CMPS109 Spring 2018 : Lab 4

In this lab you will implement an automated configurable testing system for a C++ two and three-dimension geometric bounds checker derived from those developed in Lab2 and Lab 3.

This lab is worth 7% of your final grade.

Submissions are due NO LATER than 23:59, Sunday April 29, 2018.

Late submissions will not be graded.

Background information

As you continue to become familiar with C++, remember that Google is your friend and continue to make extensive use of the myriad of resources available on-line, including UCSC Canvas discussions.

In addition, one of the TAs will be holding extended office hours from this week onwards, so watch out for Canvas notifications of times and locations.

Setup

SSH in to any of the CMPS109 teaching servers using your CruzID Blue credentials:

```
$ ssh <cruzid>@<server>.soe.ucsc.edu
```
(use Putty http://www.putty.org/ if on Windows)

Authenticate with Kerberos and AFS: ***(do this EVERY time you log in)***

```
$ kinit
$ aklog
```

Create a suitable place to work: ***(only do this the FIRST time you log in)***

```
$ mkdir -p ~/CMPS109/Lab4
$ cd ~/CMPS109/Lab4
```

Recursively copy your Lab 3 solution: ***(only do this once)***

```
$ cp -r ~/CMPS109/Lab3/* .
```

Install the lab environment: ***(only do this once)***

```
$ tar xvf /var/classes/CMPS109/Spring18/Lab4.tar.gz
```

Build the skeleton system:

```
$ make
```

Check your implementation:

```
$ make check
```

Calculate your expected grade:

```
$ make grade
```

Check your submission for compliance with naming conventions:

```
$ make comply
```

No response indicates your solution is in compliance with the naming requirements.

What to submit

In a command prompt:

```
$ cd ~/CMPS109/Lab4
$ make submit
```

This creates a gzipped tar archive named `CMPS109-Lab4.tar.gz` in your home directory. Note that the submission archive will not be created if your solution fails to comply with the naming conventions outlined below.

UPLOAD THIS FILE TO THE APPROPRIATE CANVAS ASSIGNMENT.

Requirements

In Lab 3 you completed the implementation of the `Circle` and `RegularConvexPolygon` classes and implement from scratch the `ReuleauxTriangle` class.

You now need to implement the `Cube`, `Sphere`, and `ReuleauxTetrahedron` classes, ensuring you adhere to required include file and class naming conventions.

More importantly, rather than laboriously implementing numerous individual tests, you significantly raise your programming game and implement an automated testing system driven by a simple ASCII configuration file with the following format for each line:

```
inner i-arg1 ... i-argn outer o-arg1 ... o-argn expected [text]
```

Where `inner` and `outer` are one of the six classes implemented, and `i-arg1` to `i-argn` are the arguments to the non-default constructor on the inner shape. Likewise, `o-arg1` to `o-argn` are the arguments for the `outer` shape non-default constructor.

The penultimate element on the line, `expected` (either `true` or `false`, is the expected result from calling `inner.containedWithin(outer)`.

The final element on the line, `text`, is an optional string for inclusion in log messages about the test.

Examples:

```
RegularConvexPolygon -4,-4 -4,4 4,4 4,-4 Circle 0,0 8 true square inside circle
```

Indicates an 8x8 square centered on the origin is contained by a circle of radius 8 centered on the origin.

```
Circle 3,3 4 RegularConvexPolygon -4,-4 -4,4 4,4 4,-4 false circle intersects square
```

Indicates a circle of radius 4 centered at 3,3 is not contained within an 8x8 square centered on the origin.

In this configuration file:

- The token delimiter is a space
- Lines beginning with # should be regarded as a comment and be ignored
- Blank lines should be ignored
- Leading and trailing whitespace should be ignored
- Two dimensional coordinates are given as `x,y` with no spaces
- Three dimensional coordinates are given as `x,y,z` with no spaces
- Coordinates for polygons should be given either clockwise or anti-clockwise
- Coordinates for cubes should be given as eight vertices, the first four-tuple representing the top face, the second four-tuple representing the bottom face. Both four-tuples should follow the anti-clockwise or clockwise presentation required for polygons
- Coordinates for reuleaux shapes can, naturally, be given in any order

What you need to do

Assuming you completed Lab 3, the `Circle` and `RegularConvexPolygon` and `ReuleauxTriangle` classes are now complete. If they are not, you need to finish them off, testing as you go.

As in Lab 3, if there's something you don't understand, do this, in this order:

1. Google it
2. Post a discussion on Canvas
3. Come along to a section and/or office hours and ask questions

Before starting on the three-dimensional shape implementations, you should implement your test harness in `src/main.cc`.

You can write your test harness any way you like, but some of the functionality you'll almost certainly need to implement is as follows:

- Open the test file which is passed as the first command line argument
- Parse the test file line-by-line rejecting any that are badly formed
- From well-formed lines, construct the required inner and outer objects
- Call `inner.containedWithin(outer)`
- Check the calculated result against the expected result
- Print a message containing the strings "PASS" or "FAIL" as appropriate and the test text if supplied

To execute your tests stand-alone after running `make`:

```
$ ./bounds test-file
```

Or:

```
$ ./check.sh
```

The latter will assume the test file is named "bounds.tests".

Note that the automated grading system has its own implementation of the test harness. The results from your implementation will be evaluated against those of the grading system

Header file naming & missing constructors

The header file you create for the missing classed must have the following names:

```
reuleauxtriangle.h      (will exist if you completed Lab 2)
sphere.h
cube.h
reuleauxtetrahedron.h
```

You can implement the missing class any way you like, but you must include the following constructors:

```
ReuleauxTriangle(Point2D vertices[3]);     (will exist if you completed Lab 2)
Sphere(Point3D center, double radius);
Cube(Point3D upperFace[4], Point3D lowerFace[4]);
ReuleauxTetrahedron(Point3D vertices[4]);
```

Failing to include these constructors and header files will cause a compilation error in the automated grading system and your submission will receive no marks.

To check your solution for compliance with these naming conventions:

```
$ make comply
```

No response indicates your solution is in compliance with the naming requirements.

A word of advice

Do NOT assume the Lab 3 supplied geometric code is 100% accurate.

Test driven development is a consistent theme across all the labs; whenever you get some code from a third party, most more typically copied from an on-line resource, always be wary of its accuracy. If something isn't working as expected, investigate and modify as necessary.

Most of you will have already modified the supplied code from Lab 3, and if you have a collection of robust three-dimensional geometric functions from Lab 2 feel free to use them.

Grading scheme

The following aspects will be assessed by executing your code on a machine with an identical configuration to the CMPS109 teaching servers:

1. (70 Marks) **Does it work?**

    a. Geometric Calculations (60 marks)
    b. Code free of warnings (10 marks)

For a, marks are deducted for any geometric calculations failing to produce the correct answer.

Note that the test script used by the automated grading system will **NOT** be the same as the trivial skeleton script provided in the lab environment you installed.

2. (-100%) **Did you give credit where credit is due?**

    a. Your submission is found to contain code segments copied from on-line resources and you failed to give clear and unambiguous credit to the original author(s) in your source code (-100%)

    b. Your submission is determined to be a copy of another CMPS109 student's submission (-100%)

    c. Your submission is found to contain code segments copied from on-line resources that you did give a clear an unambiguous credit to in your source code, but the copied code constitutes too significant a percentage of your submission:

        - < 50% copied code No deduction
        - 50% to 75% copied code (-50%)
        - > 75% (-100%)

