Lab 5: Conditional Statements

Due date: Tuesday, October 22, 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

This assignment continues the work on the Grower's Grove Game. We will address another aspect of gaming, *awarding achievement badges* to the player who reached certain milestones in their groveberry growing.

From the programming standpoint, most of the functions you need to develop for this lab require significant use of conditional statements (`if` statements, nested `if` statements and, at times, `switch` statements).

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

Part 1: Getting your functions to produce the correct output

The new functions in this lab require you to use the functions from the `grove.c` library that you have developed in **Lab 3** and **Lab 4**. To make sure your new functions work correctly and pass all the tests, you need to ensure that all functions from **Lab 3** and **Lab 4** operate properly. Specifically, before starting work on **Lab 5**, you need to ensure that the following functions work correctly:

```
double soilQuality(int x, int y);       /* computes soil quality of the field         */
double sunExposure(int x, int y);       /* computes sun exposure levels of the field   */
double irrigationExposure(int x, int y);/* computes the irrigation levels of the field */

double estimateYield(int x, int y);     /* estimates crop yield from the field        */
double estimateQuality(int x, int y);   /* estimates quality of the crop from the field*/
double harvestTime(int x, int y);       /* estimates time to harvest for the field     */

double fieldScore(int x, int y);        /* computes the overall score of the field     */

double plantingCost(int x, int y);      /* computes cost of planting                               */
double pricePerUnit(int x, int y);      /* computes price of collected harvest based on its quality  */
double revenue(int x, int y);           /* computes revenue from the harvest sales                 */
double fieldProfit(int x, int y);       /* computes computes profit from the harvest               */
double roi(int x, int y);               /* computes return on investment for a single harvest      */
double annualRevenue(int x, int y);     /* computes annual revenue from the field                  */
```

You can observe the output of the instructor's versions of these functions by running the `print-all` program (`print-all` or `print-all32` programs shared with you as part of **Lab 4** materials).

You can compile your `grove.c` file together with the instructor's `print-all` program, by downloading either `print-all.o` or `print-all32.o` object file into your lab directory and executing the following command:

```
$ gcc -ansi -Wall -Werror -lm -o myprint-all print-all32.o grove.c
```

You can then compare your results to the instructor's results by running the following commands:

```
$ myprint-all > myOutput.txt
$ print-all32 > alexOutput.txt
$ diff myOutput.txt alexOutput.txt > lab4.diff
$ more lab4.diff
```

The commands above run your and the instructor's versions of `print-all` and output the results in `myOutput.txt` and `alexOutput.txt` files respectively. The `diff` command compares the two outputs and writes the difference to the file `lab4.diff`. The last command outputs that file to screen. *If you did everything right,* `lab4.diff file will be empty`, and the last command will not produce any visible output.

If you have errors in your code that lead to different outputs, find out which functions produce incorrect output, and fix the problems. Make certain you can produce all the same numbers as the instructor's functions before proceeding further - if you cannot, your new functions won't work properly.

Badges

The Grower's Grove Game will award a number of badges for different achievements in planting and harvesting groveberries. In this lab, we will concentrate on the badges that can be obtained *for a single planting groveberries on a single field*, but in the future, we may add badges that take multiple turns to get awarded.

In the Grower's Grove Game, an achievement badge is an in-game mini-award for the player fulfilling a certain predefined condition or a set of conditions while playing the game. Later, the game mechanics may involve awarding points, or extra money for achieving the badges.

For this lab assignment, we will define a number of badges that depend on a single planting of groveberries at a single field. You are asked to develop functions that take as input the field coordinates and determine (yes/no) if a badge can be awarded for this particular planting.

Setup

You will write all new functions for this lab in a new file `badges.c`. The file shall contain the following preprocessor instructions:

```
#include <math.h>
#include <stdio.h>
#include "badges.h"
#include "grove.h"
```

The `grove.h` file shall contain the function declarations for all the new functions in the lab. The instructor's version of this file is provided to you for your convenience. Like `grove.c` before it, `badges.c` shall contain only the definitions of the new functions, and **shall not** contain a `main()` function. See the **Testing** section below for further instructions on how to compile and run programs that use the new functionality.

Writing Code

Here are some comments/advice concerning writing code for this assignment.

Use of local variables. If your function must call another function with exactly the same parameters *more than once*, define a local variable, and assign the value returned from this function call to it. Then use the variable elsewhere.

For example, the following code:

```
if ( f(x,y) > 20 && f(x,y) < 40) {
   ...
 }
```

can be replaced with

```
double value;
```

```
value = f(x,y);
if (value > 20 && value < 40) {
  ...
}
```

(*Short explanation:* function calls are "expensive" in terms of number of machine instructions. You can get away with calling the function only once, by "archiving" the computed value and using it in the rest of the code multiple times.)

Counting with comparisons. Recall that comparisons are *valid* `C` expression that return an `int` value. You can use this when asked to count how many values are, for example, equal to 10. Here is a simple way to determine how many of the four variables `int a,b,c,d` have values equal to 10:

```
int a,b,c,d;
int counter;

.... /* code establishing values for a,b,c,d */

counter = (a==10) + (b==10) + (c==10) + (d==10);
```

if, or if-else? Some of your tasks may be performed by either an `if-else` conditional statement, or by a careful use of a pure `if` statement. The trick is to ensure that you `return` at the end of the `if` code block. The following two code fragments do the same thing.

```
int foo(int a, int b) {
   if (a > b) {
      return (a+b);
   }
   else {
      return 0;
   }
}
```

and

```
int foo(int a, int b) {
    if (a > b) {
       return (a+b);
    }

    return 0;
}
```

The reason why this works is because the code block inside the `if` clause ends with a `return`, which means that when $a > b$, the execution of the function will never go beyond the end of the `if` code block. So, the only case in which the execution can get to the `return 0;` statement in the second code fragment is when $!(a > b)$, i.e., in the situation logically equivalent to the `else` clause in the first code fragment.

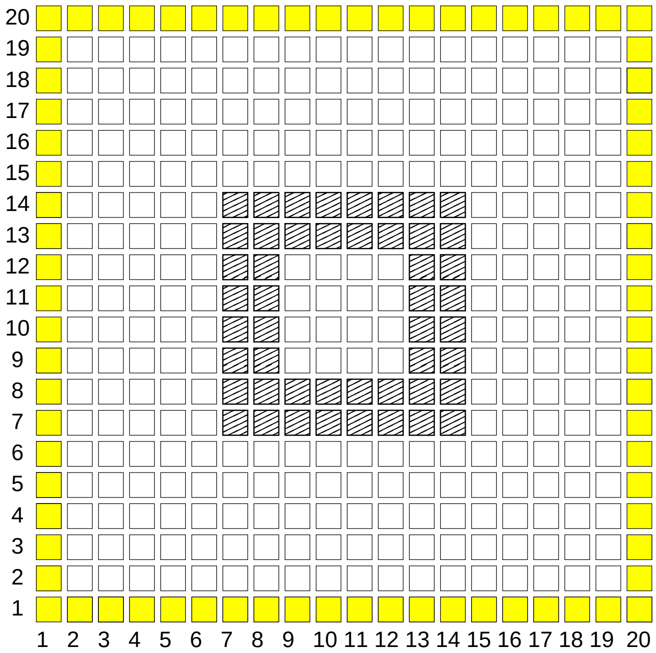

Figure 1: Geolocation-based badges. Light shade (yellow) shows the applicability of the Boundary Maven badge, while the patterned shade shows the applicability of the Inner Circle badge.

Part 2: Bronze Badge Awards

Bronze badge awards are the simplest of all. They depend solely on the location of the field in which groveberries were planted, and the growing conditions/other parameters that can be computed using the functions from **Lab 3** and **Lab 4** listed in **Part 1** of this assignment.

The Boundary Maven Badge

The Boundary Maven badge is given to a player who plants groveberries on the boundary of the Grower's Grove property and who grows a decent harvest. More formally, the Boundary Maven badge is awarded to a player who plants groveberries on a field (x, y) that satisfies the following conditions:

1. The field is on the boundary of the property, i.e., either x or y take value of either 1 or 20. See Figure 1 (light/yellow shade) for the region of applicability of this badge.
2. Expected quality of the harvest exceeds 6.
3. Expected yield of the harvest exceeds 100.

Function. Write a function

```
int badgeBoundaryMaven(int x, int y)
```

to determine whether a field (x, y) earns a Boundary Maven badge. The function shall validate the input, i.e., ensure that both x and y are between 1 and 20 and output the following values:

return value	condition
-1	field (x, y) is invalid
0	field (x, y) does not earn the Boundary Maven badge
1	field (x, y) earns the Boundary Maven badge

The Inner Circle Badge

The Inner Circle badge is given to a player who plants groveberries in one of the fields that form a small "inner" circle inside the Grower's Grove property, and grows a reasonable harvest. More formally, the Inner Circle badge is awarded to a player who plants groveberries on a field (x, y) that satisfies the following conditions:

1. The field is inside the square whose southwest corner is $(7, 7)$ and northeast corner is $(14, 14)$, but not in the inner square bounded by field $(9, 9)$ on the southwest and the field $(12, 12)$ on the northeast. See Figure 1 (dark patterned shade) for the region of applicability of this badge.
2. The field turns a profit of 10 or above on a single planting of groveberries.
3. The field has an overall field score of 10 or above.

Function. Write a function

```
int badgeInnerCircle(int x, int y)
```

to determine whether a field (x, y) earns an Inner Circle badge. The function shall validate the input, i.e., ensure that both x and y are between 1 and 20 and output the following values:

return value	condition
-1	field (x, y) is invalid
0	field (x, y) does not earn the Inner Circle badge
1	field (x, y) earns the Inner Circle badge

The Local Hero Badge

The Local Hero badge is given to a player who plants groveberries in a "locally optimal" field, i.e., a field that brings in more money than any neighboring field. More formally, the Local Hero badge is awarded to a player who plants groveberries on a field (x, y) that satisfies the following conditions:

1. The (x, y) *does not* lie on the Grover's Grove boundary (see Figure 1: any field that is not lightly shaded (yellow) is a candidate for this badge).
2. The profit generated by a single planting on the field is higher than the profit generated by a single planting on any neighboring fields. For the purpose of awarding this badge, all eight adjacent fields around (x, y) (North, South, East, West, North-East, North-West, South-East and South-West) are considered as neighboring fields.

Function. Write a function

```
int badgeLocalHero(int x, int y)
```

to determine whether a field (x, y) earns a Local Hero badge. The function shall validate the input, i.e., ensure that both x and y are between 1 and 20 and output the following values:

return value	condition
-1	field (x, y) is invalid
0	field (x, y) does not earn the Local Hero badge
1	field (x, y) earns the Local Hero badge

The Boring Weather Badge

The Boring Weather badge is given to a player who plants groveberries at a field with "moderate" conditions (sun exposure and irrigation exposure) and manages to grow a meaningful harvest out of it. More formally, the Local Hero badge is awarded to a player who plants groveberries on a field (x, y) that satisfies the following conditions:

1. The sun exposure of the field is between 6 and 8 (inclusive on both ends).
2. The irrigation exposure of the field is between 4 and 7 (inclusive on both ends).
3. The field yields over 100 units of harvest.
4. The quality of the harvest is 3 or above.

Function. Write a function

```
int badgeBoringWether(int x, int y)
```

to determine whether a field (x, y) earns a Boring Weather badge. The function shall validate the input, i.e., ensure that both x and y are between 1 and 20 and output the following values:

return value	condition
-1	field (x, y) is invalid
0	field (x, y) does not earn the Boring Weather badge
1	field (x, y) earns the Boring Weather badge

The In Quattro Badge

The In Quattro badge is given to a player who plants groveberries at a field which can yield four full harvests a year, with reasonable demands on resources it takes to grow the berries on the field. More formally, the In Quattro badge is awarded to a player who plants groveberries on a field (x, y) that satisfies the following conditions:

1. The field supports four full harvests a year (but does not support five full harvests).
2. The cumulative annual cost[2] of planting at the field is between 190 and 250 units.

[2]Based on the four full plantings.

3. The field brings profit.

Function. Write a function

```
int badgeInQuattro(int x, int y)
```

to determine whether a field (x, y) earns a In Quattro badge. The function shall validate the input, i.e., ensure that both x and y are between 1 and 20 and output the following values:

return value	condition
-1	field (x, y) is invalid
0	field (x, y) does not earn the Boring Weather badge
1	field (x, y) earns the Boring Weather badge

Part 3: Silver Badge Awards

Silver badge awards are more complicated. They have conditions on their own, which can include presence and/or absense of some other awards either for a given field, or for a given vicinity of the field. There are two silver badge awards that you need to implement in this lab.

The Poly Badge

The Poly Silver Badge is given to the player who plants on a field that yields multiple bronze badges. The formal definition is as follows. The Poly Silver Badge award is awarded to a player who plants groveberries on a field (x, y) that receives *two or more* bronze awards.

Function. Write a function

```
int badgePoly(int x, int y)
```

to determine whether a field (x, y) earns a Poly badge. The function shall validate the input, i.e., ensure that both x and y are between 1 and 20, and output the following values:

return value	condition
-1	field (x, y) is invalid
0	field (x, y) does not earn the Poly badge
1	field (x, y) earns the Poly badge

Challenge. Write this function **without** using `if` or `switch` statements (except for validity check/variable guard check).

The Any Color You Like Badge

The Any Color You Like Silver Badge can be earned in a number of ways. These ways are listed below.

The Anyt Color You Like badge is awarded to a player who plants groveberries on a field (x, y) that satisfies the following conditions:

1. Any field can earn this badge, but the conditions for earning it differ based on the total number of bronze badges the field qualifies for.
2. **0 badges.** If the field does not earn any bronze badges, then it earns the Any Color You Like Badge if *all neighboring fields earn no* bronze badges. Each field outside the boundary has eight neighbors (North, South, East, West, North East, North West, South East, South West). Corner field have three neighbors. Other boundary fields have five neighbors.
3. **1 badge.** A field with one bronze badge earns the Any Color You Like badge if its overall field score is 20 or above.
4. **2 badges.** A field with two bronze badges earns the Any Color You Like badge if **at least one** of the following conditions is satisfied:
   - The field **does not** turn profit.
   - It takes less than 80 days to harvest groveberries on the field.
   - Its sun exposure is greater than its irrigation exposure.
5. **3 or more badges.** A field with three or more bronze badges if at least two of its *direct neighbors* earn two or more bronze badges. A *direct neighbor* of a field is any field to the North, South, East or West of it. (Note, fields on the boundary have three direct neighbors, corner fields have two direct neighbors)

Function. Write a function

```
int badgeAnyColor(int x, int y)
```

to determine whether a field (x, y) earns an Any Color You Like badge. The function shall validate the input, i.e., ensure that both x and y are between 1 and 20, and output the following values:

return value	condition
-1	field (x, y) is invalid
0	field (x, y) does not earn the Any Color You Like badge
1	field (x, y) earns the Any Color You Like badge

Requirement. Write this function using a switch statement. (you will also need plenty of `if` statements there, but at least one `switch` must be present).

Part 4: Badge Display function

The final assignment for this lab is the function

```
void describeBadges(int x, int y);
```

When called with the (x, y) coordinates of a field, the function must produce text output that specifies which badges the field can earn, if the player chooses it for planting of groveberries.

Your function **must match instructor's output character-for-character**, so that `diff` could not find any differences between the outputs. This includes whitespace used to position words, punctuation, new lines, and all text.

For each badge, the function will print the name of the badge and then will print `Yes` or `No` depending on whether the badge can be awarded for planting of groveberries on the given field.

Here is how the output of the `describeBadges(13,15)` function call looks like:

```
------------------------------

Field: 13, 15

=== Bronze Badges ===

Boundary Maven Badge: No
  Inner Circle Badge: No
    Local Hero Badge: No
Boring Weather Badge: Yes
    In Quattro Badge: Yes

=== Silver Badges ===

       Poly Silver Badge: No
Any Color You Like Badge: Yes

------------------------------
```

The instructor's program printing the badge descriptions for all fields in the Grower's Grove, as well as the `.o` file(s) to allow you to compile this program with your `badges.c` library of functions are provided to you.

Testing.

We share the following files with you to simplify testing of your code:

- `printall`, `printall32`: the executables of the instructor's program testing all functions from **Part 1** of this lab. For each function, you get a 20×20 "matrix" of values returned by it for each feasible field.
- `print-all.o`, `print-all32.o`: the so-called "object" files for the instructor's `print-all.c` program. You can compile them with your `grove.c` file to obtain an executable that will test **your implementations** of **Part 1** functions. The compilation command is

  ```
  $ gcc -ansi -Wall -Werror -lm -o myprint-all print-all32.o grove.c
  ```

- `badges.h`: instructor's header file with the list of function declarations for **Parts 2**, **3** and **4** of this lab. This is shared with you to give you a starting point for code development for this lab. You *may* edit the contents of the file as it suits you, but make sure that all function declarations stay intact.

- `print-badges`, `print-badges32`: instructor's executables printing information about each badge in the same way, `print-all` programs print various computed data about the fields. Instructor's `print-badges.c` program prints a 20×20 matrix of results of calling each badge check function (**Parts 2-3** of this lab). This gives you a good idea where the badges are in the field.
- `print-badges.o`, `print-badges32.o`: object files for `print-badges.c` to be compiled with your `grove.c` and `badges.c` files as follows:

  ```
  $ gcc -ansi -Wall -Werror -lm -o myprint-badges print-badges32.o grove.c badges.c
  ```

  To compare with instructor's output, run the following commands:

  ```
  $ myprint-badges > my.out
  $ print-badges > alex.out
  $ diff my.out alex.out output.diff
  $ more output.diff
  ```

  If your program produces exactly the same output as mine, you will see no output after the last command. The `output.diff` file will be empty.
- `show-badges.o`, `show-badges32.o`: instructor's program for testing `describeBadges()` function. It calls `describeBadges()` on every pair of valid inputs and thus produces a lot of output. You can compile it with your files as follows:

  ```
  $ gcc -ansi -Wall -Werror -lm -o myshow-badges show-badges32.o grove.c badges.c
  ```

- `badges-alex.out`: instructor's output from the `show-badges.c` program. Use it to compare to your output as follows:

  ```
  $ myshow-badges > mybadges.out
  $ diff mybadges.out badges-alex.out > badges.diff
  $ more badges.diff
  ```

  If things went well, the last command results in no output (`badges.diff` is empty).

Submission.

Files to submit. You shall submit the following files: `grove.h`, `grove.c`, `badges.h` and `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 lab05  <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.)