CS 162
Intro to CS II

Finish Polymorphism
Redefine/Override vs. Polymorphism

```cpp
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;

class employee {
    public:
        employee() { }
        employee(int y) { years = y; }
        int get_vacation_days() { //This always calls employee w/o virtual
            return 10 + get_seniority_bonus();
        }
        //When the virtual is missing it is redefined if in a child too
        int get_seniority_bonus() { return 2 * years; }
        friend void test(employee &s);
        ~employee() { }
    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;
        }
    }
};
```
What is polymorphism?

• Vehicle, Bike example...
• Revisit our code
Extending Types/Polymorphism

• Can upcast, but not down
  Parent p; Child c;
  p = c;  //what will the polymorph function call now?

• What if we made pointers?
  Parent *p; Child *c = new Child;
  p = c;
Make Destructors Virtual

• What does this do if destructor isn’t virtual?
  Child *c=new Child; vs. Parent *p = new Child;
  delete c;
  delete p;

• Example:
  class parent {
    public:
      parent() { //Have a constructor
        shared_ptr = new int;
      }
      virtual ~parent() { //Have a destructor
        delete shared_ptr;
      }
    private:
      int *shared_ptr;
  };

Pure Virtual

• Definition
  – Don’t need to define function in base/parent class
  – Why?

• Abstract class
  – One or more pure virtual functions in class
Pure Virtual

class figure {
public:
    figure();
    ~figure();
    virtual void draw() = 0;
    center() { ... draw(); ...}
};

class circle : public figure {
public:
    circle();
    ~circle();
    void draw() { ... }
};

class rectangle : public figure {
public:
    rectangle();
    ~rectangle();
    void draw() { ... }
};