More About Inheritance

<u>Composition</u>

- When a class **"has a"** something, just make that thing a **member variable**
- <u>Ex</u>: DOG has a breed, VEHICLE has an engine, PERSON has a name

<u>Public Inheritance</u>

- When a class **"is a"** something else (which is more general), make that class a **public subclass** of the more general class
- <u>Ex</u>: DOG is a PET, MANAGER is an EMPLOYEE, BAG is a COLLECTION

<u>Private and Protected Inheritance</u>

- "Promotes" access levels
- **Protected** inheritance: **protected** will be treated like **private**, **public** treated like **protected**
- **Private** inheritance: everything treated like **private**
- Represents an **"is implemented in terms of"** relationship
- Forces derived classes to implement their own versions of functions, while still giving them info about what they *should* define

<u>Virtual Functions</u>

- Declaring a member function as **virtual** tells the compiler **"I don't know how this function is implemented. Wait until it is used in a program."**
- **Late (or dynamic) binding**: waiting until runtime to determine implementation
- If a function will have a **different definition in a derived class** than in the base class, then declare it as **virtual** in the base class
- **Polymorphism**: ability to associate multiple meanings to one function name (e.g., late binding, virtual functions)
- **Overriding**: when a **virtual** function definition is changed in a derived class
- **Redefining**: when a **non-virtual** function is changed in a derived class
- Overriding and redefining treated differently by the compiler!

<u>Pure Virtual Function</u>

- Class doesn't provide **<u>any</u>** implementation; just has **= 0** before **;** in prototype
- If any pure virtual function, class is **abstract**; can't declare an instance of it

Example

```
// SALE is superclass (also called base class)
class SALE {
protected:
        double price;

public:
        SALE() : price(0) { }
        SALE(const double p) : price(p) { }

        virtual double bill( ) const { return(price); }

        double savings(const SALE& other) const {
                return(bill() - other.bill());
        }

        bool operator < (const SALE& other) {
                return(bill() < other.bill());
        }
};

// DISCOUNT_SALE is subclass of SALE (also called derived class)
class DISCOUNT_SALE : public SALE {
private:
        double discount;

public:
        DISCOUNT_SALE(): SALE( ), discount(0) { }
        DISCOUNT_SALE(const double p, const double d): SALE(p), discount(d) { }

        virtual double bill() const {  // Note: 'virtual' not required here
                double fraction = discount /100;
                return((1 - fraction) * price);
        }
};

int main() {
SALE s(10.00);  // one item at $10
DISCOUNT_SALE d(11.00, 10); // one item at $11 with 10% discount

  if (d < s) {
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << "Discounted item is cheaper.\n";
    cout << "Savings is $" << s.savings(d) << endl;
  }
  else cout << "Discounted item is not cheaper.\n";

  return(0);
}
```