Bachelor of Computer Science
MIDTERM

Subject Code: COS30008
Subject Title: Data Structures and Patterns
Assignment number and title: Midterm, Convex Hull
Due date: Check Canvas

Your name:___Ngo Cong Thanh_____ **Your student ID: 103433609___**

Marker's comments:

Problem	Marks	Obtained
1	20	
2	30	
3	50 (buildConvexHull:20)	
Total	100	

Detailed comments:

Point2D.cpp:

```
#include "Point2D.h"
#include <cmath>

using namespace std;

static const double gEpsilon = 0.0001;
static const Point2D gCoordOrigin;

double Point2D::directionTo(const Point2D& aOther) const {
	Point2D temp = *this;
	return atan2(temp.fPosition.getX() - aOther.fPosition.getY(), temp.fPosition.getX()
- aOther.fPosition.getY());
}

double Point2D::magnitudeTo(const Point2D& aOther) const {
	Point2D temp = *this;
	return (*this - aOther).magnitude();
}

Point2D::Point2D() : fId(""), fPosition(0,0), fOrigin(&gCoordOrigin) {

}

Point2D::Point2D(const std::string& aId, double aX, double aY) : fId(aId), fPosition(aX,
aY), fOrigin(&gCoordOrigin) {

}

Point2D::Point2D(istream& aIStream) : fOrigin(&gCoordOrigin){
	double lX, lY;
	aIStream >> fId >> lX >> lY;

	fPosition.setX(lX);
	fPosition.setY(lY);
}

const string& Point2D::getId() const {
	return fId;
}

void Point2D::setX(const double& aX) {
	fPosition.setX(aX);
}
const double Point2D::getX() const {
	return fPosition.getX();
}
void Point2D::setY(const double& aY) {
	fPosition.setY(aY);
}
const double Point2D::getY() const {
	return fPosition.getY();
}
void Point2D::setOrigin(const Point2D& aPoint) {
	Point2D temp = *this;
	temp.fOrigin = aPoint.fOrigin;
}
const Point2D& Point2D::getOrigin() const {
	return *fOrigin;
}
```

```
Vector2D Point2D::operator-(const Point2D& aRHS) const {
	return Vector2D(fPosition.getX() - aRHS.getX(), fPosition.getY() - aRHS.getY());
}

double Point2D::direction() const {
	return directionTo(*fOrigin);
}

double Point2D::magnitude() const {
	return magnitudeTo(*fOrigin);
}

bool Point2D::isCollinear(const Point2D& aOther) const {
	double lResult = abs(direction() - aOther.direction());
	return lResult <= gEpsilon && lResult >= 0 || lResult <= 3.1416 && lResult >=
3.1415;
}

bool Point2D::isClockwise(const Point2D& aP0, const Point2D& aP2) const {
	return Vector2D(*this - aP0).cross(Vector2D(aP2 - aP0)) > 0;
}

bool Point2D::operator<(const Point2D& aRHS) const {
	Vector2D lResult = *this - aRHS;
	if (lResult.getY() <= -gEpsilon || lResult.getY() == 0 && lResult.getX() <= -
gEpsilon)
		return true;
	return false;
}

std::ostream& operator<<(std::ostream& aOStream, const Point2D& aObject) {
	aOStream << aObject.fId << ": (" << aObject.fPosition.getX() << ", " <<
aObject.fPosition.getY() << ")";
	return aOStream;
}

std::istream& operator>>(std::istream& aIStream, Point2D& aObject) {
	aObject = Point2D(aIStream);
	return aIStream;

}
```

Vector2D.cpp:

```
#include "Vector2D.h"
#include <math.h>

using namespace std;

Vector2D::Vector2D(double aX, double aY) : fX(aX), fY(aY) {

}

void Vector2D::setX(double aX) {
	fX = aX;
}

double Vector2D::getX() const {
	return fX;
}
```

```
void Vector2D::setY(double aY) {
	fY = aY;
}
double Vector2D::getY() const {
	return fY;
}

Vector2D Vector2D::operator+(const Vector2D& aRHS) const {
	Vector2D temp = *this;
	return  Vector2D(temp.fX + aRHS.fX, temp.fY + aRHS.fY);
}

Vector2D Vector2D::operator-(const Vector2D& aRHS) const {
	Vector2D temp = *this;
	return  Vector2D(temp.fX - aRHS.fX, temp.fY - aRHS.fY);
}

double Vector2D::magnitude() const {
	return sqrt(pow(fX, 2) + pow(fY, 2));
}

double Vector2D::direction() const {
	return atan(fY / fX);
}

double Vector2D::dot(const Vector2D& aRHS) const {
	Vector2D temp = *this;
	return  temp.fX*aRHS.fX + temp.fY*aRHS.fY;
}

double Vector2D::cross(const Vector2D& aRHS) const {
	Vector2D temp = *this;
	return  temp.fY * aRHS.fX - temp.fX * aRHS.fY ;
}

double Vector2D::angleBetween(const Vector2D& aRHS) const {
	Vector2D temp = *this;
	return atan2(temp.fX * aRHS.fX + temp.fY * aRHS.fY, temp.fY * aRHS.fX - temp.fX *
aRHS.fY);
}

ostream& operator<<(ostream& aOutStream, const Vector2D& aObject) {
	aOutStream << "[" << aObject.fX << "," << aObject.fY << "]" << endl;
	return aOutStream;
}
istream& operator>>(istream& aInStream, Vector2D& aObject) {
	aInStream >> aObject.fX;
	aInStream >> aObject.fY;
	return aInStream;

}
```

Point2DSet.cpp:

```
#include "Point2DSet.h"
#include <fstream>
#include <algorithm>

using namespace std;


static const double gEpsilon = 0.0001;
using Iterator = std::vector<Point2D>::const_iterator;
```

```
void Point2DSet::add(const Point2D& aPoint) {
	fPoints.push_back(aPoint);
}

void Point2DSet::add(Point2D&& aPoint) {
	fPoints.push_back(aPoint);
}

void Point2DSet::removeLast() {
	fPoints.pop_back();
}

bool Point2DSet::doesNotTurnLeft(const Point2D& aPoint) const {
	return aPoint.isClockwise(fPoints[size() - 2], fPoints[size() - 1]);
}

void Point2DSet::populate(const std::string& aFileName) {
	int lPointCount;
	Point2D lPoint2D;

	ifstream aInStream(aFileName, ifstream::in);
	aInStream >> lPointCount;
	for (int i = 0; i < lPointCount; i++)
	{
		aInStream >> lPoint2D;
		add(lPoint2D);
	}
}

bool orderByCoordinates(const Point2D& aLeft, const Point2D& aRight) {
	return aLeft < aRight;
}

bool orderByPolarAngle(const Point2D& aLHS, const Point2D& aRHS) {
	if (aLHS.isCollinear(aRHS)) {
		return aLHS.magnitude() - aRHS.magnitude() <= -gEpsilon;
	}

	return aLHS.direction() - aRHS.direction() <= -gEpsilon;
}

void Point2DSet::sort(Comparator aComparator) {
	stable_sort(fPoints.begin(), fPoints.end(), aComparator);
}

void Point2DSet::buildConvexHull(Point2DSet& aConvexHull) {
	//sort bt the coordinates
	sort(orderByCoordinates);

	//Assign new value for new Origin
	for (Point2D& point2D : fPoints)
	{
		point2D.setOrigin(fPoints[0]);
	}

	//sort by polar angle
	sort(orderByPolarAngle);

	//add 3 first point
	for (size_t i = 0; i < 3; i++)
	{
		aConvexHull.add(move(fPoints[i]));
	}
```

```
        //Graham Scan Iterator
        for (size_t i = 3; i < size(); i++)
        {
                while (aConvexHull.doesNotTurnLeft(fPoints[i]))
                        aConvexHull.removeLast();
                aConvexHull.add(move(fPoints[i]));
        }
}

size_t Point2DSet::size() const {
        return fPoints.size();
}

void Point2DSet::clear() {
        fPoints.clear();
}

const Point2D& Point2DSet::operator[](size_t aIndex) const {
        return fPoints[aIndex];
}

Iterator Point2DSet::begin() const {
        return fPoints.begin();
}

Iterator Point2DSet::end() const {
        return fPoints.end();

}
```