Assignment 1

You will complete the code for a number of C++ classes that model a number of (up to) two-dimensional objects in the standard x-y coordinate system (see the first picture in [this](https://en.wikipedia.org/wiki/Cartesian_coordinate_system)), together with a number of transformations that can be applied to them, and some other operations.

**Overview and the class hierarchy**

A number of different classes are used to model different geometric shapes, all derived from a common base class called Shape. The class hierarchy is as shown in this diagram:

Shape

|

-----------------------

| | |

Point LineSegment TwoDShape

|

-----------

| |

Rectangle Circle

In addition to attributes specific to each type of objects, all objects have a "depth". The *depth* of an object is a non-negative integer indicating which "layer" or "plane" it is in: imagine that there are multiple "planes" in the coordinate system, or like "layers" in some photo-editing software; see figure below.

A picture containing diagram

Description automatically generated

**Constructors of Shape and its subclasses**

The following explains each of the classes and their constructors:

* Shape(int d):  
  This is the base class and should be abstract (no object of this class should be constructed). Nevertheless, it has a constructor with a parameter d specifying the depth of the object. If d is negative, the constructor should throw a std::invalid\_argument exception.
* Point(float x, float y, int d=0):  
  This class models a point, which is a zero-dimensional object specified by its x-coordinate, y-coordinate and its depth. The =0 here indicates that d is a *default argument*, namely that it will by default set to 0 if it is not supplied when the constructor is invoked. (See e.g. <https://www.geeksforgeeks.org/default-arguments-c/> for some explanation.) You don't need to do anything about it (and it has nothing to do with pure virtual functions!)
* LineSegment(const Point& p, const Point& q):   
  This class models a line segment, which is a one-dimensional object, the portion of a straight line passing between the two points. Here we model "axis-aligned" line segments only, i.e. the line segment must be parallel to either the x-axis or the y-axis. The constructor specifies the two endpoints of the line segment. The two points p and q are not necessarily given in any order; the line segment has no "direction". The two endpoints should have the same depth, and the resulting line segment has a depth equal to that of its endpoints. If the two endpoints have different depths, or if both their x- and y-coordinates are different (line not horizontal/vertical), or if both their x- and y-coordinates are the same (the two endpoints coincide), the constructor should throw a std::invalid\_argument exception.
* TwoDShape(int d):  
  The class models any two-dimensional object, and should be an abstract class. The parameter d specifies the depth of the object.
* Rectangle(const Point& p, const Point& q):

This class models a rectangle, which is a two-dimensional object specified by two **opposite** corners (not adjacent corners) p and q. Note that the two points could be either the top-left and bottom-right corners, or the top-right and bottom-left corners; and in no particular order. Only axis-aligned rectangles are allowed, which means all edges are parallel to either the x- or y-axis; thus two corners are sufficient to define a rectangle. For example, if two of the corners are (1,2) and (3,4), then the other two corners must be (3,2) and (1,4).

The two points p and q should have the same depth which is also the depth of the rectangle. If the two given points have different depths, or have the same x-coordinate and/or y-coordinate (which means they are on the same horizontal/vertical line or are even the same point, and no rectangle can be formed), the constructor should throw a std::invalid\_argument exception.

* Circle(const Point& c, float r):  
  This models a circle, specified by its centre point c and its radius r. The depth of the circle is the same as that of c. If the radius is 0 or negative, the constructor should throw a std::invalid\_argument exception.

**Other functions of Shape and its subclasses**

The Shape class (and all its subclasses) should support the following functions:

* int getDepth()  
  bool setDepth(int d)  
  Get/set the depth of the object. If d is negative, return false and do not update the depth.
* int dim():   
  Return the dimension (0, 1 or 2) of the object.
* void translate(float x, float y):   
  Translate, i.e. move, the whole object, to the right by a distance of x, and to the top by a distance of y. A negative x or y value means it will move to the left or the bottom, respectively.
* void rotate():   
  Rotate the object 90 degrees around its centre. Since all objects under consideration are "symmetric", it makes no difference whether it is rotated clockwise or anticlockwise.

For example, for a line segment with two endpoints (0,0) and (10,0), rotating will change its endpoints to (5,5) and (5,-5); for a rectangle with four corners (0,0), (10,0), (0,4) and (10,4), rotating will change the corners to (3,-3), (7,-3), (3,7) and (7,7). Rotation has no effect (but is still a valid operation) on Point or Circle.

* void scale(float f):   
  Scale up/down the size of the object by a factor f, relative to its centre. A factor f > 1 indicates the object becomes bigger, and a factor 0 < f < 1 indicates the object becomes smaller. If f is zero or negative, throw a std::invalid\_argument exception, and do not change the object. "Relative to its centre" means that the object's centre remains at the same position.

For example, if a rectangle with four corners (0,0), (0,10), (2,0) and (2,10) is scaled up by a factor of 2, the corners become (-1,-5), (3,15), (3,-5) and (-1,15). And a circle with centre (1,2) and radius 10, scaled by a factor f = 0.5, will still have centre (1,2) but the radius becomes 5. Scaling has no effect (but is still a valid operation) on Point.

* bool contains(const Point& p):

Return true or false indicating whether the point p is "inside" the current object (let's call it o here) or not. The depths of o and p are disregarded for the purpose of this function. That is, even if o and p have different depths, this function should return true if p is "inside" o as if they had the same depth.

If this object o is a Point, the function returns true if and only if o and p have the same x- and y-coordinates. If o is a line segment, it returns true if and only if p lies between the two endpoints of o, including the two endpoints themselves. If o is a rectangle or circle, it returns true if and only if p is on the inside or the boundary of the rectangle/circle. Just to remind you, the distance between two points (x1,y1) and (x2,y2) is given by the formula √( (x1-x2)2 + (y1-y2)2 ).

* Destructor, copy constructor and copy assignment operator are assumed to be not required (i.e. the default is adequate). If for some reason they are not adequate for your implementation then you should supply them.

The figures below illustrates how rotate and scale work on a rectangle: (left) blue rectangle rotates to become the orange one (and vice versa); (right) blue rectangle scaled up with f=2 to become the orange one (or the orange one scaled down with f=1/2 to become the blue one).

Chart, bar chart

Description automatically generated

**Other functions of Point**

In addition to those inherited from the parent class, the Point class should support the following operations:

* float getX(): return the x-coordinate of the point.
* float getY(): return the y-coordinate of the point.

**Other functions of LineSegment**

In addition to those inherited from the parent class, the LineSegment class should support the following operations:

* float getXmin(): return the x-coordinate of the left endpoint (or both endpoints if the line is vertical).
* float getXmax(): return the x-coordinate of the right endpoint (or both endpoints if the line is vertical).
* float getYmin(): return the y-coordinate of the bottom endpoint (or both endpoints if the line is horizontal).
* float getYmax(): return the y-coordinate of the top endpoint (or both endpoints if the line is horizontal).
* float length(): return the length of this line segment.

**Other functions of TwoDShape and its subclasses**

In addition to those inherited from the parent class, all TwoDShape objects should support the following operation:

* float area(): return the area of the object. (In case you need reminding, for rectangles it is the width multiplied by the height, and for circle it is πr2 where r is the radius and the constant PI is defined in the Shape class.)

**Other functions of Rectangle**

In addition to those inherited from the parent classes, the Rectangle class should support the following operations:

* float getXmin(): return the x-coordinate of the left edge of the rectangle.
* float getXmax(): return the x-coordinate of the right edge of the rectangle.
* float getYmin(): return the y-coordinate of the bottom edge of the rectangle.
* float getYmax(): return the y-coordinate of the top edge of the rectangle.

**Other functions of Circle**

In addition to those inherited from the parent classes, the Circle class should support the following operations:

* float getX(): return the x-coordinate of the centre.
* float getY(): return the y-coordinate of the centre.
* float getR(): return the radius.

**The Scene class**

This class stores a collection of shared pointers to some Shape objects, so that the objects can be "drawn" on the screen. It should support the following operations:

* void addObject(shared\_ptr<Shape> p):  
  Add the shared pointer p that points to some Shape (or its subclasses) object to this Scene object. After the call, the pointer p (and the object that it points to) must remain "intact", and both the caller of this function and this Scene object share the "ownership" of the object being pointed to. In other words, any changes made to one of them is reflected in the other. For example, in the following code:
* shared\_ptr<Rectangle> rp = make\_shared<Rectangle>(...);
* Scene s;
* s.addObject(rp);
* rp->translate(1,2);

After the rectangle pointed to by rp is translated, the rectangle added to s is translated as well.

* void setDrawDepth(int d):  
  Set the "drawing depth" to d, which means that when operator<< is called (see the next bullet point), it draws all objects at depth d or less. For example, if d=2, it draws all objects with depth 0, 1, 2, but not those with depth 3, 4, 5, etc. If this function is never called, operator<< should draw all objects of any depth.
* ostream& operator<<(ostream& out, const Scene& s):

Overloaded output stream redirection operator, to be implemented as a friend function (not a member function) of this class. It "draws" all objects to the screen, as follows. There is a rectangular "drawing area" of size defined in the constants WIDTH and HEIGHT in the Scene class. This function outputs to the stream a number of lines equal to HEIGHT, where each line has exactly WIDTH characters. These lines are joined by the newline ('\n') character, and the final line is followed by a newline character as well. The j-th line from the **bottom**, and the i-th character from the left in this line (count starts from 0), correspond to the point with integer coordinates (i,j). That is, the rectangular area corresponds to the coordinates from (0,0) in the bottom left, to (WIDTH-1, HEIGHT-1) in the top right.

Each character in the above lines is the character '\*', if the corresponding integer coordinates (i,j) is inside part of any object (as defined by the contains() function) and that object has depth no more than the drawing depth (see previous bullet point); otherwise it is the blank space ' ' character.

Effectively, all objects of different depths (that are at most the drawing depth) are "overlayed" on top of each other. Also note that some objects may be partially or wholly outside of the drawing area, and they (or those parts) are not drawn.

For example, with the default WIDTH and HEIGHT values of 20 and 60, a scene with a point (0,0), a line segment with endpoints (0,19) and (59,19), a rectangle with opposite corners (59,0) and (55,19), and a circle with centre (30,0) and radius 10, all with depth 0, may be drawn like this if the drawing depth is also 0:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\* \*\*\*\*\*

\*\*\*\*\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*

(yes, I know it doesn't look like a semi-circle...) Due to the floating point nature of the coordinates, this function will have the interesting "feature" that it only draws things with integer coordinates; for example a line segment between (1,1) and (1,5) will be drawn, while one between (1.5, 1) and (1.5, 5) will not.

In all places that require comparing floating point values, you can ignore the issue of floating-point inaccuracies. That is, you can just compare two floats with x==y.

**What needs to change and what cannot be changed**

All the classes have no member variables at the moment; you will need to decide what protected/private data members to add to the classes. There are multiple ways to represent the various geometric objects, and there is not necessarily a "best" way. For example, a rectangle may be represented by the coordinates of its four corners, or in fact only two opposite corners are enough; or the coordinates of its centre plus its height and width; or many other ways. You will also need some data structure (preferably from STL) to store the shared pointers in the Scene class.

The classes Shape, Point, LineSegment, TwoDShape, Rectangle and Circle, as well as their inheritance relationships, have been defined for you in the Geometry.h file. However, all the required member functions are only declared in the highest class in the class hierarchy where they are relevant. You will need to decide whether some subclasses should override some of the functions, where in the hierarchy should they be implemented, and whether each function should be made virtual. In addition, all non-leaf classes should be abstract; only instances of Point, LineSegment, Rectangle and Circle should be allowed to be created. This means each non-leaf class must have some pure virtual member functions.

Therefore, you will need to decide whether each function should have the virtual, override, final keywords and/or the = 0 pure specifier. You may need to duplicate some function declarations and/or their implementations (bodies) to some subclasses, or move them to some other classes (even when the comments say IMPLEMENT ME). Please see further notes in the marking criteria about this.

You must not change the existing public interface of the classes, other than in relation to inheritance, virtual-ness and pure-ness as explained above. You are allowed to add other public/protected/private member functions, should you want to.

**Submit only the files Geometry.h and Geometry.cpp**

**Files Provided**

* [Geometry.h](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/Geometry.h)
* [Geometry.cpp](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/Geometry.cpp)

These are the only two files you need to modify/submit. All code written by you should be in there.

* [main.cpp](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/main.cpp)

This is just an example that illustrates how the functions can be called.

* [GeometryTester.h](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/GeometryTester.h)
* [GeometryTester.cpp](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/GeometryTester.cpp)
* [GeometryTesterMain.cpp](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/GeometryTesterMain.cpp)

They are used for the execution testing part (see the next section).

* [makefile](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/makefile)

This is a makefile that will compile the main executable and the testing suite executable.

**Marking Criteria and Test Suite**

See [this separate page](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/ms.html) for the marking criteria.

To use the test suite (which is used in the execution testing part of the marking), simply type "make" in a linux terminal (with all the above files in the same folder). It will (assuming you did not break Geometry.h or Geometry.cpp) produce a GeometryTesterMain executable file. Run the program by typing

./GeometryTesterMain a

or

./GeometryTesterMain a b c d

which runs a single test case or multiple test cases respectively. We will also demonstrate its use in class.

The given Geometry.h and Geometry.cpp files (without your contributions) are already compilable with the test suite. It might even pass a few test cases. So, whatever you do, please don't break them...

The test suite may not cover all corner cases, so passing all test cases does not guarantee your program is 100% correct. Also, unfortunately we cannot isolate the testing of the various functions. For example to pass those test cases meant for LineSegment you need to also implement Point at least partially correctly.

Files provided :

[Geometry.h](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/Geometry.h)

#ifndef GEOMETRY\_H\_

#define GEOMETRY\_H\_

#include <iostream>

#include <memory>

class Point; // forward declaration

class Shape {

public:

// Default constructor, just to make this release version compilable.

// If your implementation is correct this should be removed

Shape();

// Constructor specifying the depth of the object.

// If d is negative, throw a std::invalid\_argument exception.

Shape(int d);

// Set depth of object to d. If d is negative, return false and

// do not update depth. Otherwise return true

bool setDepth(int d);

// Return the depth of object

int getDepth() const;

// Return the dimension of the object (0, 1 or 2)

int dim() const;

// Translate the object horizontally by x and vertically by y

void translate(float x, float y);

// Rotate the object 90 degrees around its centre

void rotate();

// Scale the object by a factor f relative to its centre.

// If f is zero or negative, throw a std::invalid-argument exception.

void scale(float f);

// Return true if the object contains p and false otherwise.

// Depths are ignored for purpose of comparison

bool contains(const Point& p) const;

// the constant pi

static constexpr double PI = 3.1415926;

protected:

private:

// add any protected/private member variables you need

};

class Point : public Shape {

public:

// Constructor. Depth defaults to 0

Point(float x, float y, int d=0);

// Return basic information (see assignment page)

float getX() const;

float getY() const;

private:

// add any member variables you need

};

class LineSegment : public Shape {

public:

// Constructor.

// If the two points have different depths, or have the same x- and

// y-coordinate, or if the line is not axis-aligned, throw a

// std::invalid\_argument exception

LineSegment(const Point& p, const Point& q);

// Return basic information (see assignment page)

float getXmin() const;

float getXmax() const;

float getYmin() const;

float getYmax() const;

// Return the length of the line segment

float length() const;

private:

// add any member variables you need

};

class TwoDShape : public Shape {

public:

// Default constructor.

// Similar comment to Student default constructor applies

TwoDShape();

// Constructor specifying the depth d

TwoDShape(int d);

// Return the area of the object

float area() const;

protected:

private:

// add any protected/private member variables you need

};

class Rectangle : public TwoDShape {

public:

// Constructor.

// If the two points have different depths, or have the same x-

// and/or y-coordinate, throw a std::invalid\_argument exception

Rectangle(const Point& p, const Point& q);

// Return basic information (see assignment page)

float getXmin() const;

float getYmin() const;

float getXmax() const;

float getYmax() const;

private:

// add any member variables you need

};

class Circle : public TwoDShape {

public:

// Constructor.

// If r is zero or negative, throw a std::invalid-argument exception.

Circle(const Point& c, float r);

// Return basic information (see assignment page)

float getX() const;

float getY() const;

float getR() const;

private:

// add any member variables you need

};

class Scene {

public:

// Constructor

Scene();

// Add the pointer to the collection of pointers stored

void addObject(std::shared\_ptr<Shape> ptr);

// Set the drawing depth to d

void setDrawDepth(int d);

// Constants specifying the size of the drawing area

static constexpr int WIDTH = 60;

static constexpr int HEIGHT = 20;

private:

// add any member variables you need

// Draw objects as specified in the assignment page

friend std::ostream& operator<<(std::ostream& out, const Scene& s);

};

#endif /\* GEOMETRY\_H\_ \*/

[Geometry.cpp](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/Geometry.cpp)

#include "Geometry.h"

// ============ Shape class =================

Shape::Shape() {} // REMOVE ME

Shape::Shape(int d) {

// IMPLEMENT ME

}

bool Shape::setDepth(int d) {

// IMPLEMENT ME

return false; // dummy

}

int Shape::getDepth() const {

// IMPLEMENT ME

return -999; // dummy

}

int Shape::dim() const {

// IMPLEMENT ME

return -999; // dummy

}

void Shape::translate(float x, float y) {

// IMPLEMENT ME

}

void Shape::rotate() {

// IMPLEMENT ME

}

void Shape::scale(float f) {

// IMPLEMENT ME

}

bool Shape::contains(const Point& p) const {

// IMPLEMENT ME

return false; // dummy

}

// =============== Point class ================

Point::Point(float x, float y, int d) {

// IMPLEMENT ME

}

float Point::getX() const {

// IMPLEMENT ME

return -999; // dummy

}

float Point::getY() const {

// IMPLEMENT ME

return -999; // dummy

}

// =========== LineSegment class ==============

LineSegment::LineSegment(const Point& p, const Point& q) {

// IMPLEMENT ME

}

float LineSegment::getXmin() const {

// IMPLEMENT ME

return -999; // dummy

}

float LineSegment::getXmax() const {

// IMPLEMENT ME

return -999; // dummy

}

float LineSegment::getYmin() const {

// IMPLEMENT ME

return -999; // dummy

}

float LineSegment::getYmax() const {

// IMPLEMENT ME

return -999; // dummy

}

float LineSegment::length() const {

// IMPLEMENT ME

return -999; // dummy

}

// ============ TwoDShape class ================

TwoDShape::TwoDShape(){} // REMOVE ME

TwoDShape::TwoDShape(int d) {

// IMPLEMENT ME

}

float TwoDShape::area() const {

// IMPLEMENT ME

return -999; // dummy

}

// ============== Rectangle class ================

Rectangle::Rectangle(const Point& p, const Point& q) {

// IMPLEMENT ME

}

float Rectangle::getXmin() const {

// IMPLEMENT ME

return -999; // dummy

}

float Rectangle::getYmin() const {

// IMPLEMENT ME

return -999; // dummy

}

float Rectangle::getXmax() const {

// IMPLEMENT ME

return -999; // dummy

}

float Rectangle::getYmax() const {

// IMPLEMENT ME

return -999; // dummy

}

// ================== Circle class ===================

Circle::Circle(const Point& c, float r) {

// IMPLEMENT ME

}

float Circle::getX() const {

// IMPLEMENT ME

return -999; // dummy

}

float Circle::getY() const {

// IMPLEMENT ME

return -999; // dummy

}

float Circle::getR() const {

// IMPLEMENT ME

return -999; // dummy

}

// ================= Scene class ===================

Scene::Scene() {

// IMPLEMENT ME

}

void Scene::addObject(std::shared\_ptr<Shape> ptr) {

// IMPLEMENT ME

}

void Scene::setDrawDepth(int depth) {

// IMPLEMENT ME

}

std::ostream& operator<<(std::ostream& out, const Scene& s) {

// IMPLEMENT ME

return out;

}

[main.cpp](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/main.cpp)

#include <iostream>

#include "Geometry.h"

using namespace std;

int main() {

Point p1(-1,-2);

Point p2(10,-2);

cout << p1.getX() << endl;

LineSegment l(p1, p2);

cout << l.getXmin() << " " << l.getXmax() << endl;

cout << l.length() << endl;

p2.translate(0,2);

Rectangle r(p1,p2);

r.rotate();

cout << r.getYmin() << " " << r.getYmax() << endl;

cout << r.area() << endl;

Point p3(1,2,3);

Circle c(p3, 2);

cout << c.area() << endl;

c.scale(10);

cout << c.area() << endl;

auto pp1 = make\_shared<Point>(0,0);

auto pp2 = make\_shared<Point>(0,19);

auto pp3 = make\_shared<Point>(59,19);

auto pp4 = make\_shared<Point>(59,0);

auto pp5 = make\_shared<Point>(55,19);

auto pp6 = make\_shared<Point>(30,0);

auto lp = make\_shared<LineSegment>(\*pp2,\*pp3);

auto rp = make\_shared<Rectangle>(\*pp4,\*pp5);

auto cp = make\_shared<Circle>(\*pp6,10);

Scene s;

s.addObject(pp1);

s.addObject(lp);

s.addObject(rp);

s.addObject(cp);

cout << s;

cout << endl;

rp->rotate();

cp->translate(0,-5);

cout << s;

}

[GeometryTester.h](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/GeometryTester.h)

#ifndef GEOMETRYTESTER\_H\_

#define GEOMETRYTESTER\_H\_

#include <string>

#include "Geometry.h"

class GeometryTester {

public:

GeometryTester();

// point

void testa();

void testb();

void testc();

// line

void testd();

void teste();

void testf();

// rect

void testg();

void testh();

void testi();

// circle

void testj();

void testk();

void testl();

// depth, dim, area, polymorphic

void testm();

void testn();

void testo();

void testp();

void testq();

// exceptions

void testr();

void tests();

void testt();

// scene

void testu();

void testv();

void testw();

void testx();

void testy();

// unused

void testz();

private:

// three overloaded versions

void errorOut\_(const std::string& errMsg, unsigned int errBit);

void errorOut\_(const std::string& errMsg, const std::string& errRes, unsigned int errBit);

void errorOut\_(const std::string& errMsg, int errRes, unsigned int errBit);

void passOut\_();

char error\_;

std::string funcname\_;

std::string blankpage\_; // for this cw only!

};

#endif /\* GEOMETRYTESTER\_H\_ \*/

[GeometryTester.cpp](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/GeometryTester.cpp)

#include <iostream>

#include <sstream>

#include <stdexcept>

#include "Geometry.h"

#include "GeometryTester.h"

using namespace std;

GeometryTester::GeometryTester() : error\_(false), funcname\_("") {

string blankline(Scene::WIDTH, ' ');

for(int i=0;i<Scene::HEIGHT;i++) blankpage\_ += (blankline + "\n");

}

// Point ctor, basics

void GeometryTester::testa() {

funcname\_ = "GeometryTester::testa";

{

// default depth

Point p(0,0);

if (p.getX() != 0)

errorOut\_("p x-coord reported as" , p.getX(), 1);

if (p.getY() != 0)

errorOut\_("p y-coord reported as ", p.getY(), 1);

if (p.getDepth() != 0)

errorOut\_("p depth reported as ", p.getDepth(), 1);

// specified depth

Point q(1,-2,3);

if (q.getX() != 1)

errorOut\_("q x-coord reported as ", q.getX(), 2);

if (q.getY() != -2)

errorOut\_("q y-coord reported as ", q.getY(), 2);

if (q.getDepth() != 3)

errorOut\_("q depth reported as ", q.getDepth(), 2);

}

passOut\_();

}

// point translate, contains

void GeometryTester::testb() {

funcname\_ = "GeometryTester::testb";

{

Point p(1,2);

p.translate(3,-4);

if (p.getX() != 4)

errorOut\_("translated p x-coord reported as ", p.getX(), 1);

if (p.getY() != -2)

errorOut\_("translated p y-coord reported as ", p.getY(), 1);

Point q(-1,-2);

if (!q.contains(q))

errorOut\_("q does not contain q?", 2);

if (q.contains(Point(1,-2)) || q.contains(Point(-1,2)) || q.contains(p))

errorOut\_("q contains another pt?", 2);

}

passOut\_();

}

// point rotate, scale no effect

void GeometryTester::testc() {

funcname\_ = "GeometryTester::testc";

{

Point p(-1,-2);

p.rotate();

if (p.getX() != -1)

errorOut\_("rotated p x-coord reported as ", p.getX(), 1);

if (p.getY() != -2)

errorOut\_("rotated p y-coord reported as ", p.getY(), 1);

Point q(-5,6);

q.scale(2);

if (q.getX() != -5)

errorOut\_("scaled up q x-coord reported as ", q.getX(), 2);

if (q.getY() != 6)

errorOut\_("scaled up q y-coord reported as ", q.getY(), 2);

q.scale(0.25);

if (q.getX() != -5)

errorOut\_("scaled down q x-coord reported as ", q.getX(), 2);

if (q.getY() != 6)

errorOut\_("scaled down q y-coord reported as ", q.getY(), 2);

}

passOut\_();

}

// LineSegment ctor, basics, length

void GeometryTester::testd() {

funcname\_ = "GeometryTester::testd";

{

// horizontal line, default depth

Point p1(1,2);

Point p2(5,2);

LineSegment l1(p1,p2);

if (l1.getXmin() != 1) errorOut\_("l1 Xmin reported as ",l1.getXmin(),1);

if (l1.getXmax() != 5) errorOut\_("l1 Xmax reported as ",l1.getXmax(),1);

if (l1.getYmin() != 2) errorOut\_("l1 Ymin reported as ",l1.getYmin(),1);

if (l1.getYmax() != 2) errorOut\_("l1 Ymax reported as ",l1.getYmax(),1);

if (l1.getDepth() != 0) errorOut\_("l1 depth reported as ",l1.getDepth(),1);

if (l1.length() != 4) errorOut\_("l1 length reported as ",l1.length(),1);

// vertical line, opp dir, specified depth

Point p3(5,4,1);

Point p4(5,2,1);

LineSegment l2(p3,p4);

if (l2.getXmin() != 5) errorOut\_("l2 Xmin reported as ",l2.getXmin(),2);

if (l2.getXmax() != 5) errorOut\_("l2 Xmax reported as ",l2.getXmax(),2);

if (l2.getYmin() != 2) errorOut\_("l2 Ymin reported as ",l2.getYmin(),2);

if (l2.getYmax() != 4) errorOut\_("l2 Ymax reported as ",l2.getYmax(),2);

if (l2.getDepth() != 1) errorOut\_("l2 depth reported as ",l2.getDepth(),2);

if (l2.length() != 2) errorOut\_("l2 length reported as ",l2.length(),2);

}

passOut\_();

}

// line segment translate, rotate

void GeometryTester::teste() {

funcname\_ = "GeometryTester::teste";

{

Point p1(1,2);

Point p2(5,2);

Point p3(5,4);

LineSegment l1(p1,p2);

LineSegment l2(p2,p3);

l1.translate(2,3);

if (l1.getXmin() != 3) errorOut\_("translated l1 xmin reported as ",l1.getXmin(),1);

if (l1.getXmax() != 7) errorOut\_("translated l1 xmax reported as ",l1.getXmax(),1);

if (l1.getYmin() != 5) errorOut\_("translated l1 ymin reported as ",l1.getYmin(),1);

if (l1.getYmax() != 5) errorOut\_("translated l1 ymax reported as ",l1.getYmax(),1);

l2.translate(-2,-3);

if (l2.getXmin() != 3) errorOut\_("translated l1 xmin reported as ",l2.getXmin(),1);

if (l2.getXmax() != 3) errorOut\_("translated l2 xmax reported as ",l2.getXmax(),1);

if (l2.getYmin() != -1) errorOut\_("translated l2 ymin reported as ",l2.getYmin(),1);

if (l2.getYmax() != 1) errorOut\_("translated l2 ymax reported as ",l2.getYmax(),1);

l1.translate(-2,-3);

l2.translate(2,3);

l1.rotate();

if (l1.getXmin() != 3) errorOut\_("rotated l1 xmin reported as ",l1.getXmin(),2);

if (l1.getXmax() != 3) errorOut\_("rotated l1 xmax reported as ",l1.getXmax(),2);

if (l1.getYmin() != 0) errorOut\_("rotated l1 ymin reported as ",l1.getYmin(),2);

if (l1.getYmax() != 4) errorOut\_("rotated l1 ymax reported as ",l1.getYmax(),2);

l2.rotate();

if (l2.getXmin() != 4) errorOut\_("rotated l2 xmin reported as ",l2.getXmin(),2);

if (l2.getXmax() != 6) errorOut\_("rotated l2 xmax reported as ",l2.getXmax(),2);

if (l2.getYmin() != 3) errorOut\_("rotated l2 ymin reported as ",l2.getYmin(),2);

if (l2.getYmax() != 3) errorOut\_("rotated l2 ymax reported as ",l2.getYmax(),2);

}

passOut\_();

}

// line segment contains, scale

void GeometryTester::testf() {

funcname\_ = "GeometryTester::testf";

{

Point p1(1,2);

Point p2(5,2);

Point p3(5,4);

LineSegment l1(p2,p1);

LineSegment l2(p3,p2);

if (!l1.contains(Point(3,2))) errorOut\_("l1 does not contain (3,2)?",1);

if (!l1.contains(Point(1,2))) errorOut\_("l1 does not contain (1,2)?",1);

if (l1.contains(Point(-1,2))) errorOut\_("l1 contains (-1,2)?",1);

if (l1.contains(Point(2,1))) errorOut\_("l1 contains (2,1)?",1);

if (!l2.contains(Point(5,3))) errorOut\_("l2 does not contain (5,3)?",1);

if (!l2.contains(Point(5,4))) errorOut\_("l2 does not contain (5,4)?",1);

if (l2.contains(Point(5,0))) errorOut\_("l2 contains (5,0)?",1);

if (l2.contains(Point(0,0))) errorOut\_("l2 contains (0,0)?",1);

l1.scale(0.5);

if (l1.getXmin() != 2) errorOut\_("scaled l1 xmin reported as ",l1.getXmin(),2);

if (l1.getXmax() != 4) errorOut\_("scaled l1 xmax reported as ",l1.getXmax(),2);

if (l1.getYmin() != 2) errorOut\_("scaled l1 ymin reported as ",l1.getYmin(),2);

if (l1.getYmax() != 2) errorOut\_("scaled l1 ymax reported as ",l1.getYmax(),2);

l2.scale(3);

if (l2.getXmin() != 5) errorOut\_("scaled l2 xmin reported as ",l2.getXmin(),2);

if (l2.getXmax() != 5) errorOut\_("scaled l2 xmax reported as ",l2.getXmax(),2);

if (l2.getYmin() != 0) errorOut\_("scaled l2 ymin reported as ",l2.getYmin(),2);

if (l2.getYmax() != 6) errorOut\_("scaled l2 ymax reported as ",l2.getYmax(),2);

}

passOut\_();

}

// rect ctor, basics

void GeometryTester::testg() {

funcname\_ = "GeometryTester::testg";

{

// BL-TR, default depth

Point p1(2,1);

Point q1(6,5);

Rectangle r1(p1,q1);

if (r1.getXmin() != 2) errorOut\_("r1 xmin reported as ",r1.getXmin(),1);

if (r1.getXmax() != 6) errorOut\_("r1 xmax reported as ",r1.getXmax(),1);

if (r1.getYmin() != 1) errorOut\_("r1 ymin reported as ",r1.getYmin(),1);

if (r1.getYmax() != 5) errorOut\_("r1 ymin reported as ",r1.getYmax(),1);

if (r1.getDepth() != 0) errorOut\_("r1 depth reported as ",r1.getDepth(),1);

// TL-BR, specified depth

Point p2(-1,6,2);

Point q2(1,-2,2);

Rectangle r2(p2,q2);

if (r2.getXmin() != -1) errorOut\_("r2 xmin reported as ",r2.getXmin(),2);

if (r2.getXmax() != 1) errorOut\_("r2 xmax reported as ",r2.getXmax(),2);

if (r2.getYmin() != -2) errorOut\_("r2 ymin reported as ",r2.getYmin(),2);

if (r2.getYmax() != 6) errorOut\_("r2 ymax reported as ",r2.getYmax(),2);

if (r2.getDepth() != 2) errorOut\_("r2 depth reported as ",r2.getDepth(),2);

}

passOut\_();

}

// rect contains, translate

void GeometryTester::testh() {

funcname\_ = "GeometryTester::testh";

{

Point p(2,1);

Point q(6,5);

Rectangle r(p,q);

if (!r.contains(Point(3,3))) errorOut\_("r does not contain (3,3)?",1);

if (!r.contains(Point(6,3))) errorOut\_("r does not contain (6.3)?",1);

if (!r.contains(Point(4,1))) errorOut\_("r does not contain (4,1)?",1);

if (!r.contains(Point(6,1))) errorOut\_("r does not contain (6,1)?",1);

if (r.contains(Point(7,3))) errorOut\_("r contains (7,3)?",1);

if (r.contains(Point(5,0))) errorOut\_("r contains (5,0)?",1);

r.translate(-1,3);

if (r.getXmin() != 1) errorOut\_("translated r xmin reported as ",r.getXmin(),2);

if (r.getXmax() != 5) errorOut\_("translated r xmax reported as ",r.getXmax(),2);

if (r.getYmin() != 4) errorOut\_("translated r ymin reported as ",r.getYmin(),2);

if (r.getYmax() != 8) errorOut\_("translated r ymax reported as ",r.getYmax(),2);

}

passOut\_();

}

// rect rotate, scale

void GeometryTester::testi() {

funcname\_ = "GeometryTester::testi";

{

Point p(6,3);

Point q(2,1);

Rectangle r(p,q);

r.rotate();

if (r.getXmin() != 3) errorOut\_("rotated r xmin reported as ",r.getXmin(),1);

if (r.getXmax() != 5) errorOut\_("rotated r xmax reported as ",r.getXmax(),1);

if (r.getYmin() != 0) errorOut\_("rotated r ymin reported as ",r.getYmin(),1);

if (r.getYmax() != 4) errorOut\_("rotated r ymax reported as ",r.getYmax(),1);

r.rotate();

if (r.getXmin() != 2) errorOut\_("rotated again r xmin reported as ",r.getXmin(),1);

if (r.getXmax() != 6) errorOut\_("rotated again r xmax reported as ",r.getXmax(),1);

if (r.getYmin() != 1) errorOut\_("rotated again r ymin reported as ",r.getYmin(),1);

if (r.getYmax() != 3) errorOut\_("rotated again r ymax reported as ",r.getYmax(),1);

Rectangle r2(p,q);

r2.scale(2);

if (r2.getXmin() != 0) errorOut\_("scaled r2 xmin reported as ",r2.getXmin(),2);

if (r2.getXmax() != 8) errorOut\_("scaled r2 xmax reported as ",r2.getXmax(),2);

if (r2.getYmin() != 0) errorOut\_("scaled r2 ymin reported as ",r2.getYmin(),2);

if (r2.getYmax() != 4) errorOut\_("scaled r2 ymax reported as ",r2.getYmax(),2);

r2.scale(0.25);

if (r2.getXmin() != 3) errorOut\_("scaled again r2 xmin reported as ",r2.getXmin(),2);

if (r2.getXmax() != 5) errorOut\_("scaled again r2 xmax reported as ",r2.getXmax(),2);

if (r2.getYmin() != 1.5) errorOut\_("scaled again r2 ymin reported as ",r2.getYmin(),2);

if (r2.getYmax() != 2.5) errorOut\_("scaled again r2 ymax reported as ",r2.getYmax(),2);

}

passOut\_();

}

// circle ctor, basics

void GeometryTester::testj() {

funcname\_ = "GeometryTester::testj";

{

Point p(1,2);

Circle c(p,3);

if (c.getX() != 1) errorOut\_("c x reported as ",c.getX(),1);

if (c.getY() != 2) errorOut\_("c y reported as ",c.getY(),1);

if (c.getR() != 3) errorOut\_("c r reported as ",c.getR(),1);

if (c.getDepth() != 0) errorOut\_("c depth reported as ",c.getDepth(),1);

Point p2(-1,-2,1);

Circle c2(p2,3);

if (c2.getX() != -1) errorOut\_("c2 x reported as ",c2.getX(),2);

if (c2.getY() != -2) errorOut\_("c2 y reported as ",c2.getY(),2);

if (c2.getR() != 3) errorOut\_("c2 r reported as ",c2.getR(),2);

if (c2.getDepth() != 1) errorOut\_("c2 depth reported as ",c2.getDepth(),2);

}

passOut\_();

}

// circle contains, rotate no effect

void GeometryTester::testk() {

funcname\_ = "GeometryTester::testk";

{

Point p(1,2);

Circle c(p,3);

if (!c.contains(p)) errorOut\_("c does not contain p?",1);

if (!c.contains(Point(3.9,2))) errorOut\_("c does not contain (3.9,2)?",1);

if (!c.contains(Point(3.1,4.1))) errorOut\_("c does not contain (3.1,4.1)?",1);

if (!c.contains(Point(-1.1,4.1))) errorOut\_("c does not contain (-1.1,4.1)?",1);

if (!c.contains(Point(-1.1,-0.1))) errorOut\_("c does not contain (-1.1,-0.1)?",1);

if (!c.contains(Point(3.1,-0.1))) errorOut\_("c does not contain (3.1,-0.1)?",1);

if (c.contains(Point(3.2,4.2))) errorOut\_("c contains (3.2,4.2)?",1);

if (c.contains(Point(-1.2,4.2))) errorOut\_("c contains (-1.2,4.2)?",1);

if (c.contains(Point(-1.2,-0.2))) errorOut\_("c contains (-1.2,-0.2)?",1);

if (c.contains(Point(3.2,-0.2))) errorOut\_("c contains (3.2,-0.2)?",1);

c.rotate();

if (c.getX() != 1) errorOut\_("rotated c x reported as ",c.getX(),2);

if (c.getY() != 2) errorOut\_("rotated c y reported as ",c.getY(),2);

if (c.getR() != 3) errorOut\_("rotated c r reported as ",c.getR(),2);

}

passOut\_();

}

// circle translate, scale

void GeometryTester::testl() {

funcname\_ = "GeometryTester::testl";

{

Point p(1,2);

Circle c(p,3);

c.translate(-5,-7);

if (c.getX() != -4) errorOut\_("translated c x reported as ",c.getX(),1);

if (c.getY() != -5) errorOut\_("translated c y reported as ",c.getY(),1);

if (c.getR() != 3) errorOut\_("translated c r reported as ",c.getR(),1);

Circle c2(p,3);

c2.scale(3);

if (c2.getX() != 1) errorOut\_("scaled up c2 x reported as ",c2.getX(),2);

if (c2.getY() != 2) errorOut\_("scaled up c2 y reported as ",c2.getY(),2);

if (c2.getR() != 9) errorOut\_("scaled up c2 r reported as ",c2.getR(),2);

c2.scale(0.5);

if (c2.getX() != 1) errorOut\_("scaled down c2 x reported as ",c2.getX(),2);

if (c2.getY() != 2) errorOut\_("scaled down c2 y reported as ",c2.getY(),2);

if (c2.getR() != 4.5) errorOut\_("scaled down c2 r reported as ",c2.getR(),2);

}

passOut\_();

}

// depth

void GeometryTester::testm() {

funcname\_ = "GeometryTester::testm";

{

bool b;

// point

Point p(1,-2,3);

b = p.setDepth(4);

if (!b) errorOut\_("p non-neg setDepth returned false", 1);

if (p.getDepth() != 4)

errorOut\_("p new depth reported as ", p.getDepth(), 2);

b = p.setDepth(-1);

if (b) errorOut\_("p neg setDepth returned true", 1);

if (p.getDepth() != 4)

errorOut\_("p neg setDepth set to ", p.getDepth(), 2);

// line

LineSegment l(Point(0,0,5), Point(0,2,5));

b = l.setDepth(2);

if (!b) errorOut\_("l non-neg setDepth returned false", 1);

if (l.getDepth() != 2)

errorOut\_("l new depth reported as ", l.getDepth(), 2);

b = l.setDepth(-2);

if (b) errorOut\_("l neg depth returned true", 1);

if (l.getDepth() != 2)

errorOut\_("l neg setDepth set to ", l.getDepth(), 2);

// rect

Rectangle r(Point(-3,-4), Point(5,6));

b = r.setDepth(3);

if (!b) errorOut\_("r non-neg setDepth returned false", 1);

if (r.getDepth() != 3)

errorOut\_("r new depth reported as ", r.getDepth(), 2);

b = r.setDepth(-3);

if (b) errorOut\_("r neg depth returned true", 1);

if (r.getDepth() != 3)

errorOut\_("r neg setDepth set to ", r.getDepth(), 2);

// circle

Circle c(Point(0,-2,4), 10);

b = c.setDepth(0);

if (!b) errorOut\_("c non-neg setDepth returned false", 1);

if (c.getDepth() != 0)

errorOut\_("c new depth reported as ", c.getDepth(), 2);

b = c.setDepth(-4);

if (b) errorOut\_("c neg depth returned true", 1);

if (c.getDepth() != 0)

errorOut\_("c neg setDepth set to ", c.getDepth(), 2);

// poly

Shape\* sp[4];

sp[0] = &p; sp[1] = &l; sp[2] = &r; sp[3] = &c;

for(int i=0;i<4;i++) {

sp[i]->setDepth(i);

if (sp[i]->getDepth() != i)

errorOut\_("get/setdepth wrong for sp", i, 2);

}

}

passOut\_();

}

// dim, polymorphic

void GeometryTester::testn() {

funcname\_ = "GeometryTester::testn";

{

Shape\* sp[4];

sp[0] = new Point(0,0);

sp[1] = new LineSegment(Point(0,0), Point(4,0));

sp[2] = new Rectangle(Point(0,0), Point(6,4));

sp[3] = new Circle(Point(0,0), 7);

if (sp[0]->dim() != 0)

errorOut\_("dim sp[0] reported as ",sp[0]->dim(),0);

if (sp[1]->dim() != 1)

errorOut\_("dim sp[1] reported as ",sp[1]->dim(),0);

if (sp[2]->dim() != 2)

errorOut\_("dim sp[2] reported as ",sp[2]->dim(),0);

if (sp[3]->dim() != 2)

errorOut\_("dim sp[3] reported as ",sp[3]->dim(),0);

}

passOut\_();

}

// area, polymorphic

void GeometryTester::testo() {

funcname\_ = "GeometryTester::testo";

{

Rectangle r(Point(-1,-2), Point(3,4));

if (r.area() != 24) errorOut\_("r area reported as ", r.area(), 1);

Rectangle r2(Point(5,-6), Point(-7,8));

if (r2.area() != 168) errorOut\_("r2 area reported as ", r2.area(), 1);

Circle c(Point(2,3), 10);

if (c.area() < 314.1 || c.area() > 314.2)

errorOut\_("c area reported as ", c.area(), 2);

Circle c2(Point(4,5), 6);

if (c2.area() < 113.0 || c2.area() > 113.1)

errorOut\_("c2 area reported as ", c.area(), 2);

TwoDShape& s1 = r;

TwoDShape& s2 = c;

if (s1.area() != 24) errorOut\_("s1 area reported as ", s1.area(), 1);

if (s2.area() < 314.1 || s2.area() > 314.2)

errorOut\_("s2 area reported as ", s2.area(), 2);

}

passOut\_();

}

// polymorphic contains, translate

void GeometryTester::testp() {

funcname\_ = "GeometryTester::testp";

{

Shape\* sp[4];

sp[0] = new Point(0,0);

sp[1] = new LineSegment(Point(0,0), Point(4,0));

sp[2] = new Rectangle(Point(0,0), Point(6,4));

sp[3] = new Circle(Point(0,0), 7);

for(int i=0;i<4;i++) {

if (!sp[i]->contains(Point(0,0)))

errorOut\_("sp" + std::to\_string(i) + " not contain (0,0)?",1);

}

for(int i=0;i<4;i++) sp[i]->translate(3,-2);

for(int i=0;i<4;i++) {

if (!sp[i]->contains(Point(3,-2)))

errorOut\_("sp" + std::to\_string(i) + " contains wrong after translate",2);

}

}

passOut\_();

}

// polymorphic rotate, scale

void GeometryTester::testq() {

funcname\_ = "GeometryTester::testq";

{

Shape\* sp[4];

sp[0] = new Point(1,1);

sp[1] = new LineSegment(Point(0,3), Point(6,3));

sp[2] = new Rectangle(Point(0,2), Point(6,4));

sp[3] = new Circle(Point(0,0), 7);

for(int i=0;i<4;i++) sp[i]->rotate();

// should use dynamic\_cast but won't work until you coded correctly!

Point\* p0 = static\_cast<Point\*>(sp[0]);

LineSegment\* p1 = static\_cast<LineSegment\*>(sp[1]);

Rectangle\* p2 = static\_cast<Rectangle\*>(sp[2]);

Circle\* p3 = static\_cast<Circle\*>(sp[3]);

if (p0->getX() != 1 || p0->getY() != 1)

errorOut\_("sp[0] rotate wrong",1);

if (p1->getXmin() != 3 || p1->getXmax() != 3 || p1->getYmin() != 0 || p1->getYmax() != 6)

errorOut\_("sp[1] rotate wrong",1);

if (p2->getXmin() != 2 || p2->getXmax() != 4 || p2->getYmin() != 0 || p2->getYmax() != 6)

errorOut\_("sp[2] rotate wrong",1);

if (p3->getX() != 0 || p3->getY() != 0 || p3->getR() != 7)

errorOut\_("sp[3] rotate wrong",1);

for(int i=0;i<4;i++) sp[i]->scale(5);

if (p0->getX() != 1 || p0->getY() != 1)

errorOut\_("sp[0] scale wrong",2);

if (p1->getXmin() != 3 || p1->getXmax() != 3 || p1->getYmin() != -12 || p1->getYmax() != 18)

errorOut\_("sp[1] scale wrong",2);

if (p2->getXmin() != -2 || p2->getXmax() != 8 || p2->getYmin() != -12 || p2->getYmax() != 18)

errorOut\_("sp[2] scale wrong",2);

if (p3->getX() != 0 || p3->getY() != 0 || p3->getR() != 35)

errorOut\_("sp[3] scale wrong",2);

}

passOut\_();

}

// point, line ctor exception

void GeometryTester::testr() {

funcname\_ = "GeometryTester::testr";

{

// point

try {

Point p(-1,-2,-1);

errorOut\_("negative depth should throw exception", 1);

}

catch(const std::invalid\_argument& e) {}

// line

Point p1(1,2);

Point p2(5,2);

Point p3(5,4,1);

try {

LineSegment l(p2,p3);

errorOut\_("two endpoints diff depth should throw exception",2);

}

catch(const std::invalid\_argument& e) {}

try {

LineSegment l(p1,Point(1,2));

errorOut\_("two endpoints same should throw exception",2);

}

catch(const std::invalid\_argument& e) {}

p3.setDepth(0);

try {

LineSegment l(p1,p3);

errorOut\_("line not orthogonal should throw exception",2);

}

catch(const std::invalid\_argument& e) {}

}

passOut\_();

}

// rect, circle ctor exception

void GeometryTester::tests() {

funcname\_ = "GeometryTester::tests";

{

Point p(2,1);

Point q(1,-2,2);

// rect

try {

Rectangle r(p,q);

errorOut\_("two points diff depth should throw exception",1);

}

catch(const std::invalid\_argument& e) {}

try {

Rectangle r(p,Point(2,6));

errorOut\_("two points same x-coord should throw exception",1);

}

catch(const std::invalid\_argument& e) {}

try {

Rectangle r(p,Point(-2,1));

errorOut\_("two points same y-coord should throw exception",1);

}

catch(const std::invalid\_argument& e) {}

// circle

try {

Circle c(p,0);

errorOut\_("circle zero r should throw exception",2);

}

catch(const std::invalid\_argument& e) {}

try {

Circle c(p,-1);

errorOut\_("circle -ve r should throw exception",2);

}

catch(const std::invalid\_argument& e) {}

}

passOut\_();

}

// scale exception

void GeometryTester::testt() {

funcname\_ = "GeometryTester::testt";

{

Point p(2,1);

Point q1(6,1);

Point q2(6,3);

LineSegment l(p,q1);

Rectangle r(p,q2);

Circle c(p,4);

try {

p.scale(0);

errorOut\_("p scale zero f should throw exception",1);

} catch(const std::invalid\_argument& e) {}

try {

p.scale(-1);

errorOut\_("p scale -ve f should throw exception",2);

} catch(const std::invalid\_argument& e) {}

try {

l.scale(0);

errorOut\_("l scale zero f should throw exception",1);

} catch(const std::invalid\_argument& e) {}

try {

l.scale(-2);

errorOut\_("l scale -ve f should throw exception",2);

} catch(const std::invalid\_argument& e) {}

try {

r.scale(0);

errorOut\_("r scale zero f should throw exception",1);

} catch(const std::invalid\_argument& e) {}

try {

r.scale(-0.5);

errorOut\_("r scale -ve f should throw exception",2);

} catch(const std::invalid\_argument& e) {}

try {

c.scale(0);

errorOut\_("c scale zero f should throw exception",1);

} catch(const std::invalid\_argument& e) {}

try {

c.scale(-0.25);

errorOut\_("c scale -ve f should throw exception",2);

} catch(const std::invalid\_argument& e) {}

}

passOut\_();

}

// scene draw blank, point

void GeometryTester::testu() {

funcname\_ = "GeometryTester::testu";

{

// blank

Scene s;

stringstream ss;

ss << s;

string page = blankpage\_;

if (ss.str() != page) {

errorOut\_("blank scene drawn wrongly",1);

cout << "Expected output:\n" << page;

cout << "Your output:\n" << ss.str();

}

// point

auto p1 = make\_shared<Point>(1,2);

s.addObject(p1);

stringstream ss2;

ss2 << s;

page[17\*(Scene::WIDTH+1)+1] = '\*';

if (ss2.str() != page) {

errorOut\_("point drawn wrongly",2);

cout << "Expected output:\n" << page;

cout << "Your output:\n" << ss2.str();

}

}

passOut\_();

}

// scene draw line, shared w caller

void GeometryTester::testv() {

funcname\_ = "GeometryTester::testv";

{

// line

Scene s;

auto p = make\_shared<LineSegment>(Point(1,4), Point(6,4));

s.addObject(p);

stringstream ss;

ss << s;

string page = blankpage\_;

for(int i=1;i<=6;i++) page[15\*(Scene::WIDTH+1)+i] = '\*';

if (ss.str() != page) {

errorOut\_("line drawn wrongly",1);

cout << "Expected output:\n" << page;

cout << "Your output:\n" << ss.str();

}

// moved line

p->translate(0,3);

stringstream ss2;

ss2 << s;

for(int i=1;i<=6;i++) {

page[15\*(Scene::WIDTH+1)+i] = ' ';

page[12\*(Scene::WIDTH+1)+i] = '\*';

}

if (ss2.str() != page) {

errorOut\_("line after translate drawn wrongly",2);

cout << "Expected output:\n" << page;

cout << "Your output:\n" << ss2.str();

}

}

passOut\_();

}

// scene draw rect, circle

void GeometryTester::testw() {

funcname\_ = "GeometryTester::testw";

{

// rect

Scene s1;

auto p1 = make\_shared<Rectangle>(Point(0,15), Point(10,19));

s1.addObject(p1);

stringstream ss1;

ss1 << s1;

string page = blankpage\_;

for(int j=0;j<5;j++)

for(int i=0;i<=10;i++)

page[j\*(Scene::WIDTH+1)+i] = '\*';

if (ss1.str() != page) {

errorOut\_("rect drawn wrongly",1);

cout << "Expected output:\n" << page;

cout << "Your output:\n" << ss1.str();

}

// circle

Scene s2;

auto p2 = make\_shared<Circle>(Point(30,10), 9);

s2.addObject(p2);

stringstream ss2;

ss2 << s2;

string page2 = blankpage\_;

for(int i=30;i<=30;i++) page2[0\*(Scene::WIDTH+1)+i] = '\*';

for(int i=26;i<=34;i++) page2[1\*(Scene::WIDTH+1)+i] = '\*';

for(int i=25;i<=35;i++) page2[2\*(Scene::WIDTH+1)+i] = '\*';

for(int i=24;i<=36;i++) page2[3\*(Scene::WIDTH+1)+i] = '\*';

for(int i=23;i<=37;i++) page2[4\*(Scene::WIDTH+1)+i] = '\*';

for(int i=22;i<=38;i++) page2[5\*(Scene::WIDTH+1)+i] = '\*';

for(int i=22;i<=38;i++) page2[6\*(Scene::WIDTH+1)+i] = '\*';

for(int i=22;i<=38;i++) page2[7\*(Scene::WIDTH+1)+i] = '\*';

for(int i=22;i<=38;i++) page2[8\*(Scene::WIDTH+1)+i] = '\*';

for(int i=21;i<=39;i++) page2[9\*(Scene::WIDTH+1)+i] = '\*';

for(int i=22;i<=38;i++) page2[10\*(Scene::WIDTH+1)+i] = '\*';

for(int i=22;i<=38;i++) page2[11\*(Scene::WIDTH+1)+i] = '\*';

for(int i=22;i<=38;i++) page2[12\*(Scene::WIDTH+1)+i] = '\*';

for(int i=22;i<=38;i++) page2[13\*(Scene::WIDTH+1)+i] = '\*';

for(int i=23;i<=37;i++) page2[14\*(Scene::WIDTH+1)+i] = '\*';

for(int i=24;i<=36;i++) page2[15\*(Scene::WIDTH+1)+i] = '\*';

for(int i=25;i<=35;i++) page2[16\*(Scene::WIDTH+1)+i] = '\*';

for(int i=26;i<=34;i++) page2[17\*(Scene::WIDTH+1)+i] = '\*';

for(int i=30;i<=30;i++) page2[18\*(Scene::WIDTH+1)+i] = '\*';

if (ss2.str() != page2) {

errorOut\_("circle drawn wrongly",2);

cout << "Expected output:\n" << page2;

cout << "Your output:\n" << ss2.str();

}

}

passOut\_();

}

// scene draw overlapping, out of view

void GeometryTester::testx() {

funcname\_ = "GeometryTester::testx";

{

auto p1 = make\_shared<Point>(0,0);

auto p2 = make\_shared<Point>(0,2);

auto p3 = make\_shared<Point>(59,2);

auto p4 = make\_shared<Point>(28,0);

auto p5 = make\_shared<Point>(32,19);

auto p6 = make\_shared<Point>(30,0);

auto l = make\_shared<LineSegment>(\*p2,\*p3);

auto r = make\_shared<Rectangle>(\*p4,\*p5);

auto c = make\_shared<Circle>(\*p6,10);

Scene s;

s.addObject(p1);

s.addObject(l);

s.addObject(r);

s.addObject(c);

string page = blankpage\_;

for(int j=0;j<=9;j++)

for(int i=28;i<=32;i++)

page[j\*(Scene::WIDTH+1)+i] = '\*';

for(int i=26;i<=34;i++) page[10\*(Scene::WIDTH+1)+i] = '\*';

for(int i=24;i<=36;i++) page[11\*(Scene::WIDTH+1)+i] = '\*';

for(int i=23;i<=37;i++) page[12\*(Scene::WIDTH+1)+i] = '\*';

for(int i=22;i<=38;i++) page[13\*(Scene::WIDTH+1)+i] = '\*';

for(int i=22;i<=38;i++) page[14\*(Scene::WIDTH+1)+i] = '\*';

for(int i=21;i<=39;i++) page[15\*(Scene::WIDTH+1)+i] = '\*';

for(int i=21;i<=39;i++) page[16\*(Scene::WIDTH+1)+i] = '\*';

for(int i=0;i<=59;i++) page[17\*(Scene::WIDTH+1)+i] = '\*';

for(int i=21;i<=39;i++) page[18\*(Scene::WIDTH+1)+i] = '\*';

for(int i=20;i<=40;i++) page[19\*(Scene::WIDTH+1)+i] = '\*';

for(int i=0;i<=0;i++) page[19\*(Scene::WIDTH+1)+i] = '\*';

stringstream ss;

ss << s;

if (ss.str() != page) {

errorOut\_("scene drawn wrongly",0);

cout << "Expected output:\n" << page;

cout << "Your output:\n" << ss.str();

}

}

passOut\_();

}

// drawdepth

void GeometryTester::testy() {

funcname\_ = "GeometryTester::testy";

{

auto p1 = make\_shared<Point>(0,0,10);

auto p2 = make\_shared<Point>(0,19,20);

auto p3 = make\_shared<Point>(59,0,30);

auto p4 = make\_shared<Point>(59,19,40);

Scene s;

s.addObject(p1);

s.addObject(p2);

s.addObject(p3);

s.addObject(p4);

// def draw depth

stringstream ss0;

ss0 << s;

string page = blankpage\_;

page[0\*(Scene::WIDTH+1)+0] = '\*';

page[0\*(Scene::WIDTH+1)+59] = '\*';

page[19\*(Scene::WIDTH+1)+0] = '\*';

page[19\*(Scene::WIDTH+1)+59] = '\*';

if (ss0.str() != page) {

errorOut\_("ss0 drawn wrongly",1);

cout << "Expected output:\n" << page;

cout << "Your output:\n" << ss0.str();

}

// successively reduce drawdepth

s.setDrawDepth(35);

stringstream ss1;

ss1 << s;

page[0\*(Scene::WIDTH+1)+59] = ' ';

if (ss1.str() != page) {

errorOut\_("ss1 drawn wrongly",1);

cout << "Expected output:\n" << page;

cout << "Your output:\n" << ss1.str();

}

s.setDrawDepth(21);

stringstream ss2;

ss2 << s;

page[19\*(Scene::WIDTH+1)+59] = ' ';

if (ss2.str() != page) {

errorOut\_("ss2 drawn wrongly",1);

cout << "Expected output:\n" << page;

cout << "Your output:\n" << ss2.str();

}

s.setDrawDepth(10);

stringstream ss3;

ss3 << s;

page[0\*(Scene::WIDTH+1)+0] = ' ';

if (ss3.str() != page) {

errorOut\_("ss3 drawn wrongly",2);

cout << "Expected output:\n" << page;

cout << "Your output:\n" << ss3.str();

}

s.setDrawDepth(5);

stringstream ss4;

ss4 << s;

page[19\*(Scene::WIDTH+1)+0] = ' ';

if (ss4.str() != page) {

errorOut\_("ss4 drawn wrongly",2);

cout << "Expected output:\n" << page;

cout << "Your output:\n" << ss4.str();

}

// change depth of points

p1->setDepth(1); p2->setDepth(2); p3->setDepth(3); p4->setDepth(4);

stringstream ss5;

ss5 << s;

page[0\*(Scene::WIDTH+1)+0] = '\*';

page[0\*(Scene::WIDTH+1)+59] = '\*';

page[19\*(Scene::WIDTH+1)+0] = '\*';

page[19\*(Scene::WIDTH+1)+59] = '\*';

if (ss5.str() != page) {

errorOut\_("ss5 drawn wrongly",2);

cout << "Expected output:\n" << page;

cout << "Your output:\n" << ss5.str();

}

}

passOut\_();

}

// reserved for new test cases

void GeometryTester::testz() {

funcname\_ = "GeometryTester::testz";

}

void GeometryTester::errorOut\_(const string& errMsg, unsigned int errBit) {

cerr << funcname\_ << ":" << " fail" << errBit << ": ";

cerr << errMsg << endl;

error\_ |= (1<<errBit);

cerr << std::flush;

}

void GeometryTester::errorOut\_(const string& errMsg, const string& errResult, unsigned int errBit) {

cerr << funcname\_ << ":" << " fail" << errBit << ": ";

cerr << errMsg << errResult << endl;

error\_ |= (1<<errBit);

cerr << std::flush;

}

void GeometryTester::errorOut\_(const string& errMsg, int errResult, unsigned int errBit) {

cerr << funcname\_ << ":" << " fail" << errBit << ": ";

cerr << errMsg << std::to\_string(errResult) << endl;

error\_ |= (1<<errBit);

cerr << std::flush;

}

void GeometryTester::passOut\_() {

if (!error\_) {

cerr << funcname\_ << ":" << " pass" << endl;

}

cerr << std::flush;

}

[GeometryTesterMain.cpp](https://blackboard.le.ac.uk/bbcswebdav/pid-3154775-dt-content-rid-14858861_2/courses/CO3105_2021-22_SEM1/Asg/cw2a/GeometryTesterMain.cpp)

#include <iostream>

#include "GeometryTester.h"

using std::cout;

using std::endl;

int main(int argc, char\* argv[]) {

for(int i=1; i<argc; i++) {

switch (argv[i][0]) {

case 'a': { GeometryTester t; t.testa(); } break;

case 'b': { GeometryTester t; t.testb(); } break;

case 'c': { GeometryTester t; t.testc(); } break;

case 'd': { GeometryTester t; t.testd(); } break;

case 'e': { GeometryTester t; t.teste(); } break;

case 'f': { GeometryTester t; t.testf(); } break;

case 'g': { GeometryTester t; t.testg(); } break;

case 'h': { GeometryTester t; t.testh(); } break;

case 'i': { GeometryTester t; t.testi(); } break;

case 'j': { GeometryTester t; t.testj(); } break;

case 'k': { GeometryTester t; t.testk(); } break;

case 'l': { GeometryTester t; t.testl(); } break;

case 'm': { GeometryTester t; t.testm(); } break;

case 'n': { GeometryTester t; t.testn(); } break;

case 'o': { GeometryTester t; t.testo(); } break;

case 'p': { GeometryTester t; t.testp(); } break;

case 'q': { GeometryTester t; t.testq(); } break;

case 'r': { GeometryTester t; t.testr(); } break;

case 's': { GeometryTester t; t.tests(); } break;

case 't': { GeometryTester t; t.testt(); } break;

case 'u': { GeometryTester t; t.testu(); } break;

case 'v': { GeometryTester t; t.testv(); } break;

case 'w': { GeometryTester t; t.testw(); } break;

case 'x': { GeometryTester t; t.testx(); } break;

case 'y': { GeometryTester t; t.testy(); } break;

case 'z': { GeometryTester t; t.testz(); } break;

default: { cout << "Options are a -- y." << endl; } break;

}

}

return 0;

}