CS 162
Intro to CS II

Finish Inheritance, Operator Overload, and Friend Functions
Odds and Ends

• Last few days to get Assignment 2 graded!!!
• IT club: Build Home Network
  – 7pm to 9pm this Thursday
  – http://corvallisitclub.com
```cpp
#include "Vehicle.h"
#include "Bike.h"
#include <iostream>
using namespace std;

int main() {
    Vehicle v;
    Bike b;

    v.set_wheels(4);
    b.set_wheels(2);

    v=b;  // it is fine to assign a derived (child) to a base (parent) obj
    // b=v;  // you shouldn't assign a base (parent) to a derived (child) obj
    if(b==v)
        cout << "vehicles are equal" << endl;

    cout << "Vehicle wheels: " << v.get_wheels() << endl;
    cout << "Bike wheels: "   << b.get_wheels()  << endl;
    cout << "Vehicle toll:  " << v.get_toll()   << endl;
    cout << "Bike toll:    " << b.get_toll()   << endl;
}
```
Operator Overload Non-Member...

bool operator == (const Point & p1, const Point & p2) {
    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?
Operator Overload/Friends...

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;
}
Operator Overload Non-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
    friend bool operator == (const Point & l, const Point & r);
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; }
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 &p2) 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;
}
```cpp
#include "Vehicle.h"
#include "Bike.h"
#include <iostream>
using namespace std;

// operator overload non-member not as a friend must use accessors for members
bool operator==(Vehicle &v, Vehicle &v1) {
    if(v.get_wheels()==v1.get_wheels())
        return true;
    return false;
    // return v.get_wheels()==v1.get_wheels(); // same logic as above
}

int main() {
    Vehicle v;
    Bike b;

    v.set_wheels(4);
    b.set_wheels(2);

    v=b;    // it is fine to assign a derived (child) to a base (parent) obj
// b=v;     // you shouldn't assign a base (parent) to a derived (child) obj
    if(b==v)
        cout << "vehicles are equal" << endl;

    cout << "Vehicle wheels: " << v.get_wheels() << endl;
    cout << "Bike wheels: "  << b.get_wheels() << endl;
    cout << "Vehicle toll: "  << v.get_toll()  << endl;
    cout << "Bike toll: "    << b.get_toll()  << endl;
~
```
```cpp
#include "Vehicle.h"
#include "Bike.h"
#include <iostream>
using namespace std;

// operator overload non-member as friend to Vehicle, so we have access to
// private members!!!
bool operator==(Vehicle &v, Vehicle &v1) {
    if(v.wheels==v1.wheels)
        return true;
    return false;
    //return v.get_wheels()==v1.get_wheels(); // same logic as above
}

int main() {
    Vehicle v;
    Bike b;

    v.set_wheels(4);
    b.set_wheels(2);

    v=b;  // it is fine to assign a derived (child) to a base (parent) obj
    // b=v;  // you shouldn't assign a base (parent) to a derived (child) obj
    if(b==v)
        cout << "vehicles are equal" << endl;

    cout << "Vehicle wheels: " << v.get_wheels() << endl;
    cout << "Bike wheels: " << b.get_wheels() << endl;
    cout << "Vehicle toll: " << v.get_toll() << endl;
    cout << "Bike toll: " << b.get_toll() << endl;
}"
```
#define VEHICLE_H

class Vehicle {
    protected:
        int wheels;
    public:
        void set_wheels(int);
        int get_wheels();
        int get_toll();
        //make non-member operator overload a friend to access private
        //members
        friend bool operator==(Vehicle &, Vehicle &);
};

#endif
```cpp
#define VEHICLE_H

class Vehicle {
    protected:
        int wheels;
    public:
        void set_wheels(int);
        int get_wheels();
        int get_toll();
        // operator overload as a member function, which means first operand had to be a Vehicle or derived from Vehicle
        bool operator==(Vehicle &);
};
#endif
```
```cpp
#include "Vehicle.h"

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

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

int Vehicle::get_toll() {
    return 1 * wheels;
}

// operator overload as a member function. The first operand is
// the calling object to the function, which is the first argument
// implicit to member functions of a class.
bool Vehicle::operator==(Vehicle &v1) {
    if (this->wheels == v1.wheels)
        return true;
    return false;
}
```