SHAROME BURTON

2143 - OOP
Spring 2021
Take Home Exam
April 22, 2021

Name: Sharome Burton

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?

   Classes are more secure than structs, and the members of class are private by default and struct are public by default. Classes are stored in the heap while struct are stored in the stack. Classes are reference type while struct are value type. Classes provide re-usability by using inheritance and provides flexible combining data and methods. Default access to specifier is private in a class while struct default access to specifier is public.

   (b) (7 points) What is the difference between a *class* and an *object*? A class is a blueprint from which you can create the instance, i.e., objects. An object is the instance of the class. A class also describes object behavior. An object is a member or an "instance" of a class. A class is a definition of an abstract data type that includes methods and data members. An object is an instance of a class that has been declared.

   (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 involves creating a new class (derived class) from an existing one (base class). With composition, a class is defined using other classes as components. Using inheritance or composition in practical applications depends on the nature of the problem at hand. When a class could be described as a simple subset of another class, inheritance may be preferred. When a class is simply a component of another class eg. an engine is a component of a car class, then composition should probably be used.

   (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 a shallow copy, an object is created by simply copying the data of all variables of the original object. This works well if none of the variables of the object are defined in the heap section of memory, otherwise Since both objects will reference the same memory location, then a change made by one will reflect those changes in another object as well. In order to create a copy of the object where variables have been allocated dynamically, a deep copy is required. Compilers like that of C++ implicitly create copy constructors and overloaded assignment operators that allow

for shallow copies. To make a deep copy happen, we need to define a copy constructor ourselves to assign dynamic memory to the variables.

(e) (7 points) What is the difference between a *constructor* and a *destructor*? Are they both mandatory or even necessary? A constructor is a member function of a class that has the same name as the class name. It helps to initialize the object of a class. It can either accept the arguments or not. It is used to allocate the memory to an object of the class. It is called whenever an instance of the class is created.. Destructors help to deallocate the memory of an object. It is called while object of the class is freed or deleted. In a class, there is always a single destructor without any parameters so it can't be overloaded.

(f) (7 points) What is *static* vs *dynamic* typing? Which does C++ employ and which does Python employ?
Dynamically typed languages do type checking of variables only as code runs, and the type of a variable is allowed to change over its lifetime. Static type checks are performed without running the program. The type of a variable is not allowed to change over its lifetime. C++ employs static variables, while Python employs dynamic variables.

(g) (7 points) What is *encapsulation* vs *abstraction*? Please give some examples!
Abstraction's main goal is to handle complexity by hiding unnecessary details from the user. That enables the user to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity. Encapsulation describes the idea of bundling data and methods that work on that data within one unit. This concept is also often used to hide the internal representation, or state, of an object from the outside. This is called information hiding. The general idea of this mechanism is simple. If you have an attribute that is not visible from the outside of an object, and bundle it with methods that provide read or write access to it, then you can hide specific information and control access to the internal state of the object.

(h) (7 points) What is the difference between an *abstract class* and an *interface*?
Both abstract classes and interfaces are used for abstraction, however, a major difference is that Interfaces involve *pure* virtual functions (which are not implemented) while abstract classes may define regular virtual functions. A virtual function is one that is declared within a base class and is overwritten by a derived class while a pure virtual function is a virtual function that does not have an implementation. Classes of virtual functions are not abstract while classes of pure virtual functions turn abstract.
Interfaces can allow for multiple inheritance while abstract classes cannot. A class may extend only one abstract class, but it can benefit from multiple interfaces. An abstract class is a special type of class that cannot be instantiated. An abstract class is designed to be inherited by subclasses that either implement or override its methods. In other words, abstract classes are either partially implemented or not implemented at all. On the other hand, interfaces don't have any

implementation. An interface can contain only method declarations; it cannot contain method definitions. There also cannot be any member data in an interface. Whereas an abstract class may contain method definitions, fields, and constructors, an interface may only have declarations of methods, and properties. Methods declared in an interface must be implemented by the classes that implement the interface.

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

A virtual function is one that is declared within a base class and is overwritten by a derived class while a pure virtual function is a virtual function that does not have an implementation. Classes of virtual functions are not abstract while classes of pure functions turn abstract.

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

Function overloading is multiple definitions of the same function using different parameters. Function overriding is the redefining of a function with the same parameters. Overriding of functions can happen when one class is inherited from another class. Overloading can occur without inheritance. Overloaded functions must have different parameters. In overriding, function parameters are the same.

---

2. Define the following and give examples of each :

(a) (5 points) Polymorphism

Polymorphism is when many classes that are related to each other by inheritance or customizing functions or methods. Polymorphism is taking up more than one form. Examples of this are the overloading of functions and operators, the overriding of functions and the definition of virtual functions. Any example of this is a class called Animal has a method called greet(). Then other classes derived from Animal for example lion or chicken will be able to use their own version of greet() through the use of virtual functions for example.

(b) (5 points) Encapsulation

Encapsulation is basically bundling data into one unit. This concept is also often used to hide the internal representation, or state, of an object from the outside. This is called information hiding. The general idea of this mechanism is simple. If you have an attribute that is not visible from the outside of an object, and bundle it with methods that provide read or write access to it, then you can hide specific information and control access to the internal state of the object. An example of that is in the human resource office there are many departments. Within department there is someone in charge. If a

head of department wants to get info from another department, they must go to another head to get it.

(c) (5 points) Abstraction

Abstraction is only displaying the data that is useful to the user and hiding the rest. Its main goal is to handle complexity by hiding unnecessary details from the user. That enables the user to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity. For example, when you use a blender all you do is place the contents into it and switch it on. You do not know how exactly the motors in the blender work for or how long it remains working before it stops

---

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

A default constructor is a member function of a class which has the same name as the class, that can be called with no arguments, and is used to instantiate an object of a class using default values (or parameters). Default constructors are called when an object is instantiated without any parameters in the parameter list at that point in the program.

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

An overloaded constructor is a member function of a class that is used to instantiate an object of that class using a different number of parameters than the default constructor. The number and type of parameters that are used to instantiate an object determine which constructor is called. For example, while a person class can be instantiated without passing any arguments, passing the int age = 23 and char sex = 'F' arguments into the call to create the person object will call the overloaded constructor that will use those two arguments to instantiate an object with those two data members in their definition.

(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 used to copy the contents from one objects to another object of the same class. We do not need a copy constructor for every class. We only need one when there is dynamic memory involved and there needs to be a deep copy done.

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

A deep copy is a way of creating an exact replica of the object having the same literal value, data type, and resources. A deep copy is required if the variables of an object have been dynamically allocated, in order to create a copy of the object.

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

Programmer-defined copy constructors are required to allow for deep copying. This is because the programmer needs to create a constructor that dynamically allocates memory for the member variables in order to allow deep copying of an object.

(f) (5 points) Is a copy constructor the same as overloading the assignment operator?
The purpose of the copy constructor and the assignment operator are almost the same except that the copy constructor initializes new objects while the assignment operator replaces the contents of existing objects.

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

One major reason a class needs a destructor is dynamic memory management or 'garbage collecting'. When objects are instantiated, they take up memory until they are destroyed, which unallocated memory that was used for the object. Without destructors, memory from objects that are not being used and have not been destructed is still taken up, which can cause occupation of the computer's resources, leading to a slowdown in performance.

Classes may also need destructors if the programmer intends to keep track of how many objects have existed/have been destroyed during runtime.

---

4. (10 points) What is the difference between an abstract class and an interface?
Both abstract classes and interfaces are used for abstraction, however, a major difference is that Interfaces involve *pure* virtual functions (which are not implemented) while abstract classes may define regular virtual functions. A virtual function is one that is declared within a base class and is overwritten by a derived class while a pure virtual function is a virtual function that does not have an implementation. Classes of virtual functions are not abstract while classes of pure virtual functions turn abstract.
Interfaces can allow for multiple inheritance while abstract classes cannot. A class may extend only one abstract class, but it can benefit from multiple interfaces. An abstract class is a special type of class that cannot be instantiated. An abstract class is designed to be inherited by subclasses that either implement or override its methods. In other words, abstract classes are either partially implemented or not implemented at all. On the other hand, interfaces don't have any implementation. An interface can contain only method declarations; it cannot contain method definitions. There also cannot be any member data in an interface. Whereas an abstract class may contain method definitions, fields, and constructors, an interface may only have declarations of methods, and properties. Methods declared in an interface must be implemented by the classes that implement the interface.

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

   The term public means that a class's member variables and functions can be accessed from outside the class.

   (b) (5 points) Private

   The term private means that a class's member variables and functions cannot be accessed from outside the class.

```
// example code for private and public
class Example {
  public:    // Public access
    int x;
  private:   // Private access
    int y;
};

int main() {
  Example Obj;
  Obj.x = 3;  // Allowed access (public)
  Obj.y = 4;  // Not allowed access (private)
  return 0;
}
```

   (c) (5 points) Protected

   The term protected means that members cannot be accessed from outside the class but they can be accessed in inherited classes.

Hint:

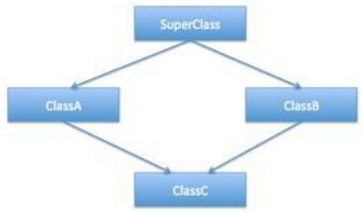
- Make sure you define each item individually as well.
- Use examples.
- If your not sure, use examples to make your point.
- Ummm, example code is always welcome.

---

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

   The diamond problem occurs when two derived classes have a common parent class, and in turn, those classes are the parents to a derived class. This causes an issue as the two derived classes may

have data members of the same type but different values. Thus when the derived class tries to retrieve information from the two parent classes an error may occur.
An example of the diamond problem is at a school, a person(base class) will have a name and an age. He will also be a student (derived class) and belong to a faculty(derived class). The problem comes around if the student is also a custodian(new derived class), which also requires a name and age. At this point, the custodian would be defined with those pieces of information twice. This causes an issue – which is the result of the diamond problem.



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 occurs when the compiler maps out what function call goes to which function and turns that communication into machine code. Late binding happens during runtime when the compiler adds code that identifies the kind of object it then maps in machine code the object to the function call. By default (in C++), early binding takes place, where a function call is made and executed. By this time, the compiler has already gone through the code and has mapped the calls to the function.

Since virtual functions are altered at runtime, they are used for late binding. This is mainly because the corrected version of the virtual function must be selected first then the mapping from object to function can be done in machine code.

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.

```
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.

Base* baseptr; // pointer to Base class

baseptr.show(); // displays contents of show function from Base class

Derived d; // object from Derived class

baseptr = &d; // makes the Base pointer equal to that of the address of the Derived object.

baseptr.show(); // displays contents of show function from Derived class

---

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).

class Automobile{

public:

string type = “Sedan”;

};

class engine{

public:

int date;

```
    int numCylinders = 6;

    string type = "V6 Turbo";

    Engine(){date = 2021;}

};

class Car: public Automobile{

    public:

    Engine CarEngine;

    string gearing = "Manual";

    int seats;

    Car(int num_seats){seats = num_seats;}

};
```

I decided to go with composition in this case because it seemed easier to implement and was a more intuitive solution. A car is by definition *composed* of an engine in the real world, so using composition to have an engine object as a member variable of the car class makes most sense, especially since composing the Car class with an engine was as simple as declaring an engine object as a public member variable of the car class.

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.)

```
class LivingObjects{

        public:
        static int numBorn=0;
        int numLiving;

        LivingObjects(){numBorn++;};

        static void total(){
                numLiving = numBorn;
                return numLiving;
};
```

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.

```
class Quad {
        public:
        int length1;
        int width1;
        int length2;
        int width2;
        virtual void setlength1(int L){length1 = L;};
        virtual void setlength2(int L){length2 = L;};
        virtual void setwidth1(int w){width1 = w;};
        virtual void setwidth2(int w){width2 = w;};

        virtual int Area()
        {area = length1 *width1;
        return area;
        };

};
```

(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.

```
class Rectangle: public Quad{
    public:
        Rectangle(){length1 = 1; width1 =2; length2 = length1; width2=width1;}

        void setlength1(int L){length1 = L; length2 = L;};};
        void setwidth1(int w){width1 = w; width2 = w;};};

        int Area(){
            area = length1 *width1;
            return area;
        };
};
Int main(){
```

```
Rectangle A;
A.setlength1(3);
A.setwidth1(2);

Return 0;
}
```

(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.

---

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 in a concept that suggests that if a class defines any of the following (destructor, copy constructor, copy assignment operator) then it should probably explicitly define all three. When there is no copy constructor and assignment operator for an object, the destructor runs only once to destroy an object. However there is a copy constructor and assignment operator the objects are altered via deep copy. Thus there are two copies of the same value in the objects. This causes the destructor to run twice.

---

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

OOP can be inefficient in that the processing power used to run the code places too much strain on the CPU as opposed to other code standards.

OOP can be too bulky by using excess lines of code that could be used more efficiently. Unnecessary code can be accumulated if OOP is left unchecked. This waste space making it difficult to keep time and cost down of programming down.

OOP projects are easier to design than to implement. This is because these classes are flexible in application. Projects may be up and running quickly but there is a large possibility that the project may seem cloned.

BIBLIOGRAPHY

Alan Hinchcliffe (n.d.). When should you use a class vs a struct in C ? Retrieved from https://stackoverflow.com/questions/54585/when-should-you-use-a-class-vs-a-struct-in-c

Au, E. (2020, April 04). Static Typing in Python. Retrieved April 26, 2021, from https://towardsdatascience.com/static-typing-in-python-55aa6dfe61b4

BestProgISch. (2018, November 25). Overloading of methods in classes. Examples The advantage of methods overloading. Constructors overloading. Retrieved April 26, 2021, from https://www.bestprog.net/en/2018/11/25/overloading-the-methods-in-classes-examples-the-advantage-of-methods-overloading-constructors-overloading/

Constructor Overloading in C. (2018, September 03). Retrieved April 26, 2021, from https://www.geeksforgeeks.org/constructor-overloading-c/

Default constructors. (n.d.). Retrieved April 26, 2021, from https://en.cppreference.com/w/cpp/language/default_constructor

Difference between Abstract class and Interface - Javatpoint. (n.d.). Retrieved April 26, 2021, from https://www.javatpoint.com/difference-between-abstract-class-and-interface

Difference between Abstract Class and Interface in Java. (2021, March 31). Retrieved April 26, 2021, from https://www.geeksforgeeks.org/difference-between-abstract-class-and-interface-in-java/

Dynamic typing vs. Static typing. (n.d.). Retrieved April 26, 2021, from https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html

Griffin, T. R. (n.d.). Test 1 - OOP Concepts. Retrieved April 26, 2021, from https://github.com/rugbyprof/2143-Object-Oriented-Programming/tree/master/Assignments/06-T01

Kanjilal, J. (2018, January 01). When to use an abstract class vs. interface in C#. Retrieved April 26, 2021, from https://www.infoworld.com/article/2928719/when-to-use-an-abstract-class-vs-interface-in-csharp.html

Shallow Copy and Deep Copy in C. (2020, December 29). Retrieved April 26, 2021, from https://www.geeksforgeeks.org/shallow-copy-and-deep-copy-in-c/

The rule of three/five/zero. (n.d.). Retrieved April 26, 2021, from https://en.cppreference.com/w/cpp/language/rule_of_three