CS 162
Intro to CS II

Finish Inheritance and
Operator Overload/Friends
Protected vs. Private Inheritance

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

- Public, protected, and private inheritance
  - What does this mean?

```cpp
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:**
  - Breaks “is a” relationship
Operator Overload Non-Member...

```cpp
bool operator == (const Point &, const Point &);  
int main () {
    Point p1, p2(2, 2);
    // How do we test if(p1 == p2)?
    return 0;
}
bool operator == (const Point &p1, const Point &p2) {
    if(p1.get_x() == p2.get_x() && p1.get_y() == p2.get_y())
        return true;
    else
        return false;
}
What if you don’t want to go through the accessor/mutator functions?
```

```
if (p1 == p2)

String s = "hi";
cout <"there";
String const char *
cout << "there" + s.
```
bool operator ==(const Point &p1, const Point &p2);  
int main () {  
    Point p1, p2(2, 2);  
    //How do we test if(p1 == p2)? return 0;  
}  
bool operator !==(const Point &p1, const Point &p2) {  
    if(p1.x == p2.x && p1.y == p2.y) //How do we have direct access to data members!  
        return true;  
    else  
        return false;  
}
class Point {
public:
    Point(); // Default Constructor
    Point(const int x_val, const int y_val); // Non-Default Constructor
    void set_xy(const int theX, const int theY); // Mutator Function
    int get_y() const; // Accessor Function
    int get_x() const; // Accessor Function
    friend bool operator ==(const Point &, const Point &);

private:
    int x;
    int y;
};
Point::Point(const int x_val, const int y_val) { x=x_val; y=y_val; }
Point::Point() { x=0; y=0; }
Operator Overload Member

class Point {
public:
  Point(); // Default Constructor
  Point(const int x_val, const int y_val); // Non-Default Constructor
  void set_xy(const int theX, const int theY); // Mutator Function
  int get_y() const; // Accessor Function
  int get_x() const; // Accessor Function
  bool operator ==(const Point &) const;
private:
  int x;
  int y;
};

Point::Point(const int x_val, const int y_val) { x=x_val; y=y_val; }
Point::Point() { x=0; y=0; }
bool Point::operator ==(const Point &p2) const {
  if(x == p2.x && y == p2.y)
    return true;
  else
    return false;
}
What is polymorphism?

• Vehicle and Bike Example...
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;
```cpp
#include "bike.h"
#include <iostream>

using namespace std;

int main() {
    vehicle v(4), *vptr;
    Bike b(2);
    // b=v; // don't assign parent to child
    v=b;    // can put child in parent, but v is statically bound to vehicle
    vptr=&b;    // can make vptr point to vehicle or bike for morphing
    cout << vptr->get_seats() << endl;
    cout << vptr->get_toll() << endl;    // this will only morph if virtual
    cout << b.get_seats() << endl;
    cout << b.get_toll() << endl;
    cout << v.get_seats() << endl;
    cout << v.get_toll() << endl;
}
```
Demo...

```cpp
class vehicle {
    protected:
        int seats;
    public:
        vehicle(int);
        int get_seats();
    virtual int get_toll(); // we want this to morph with dynamic binding
};
```
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();
        }
    
        // 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 &s);
    private:
        int years;
};

class secretary : public employee{
    public:
        secretary(int y) : employee(y) { }
        virtual int get_seniority_bonus() { return 0; } // Secretary doesn't get bonus

        void take_dictation(string txt) {
            cout << "Taking Dictation: " + txt << endl;
        }
};
```
Make Destructors Virtual

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

  ...
  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() { ... }
};