```
Script started on 2023-07-27 14:59:11-05:00 [TERM="xterm" TTY="/dev/pts/1" COLUMNS=
bj94684@ares:~$ pwd
/home/students/bj94684
bj94684@ares:~$ point.info
point.info: command not found
bj94684@ares:~$ cat point.info
/*************************************************************************
 *                                                                       *
 *  NAME:  Jose Barron                          CLASS:  CSC122-002       *
 *                                                                       *
 *          Lab: Operate on this!                Level:   2              *
 *       Option: Overload operator []            Level: + 1              *
 *                                          Total Level:   3.0           *
 *                                                                       *
 * This program is designed to add operator overloading to a point class *
 * given by a friend. The operators that were overloaded in the class are *
 * the distance between two points (operator-), input (operator>>),      *
 * output (operator<<), equality (operator==),  inequality (operator!=),  *
 * midpoint (operator/), and operator[] to return the x or y part of the  *
 * point. The overloaded operators are tested in the program.            *
 *                                                                       *
*************************************************************************/
bj94684@ares:~$ show-code point.h


point.h:


```

```
 1  #ifndef POINT_CLASS_HEADER_INCLUDED
 2  #define POINT_CLASS_HEADER_INCLUDED
 3
 4  #include <iostream>
 5  #include <cmath>
 6
 7  // A 2D point class
 8  class Point
 9  {
10      double x, // x coordinate of point
11             y; // y coordinate of point
12
13  public:
14      Point(): x(0.0), y(0.0) {}
15      Point(double new_x, double new_y);
16
17
18      void Output(void);   // output this point
19      void Input(void);    // input this point
20      double distance(Point other);   // distance between this point and othe
21
22      double get_x(void) { return x; }
23      double get_y(void) { return y; }
24
25      void set_x(double new_x);
26      void set_y(double new_y);
27
28      Point flip_x(void);
29      Point flip_y(void);
30
31      Point shift_x(double move_by);
32      Point shift_y(double move_by);
33
34      Point operator - (const Point & other) const;
35      Point operator + (const Point & other) const;
36      Point operator / (double d) const;
37
38      bool operator == (const Point & other) const;
39      bool operator != (const Point & other) const;
40
41      friend std::istream & operator >> (std::istream & is, Point & p);
42      friend std::ostream & operator << (std::ostream & os, const Point & p)
43
44      Point & operator = (const Point & other);
45      Point (const Point & other) = default;
46
47      double & operator[] (char c);
48      const double & operator[] (char c) const;
49
50  };
51
52  #endif
```

```
bj94684@ares:~$ show-code point.cpp


point.cpp:


```

```
 1  #include "point.h"
 2  #include <iostream>
 3  #include <cmath>
 4
 5
 6  // read standard 2D point notation (x,y) -- ignore
 7  // window dressing
 8  void Point::Input(void)
 9  {
10      char dummy;
11      std::cin >> dummy >> x >> dummy >> y >> dummy;
12      return;
13  }
14
15  // output standard 2D point notation (x,y)
16  void Point::Output(void)
17  {
18      std::cout << '(' << x << ", " << y << ')';
19      return;
20  }
21
22  // 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 given initial x,y values
Point::Point(double new_x, double new_y) : x(new_x), 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);
}

Point Point::operator - (const Point & other) const
{
    return Point(x - other.x, y - other.y);
}

Point Point::operator + (const Point & other) const
{
    return Point(x + other.x, y + other.y);
}

Point Point::operator / (double d) const
{
    return Point(x / d, y / d);
}

bool Point::operator == (const Point & other) const
{
    return (x == other.x) && (y == other.y);
}

bool Point::operator != (const Point & other) const
{
    return !(*this == other);
}

std::istream & operator >> (std::istream & is, Point & p)
{
    char filler;
    is >> filler >> p.x >> filler >> p.y >> filler;
    return is;
}

std::ostream & operator << (std::ostream & os, const Point & p)
{
    os << '(' << p.x << ", " << p.y << ')';
    return os;
}


Point & Point::operator = (const Point & other)
{
    if (this == & other)
        return *this;

    x = other.x;
    y = other.y;
    return *this;
}

double & Point::operator[](char c)
{
    if (c == 'x' || c == 'X')
    {
        return x;
    }
    else if (c == 'y' || c == 'Y')
    {
        return y;
    }
    else
```

```
    {
        std::cout << "\nInvalid choice. Use 'x' or 'y'.";
        return x;

    }
}

const double & Point::operator[](char c) const
{
    if (c == 'x' || c == 'X')
    {
        return x;
    }
    else if (c == 'y' || c == 'Y')
    {
        return y;
    }
    else
    {
        std::cout << "\nInvalid choice. Use 'x' or 'y'.";
        return x;
    }
}
bj94684@ares:~$ show-code p.cpp


p.cpp:



#include <iostream>
#include <limits>
#include "point.h"

using namespace std;

int main()
{
    Point p1;
    Point p2;
    cout << "Enter x and y coordinates for p1: ";
    cin >> p1;
    cout << "Enter x and y coordinates for p2: ";
    cin >> p2;
    Point origin;

    cout << "p1: " << p1 << '\n';
    cout << "p2: " << p2 << '\n';
    cout << "origin:" << origin << '\n';

    Point midpt = (p1 + p2) / 2.0;
    cout << "Midpoint between p1 and p2: " << midpt << '\n';

    double distance = (p2 - p1).distance(origin);
    cout << "Distance between p1 and p2: " << distance << '\n';


    bool equal = (p1 == p2);
    bool not_equal = (p1 != p2);

    cout << "Equal test:\n";

    if ( equal )
    {
        cout << "p1 and p2 are equal:\n";
    }
    else
    {
        cout << "p1 and p2 are not equal.\n";
    }

    cout << "Not equal test:\n";

    if ( not_equal )
    {
        cout << "p1 and p2 are not equal:\n" ;
    }
    else
    {
        cout << "p1 and p2 are equal.\n";
    }

    cout << "Enter new x and y coordinates for p1: ";
    cin >> p1;
    cout << "Updated p1: " << p1 << '\n';

    double my_x, my_y;
    my_x = p1['x'];
    my_y = p1['Y'];

    cout << "my_x: " << my_x << '\n';
    cout << "my_y: " << my_y << '\n';

    double new_x;
    cout << "Enter new value for my_x: ";
    cin >> new_x;
    p1['x'] = new_x;

    cout << "Modified my_x: " << p1 << '\n';


    return 0;
}
bj94684@ares:~$ CPP point p
p.cpp***
point.cpp...
point.cpp: In member function ‘bool
Point::operator==(const Point&) const’:
point.cpp:88:15: warning: comparing
floating-point with ‘==’ or ‘!=’
```

```
is unsafe [-Wfloat-equal]
   88 |     return (x == other.x) && (y == other.y);
      |             ~~^~~~~~~~~~
point.cpp:88:33: warning: comparing
floating-point with '==' or '!='
is unsafe [-Wfloat-equal]
   88 |     return (x == other.x) && (y == other.y);
      |                               ~~^~~~~~~~~~


bj94684@ares:~$ ./p.out
Enter x and y coordinates for p1: (6.5,7.8)
Enter x and y coordinates for p2: (7.8,5.4)
p1: (6.5, 7.8)
p2: (7.8, 5.4)
origin:(0, 0)
Midpoint between p1 and p2: (7.15, 6.6)
Distance between p1 and p2: 2.72947
Equal test:
p1 and p2 are not equal.
Not equal test:
p1 and p2 are not equal:
Enter new x and y coordinates for p1: (6.2,3.4)
Updated p1: (6.2, 3.4)
my_x: 6.2
my_y: 3.4
Enter new value for my_x: 1.2
Modified my_x: (1.2, 3.4)
bj94684@ares:~$ ./p.out
Enter x and y coordinates for p1: (3.4,7.3)
Enter x and y coordinates for p2: (4.5,2.1)
p1: (3.4, 7.3)
p2: (4.5, 2.1)
origin:(0, 0)
Midpoint between p1 and p2: (3.95, 4.7)
Distance between p1 and p2: 5.31507
Equal test:
p1 and p2 are not equal.
Not equal test:
p1 and p2 are not equal:
Enter new x and y coordinates for p1: (5.6,4.2)
Updated p1: (5.6, 4.2)
my_x: 5.6
my_y: 4.2
Enter new value for my_x: 7.2
Modified my_x: (7.2, 4.2)
bj94684@ares:~$ ./p.out
Enter x and y coordinates for p1: (7.8,5.6)
Enter x and y coordinates for p2: (7.8,5.6)
p1: (7.8, 5.6)
p2: (7.8, 5.6)
origin:(0, 0)
Midpoint between p1 and p2: (7.8, 5.6)
Distance between p1 and p2: 0
Equal test:
p1 and p2 are equal:
Not equal test:
p1 and p2 are equal.
Enter new x and y coordinates for p1: (4.3,2.3)
Updated p1: (4.3, 2.3)
my_x: 4.3
my_y: 2.3
Enter new value for my_x: 6.3
Modified my_x: (6.3, 2.3)
bj94684@ares:~$ cat point.tpq
/*****************************************************************************
*                                                                            *
* TPQs:                                                                      *
* 1. All the operators are members of the class except for the operator      *
* << and operator >> since they are friends of the class not members. The    *
* operator + and operator [] always have to be members.                      *
* 2. The operators that are const, are ones that dont modify the Point       *
* object (essentially just reads the object). Like the operator << just      *
* reads the object and displays it but doesnt change it in any way.          *
* 3. Equality and inequality return a bool that indicates whether the        *
* coordinates compared are equal or not. The input returns istream and       *
* the output returns ostream.                                                *
* 4. No, I think a midpoint() wouldve been more straightforward and          *
* easier to understand than the overloaded operator /.                       *
* 5. When comparing two coordinates you dont compare to find which is        *
* greater or smaller like you would for integer since coordinates involve    *
* a number for x and a number for y, not just one.                           *
* 6. Overloading those operators for those methods would not be intuitive    *
* and would make using those operators confusing.                            *
* 7. No, you shouldnt remove the old methods since they can still serve      *
* a purpose and other methods may rely on the old methods.                   *
* Additional TPQs:                                                           *
* 1. No since when refering to the x coordinate or y coordinate its          *
* common practice to use 'x' or 'y'.
* 2. It isnt difficult to add case-insesitive to the operator [], to         *
* allow for either 'X' or 'x', for input processing you would add both       *
* options.                                                                   *
*                                                                            *
*****************************************************************************/
bj94684@ares:~$ exit
exit

Script done on 2023-07-27 15:01:58-05:00 [COMMAND_EXIT_CODE="0"]
```