Introduction to C++

Pointers and Inheritance

Pointers and Inheritance

- **A pointer to a derived class can be stored as a pointer to the base class instead**
  - Respects the "is a" relationship
  - Vital to Liskov substitutability
- **Any base class function can then be called through the pointer**
  - Virtual function – derived class function executes
  - Nonvirtual function – base class function executes
  - This is C++ - you get to choose
- **Can't put a base class pointer into a derived class pointer**
  - Some member variables will be missing in the base class
- **Same rules when using smart pointers**
  - Smart pointers act like regular pointers and that includes polymorphism

Slicing

- **If you copy objects around, slicing can occur**
  - Copy a derived object into a base object – extra member variables fall away
  - Can't copy a base object into a derived object
- **Same rules apply when passing to a function by value**
  - A copy is made
  - Slicing will happen
- **Use references or pointers to avoid slicing**
  - References use same syntax as solid objects

Cast Operators

- **(type)**
  - C style cast
  - Super dangerous, Doesn't tell humans much when they read your code
- **static_cast<type>**
  - Compile time only
  - Up to you to be sure it's safe
- **dynamic_cast<type>**
  - Runtime check
  - Works only when casting to pointer to a class with a virtual table
  - Returns null if cast fails
  - Slower but safer
- **const_cast**
  - For casting away const (not a beginner technique)
- **reinterpret_cast**
  - For bit twiddling

Summary

- **Polymorphism lets you write general code that relies on specific implementations**
  - Update all the accounts, ship all the orders, pay all the employees
- **Raw pointers, smart pointers, and references all support polymorphism**
- **Copying solid objects derived to base can cause slicing**
- **Cast operators give you more safety and expressiveness**