Static vs dynamic binding, keywords

- with inheritance we might have multiple methods with the same name declared at different points amongst the base and derived classes
- which one gets called in a given situation depends on how we make the call and how the inheritance has been set up
- need to introduce static vs dynamic binding
- will use a variety of keywords: virtual, override, final
- tomorrow we'll get deeper into dynamic binding, pure virtual methods, and abstract base classes

Assigning ancestors/descendants

- suppose child is derived from parent, and have variables child C and parent P
- P = C; treated as valid, knows how to fill P's fields, ignores the extras
- C = P; treated as invalid (unless we overload = in child class) since it doesn't know how to initialize some of C

Passing child to parent* as param

- suppose function expects a parameter that is a pointer to the parent class
- we can still pass a pointer to a child class derived from parent, and call methods inside the function
- with our syntax so far, only the parent methods can get called in the function (since it's a pointer to a parent)
- called static binding: the compiler determines the method type based solely on what kind of object/pointer the method is called through

static binding example

```
class parent {
   protected:
      int pVal;
   public:
      parent(int v = -1) { pVal = v; }
      void print() {
          cout << “pVal is “ << pVal << endl;
      }
};
class child: public parent {
   protected:
      int cVal;
   public:
      parent(int v = -2) { cVal = v; }
      void print() {
          cout << “cVal,pVal are “;
          cout << cVal << “,” << pVal << endl;
      }
};
```

```
void show(parent *p) {
    p->print();
}

int main() {
   parent x(1);
   child y(2);
}

// both times it calls the parent print method
// since p->print() is calling through parent ptr

// first call displays pVal is 1
// second call displays pVal is -1

// (when child is declared it uses default
//   parent constructor)
```

Static vs dynamic binding

- suppose we wanted the second call to actually run the child's print:
- i.e. pick the method based on the kind of object passed, not simply the parameter type, with desired effect:
  - show(y); // want it to run y.child::print() since y is child
  - show(x); // want it to run x.parent::print() since x is parent
- called dynamic binding
  - static: specific method is chosen at compile time
  - dynamic: specific method not chosen until call actually takes place during execution, and is based on what was passed

Keywords: virtual, override, final

- we can preface method declarations with the ***virtual*** keyword
  - indicates they're meant to be overridden by descendants and dynamically bound
  - virtual void print(); // inside class declaration
- when overriding a method we can explicitly add ***override*** keyword
  - triggers a compile-time error if we accidentally hide an inherited method
  - virtual void print() override; // inside class declaration
- we can add keyword ***final*** to a method to indicate this cannot be overriden
  - virtual void print() final; // inside class declaration

Hiding inherited methods

- suppose we have following scenario:
  - parent declares print method with no params
  - child declares print method with one param
  - main tries to call print() on a child object
- call fails: the child's declared print hides inherited prints with different parameter profiles
- we could still call the parent's print explicitly, e.g. c.parent::print()