CS 162
Intro to CS II
Finish Polymorphism
Redefine 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();
    }
};  

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

//When the virtual is missing it is redefined if in a child too

virtual int get_seniority_bonus() { return 2 * years; }
friend void test(employee &e);
~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
    < you can’t define an object of this type
Pure Virtual

```cpp
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() { ... }
};
```

When you have one or more pure virtual functions, you do not define the functionality here. Then you have an abstract class.

- base
In Class Exercise

• Get into groups 4-5.

• Discuss Lab#5: vehicle, bike, skateboard, car, motorcycle, and date classes.
  — What is the relationship?
  — What did you learn about polymorphism and upcasting?
  — Are any of these classes abstract classes?
#ifndef V_H
#define V_H

class vehicle {
    private:
    int seats;
    
    public:
    vehicle(int);
    int get_seats();
    // This makes a pure virtual function
    // and the vehicle class is an abstract class,
    // which means you cannot make a direct object
    // of this type.
    virtual int get_toll()=0;
}
#endif
Vehicle.cpp

```cpp
#include "./Vehicle.h"

Vehicle::Vehicle(int x) {
    seats=x;
}

int Vehicle::get_seats() {
    return seats;
}

// We made this a pure virtual function,
// which means we are not defining it in
// this class.

// int Vehicle::get_toll() {
//     return 20 * seats;
// }
```

"Vehicle.cpp" 15L, 256C written
```cpp
#include <iostream>
#include "./Vehicle.h"
#include "./bike.h"
using std::cout;
using std::endl;
int main()
{
    //vehicle v(4); //Cannot make object of abstract class
    bike b(1);
    vehicle *vp = &b; //Polymorphism is late binding with pointer

    //v=b; //upcasting is not polymorphism
    //b=v; //downcasting not advised

    cout << v.get_seats() << endl;
    cout << b.get_seats() << endl;
    cout << v.get_toll() << endl;
    cout << b.get_toll() << endl;

    cout << vp->get_seats() << endl;
    cout << vp->get_toll() << endl;

    return 0;
}
```