CS 162
Intro to CS II

Classes: Polymorphism
Chap. 15.1
Remember: What is not inherited?

- Constructors
- Private Members
- Destructors
- Copy Constructor (by default content)
- Overloaded Assignment Ops

What does this mean to not be inherited?
- Notice most of these cases are functions...
- Inherited basically gets inlined in class, not called
Copy Constructor Example

```cpp
class parent {
public:
    parent() { 
        //Have a constructor
        shared_ptr = new int;
        *shared_ptr = 4;
    }
    parent(const parent &p) {  
        //Copy Constructor
        shared_ptr = new int; //Why do this???
        *shared_ptr = *(p.shared_ptr);
    }
    ~parent() { 
        //Have a destructor
        delete shared_ptr;
    }
private:
    int *shared_ptr;
};
```
Protected vs. Private Inheritance

• Public, protected, and private members
  – Who has access to these?

• Public, protected, and private inheritance
  – What does this mean?

  class child : public parent { ... }
  class child : protected parent { ... }
  class child : private parent { ... }
Protected vs. Private Inheritance

• Protected
  – Public members are protected in child

• Private
  – All members are private to child

• Either case:
  – Cannot be used as parent, i.e. breaks “is a”
Polymorphism

• Webster definition:
  – The quality or state of existing in or assuming different form

• C++ definition:
  – Associate many meanings to one function by virtual functions, i.e. late/dynamic binding
What exactly is virtual?

• If a function is virtual in the parent/base and a new definition is provided in the child/derived class, then the virtual function in the child is always invoked, even if the call was in the parent class.

• This call to the virtual function in the parent class is dynamic/late binding.
Employee Class...

class Employee {
    public:
        employee(int y) { years = y; }
        virtual int get_seniority_bonus() { return 2*years; }
        int get_vacation_days() {
            return 10 + get_seniority_bonus();
        }
    private:
        int years;
};
class secretary : public employee {
public:
  secretary(int y) : employee(y) {
  }
  int get_seniority_bonus() { return 0; }
  void take dictation(string txt) {
    cout << "Taking dictation: " + txt << endl;
  }
};
#include <iostream>
#include <cstring>
using namespace std;

class employee {
    public:
        employee(int y) { years = y; }
        int get_vacation_days() { // This is late/dynamic binding
            return 10 + get_seniority_bonus();
        }
        virtual int get_seniority_bonus() { return 2 * years; }
    private:
        int years;
};

class secretary : public employee {
    public:
        secretary(int y) : employee(y) { }
        int get_seniority_bonus() { return 0; } // Secretary doesn't get bonus
        void take_dictation(string txt) {
            cout << "Taking Dictation: " + txt << endl;
        }
};

int main() {
    employee e(5); // Employee and Secretary have same # years...
    secretary s(5);
    cout << e.get_vacation_days() << endl;
    cout << s.get_vacation_days() << endl; // Poor secretary, no love:
    return 0;
}
Quiz #7

• Get into groups of 4 – 5.
• What is the difference in a friend function and inherited function?
• What is the difference between redefining a function and polymorphism?