13. Inheritance and Object Oriented Design

- **What is it?**
  - You’ve used it, often without knowing it.
    - Inheritance Hierarchy for Streams
      - istream: istrstream, ifstream, (cin)
    - Somewhat more systematic look
  - Way of classifying a set of objects by similarities and differences
  - Way of "extending" another class
  - Derived class inherits data and member functions of base class
  - Must use pointers (or references) to use effectively

- **"Is a" Relationship**
  - circle is a shape
  - square is a shape
  - ??? is a shape

- **Base Class Syntax**
  - No different from classes we have seen so far
  - Thus, *any class can be a base class*
  - May declare some member data protected
    - Data available to derived (sub) class
    - Do not need to make it public
  - May declare some member functions virtual
    - Allows dynamic (run-time) binding
    - Otherwise uses static (compile-time) binding

```cpp
class base // simplest situation
{
public:
    constructor, destructor
    public members
virtual public members
protected:
    data accessible to derived classes

private:
    data members
    private members functions
}
```

- **Whimsical Example: Students at a University**
  - school.cc
  - student.h, student.cc
- Derived Class Syntax
  - Need to declare which class is base class
  - Can access public and protected members of base class
  - Can add new member functions and member variables
  - Thus, only need to add or override things that are different

```cpp
class derived: public base
{
    any members not listed are derived unchanged
    (except constructor, destructor, copy constructor,
     operator= )

public:
    constructor, destructor if defaults are not good
    base members to be changed
    new public members

protected:
    possible data for classes using this as base

private:
    additional data members
    additional private members functions
    base member functions to be disabled
}
```

- Overriding Member Functions
  - Member functions can be overridden by derived class
  - Which member function called depends on instance’s class
  - Which member function called is delayed until runtime for virtual member functions
  - Most specific member function is called
  - Derived Class member functions are more specific

- The Payoff
  - Can add functionality later without changing working code
  - New implementation must conform to same interface