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;
#ifndef VEHICLE_H
#define VEHICLE_H

class Vehicle {
    private:
        int wheels;
    public:
        int get_wheels();
        void set_wheels(int);
        // If you have one pure virtual function in the class,
        // then you have an abstract base class, which means you
        // cannot create objects of this type
        virtual int get_toll(); // get_toll virtual for polymorphism
        // friend bool operator==(Vehicle & , Vehicle &);
        bool operator==(Vehicle &);
};

#endif
```cpp
int main() {
    Bike b, b1;
    Vehicle v; // cannot create object of abstract base class
    Vehicle *v1;

    v.set_wheels(4);
    b.set_wheels(2);
    b1.set_wheels(3);
    b1=b;
    if(b==b1)
        cout << "our bike and vehicle are equal" << endl;

    v=b; // this will never be polymorphic!!!
    // You can only get polymorphic behavior with a pointer!!!
    v1=&b; // if get_toll isn't virtual, then always calls base

    // v1->new_fun(); // cannot call a function only defined in child

    // never polymorphic without pointer
    cout << "Vehicle Toll: " << v.get_toll() << endl;
    // only polymorphic if the get_toll is virtual
    cout << "Vehicle pointer Toll: " << v1->get_toll() << endl;
    cout << "Bike Toll: " << b.get_toll() << endl;
}
```
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:**
  ```cpp
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
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() { ... }
};
#ifndef VEHICLE_H
#define VEHICLE_H

class Vehicle {

private:
    int wheels;

public:
    int get_wheels();
    void set_wheels(int);

    // If you have one pure virtual function in the class,
    // then you have an abstract base class, which means you
    // cannot create objects of this type.
    virtual int get_toll() = 0; // make get_toll pure virtual

    // friend bool operator==(Vehicle &, Vehicle &);
    bool operator==(Vehicle &);
};

#endif
```cpp
#include "Vehicle.h"
#include <iostream>
using namespace std;

bool Vehicle::operator==(Vehicle &v) {
    return this->wheels == v.wheels;
}

int Vehicle::get_wheels() {
    return wheels;
}

void Vehicle::set_wheels(int w) {
    wheels = w;
}

// If the function is pure virtual, then you do not define it.
*/
```

```cpp
return true;
return false;

//return v.get_wheels()==v1.get_wheels();
*/

int main() {
Bike b, b1;
//Vehicle v; //cannot create object of abstract base class
Vehicle *v1;

//v.set_wheels(4);
b.set_wheels(2);
b1.set_wheels(3);
b1=b;
if(b==b1)
    cout << "our bike and vehicle are equal" << endl;

//v=b;
v1=&b;
//v1->new_fun(); //cannot call a function only defined in child
//cout << "Vehicle Toll: " << v.get_toll() << endl;
    cout << "Vehicle pointer Toll: " << v1->get_toll() << endl;
cout << "Bike Toll: " << b.get_toll() << endl;
}
```
Fun Friday: Extra Credit

• Get into groups 4-5.
• Discuss Assignment #3 classes: World, Creature, Demon, Human, Elf, Cyberdemon, and Balrog.
  – What is the relationship?
  – Where would you use polymorphism, instead of upcasting?
  – Are any of these classes abstract classes?
virtual int getDamage();

Creature, Demon

void battle(Creature &a, Creature &b);