EECS 211 Homework 1

Winter 2019

Due:	January 17, 2019 at 11:59 PM
Partners:	No; must be completed by yourself

Purpose

The goal of this assignment is to get you programming in C, including simple I/O, separate compilation, and `assert`-based testing.

Preliminaries

Login to the server of your choice and *cd* to the directory where you keep your EECS 211 work. Then download and unarchive the starter code, and change into the project directory:

This homework assignment must be completed on Linux using the T-Lab or Wilkinson Lab machines. Each time you login to work on EECS 211, you need to run the *dev* command (as set up in Lab 1).

```
$ curl $URL211/hw/hw01.tgz | tar zvx
⋮
$ cd hw01
```

You can check that you have correctly downloaded and configured everything by building the project:

```
$ make all
⋮
3 warnings generated.
cc -o build/overlapped build/overlapped.o build/cir...
$
```

You will see warnings because several function definitions are incomplete, but the build should complete successfully.

Orientation

In this project, you will write:

- a tiny computational geometry library (src/circle.h and src/circle.c),
- a tiny client program that uses it (src/overlapped.c), and
- some tests for the library (test/test_circle.c).

Type definitions and function signatures for the library are provided for you in src/circle.h; since the grading tests expect to interface with your code via this header file, **you must not modify src/circle.h in any way.** All of your code will be written in the three .c files.

This multifile setup mirrors the structure discussed in Lecture 3, so you may want to refer to those slides for reference.

The project also provides a Makefile with several targets:

target	description
all	builds everything * †
test	builds and runs the tests †
build/test_circle	builds (but doesn't run) the tests
build/overlapped	builds the client program
clean	removes all build products †

* default † phony

Specifications

The project comprises two functional components, which are specified in the next two subsections.

The circle *library*

The *circle* library defines one **`struct`** type and three functions, as follows:

- The `circle` structure type represents a circle positioned on a Euclidean plane in terms its center (x and y coordinates) and its radius.
- Function `valid_circle(`**`struct`**` circle c)` returns a **`bool`** indicating whether circle `c` is *valid*. A circle is valid if and only if its radius is positive.
- Function `read_circle()` parses a **`struct`**` circle` from the standard input and returns it. It should expect the values of the three fields in order: `x`, `y`, `radius`.

  Exceptional cases: The returned circle must be fully initialized even if *scanf*() fails due to bad or end of input. If the input ends or is malformed, `read_circle()` returns a circle with center $(0.0, 0.0)$ and radius -1.0.
- Function `overlapped_circles(`**`struct`**` circle, `**`struct`**` circle)` returns a **`bool`** indicating whether the two given circles overlap. Circles are considered to overlap only if they contain some area in common, not if they are merely tangent to each other.

The overlapped *client program*

The *overlapped* client program reads a first ("target") circle. If there is an error in reading the target circle, the program terminates with an exit code of 1 to indicate an error.

Then the program reads as many subsequent ("candidate") circles as are provided by the user; for each valid circle read after the target circle, it prints `"overlapped\n"` if the candidate circle overlaps the target, or `"not␣overlapped\n"` if not. If the program reads an invalid candidate circle, then it terminates with an exit code of 0 to indicate success, printing nothing.

The program does not print anything else.

Here are two examples of running build/overlapped:

```
$ build/overlapped
0 0 5
0 2 1
overlapped
0 10 1
not overlapped
2019 211 -1
$
```

```
$ build/overlapped
1 0 1
0 1 0.4
not overlapped
0 1 0.41
not overlapped
0 1 0.414
not overlapped
0 1 0.415
overlapped
1 -1 0.415
overlapped
-2019 -211 -2
$
```

Reading documentation effectively can depend on understanding typesetting conventions. In the transcripts on the left, the **bold** text is what the user types, and the `medium weight` text is what the computer responds with. Your actual prompt will probably differ from `$`, which is the convention for printing Unix shell prompts in documentation.

Hints

Definition of overlap for circles

Two circles overlap if the distance between their centers is less than the sum of their radii.

You don't actually need *sqrt*() to do this, because this statement is equivalent: Two circles overlap if the square of the distance between their centers is less than the square of the sum of their radii.

Strategy for the `read_circle` *function*

First define a **`struct`** `circle` variable, without initializer, to hold the function's result. Then, try to initialize its three fields using *scanf*(). If *scanf*() is unable to convert all three **`double`**s as indicated by its result value, then initialize the **`struct`** `circle` to the invalid state `{0.0, 0.0, -1.0}` instead (per the specification above). Then, whether or not the input succeeded, return the **`struct`** `circle`.

Algorithm for the overlapped *program*

Here is an algorithm you can use in src/overlapped.c:

1. Define a **`struct`** `circle` variable to hold the target circle, and initialize it to the result of calling `read_circle()`.

2. If the target circle is invalid according to `valid_circle()`, exit with an error code of 1.

3. Repeat indefinitely:

   (a) Define a **`struct`** `circle` variable to hold the candidate circle, and initialize it to the result of calling `read_circle()`.

   (b) If the candidate circle is invalid according to `valid_circle()`, exit with an error code of 0.

   (c) Use `overlapped_circles` in the condition of an **`if–else`** statement to check whether the target circle overlaps the candidate circle and print the correct message in either case.

   To get an infinite loop that repeats some statements, use a **`for`** loop with empty condition:

```
for (;;) {
    // Statements to repeat go here.
}
```

From `main`, exiting can be accomplished by **`return`**ing the desired error code, but to exit from another function one must call the *exit*(3) function.

(Note that the "3" in *exit*(3) is not the argument you should pass, but the section of the Unix manual system where documenation for the *exit* function is found. To see why this matters, compare the result of running `man exit` with the result of running `man 3 exit`.)

Deliverables and evaluation

For this homework you must:

1. Implement the specification for the *circle* library from the previous section in src/circle.c.

2. Implement the specification for the *overlapped* client program from the previous section in src/overlapped.c.

3. Add more test cases for the `overlapped_circles` function provided by the *circle* library in test/test_circle.c.

   In particular, file src/test_circle.c already contains two tests cases, `test_tangent` and `test_not_overlapped`, both of which are called from `main`. Your job is to add two more test cases, demonstrating that:

   - `overlapped_circles` returns **`true`** given different but overlapping circles, and
   - `overlapped_circles` returns **`true`** given the same circle for both arguments.

Grading will be based on:

- the correctness of your implementations with respect to the specifications,
- the presence of the two required test cases, and
- adherance to the EECS 211 Style Manual.

Submission

Homework submission and grading will use the GSC grading server. You must upload any files that you create or change. For this homework, that will include src/circle.c, src/overlapped.c, and test/test_circle.c. (You should not need to modify Makefile and you must not modify src/circle.h.)

Submit using the command-line GSC client **gsc**(1). Instructions are available in the **submit211**(7) manual page on the lab machines. To view it, run:

```
$ man submit211
```