/\*  Assignment:     Points Class

    Requirements:   implement the Point class from the book

                    Data Structures and Other Objects Using C++ by

                    Main and Savitch, and add the methods

                    translate, scale, rotate, and shear

    Author:         Ryan Wood

    Created On:     January 21, 2018

    Updated On:     January 28, 2018

\*/

/\*Point Header file\*/

#ifndef POINT\_H

#define POINT\_H

#include <iostream>

#include <fstream>

class Point

{

    public:

        /\*  Function:   Point()

            Purpose:    default constructor. Initializes x and y

                        coordinates to 0

            Return:     a Point instance

        \*/

        Point();

        /\*  Function:   Point(double, double)

            Purpose:    constructore for the Point class. initializes

                        x to dXpos and y to dYpos

            Parameters: the x and y positions

            Return:     an instance of a Point

        \*/

        Point(double dXpos, double dYpos);

        /\*  Function:   shift(double, double)

            Purpose:    moves the point by the given x amount and the

                        given y amount along their respective axis

            Parameters: the x amount to shift,

                        the y amount to shift

        \*/

        void shift(double xAmount, double yAmount);

        /\*  Function:   rotate90()

            Purpose:    rotates the Point clockwise by 90 degreees

        \*/

        void rotate90();

        /\*  Function:   getX() const

            Purpose:    retrieves the Point's value for the x coordinate

            Return:     the x coordinate

        \*/

        double getX() const;

        /\*  Function:   getY()

            Purpose:    retrieves the value for the Point's y coordinate

            Return:     the y coordinate

        \*/

        double getY() const;

        /\*  Function:   translate(double, double)

            Purpose:    this function does the same thing as function shift,

                        adding the given x and y amounts respectively to

                        the x and y coordinates of the point

            Parameters: the x amount to move, the y amount to move

        \*/

        void translate(double, double);

        /\*  Function:   scale(double, double)

            Purpose:    multiplies the x and y coordinates of the point by

                        the given amount for x and y respectively

            Parameters: the x amount to scale, the y amounbt to scale

        \*/

        void scale(double, double);

        /\*  Function:   rotate(double)

            Purpose:    rotates the point by the given amount of degrees

            Parameters: the amount in degrees to rotate

        \*/

        void rotate(double);

        /\*  Function:   shearX(double)

            Purpose:    shifts x by the given amount along the x axis

            Parameters: the amount to shift along the x axis

        \*/

        void shearX(double);

        /\*  Function:   shearY(double)

            Purpose:    shifts the point's y coordinate by the given amount

            Parameters: the amount to shift along the y axis

        \*/

        void shearY(double);

        /\*  Function:   operator <<(istream&, Point&)

            Purpose:    overloads the input operator, setting the x and

                        y member variables of the given Point to be the

                        next two double values given to the input stream

            Parameters: a reference to the input stream, a reference to the Point

            Return:     a reference to the input stream

        \*/

        friend std::istream& operator >>(std::istream &in, Point &target);

    private:

        double getRadians(double degrees) const;

        double x;

        double y;

};

/\*  Function:   distance(const Point&, const Point&)

    Purpose:    determines the distance between the two

                given points using pythagorean's theorem

    Parameters: the first point, the second point

    Return:     the distance between the points

\*/

double distance(const Point &pntA, const Point &pntB);

/\*  Function:   middle(const Point&, const Point&)

    Purpose:    determines the midpoint between the given points,

                constructs a Point object having those coordinates

                and returns it

    Parameters: the first point, the second point

    Return:     the Point in the middle

\*/

Point middle(const Point &pntA, const Point &pntB);

/\*  Function:   operator ==(const Point&, const Point&)

    Purpose:    determines whether the given points are equal,

                overloading the == operator

    Parameters: the first point, the second point

    Return:     whether the points have the same coordinates

\*/

bool operator ==(const Point &pntA, const Point &pntB);

/\*  Function:   operator !=(const Point&, const Point&)

    Purpose:    determines whetheer the given points are not equal,

                overloading the != operator

    Parameters: the first point, the second point

    Return:     whether the two points do not hae the same coordinates

\*/

bool operator !=(const Point &pntA, const Point &pntB);

/\*  Function:   operator +(const Point, const Point)

    Purpose:    adds the x and y coordinates of the two given points,

                and constructs a point having the new coordinates,

                overloading the + operator

    Parameters: the first point, the second point

    Return:     A point having coordinates that are the sum of the given points

\*/

Point operator +(const Point &pntA, const Point &pntB);

/\*  Function:   operator -(const Point&, const Point&)

    Purpose:    subtracts pointB from pointA and returns

                a new point with the x and y values

                having the results

    Paramters:  the first point, the second point

    Return:     the subtraction result Point instance

\*/

Point operator -(const Point &pntA, const Point &pntB);

/\*  Function:   operator >(const Point &, const Point&)

    Purpose:    determines whether the first point is

                farther from the origin Point(0,0) than

                the second point.

    Parameters: the first point, the second point

    Return:     whether the first point is futher from 0,0

\*/

bool operator >(const Point &pntA, const Point &pntB);

/\*  Function:   operator <(const Point &, const Point&)

    Purpose:    determines whether the first point is

                closer to the origin Point(0,0) than

                the second point.

    Parameters: the first point, the second point

    Return:     whether the first point is closer to 0,0

\*/

bool operator <(const Point &pntA, const Point & pntB);

/\*  Function:   operator <=(const Point&, const Point&)

    Purpose:    determines whether the first point is

                either equal to the second point or closer

                to the origin than the second point

    Paramters:  the first point, the second point

    Return:     whether the points are equal or

                the first point is less than the second point

\*/

bool operator <=(const Point &pntA, const Point &pntB);

/\*  Function:   operator >=(const Point&, const Point&)

    Purpose:    determines whether the first point is

                either equal to the second point or further

                from the origin than the second point

    Paramters:  the first point, the second point

    Return:     whether the points are equal or

                the first point is greater than the second point

\*/

bool operator >=(const Point &pntA, const Point &pntB);

/\*  Function:   operator <<(ostream&, Point)

    Purpose:    overloads the output operator and outputs

                the x and y coordnates formatted to "(x, y)"

    Parameters: a reference to the output stream, the Point to print

    Return:     a reference to the output stream

\*/

std::ostream& operator <<(std::ostream &out, const Point &source);

/\*  Function:   operator <<(ofstream&, Point)

    Purpose:    overloads the insertion operator using the ofstream

                data output to file provided as the first argument.

                Writes the given point to the file in format "(x, y)"

    Parameters: a reference to the file output stream, the Point to write

    Return:     a reference to the file output stream

\*/

std::ofstream& operator <<(std::ofstream &out, const Point &source);

#endif

/\*  Assignment:     Points Class

    Requirements:   implement the Point class from the book

                    Data Structures and Other Objects Using C++ by

                    Main and Savitch, and add the methods

                    translate, scale, rotate, and shear

    Author:         Ryan Wood

    Created On:     January 21, 2018

    Updated On:     January 28, 2018

\*/

/\* Point class implementation file\*/

#include "Point.h"

#include <iostream>

#include <fstream>

#include <math.h>

using namespace std;

const double PI = 3.1415;

Point::Point()

{

    x = 0.0;

    y = 0.0;

}

Point::Point(double dXpos, double dYpos)

{

    x = dXpos;

    y = dYpos;

}

void Point::shift(double xAmount, double yAmount)

{

    x+= xAmount;

    y+= yAmount;

}

void Point::rotate90()

{

    double tmpX = 0.0;

    double tmpY = 0.0;

    tmpX = y;

    tmpY = -x;

    x = tmpX;

    y = tmpY;

}

double Point::getX() const

{

    return x;

}

double Point::getY() const

{

    return y;

}

void Point::translate(double dXval, double dYval)

{

    shift(dXval, dYval);

}

void Point::scale(double dXval, double dYval)

{

    x\*= dXval;

    y\*= dYval;

}

void Point::rotate(double degrees)

{

    double sinDeg  = 0.0;

    double cosDeg  = 0.0;

    double radians = 0.0;

    double tmpX    = 0.0;

    double tmpY    = 0.0;

    radians = getRadians(degrees);

    tmpX = (x \* cos(radians) - y \* sin(radians));

    tmpY = (x \* sin(radians) - y \* cos(radians));

    x = tmpX;

    y = tmpY;

}

void Point::shearX(double dAmount)

{

    x+= dAmount;

}

void Point::shearY(double dAmount)

{

    y+= dAmount;

}

double Point::getRadians(double degrees) const

{

    double radians = 0.0;

    radians = (degrees \* PI) / 180 ;

    return radians;

}

double distance(const Point &pntA, const Point &pntB)

{

    double a  = 0.0;

    double b  = 0.0;

    double c2 = 0.0;

    a = pntA.getX() - pntB.getX();

    b = pntA.getY() - pntB.getY();

    //pythagorean theorem calculates the square of the distance between the points

    c2 = (a\*a) + (b\*b);

    return sqrt(c2);

}

Point middle(const Point &pntA, const Point &pntB)

{

    double xMid = 0.0;

    double yMid = 0.0;

    xMid = (pntA.getX() + pntB.getX()) /2;

    yMid = (pntA.getY() + pntB.getY()) /2;

    Point mid(xMid, yMid);

    return mid;

}

bool operator ==(const Point &pntA, const Point &pntB)

{

    bool isEqual = false;

    if(pntA.getX() == pntB.getX() &&

       pntA.getY() == pntB.getY())

        isEqual = true;

    return isEqual;

}

bool operator !=(const Point &pntA, const Point &pntB)

{

    return !(pntA == pntB);

}

Point operator +(const Point &pntA, const Point &pntB)

{

    double xSum = 0.0;

    double ySum = 0.0;

    xSum = pntA.getX() + pntB.getX();

    ySum = pntA.getY() + pntB.getY();

    Point sum(xSum, ySum);

    return sum;

}

Point operator -(const Point &pntA, const Point &pntB)

{

    double xDiff = 0.0;

    double yDiff = 0.0;

    xDiff = pntA.getX() - pntB.getX();

    yDiff = pntA.getY() - pntB.getY();

    Point diff(xDiff, yDiff);

    return diff;

}

bool operator >(const Point &pntA, const Point &pntB)

{

    double distA = 0.0;

    double distB = 0.0;

    Point origin(0,0);

    distA = distance(pntA, origin);

    distB = distance(pntB, origin);

    return (distA > distB);

}

bool operator <(const Point &pntA, const Point & pntB)

{

    double distA = 0.0;

    double distB = 0.0;

    Point origin(0, 0);

    distA = distance(pntA, origin);

    distB = distance(pntB, origin);

    return (distA < distB);

}

bool operator <=(const Point &pntA, const Point &pntB)

{

    return ((pntA < pntB) || pntA == pntB);

}

bool operator >=(const Point &pntA, const Point &pntB)

{

    return ((pntA > pntB) || pntA == pntB);

}

ostream& operator <<(ostream &out, const Point &source)

{

    out << "(" << source.getX() << ", " << source.getY() << ")";

    return out;

}

istream& operator >>(istream &in, Point &target)

{

    in >> target.x >> target.y;

    return in;

}

ofstream& operator <<(ofstream &out, const Point &source)

{

    out << "(" << source.getX() << ", " << source.getY() << ")";

    return out;

}

/\*  Assignment:     Points Class

    Requirements:   implement the Point class from the book

                    Data Structures and Other Objects Using C++ by

                    Main and Savitch, and add the methods

                    translate, scale, rotate, and shear

    Author:         Ryan Wood

    Created On:     January 21, 2018

    Updated On:     January 28, 2018

\*/

/\* Point class Main test file\*/

#include "Point.h"

#include <fstream>

#include <iostream>

#include <iomanip>

using namespace std;

ofstream outFile("Point.out");

int main()

{

    outFile << showpoint << setprecision(2);

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    Requirement 1. set up box with four points:

                   (2,2), (2, 5), (4, 5), (4, 2)

                   and print out the points

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    Point point1(2, 2);

    Point point2(2, 5);

    Point point3(4, 5);

    Point point4(4, 2);

    outFile << point1 << endl;

    outFile << point2 << endl;

    outFile << point3 << endl;

    outFile << point4 << endl << endl;

    /\*\*\*\* Requirement 2. translate x = 2 and y = 1 distance \*\*\*\*\*\*/

    point1.translate(2, 1);

    point2.translate(2, 1);

    point3.translate(2, 1);

    point4.translate(2, 1);

    outFile << "Translated by x = 2, y = 1" << endl;

    outFile << point1 << endl;

    outFile << point2 << endl;

    outFile << point3 << endl;

    outFile << point4 << endl << endl;

    /\*\*\*\*\*\*\* Requirement 3. scale box by x = 2 and y = 0.5 \*\*\*\*\*\*\*\*/

    point1.scale(2, 0.5);

    point2.scale(2, 0.5);

    point3.scale(2, 0.5);

    point4.scale(2, 0.5);

    outFile << "Scaled by x = 2 and y = 0.5" << endl;

    outFile << point1 << endl;

    outFile << point2 << endl;

    outFile << point3 << endl;

    outFile << point4 << endl << endl;

    /\*\*\*\*\*\* Requirement 4. Using original box point values, rotate 30 degrees,

                            then rotate another 60 degrees\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    Point point5(2, 2);

    Point point6(2, 5);

    Point point7(4, 5);

    Point point8(4, 2);

    point5.rotate(30);

    point6.rotate(30);

    point7.rotate(30);

    point8.rotate(30);

    outFile << "Original box rotated 30 degrees" << endl;

    outFile << point5 << endl;

    outFile << point6 << endl;

    outFile << point7 << endl;

    outFile << point8 << endl << endl;

    point5.rotate(60);

    point6.rotate(60);

    point7.rotate(60);

    point8.rotate(60);

    outFile << "Box rotated another 60 degrees" << endl;

    outFile << point5 << endl;

    outFile << point6 << endl;

    outFile << point7 << endl;

    outFile << point8 << endl << endl;

    /\*\*\*\*\* Requirement 5. shear in x direction 1.5 \*\*\*\*\*\*/

    point5.shearX(1.5);

    point6.shearX(1.5);

    point7.shearX(1.5);

    point8.shearX(1.5);

    outFile << "Box sheared by x = 1.5" << endl;

    outFile << point5 << endl;

    outFile << point6 << endl;

    outFile << point7 << endl;

    outFile << point8 << endl << endl;

    /\*\*\*\*\*\* Requirement 6. shear in y direction 1.7\*\*\*\*\*\*\*/

    point5.shearY(1.7);

    point6.shearY(1.7);

    point7.shearY(1.7);

    point8.shearY(1.7);

    outFile << "Box sheared by y = 1.7" << endl;

    outFile << point5 << endl;

    outFile << point6 << endl;

    outFile << point7 << endl;

    outFile << point8 << endl << endl;

    /\*\*\*\*\*\*\*\*\* Requirement 7. Test functions from the book for Point \*\*\*\*/

    Point ryansPoint(1, 2.5);

    Point joesPoint(-3, -7.3);

    //shift test

    outFile << "Ryan's point = " << ryansPoint << endl;

    outFile << "Joe's Point = " << joesPoint << endl;

    ryansPoint.shift(-2, 2);

    //add two points test

    outFile << "Ryan's Point shifted by (x = -1, y = 2) = " << ryansPoint << endl;

    Point ryanAndJoePoint = ryansPoint + joesPoint;

    outFile << "Ryan and Joe's Points added = " << endl;

    outFile << ryansPoint << " + " << joesPoint << " = " << ryanAndJoePoint << endl << endl;

    //rotate90 function test

    ryansPoint.rotate90();

    outFile << "Ryan's point rotated 90 degrees = " << ryansPoint << endl << endl;

    ryansPoint.rotate90();

    outFile << "Ryan's point rotated another 90 degrees = " << ryansPoint << endl << endl;

    //middle function test

    outFile << "Joe's Point = " << joesPoint << " and Ryan's Point = " << ryansPoint << endl;

    Point midPoint = middle(joesPoint, ryansPoint);

    outFile << "The midpoint between these points is " << midPoint << endl;

    //distance function test

    outFile << "The distance between Ryan and Joes Points is " << distance(ryansPoint, joesPoint) << endl << endl;

    Point newPoint(4, 5);

    Point otherPoint(4, 5);

    //equality operator test

    if(newPoint == otherPoint)

        outFile << "newPoint and otherPoint are equal" << endl;

    else

        outFile << "== function not working " << endl;

    //less than or equal to operator test

    if(newPoint <= otherPoint)

        outFile << "newPoint is less than or equal to otherPoint" << endl;

    else

        outFile << "<= function not working " << endl;

    //shift point by 1

    outFile << "otherPoint shifted by 1" << endl;

    otherPoint.shift(1, 1);

    //not equal operator test

    if(newPoint != otherPoint)

        outFile << "newPoint is now not equal to otherPoint" << endl;

    else

        outFile << "!- function not working" << endl;

    //greater than or equal to operator test

    if(otherPoint >= newPoint)

        outFile << "otherPoint is greater than or equal to newPoint" << endl;

    else

        outFile << ">= function not working" << endl;

    outFile.close();

    return 0;

}

(2.0, 2.0)

(2.0, 5.0)

(4.0, 5.0)

(4.0, 2.0)

Translated by x = 2, y = 1

(4.0, 3.0)

(4.0, 6.0)

(6.0, 6.0)

(6.0, 3.0)

Scaled by x = 2 and y = 0.5

(8.0, 1.5)

(8.0, 3.0)

(12., 3.0)

(12., 1.5)

Original box rotated 30 degrees

(0.73, -0.73)

(-0.77, -3.3)

(0.96, -2.3)

(2.5, 0.27)

Box rotated another 60 degrees

(1.0, 1.0)

(2.5, 1.0)

(2.5, 2.0)

(1.0, 2.0)

Box sheared by x = 1.5

(2.5, 1.0)

(4.0, 1.0)

(4.0, 2.0)

(2.5, 2.0)

Box sheared by y = 1.7

(2.5, 2.7)

(4.0, 2.7)

(4.0, 3.7)

(2.5, 3.7)

Ryan's point = (1.0, 2.5)

Joe's Point = (-3.0, -7.3)

Ryan's Point shifted by (x = -1, y = 2) = (-1.0, 4.5)

Ryan and Joe's Points added =

(-1.0, 4.5) + (-3.0, -7.3) = (-4.0, -2.8)

Ryan's point rotated 90 degrees = (4.5, 1.0)

Ryan's point rotated another 90 degrees = (1.0, -4.5)

Joe's Point = (-3.0, -7.3) and Ryan's Point = (1.0, -4.5)

The midpoint between these points is (-1.0, -5.9)

The distance between Ryan and Joes Points is 4.9

newPoint and otherPoint are equal

newPoint is less than or equal to otherPoint

otherPoint shifted by 1

newPoint is now not equal to otherPoint

otherPoint is greater than or equal to newPoint