Unit 3 Study Guide and Review: Object-Oriented Programming

3a. Define and compare/contrast constructors and deconstructors

1. What is object-oriented programming?

   OOP allows programmers to pack away details into neat, self-contained boxes (objects) so that they can think of the objects more abstractly and focus on the interactions between them. There are lots of definitions for OOP, but 3 primary features of it are: • Encapsulation: grouping related data and functions together as objects and defining an interface to those objects • Inheritance: allowing code to be reused between related types • Polymorphism: allowing a value to be one of several types, and determining at runtime which functions to call on it based on its type

2. How do you create a constructor and when is it deployed?
   a. A constructor is used to create an instance of the defined object and reserve memory space for said object. A constructor is most typically used within a main function or within a method that is part of the same class definition. Anytime a constructor is created, it must also be destroyed using the destructor within the same location that it was created. However, the destructor in C++ is automatically implemented when the instance of the variable is no longer in effect.
3. Explain what it means to create an overloaded constructor?
   a. Every constructor has same name as class name but they differ in terms of either number of arguments or the datatypes of the arguments or the both. As there is more than one constructor in class it is also called **multiple constructor**.
4. How do you create a destructor and when is it deployed?
   a. A **destructor** is called for a class object when that object passes out of scope or is explicitly deleted. A **destructor** is a member function with the same name as its class prefixed by a ~ (tilde)

An explanation of object-oriented programming and the concepts that accompany it, including encapsulation, inheritance, abstraction, and polymorphism are explained in these lecture notes. Object-oriented programming allows a programmer to manage the complexity of a program by grouping code into self-contained boxes (classes). Constructors are a way to “call” an instance of a class. You define the class within a class file, and then create an instance of the class, usually within the main program, although it may be created within other methods. The creation of constructors and destructors can be reviewed if you watch this video.

3b. Create overloading operators

1. What is an overloaded operator and why is it used?

Overloaded operators allow you to redefine the meaning and purpose of an operator. Imagine a world where == will have a different behavior depending on the values that are sent to it. Although you may have the capability to overload operators, it is important that you use this sparingly, as it tends to create ambiguity in your program, as operators are not behaving as they are expected. To learn more, explore this article

3c. Define and use the keyword “this” and use the static members appropriately

1. What is the keyword “this” used for?

“this” is a keyword that is used to indicate the specific instance of an object. It is required to be used when the private data members have the same name as the value being passed. For example, you would use:
this.name = name; ‘this’ indicates to use this particular instance of the class. This video explores the use of the keyword “this”.

1. When are static members used?

A private data member that is defined as static retains its value through different instances of the class. Essentially, it means that it holds a separate place in memory than the other data members of the class, and the values of that variable are shared by all other members of the class. Because it is static, no matter which instance of the class accesses the data, the same value will be retrieved.

Static members are used when multiple instances of a class are going to need access to the same data values, and this article explores the use of static members. For example, if you create a data member as static, every instance of the class will refer to the same value in member, instead of having their own variable value. An example of when you might want to use this is if you are wanting to keep a count of the number of instances of the class that occurs within a program.

3d. Design and appropriately use friend functions and classes

1. What are friend functions and classes?

    Inheritance allows us to define "is-a" relationships, but it cannot be used to implement "has-a" relationships. In order for an inheritance relationship to occur, a class must have an "is-a" relationship with the inherited class. Friends of the class are not members, although they are defined within the class, and do not have an is-a relationship. However, they do have a has-a relationship. Methods that are friends are defined as a friend in the method header, which then allows a method to be

accessed directly from outside the class. A friend method is not a member of the class.

A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions. A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends. For more information on friends, review this article.

3e. Use the class inheritance to design better code

1. What is inheritance and how is it implemented?

The idea of inheritance is that you identify those characteristics of an object that are common among all types of that object and create a class that shares these characteristics (the parent class). Then you create individual classes (child classes) to represent each specific type that identifies those characteristics that are unique to this type. These individual classes inherit the characteristics of the parent class. To learn more about inheritance, review these slides.

Inheritance types include Single inheritance, Multiple inheritance, Hierarchical inheritance, Multilevel inheritance and Hybrid inheritance. Single inheritance inherits properties and behaviors of only one base class. Multiple inheritance occurs when a derived class inherits properties and behaviors of more than one base class. Hierarchical inheritance occurs when the properties and behaviors are inherited by more than one derived class. Multilevel inheritance occurs when properties and behaviors of a derived class are inherited by another class, resulting in a chain of inheritance

3f. Explain how polymorphism is achieved through C++ code

1. What is polymorphism?

   Polymorphism is the ability to take on more than one form. A class can be used through its parent interface while a child class may override some of the parent's methods the abstract methods. If a method is abstract in the parent class, then the child class can develop their own individual method that overrides the parent.

2. How is polymorphism implemented?

Polymorphism is a way to implement multiple versions of the same method that are part of difference class definitions. When a child encounters a method that is part of their own class and a part of the parent class, the child class will implement their version of the method within their own class. It allows for invoking abstract operation. If the method is declared as abstract or virtual inside of the parent class interface it may be overridden by

child methods. Another way to implement it is through function overloading, which offers different behavior depending on the number and type of arguments that are passed to the function. To learn more information on this, review this article.

Unit 3 Vocabulary

This vocabulary list includes terms that might help you with the review items above and some terms you should be familiar with to be successful in completing the final exam for the course.

Try to think of the reason why each term is included.

- Abstraction
- Class
- Constructor
- Destructor
- Encapsulation
- Friend function
- Inheritance
- Object-oriented programming
- Overloading
- Polymorphism
- Static members
- This