```
using namespace std;
#include <iostream>
#include<cmath>//required for length calculation

//function definitions
void testAdditionalFunctions();
void requiredOutput();
void testCoreFunctions();

//classes
class point2d {
private:
    int x;
    int y;
public:
    //getters
    int getX() {
        return x;
    };
    int getY() {
        return y;
    };
    //setters
    int setX(int input) {
        x = input;
    };
    int setY(int input) {
        y = input;
    };

    //overloaded opperators
    point2d operator+(const point2d &other) {
        return point2d(x + other.x, y + other.y);
    }
    point2d operator-(const point2d& other) {
        return point2d(x - other.x, y - other.y);
    }
    bool operator==(const point2d& rhs) {
        return((x == rhs.x) && (y == rhs.y));
    }
    bool operator!=(const point2d& rhs) {
        return !((x == rhs.x) && (y == rhs.y));
    }
    //default constructor
    point2d():x(0),y(0){}

    //parameterized constructor
    point2d(int X, int Y):x(X), y(Y) {};

    //copy constructor
    point2d(const point2d& obj):x(obj.x),y(obj.y) {};

    // Overload << and >> as friend functions
    friend ostream& operator<<(ostream& out, const point2d& line);
    friend istream& operator>>(istream& in, point2d& line);


};
class lineSegment {
private:
    const point2d defaultStart;
    const point2d defaultEnd;
    point2d start;
    point2d end;
public:
    //default constructor
    lineSegment() :start(defaultStart), end(defaultEnd) {}
    //parameterized constructor
    lineSegment(point2d START, point2d END) :start(START), end(END) {};
    //copy constructor
    lineSegment(const lineSegment& obj) :start(obj.start), end(obj.end) {};

    //getters and setters
    int getStartX() {
        return start.getX();
    }
    int getStartY() {
        return start.getY();
    }
    int getEndX() {
        return end.getX();
    }
    int getEndY() {
        return end.getY();
    }
    void setStartX(int X) {
        start.setX(X);
    }
    void setStartY(int Y) {
        start.setY(Y);
    }
    void setEndX(int X) {
        end.setX(X);
    }
    void setEndY(int Y) {
        end.setY(Y);
    }

    //misc functions
    float calcSlope() {
        return((end.getY() - start.getY()) / (end.getX() - start.getX()));
    }
    float calcLength() {
        return sqrt(((end.getY() - start.getY()) ^ 2) + ((end.getX() - start.getX()) ^ 2));
    }

    //overloaded opperators
    lineSegment operator+(const lineSegment& other) {
        return lineSegment(start + other.start, end + other.end);
    }
    lineSegment operator-(const lineSegment& other) {
        return lineSegment(start - other.start, end - other.end);
    }
    bool operator==(const lineSegment& rhs) {
        return((start == rhs.start) && (start == rhs.start));
    }
    bool operator!=(const lineSegment& rhs) {
        return !((start == rhs.start) && (end == rhs.end));
    }
    // Overload << and >> as friend functions
    friend ostream& operator<<(ostream& out, const lineSegment& point);
    friend istream& operator>>(istream& in, lineSegment& point);
};

// Overload << for point output
ostream& operator<<(ostream& out, const point2d& point)
{
    out << "[" << point.x << " " << point.y << "]";
    return out;
}
// Overload >> for point input
istream& operator>>(istream& in, point2d& point)
```

```
{
    in >> point.x >> point.y;
    return in;
}
//overload << for line output
ostream& operator<<(ostream& out, const lineSegment& line)
{
    out << "point1:" << line.start << " , point2: " << line.end << "" << endl;
    return out;
}
// Overload >> for line input
istream& operator>>(istream& in, lineSegment& line)
{
    in >> line.start >> line.end;
    return in;
}

int main()
{
    requiredOutput();
    cout << endl;
    cout << "----------------------------------------------------------------------------------------------------------"<<endl;
    testCoreFunctions();
    cout << endl;
    cout << "----------------------------------------------------------------------------------------------------------"<<endl;
    testAdditionalFunctions();
}

//create output on assignment
void requiredOutput() {
    point2d a(3, 4);
    point2d b(1, 2);

    point2d c = a + b;
    point2d d = a - b;

    if (c == point2d(4, 6)) {
        cout << "Math still works.\n";
    }

    cout << a << endl;
}
//test all functions required by the assignment(constructors,+,-,++,!=, <<, and >>)
void testCoreFunctions() {
    point2d testPoint(3, 5);
    cout<<"test Point:" << testPoint.getX() << testPoint.getY() << endl;

    point2d defaultPoint;
    cout<<"default point(default constructor):" << defaultPoint.getX() << defaultPoint.getY() <<
endl;

    point2d copyPoint(testPoint);
    cout<<"copy of test point(copy constructor):" << copyPoint.getX() << copyPoint.getY() <<
endl;

    point2d pointA = testPoint + copyPoint;
    cout<<"add test point and copy point:" << pointA.getX()<<pointA.getY() << endl;

    point2d pointS = testPoint - copyPoint;
    cout<<"subtract test point and copy point" << pointS.getX()<<pointS.getY() << endl;


    if (testPoint == copyPoint) {
        cout << "(test==)points are equal" << endl;
    }
    else {
        cout << "(test==)points are not equal" << "endl";
    }

    if (testPoint != copyPoint) {
        cout << "(test!=)points are not equal" << endl;
    }
    else {
        cout << "(test!=)points are equal" << endl;
    }

    point2d IOpoint;
    cout << "enter your own point(x,y): " << endl;
    cin >> IOpoint;
    cout << "your point is: " << IOpoint << endl;
}
//test line segment functions(add, subtract, slope, length)
void testAdditionalFunctions() {
    point2d point1;
    point2d point2(5, 30);

    lineSegment line1;
    lineSegment line2(point1, point2);

    lineSegment copyLine(line2);
    lineSegment line3 = line2 + copyLine;
    lineSegment IOline;
    //output all info for one line
    cout << "line info:" << endl;
    cout << "startX:"<<line2.getStartX() << endl;
    cout << "startY:" << line2.getStartY() << endl;
    cout << "endX:" << line2.getEndX() << endl;
    cout << "endY:" << line2.getEndY() << endl;
    cout << "slope:" << line2.calcSlope() << endl;
    cout << "length:" << line2.calcLength() << endl;
    //test overloaded operators
    cout << "add lines:" << line3;
    cout << "subtract lines: " << line3 - line2;
    cin >> IOline;
    cout << IOline;
}
```