```
Script started on 2024-12-05 11:21:07-06:00 [TERM="xterm-256color" TTY="/dev/pts/6'
e_prevost@ares:~/Lab_1_p3$ pwd
/home/students/e_prevost/Lab_1_p3
e_prevost@ares:~/Lab_1_p3$ cat lab_1.info
****************

Robert Prevost

CSC 122 W01



pointyopers lab



creates a point class with operator overloading



Base Level: Level 2

Total Level: Level 2

****************e_prevost@ares:~/Lab_1_p3$ show-code point.h


point.h:


#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point {
private:
    double x;
    double y;

public:
    // Constructors
    Point() : x(0), y(0) {}
    Point(double x_coord, double y_coord) : x(x_coord), y(y_coord) {}
    Point(const Point& other) : x(other.x), y(other.y) {}

    // Original methods (kept for backward compatibility)
    double getX() const { return x; }
    double getY() const { return y; }
    void setX(double x_coord) { x = x_coord; }
    void setY(double y_coord) { y = y_coord; }
    double distance(const Point& other) const;
    Point midpoint(const Point& other) const;

    // Operator overloading - member functions
    Point& operator=(const Point& other);
    bool operator==(const Point& other) const;
    bool operator!=(const Point& other) const;

    // Friend declarations for non-member operators
    friend std::ostream& operator<<(std::ostream& os, const Point& p);
    friend std::istream& operator>>(std::istream& is, Point& p);
    friend double operator-(const Point& p1, const Point& p2);
    friend Point operator/(const Point& p1, const Point& p2);
};

#endif
e_prevost@ares:~/Lab_1_p3$ show-code point.cpp


point.cpp:


#include <cmath>
#include "point.h"
Point& Point::operator=(const Point& other) {
    if (this != &other) {
        x = other.x;
        y = other.y;
    }
    return *this;
}

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

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

double Point::distance(const Point& other) const {
    double dx = x - other.x;
    double dy = y - other.y;
    return std::sqrt(dx * dx + dy * dy);
}

Point Point::midpoint(const Point& other) const {
    return Point((x + other.x) / 2, (y + other.y) / 2);
}

// Non-member operator overloads
std::ostream& operator<<(std::ostream& os, const Point& p) {
    os << "(" << p.x << ", " << p.y << ")";
    return os;
}

std::istream& operator>>(std::istream& is, Point& p) {
    36      char ch;
    37      is >> ch >> p.x >> ch >> p.y >> ch; // Expects format (x,y)
    38      return is;
    39  }
    40
    41  double operator-(const Point& p1, const Point& p2) {
    42      return p1.distance(p2);
    43  }
    44
    45  Point operator/(const Point& p1, const Point& p2) {
    46      return p1.midpoint(p2);
    47  }
e_prevost@ares:~/Lab_1_p3$ show-code lab_1.cpp


lab_1.cpp:


     1  #include <iostream>
     2  #include "point.h"
     3  int main() {
     4      Point p1(3, 4);
     5      Point p2(6, 8);
     6      Point p3;
     7
     8      std::cout << "Point 1: " << p1 << std::endl;
     9      std::cout << "Point 2: " << p2 << std::endl;
    10      std::cout << "Point 3 (default): " << p3 << std::endl;
    11
    12      p3 = p1;
    13      std::cout << "After p3 = p1: " << p3 << std::endl;
    14
    15      std::cout << "p1 == p3? " << (p1 == p3 ? "true" : "false") <<
    16      std::endl;
    17      std::cout << "p1 != p2? " << (p1 != p2 ? "true" : "false") <<
    18      std::endl;
    19
    20      std::cout << "Distance between p1 and p2: " << (p1 - p2) << std::endl;
    21
    22      Point mid = p1 / p2;
    23      std::cout << "Midpoint of p1 and p2: " << mid << std::endl;
    24
    25      Point p4;
    26      std::cout << "Enter a point in format (x,y): ";
    27      std::cin >> p4;
    28      std::cout << "You entered: " << p4 << std::endl;
    29
    30      return 0;
    31  }
e_prevost@ares:~/Lab_1_p3$ CPP lab_1 point
lab_1.cpp***
point.cpp...
point.cpp: In member function ‘bool
Point::operator==(const Point&) const’:
point.cpp:12:15: warning: comparing
floating-point with ‘==’ or ‘!=’
is unsafe [-Wfloat-equal]
   12 |     return (x == other.x && y == other.y);
      |             ~~^~~~~~~~~~
point.cpp:12:31: warning: comparing
floating-point with ‘==’ or ‘!=’
is unsafe [-Wfloat-equal]
   12 |     return (x == other.x && y == other.y);
      |                             ~~^~~~~~~~~~


e_prevost@ares:~/Lab_1_p3$ ls
lab_1.cpp  lab_1.info  lab_1.out  lab_1.tpq  point.cpp  point.h  typescript
e_prevost@ares:~/Lab_1_p3$ ./lab_1.out
Point 1: (3, 4)
Point 2: (6, 8)
Point 3 (default): (0, 0)
After p3 = p1: (3, 4)
p1 == p3? true
p1 != p2? true
Distance between p1 and p2: 5
Midpoint of p1 and p2: (4.5, 6)
Enter a point in format (x,y): (2,3)
You entered: (2, 3)
e_prevost@ares:~/Lab_1_p3$ cat lab_1.tpq
Which operators are members and which are non-members? Do any have to be

members?



members: operator =, ==, != (== and != could be non members)

non-members: operator<<, >>, -, / (- and / could be members)



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?



comparison operators and get functions should be const. The general rule

is that if a method does not change the state of any variables (obeys the

read-only contract) it should be marked const.
```

```
What type do equality and and inequality return? Input? Output? Assignment?


equality and inequality return a bool. Input returns your istream object.

Output returns your ostream object. Assignment returns the reference

variable to your point.


Do you agree with your friend's decision to use operator/ for midpoint?

Why/Why not?


no I dont agree, midpoint being the division symbol is not intuitive.


Why didn't you overload operators for less than, greater than, etc.?


these points are in 2D space so its hard to justify less than and greater

than functions without a specified order.



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?


~ and += already have their meanings for every coder and overriding them to

create a flip and switch method isnt a great idea.



Just because you've added operators, should you necessarily remove the old

methods that did these jobs?


Yes we should keep them. It keeps a record of what we did and also some

programmers might prefer using the method names instead of the operators. e_prevos
exit

Script done on 2024-12-05 11:23:50-06:00 [COMMAND_EXIT_CODE="0"]
```