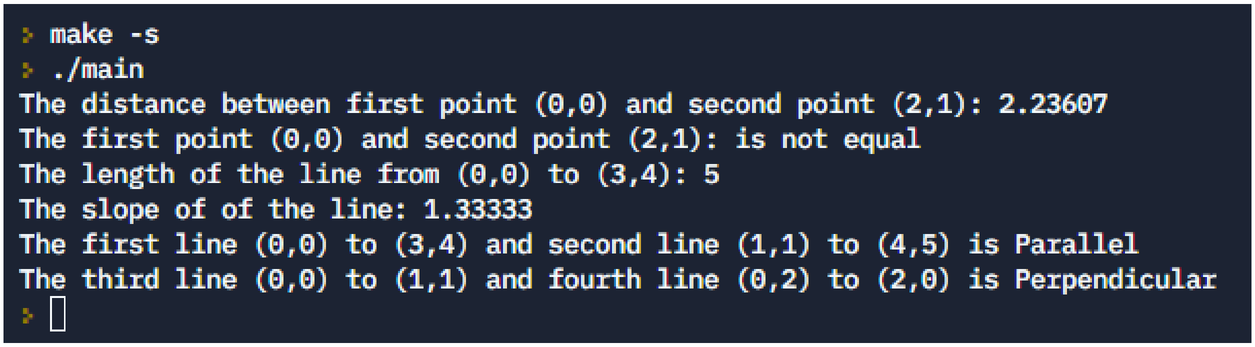

```
› make -s
› ./main
The distance between first point (0,0) and second point (2,1): 2.23607
The first point (0,0) and second point (2,1): is not equal
The length of the line from (0,0) to (3,4): 5
The slope of of the line: 1.33333
The first line (0,0) to (3,4) and second line (1,1) to (4,5) is Parallel
The third line (0,0) to (1,1) and fourth line (0,2) to (2,0) is Perpendicular
›
```

```
// Kenry Yu
// Olena Bilinska
// Brianna Soriano
// Demo at 5:45pm

#include "Line.h"
#include <iostream>
using namespace std;

int main() {
  //   Write a simple driver that outputs (annotate your output):
  // a. The distance between two points
  // b. A message showing whether or not two points are equal
  // c. The length and the slope of different lines
  Point first(0, 0), second(2, 1);
  cout << "The distance between first point (0,0) and second point (2,1): "
       << first.distance(second) << endl;
  cout << "The first point (0,0) and second point (2,1): "
       << (first == second ? "is equal" : "is not equal") << endl;
  Line first_line(0, 0, 3, 4), second_line(1, 1, 4, 5), third_line(0, 0, 1, 1),
      fourth_line(0, 2, 2, 0);
  cout << "The length of the line from (0,0) to (3,4): " << first_line.length()
       << endl;
  cout << "The slope of of the line: " << first_line.slope() << endl;

  cout << "The first line (0,0) to (3,4) and second line (1,1) to (4,5) is "
       << (first_line.slope() == second_line.slope() ? "Parallel"
           : (first_line.slope() == (-1 / second_line.slope()))
               ? "Perpendicular"
```

```
              : "Intersecting")
      << endl;
  cout << "The third line (0,0) to (1,1) and fourth line (0,2) to (2,0) is "
      << (third_line.slope() == fourth_line.slope() ? "Parallel"
          : (third_line.slope() == (-1 / fourth_line.slope()))
              ? "Perpendicular"
              : "Intersecting")
      << endl;
  return 0;
}
```

```
// Point.hpp
// Composition
//
//
#ifndef Point_h
#define Point_h
#include <stdio.h>
#include <iostream>
using namespace std;
class Point {
 private:
 int x;
 int y;

 public:
 // constructor
 Point(int x = 0, int y = 0);

 // methods
 double distance(const Point& p) const;
 int getX() const;
 int getY() const;
 void setLocation(int x, int y);
 void translate(int x, int y);

 // overloaded operators
 Point operator+(const Point& p) const;
 bool operator==(const Point& p) const;
 bool operator!=(const Point& p) const;
 friend ostream& operator<<(ostream& out, const Point& p);

};

#endif /* Point_hpp */
```

```
// Point.cpp
// Composition
//
#include <iostream>
#include <math.h>
#include "Point.h"
using namespace std;
// If no coordinates are specified, uses (0, 0).
Point::Point(int x, int y) {
 setLocation(x, y);
}
// Returns the distance between two points.
double Point::distance(const Point& p) const {
 int dx = x - p.getX();
 int dy = y - p.getY();
 return sqrt(dx*dx + dy*dy);
}
// Returns the x coordinate of this Point.
int Point::getX() const {
 return x;
}
// Returns the y coordinate of this Point.
int Point::getY() const {
 return y;
}
// Sets the x/y coordinates of this Point to the given values.
void Point::setLocation(int x, int y) {
 this->x = x;
 this->y = y;
}
// Shifts this point's location by the given amount.
void Point::translate(int dx, int dy) {
 setLocation(x + dx, y + dy);
}
// operators overloading
// add two Points.
Point Point::operator+(const Point& p) const {
 Point result(x + p.getX(), y + p.getY());
 return result;
 // return Point(this->x + p.getX(), this->y + p.getY());
}
// check Points equality ==
bool Point::operator==(const Point& p) const {
 return x == p.getX() && y == p.getY();
}
// check if two Points are not equal !=
bool Point::operator!=(const Point& p) const {
 return !(*this == p);
}
// cout << Point
ostream& operator<<(ostream& out, const Point& p) {
 out << "(" << p.getX() << ", " << p.getY() << ")";
 return out;
}
```

```
// Line.hpp
// Composition
//
//using namespace std;
#ifndef Line_hpp
#define Line_hpp
#include "Point.h"
#include <stdio.h>
class Line {
 private:
 Point* p1;
 Point* p2;
 // private initialization method (called by constructors and =)
 void init(int x1, int y1, int x2, int y2);

 public:
 // constructors/destructors
 Line(int x1, int y1, int x2, int y2);
 Line(const Line& line); // copy constructor
 ~Line(); // destructor

 // methods
 int getX1() const;
 int getY1() const;
 int getX2() const;
 int getY2() const;
 double length() const;
 double slope() const;
 void translate(int dx, int dy);
 // overloaded assignment = operator (to avoid memory leaks)
 const Line& operator=(const Line& rhs);
};
#endif /* Line_h */
```

```
// Line.cpp
// Composition
//
#include "Line.h"
#include <iostream>
using namespace std;
// helper initialization function
void Line::init(int x1, int y1, int x2, int y2) {
  this->p1 = new Point(x1, y1);
  this->p2 = new Point(x2, y2);
}

// normal constructor
Line::Line(int x1, int y1, int x2, int y2) { this->init(x1, y1, x2, y2); }

// "copy constructor"
Line::Line(const Line &line) {
  this->init(line.getX1(), line.getY1(), line.getX2(), line.getY2());
}
// destructor
Line::~Line() {
  delete p1;
  delete p2;
}
// overloaded assignment = operator
const Line &Line::operator=(const Line &rhs) {
  if (this != &rhs) {
    delete p1;
    delete p2;
    init(rhs.getX1(), rhs.getY1(), rhs.getX2(), rhs.getY2());
  }
  return *this; // always return *this from =
}
int Line::getX1() const { return p1->getX(); }
int Line::getY1() const { return p1->getY(); }
int Line::getX2() const { return p2->getX(); }
int Line::getY2() const { return p2->getY(); }

// Write the length function using the distance function
double Line::length() const { return p1->distance(*p2); }

// Write the slope function
double Line::slope() const {
  return (this->p2->getY() - this->p1->getY()) / (double)(this->p2->getX() - this->p1->getX());
}

void Line::translate(int dx, int dy) {
  p1->translate(dx, dy);
  p2->translate(dx, dy);
}
```