```
Script started on Thu 07 Dec 2017 05:48:57 PM CST
\033]0;p_stach@mars:~/pointyop\007[p_stach@mars pointyop]$ pwd
/home/students/p_stach/pointyop
\033]0;p_stach@mars:~/pointyop\007[p_stach@mars pointyop]$ cat main.info
Name: Patrick Stach

Class: CSC122-001

Program: Labs - "Operate on this"

Levels Attempted:
-This assignment is (Level 2).
Add (Level 1) to overload operator[] to return the x or y part of the point.
2+1 = Level 3

Program Description:
This library includes overloaded operators for the point class
which allows for easy inputting/ outputting, equality checks,
returning distance between 2 points and finding a midpoint
between 2 points
\033]0;p_stach@mars:~/pointyop\007[p_stach@mars pointyop]$ cat main.cpp
#include <iostream>
#include "point.h"
using namespace std;

int main()
{
    cout << "Enter point p: ";
    Point p;
    cin >> p;
    cout << "You entered " << p << endl;
    cout << "Enter point q: ";
    Point q;
    cin >> q;
    cout << "You entered " << q << endl;
    cout << "p=q? (0 false, 1 true): " << (p == q) << endl;
    cout << "p!=q? (0 false, 1 true): " << (p != q) << endl;
    cout << "Distance between p and q: " << (p - q) << endl;
    cout << "Midpoint between p and q: " << (p / q) << endl;
    cout << "Press q to quit: ";
    cin.ignore(999, 'q');
    return 0;
}
\033]0;p_stach@mars:~/pointyop\007[p_stach@mars pointyop]$ cat point.cpp
#include "point.h"

#include <iostream>
#include <math.h>
using namespace std;

// %%%%% OPERATOR OVERLOADING %%%%%

//returns whether two points are equal
bool Point::operator==(const Point & P) const
{
    // if x's and y's =, true
    return ((x == P.x) && (y == P.y) ? true : false);

}
//returns whether two points are not equal
bool Point::operator!=(const Point & P) const
{
    // if x's and y's =, true
    return ((x == P.x) && (y == P.y) ? false : true);
}

//returns midpoint of two points
Point Point::operator/(const Point & P) const
{
    return Point((x + P.x) / 2, (y + P.y) / 2);
}
//returns x or y value
double Point::operator[](const char& ch)const
{
    double rv;
    if (ch == 'x')
    {
        rv = x;
    }
    else if (ch == 'y')
    {
        rv = y;
    }
    else
    {
        rv = -999999;
    }
    return rv;
}



//returns distance of two points
double Point::operator-(const Point & P) const
{
    return sqrt(pow(x - P.x, 2.0) +
        pow(y - P.y, 2.0));
}

//outputs point
ostream& operator<<(ostream& out, const Point& p)
{
    p.Output();
    return out;
}

//inputs points
istream& operator>>(istream& in, Point& p)
{
    p.Input();
    return in;
}

// read standard 2D point notation (x,y) -- ignore
// window dressing
void Point::Input(void)
{
    char dummy;
    cin >> dummy >> x >> dummy >> y >> dummy;
    return;
}

// output standard 2D point notation (x,y)
void Point::Output(void) const
{
    cout << '(' << x << ", " << y << ')';
    return;
```

```
}

// calculate distance between two 2D points --
// the one that called us and the argument
double Point::distance(Point other)
{
    return sqrt(pow(other.x - x, 2.0) +
        pow(other.y - y, 2.0));
}

// set coordinates to programmer-specified values
void Point::set_x(double new_x)
{
    x = new_x;        // no error checking since anything is legal
    return;
}

// set coordinates to programmer-specified values
void Point::set_y(double new_y)
{
    y = new_y;        // no error checking since anything is legal
    return;
}

// construct Point by default -- no values specified
Point::Point(void)
{
    x = y = 0.0;
}

// construct Point given initial x,y values
Point::Point(double new_x, double new_y)
{
    set_x(new_x);
    set_y(new_y);
}

// creates a point flipped about the x axis from us
Point Point::flip_x(void)
{
    return Point(x, -y);
}

// creates a point flipped about the y axis from us
Point Point::flip_y(void)
{
    return Point(-x, y);
}

// creates a point shifted along the x axis from us
Point Point::shift_x(double move_by)
{
    return Point(x + move_by, y);
}

// creates a point shifted along the y axis from us
Point Point::shift_y(double move_by)
{
    return Point(x, y + move_by);
}
\033]0;p_stach@mars:~/pointyop\007[p_stach@mars pointyop]$ cat point.h
#ifndef POINT_CLASS_HEADER_INCLUDED
#define POINT_CLASS_HEADER_INCLUDED
#include <iostream>

// A 2D point class
class Point
{
    double x, // x coordinate of point
        y; // y coordinate of point

public:
    friend std::ostream& operator<<(std::ostream& out, const Point& p);
    friend std::istream& operator>>(std::istream& in, Point& p);
    double operator-(const Point & P) const;
    Point operator/(const Point & P) const;
    bool operator ==(const Point & P) const;
    bool operator !=(const Point & P) const;
    double operator[](const char& ch)const;
    Point(void);
    Point(double new_x, double new_y);

    void Output(void) const;   // output this point
    void Input(void);    // input this point
    double distance(Point other);   // distance between this point and other

    double get_x(void) { return x; }
    double get_y(void) { return y; }

    void set_x(double new_x);
    void set_y(double new_y);

    Point flip_x(void);
    Point flip_y(void);

    Point shift_x(double move_by);
    Point shift_y(double move_by);
};

#endif
\033]0;p_stach@mars:~/pointyop\007[p_stach@mars pointyop]$ CPP main point
main.cpp***
point.cpp...
point.cpp: In member function ‘bool Point::operator==(const Point&)
const’:
point.cpp:13: warning: comparing floating point with == or != is unsafe
point.cpp:13: warning: comparing floating point with == or != is unsafe
point.cpp: In member function ‘bool Point::operator!=(const Point&)
const’:
point.cpp:20: warning: comparing floating point with == or != is unsafe
point.cpp:20: warning: comparing floating point with == or != is unsafe
point.cpp: In constructor ‘Point::Point()’:
point.cpp:111: warning: ‘Point::x’ should be initialized in the member
initialization list
point.cpp:111: warning: ‘Point::y’ should be initialized in the member
initialization list
point.cpp: In constructor ‘Point::Point(double, double)’:
point.cpp:117: warning: ‘Point::x’ should be initialized in the member
initialization list
point.cpp:117: warning: ‘Point::y’ should be initialized in the member
initialization list


\033]0;p_stach@mars:~/pointyop\007[p_stach@mars pointyop]$ ./main.out
Enter point p: (3,4_
You entered (3, 4)
Enter point q: (10,8)
You entered (10, 8)
p=q? (0 false, 1 true): 0
```

```
p!=q? (0 false, 1 true): 1
Distance between p and q: 8.06226
Midpoint between p and q: (6.5, 6)
Press q to quit: q
\033]0;p_stach@mars:~/pointyop\007[p_stach@mars pointyop]$ ./main.out
Enter point p: (0,0)
You entered (0, 0)
Enter point q: (3,4)
You entered (3, 4)
p=q? (0 false, 1 true): 0
p!=q? (0 false, 1 true): 1
Distance between p and q: 5
Midpoint between p and q: (1.5, 2)
Press q to quit: q
\033]0;p_stach@mars:~/pointyop\007[p_stach@mars pointyop]$ cat main.tpq
1) Which operators are members and which are non-members? Do any
have to be members?
In my code, I made all of them members in the point class, but out of all of
them, only the [ ] operator needs to be a member function

2)Which operators should be const? What other methods might well be made const?
In general, what is the rule which determines if a method should be made const?
I made all of them constant because none of them needed to modify the original
points. This is with the exception of input and output since the stream
needs to be modified. The methods should be constant unless it is necessary
in their operation for them to be modified.

3) What type do equality and and inequality return? Input? Output? Assignment?
equality/inequality: bool
Input: istream
Output: ostream
Assignment: original type

4)Do you agree with your friend’s decision to use operator/ for midpoint?
Why/Why not?
No, the / operator is usually used for division and midpoint is not associated
with the "/" sign so it will be confusing

5)Why didn’t you overload operators for less than, greater than, etc.?
A point can’t be greater than or less than another. For example, you cant say
(4,2)>(3,4), it doesnt make sense

6)Your friend wanted to overload operators for the flip and shift methods,
too (~ and += respectively). Why did you talk them out of it?
Why wasn’t this a good idea?
We wouldn’t be able to chose whether to flip x or y or shift x or y since in
both cases it will read a double and would not be able to know whether we
want x or y to be flipped

7)Just because you’ve added operators, should you necessarily remove
the old methods that did these jobs?
No, there can be some cases where the other methods make more sense or are
more practical

8)Should the programmer be able to do:
    p['X'] = 2.0;
to change the X-coordinate of a Point object p?
No

9)If you were going to allow such behavior, how would you do it?
We would have to define the assignment operator for this to work.
\033]0;p_stach@mars:~/pointyop\007[p_stach@mars pointyop]$ exit
exit
```

```
Script done on Thu 07 Dec 2017 05:50:44 PM CST
```