Review

- 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

Recall that the composition relationship between classes (object/classes within object/classes) is informally referred to as the “has a” relationship.

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

C++ allows us to define another (more intimate) relationship between classes informally known as the “is a” relationship.

The “is a” relationship allows a programmer to define a ***derived*** class in terms of a ***base*** class.

- The derived class then becomes a type/kind 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 Basics

The “is a” relationship is realized though **Inheritance** using the following declaration syntax:

class derivedClassName : public baseClassName

Eg.

class Checking : public Account

class Baseball : public Sport

class Car : public Vehicle

class Honda : public Car

See sample1.cpp

Protection Levels

Private members of a base class are not accessible from a derived class that inherits it.

Since it may be convenient to allow some members we wish to hide from the public interface of a class to be accessible within derived classes, we introduce a new protection level called ***protected***.

Here is a summary:

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?

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 base 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