class, A **copy constructor** is a member function that initializes an object using another object of the same class, and the **copy assignment operator** that just replaces the contents of the existing objects.

Article: https://www.geeksforgeeks.org/rule-of-three-in-cpp/

13. (10 points) What are the limitations of OOP?
Most of the articles that I have been reading tend to say size is a big issue due to duplications and implications using only the CPU. They also say that the speed can also become an issue since the size of these programs can get pretty hefty. Last but not least it can be very hard to code and understand. It also requires a lot of work and since attempting to learn it while I do enjoy learning it I can see a lot of frustration in with it especially where differencing Encapsulation and Abstraction is concerned.