Some random reviews

- Why do we have 2 overloads for operator[]?
- What is the difference between a shallow copy and a deep copy?
  - Which happens by default?
- When is the copy constructor implicitly called on an object?
- Why is the parameter to the copy constructor passed by const reference?
  - What would happen if we passed by value?
- What are the differences between operator= and the copy constructor?
  - What does operator=() return?
Inheritance
Introduction

Relation among objects: the “has a” relationship

- PlayList “has a” Song
- PokerHand “has a” Card

How does C++ capture the “has a” relationship?
Another common relationship between objects is the “is-a” relationship.

- A FSU CS student is a FSU student
- A FSU student is a student.

Need a way to capture this relationship.
Introduction

The “inheritance” in C++ allows programmers to define the “is-a” relationship.

- A class can be defined as a derived class of another class (called based class).
- The derived class then becomes a type/kind of the base class.
  - An object of a derived class “is an” object of the base class.
- Another way of thinking of this is that the derived class is everything the base class is and (possibly) more.
- This allows us to use the derived class object as though it were a base class object in certain scenarios.
Inheritance, base and derived class.

The “is a” relationship is realized through Inheritance using the following declaration syntax, which means a derivedClass object is a baseClass object (can use all functions in the base class interface).

```cpp
class derivedClassName : public baseClassName
```

class Mammal {
    public:
        void PrintInfo() const;
};

class Cow: public Mammal {   // Cow is a derived class of Mammal
    public:
        void sound() const;
};

Cow xyz;
xyz.PrintInfo();  // any Cow object can call functions in both base class and derived class

See sample1.cpp
Protection Levels

A derived class can access to all public members in the base class, but not the private members.

- A derived class is however in a way different from the “public”.
- We might want to allow some members to be accessed only by derived classes, but not the general public.
  - A new protection level: **protected**.

A summary of the three protection levels:

**public** - Members that can be accessed by name from within any function.

**private** - Members that can only be accessed within member and friend functions of the class in which they are declared.

**protected** - Members that can only be accessed within member and friend functions of the class in which they are declared **AND** from within derived class's member and friend functions.

See sample2.cpp
Constructors/Destructors

Since a derived class instance (object) is also a base class instance, both the base and derived class constructors must run when a derived object is instantiated.

Constructors run from most general (most base) to most derived.

Destructors run from most specific (most derived) to most base.

See sample3.cpp

What about constructors with parameters?

Control with initialization list

See sample4.cpp
Function Overriding

One of the most useful features Inheritance include and allows us to customize the behavior of derived objects.

A derived class can declare it's own version of a function declared in the base class from which it inherits.

The derived class version of the function will supersede (override) the version in the base class.

See sample5.cpp

We can even still access the original base class version of the function through an explicit call.

This is because a derived class must, by definition, be everything the base class is and (possibly) more.

See sample6.cpp