CS 162
Intro to CS II
Operator Overloading/Friend Functions
Chap. 8
Odds and Ends

• Is everyone comfortable with all the uses of `const`?
  – Know the four ways to use it!!!
class Point {
public:
    Point();
    Point(int x_val, int y_val);  // Constructor
    int get_y() const;  // Accessor Function
    int get_x() const;  // Accessor Function
private:
    int x;
    int y;
};

int Point::get_y() const {
    return y;
}
int Point::get_x() const {
    return x;
}
Operator Overload...

```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’s the moral of the story?
Can we access x and y directly?

```c
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.x == p2.x && p1.y == p2.y)
        return true;
    return false;
}
```

Sure, we can make it a friend function...
class Point {
public:
    Point();
    Point(int x_val, int y_val); //Constructor
    int get_y() const;  //Accessor Function
    int get_x() const;  //Accessor Function
friend bool operator == (const Point &, const Point &); //Friend of the Point class
private:
    int x;
    int y;
};
int Point::get_y() const {
    return y;
}
int Point::get_x() const {
    return x;
}
Make Op Overload a Member Function

class Point {
public:
    Point();
    Point(int x_val, int y_val);    //Constructor
    int get_y() const;    //Accessor Function
    int get_x() const;    //Accessor Function
    bool operator == (const Point &p2) const; //Operator Overload Member Function
private:
    int x;
    int y;
};

bool operator == (const Point &p2) const {
    if(x == p2.x && y == p2.y)
        return true;
    return false;
}

OSU Oregon State University
Well, how does that work?

```c++
int main () {
    Point p1, p2(2, 2);
    if(p1 == p2)  //First operand is calling object, i.e. p1.==(p2)
        cout << "Point objects are equal" << endl;
    else
        cout << "Point objects are not equal" << endl;
    return 0;
}
```
```cpp
1 class Point {
2    public:
3        Point();
4        Point(int, int);
5        int get_x() const; // We need to make the function const
6        int get_y() const; // We need to make the function const
7        friend bool operator == (const Point &, const Point &); // Member takes precedence
8    private:
9        int x, y;
10 };```

```cpp
#include "Point.h"

Point::Point() {
    x = 1;
    y = 1;
}

Point::Point(int x, int y) {
    this->x = x;
    this->y = y;
}

int Point::get_x() const {  // Has to be constant here too
    return x;
}

int Point::get_y() const {  // Has to be constant here too
    return y;
}

// x and y are associated with calling object
bool Point::operator == (const Point &p2) {
    if (x == p2.x && y == p2.y) {
        return true;
    }
    return false;
}
```

class Points {
    public:
    Points();
    ~Points();
    Point * get_p() { return p; }

private:
    Point *p; //Let's make this dynamic
};
Points.cpp (Implementation)

```
#include "Points.h"

// What goes in here?
Points::Points() {
    p = new Point[10];
}

// We need a destructor
Points::~Points() {
    delete [] p;
}
```

Using == operator overload

```cpp
#include <iostream>
#include <fstream>
#include "Points.h"
using namespace std;

// friend funcion of Point, so you don't need get_x & get_y
bool operator == (const Point &p1, const Point &p2) {
    if (p1.x == p2.x && p1.y == p2.y) {
        cout << "hello friend" << endl;
        return true;
    }
    return false;
}

// Destructor is called when pts goes out of scope
void make_pts() {
    Points pts, pts2;
    std::cout << (void *) pts.get_p() << std::endl;
    std::cout << pts.get_p()[4].get_x() << std::endl;
    if (pts.get_p()[1] == pts2.get_p()[1])
        cout << "They are equal" << endl;
}
```
Other Stuff...

• Overload Function Application, ()
  – Had to be member function
  – Allows you to use object like function
  
  ```cpp
  void Point::operator ()(int translate) const {
  ...
  }
  ...
  ```
  
  Point p1;
  p1(10);

• Avoid overloading &&, ||, and ,

OSU Oregon State University
#define NUM_STUDENTS 10
#define NUM_GRADES 3

int main (int argc, char *argv[]) {
    float grades[NUM_STUDENTS][NUM_GRADES];
    int g[3]={0, 10, 20}; //Create a static array

    cout << "Static 1-d" << endl;
    cout << &g << endl; //This is where g lives
    cout << g << endl; //Contents of g, which is where g[0] lives
    cout << &g[0] << endl; //Where g[0] lives
    cout << &g[1] << endl; //Where g[1] lives, which is beside g[0]

    cout << "Static 2-d" << endl;
    cout << &grades << endl; //Where grades lives
    cout << grades << endl; //grades has where grades[0][0] lives
    cout << grades[0] << endl; //Array of const self-ref pointers
    cout << &grades[0][0] << endl; //Where grades[0][0] lives
    cout << &grades[0][1] << endl; //grades[0][1] lives beside grades[0][1]
    cout << "--------" << endl;
    //Next element in array of self-ref pointers points
    //to the next row in the array, which is why grades+1
    //takes you a whole stride/column length
    cout << grades[1] << endl; //Contains where grades[1][0] lives
    cout << &grades[1] << endl; //Lives there too because self-reference
    cout << &grades[0][2] << endl; //Where last element in 1st row lives
    cout << &grades[1][0] << endl; //Rows are consecutive in memory
    cout << &grades[1][1] << endl; //Elements in a row are consecutive

7, 0-1