Lab 6: Introduction to Loops

Due date: Thursday, October 24 11:59pm.

Lab Assignment

Assignment Preparation

Lab type. This is an **individual lab**. Each student will submit his/her set of deliverables.

Collaboration. Students are allowed to consult their peers[1] in completing the lab. Any other collaboration activities will violate the *non-collaboration* agreement. No direct sharing of code is allowed.

Purpose. You are continuing your acquaintance with C functions and their use.

Programming Style. All submitted C programs must adhere to the programming style described in detail at

http://users.csc.calpoly.edu/∼dekhtyar/CStyle.htm

When graded, the programs will be checked for style. Any stylistic violations are subject to a penalty. Significant stylistic violations, epsecially those that make grading harder, may yield reasonably strict penalties.

Testing and Submissions. Any submission that does not compile using the

```
gcc -ansi -Wall -Werror -lm
```

compiler settings will receive an automatic score of 0.

The Task

In the last few labs we used instructor's programs `print-all`, `show-badges` and `print-badges` to test your implementations of various Grower's Grove Game functions.

This lab asks you to implement your own versions of these programs.

[1] A peer for the purpose of CPE 101 is defined as *"student taking the same section of CPE 101"*.

Formatted Badge Output

In **Lab 5**, a program called `show-badges` printed the results of your `describeBadges()` function for every single field in the Grower's Grove.

Your first task is to write that program. Name the program `show-badges.c`.

The program itself shall consist of a single `int main()` function, which contains all necessary variable declarations and all the code that outputs the badge descriptions *in the same order* as the instructor's version of the program (use the executable from **Lab 5** for testing purposes).

It should be possible to compile your program using the following complilation instruction:

```
$ gcc -ansi -Wall -Werror -lm -o myshow-badges show-badges.c grove.c badges.c
```

Maps of Grower's Grove function values

In **Lab 4** and **Lab 5** we used instructor's program `print-all` do create a dump of all values returned by your Grower's Grove functions in a form of a 20×20 grid of values.

Note, that because some values have more digits than other values, the maps created this way do not always look like nice square matrices of numbers. However, outputs produced that way are easy to grade and to use for various analyses of the future game content.

Your second task in this lab is to write your own version of `print-all`. Name your file `print-all.c`.

The `print-all` program shall consist of a single `int main()` function, which shall contain all variable declarations and all code necessary to produce the output that matches the output of the instructor's version. Use intructor's executable from **Lab 5** for comparison.

Please note, that your program is expected to produce the same output as the instructor's program **character-for-character**. This means that everything, including line spacing, and the "header" lines must look exactly the same. If the `diff` between your output and the instructor's output is not empty, you won't receive full credit.

The order of the output of `print-all` shall be the same as in the instructor's program. For convenience, it is specified here:

Position	Function output
1.	`soilQuality()`
2.	`sunExposure()`
3.	`irrigationExposure()`
4.	`estimateYield()`
5.	`estimateQuality()`
6.	`harvestTime()`
7.	`plantingCost()`
8.	`pricePetUnit()`
9.	`revenue()`
10.	`fieldProfit()`
11.	`roi()`
12.	`annualRevenue()`
13.	`fieldScore()`

Final note. Take special care when printing the values. Remember that values of field $(0, 0)$ must be printed in the *bottom left corner* of the matrix output.

It should be possible to compile your `print-all.c` program using the following command:

```
$ gcc -ansi -Wall -Werror -lm -o myprint-all print-all.c grove.c
```

(notice that `print-all` does not touch any badge-related functionality).

Maps of Badge Locations

In **Lab 5**, instructor's program `print-badges` was used to create maps of badge locations in Grower's Grove. Your third, and last, task for this lab is to create your own version of this program.

Name your program `print-badges.c`. The program shall consist of a single `int main()` function, which contains all necessary variable declarations and code.

The output of the program must match instructor's output **character-for-character**. This includes the exact matches on the headers and footers of teach map, and the line spacing. To get 100% credit, the result of `diff` of your output and the instructor's output (use **Lab 5** instructor's version of `print-badges`) must be empty.

The program shall print the badge maps in the following order:

Position	Badge
1.	Boundary Maven
2.	Inner Circle
3.	Local Hero
4.	Boring Weather
5.	In Quattro
6.	Poly
7.	Any Color You Like

Final note. Take special care when printing the values. Remember that values of field $(0, 0)$ must be printed in the *bottom left corner* of the matrix output.

It should be possible to compile your `print-badges.c` program using the following command:

```
$ gcc -ansi -Wall -Werror -lm -o print-badges myprint-badges.c grove.c badges.c
```

General notes

Make certain your new programs have the right `#include` directives in them. Recall that you have two header files, `grove.h` and `badges.h` that contain your function declarations.

To test how your programs fair against the instructor's programs use the following commands (we are assuming that the instructor's binaries are named `print-all`, `print-badges` and `show-badges` and your binaries are named `myprint-all`, `myprint-badges` and `myshow-badges`. Adjust the commands below accordingly for different file names).

```
$ print-all > alex-print.out
$ myprint-all > print.out
$ diff print.out alex-print.out
```

```
$ show-badges > alex-show.out
$ myshow-badges > show.out
$ diff show.out alex-show.out
$ print-badges > alex-badges.out
$ myprint-badges > badges.out
$ diff badges.out alex-badges.out
$
```

If your test session looks exactly as above (no visibile output for all three `diff` commands), your are doing everything right.

Submission.

Files to submit. You shall submit the following files: `grove.h`, `grove.c`, `badges.h`, `badges.c`, `print-all.c`, `print-badges.c`, and `show-badges.c`.

Your file names shall be as specified above (and remember that Linux is case-sensitive). We use automated grading scripts. Any submission that has to be compiled and run manually will receive a deduction.

Submission procedure. Use `handin` to submit your work. The procedure is as follows:

- `ssh` to `unix1`, `unix2`, `unix3` or `unix4`.
- Upon login, change to your Lab 5 work directory.
- Execute the `handin` command.

```
> handin dekhtyar lab06  <files>
```

(note, you can submit files one by one, rather than using one command.)

Other submission comments. Please, DO NOT submit binary files. Please, DO NOT submit binary files. (this has been an issue in the past.)