Problem 3-2: Fifteen (Part 2)

This is CS50. Harvard University. Fall 2014.

Table of Contents

Objectives 1
Recommended Reading 1
Academic Honesty 2
    Reasonable 2
    Not Reasonable 3
Assessment 4
Getting Ready 4
On Your Mark, Get Set… 5
    Option 1: Start from a Clean Slate 5
    Option 2: Extend Your Game 8
The Game Continues 8

Questions? Feel free to head to CS50 on Reddit[1], CS50 on StackExchange[2], the `#cs50ap` channel on CS50x Slack[3] (after signing up), or the CS50 Facebook group[4].

Objectives

- Accustom you to reading someone else's code.
- Empower you with Makefiles.
- Practice debugging your code with more advanced techniques.
- Finish implementing a party favor.

Recommended Reading

- Page 17 of http://www.howstuffworks.com/c.htm.

[1] https://www.reddit.com/r/cs50
[2] http://cs50.stackexchange.com
[3] https://cs50x.slack.com
[4] https://www.facebook.com/groups/cs50

- Chapters 20 and 23 of *Absolute Beginner's Guide to C*.
- Chapters 13, 15, and 18 of *Programming in C*.

Academic Honesty

This course's philosophy on academic honesty is best stated as "be reasonable." The course recognizes that interactions with classmates and others can facilitate mastery of the course's material. However, there remains a line between enlisting the help of another and submitting the work of another. This policy characterizes both sides of that line.

The essence of all work that you submit to this course must be your own. Collaboration on problems is not permitted (unless explicitly stated otherwise) except to the extent that you may ask classmates and others for help so long as that help does not reduce to another doing your work for you. Generally speaking, when asking for help, you may show your code or writing to others, but you may not view theirs, so long as you and they respect this policy's other constraints. Collaboration on quizzes and tests is not permitted at all. Collaboration on the final project is permitted to the extent prescribed by its specification.

Below are rules of thumb that (inexhaustively) characterize acts that the course considers reasonable and not reasonable. If in doubt as to whether some act is reasonable, do not commit it until you solicit and receive approval in writing from your instructor. If a violation of this policy is suspected and confirmed, your instructor reserves the right to impose local sanctions on top of any disciplinary outcome that may include an unsatisfactory or failing grade for work submitted or for the course itself.

Reasonable

- Communicating with classmates about problems in English (or some other spoken language).
- Discussing the course's material with others in order to understand it better.
- Helping a classmate identify a bug in his or her code, such as by viewing, compiling, or running his or her code, even on your own computer.
- Incorporating snippets of code that you find online or elsewhere into your own code, provided that those snippets are not themselves solutions to assigned problems and that you cite the snippets' origins.

- Reviewing past years' quizzes, tests, and solutions thereto.
- Sending or showing code that you've written to someone, possibly a classmate, so that he or she might help you identify and fix a bug.
- Sharing snippets of your own solutions to problems online so that others might help you identify and fix a bug or other issue.
- Turning to the web or elsewhere for instruction beyond the course's own, for references, and for solutions to technical difficulties, but not for outright solutions to problems or your own final project.
- Whiteboarding solutions to problems with others using diagrams or pseudocode but not actual code.
- Working with (and even paying) a tutor to help you with the course, provided the tutor does not do your work for you.

Not Reasonable

- Accessing a solution to some problem prior to (re-)submitting your own.
- Asking a classmate to see his or her solution to a problem before (re-)submitting your own.
- Decompiling, deobfuscating, or disassembling the staff's solutions to problems.
- Failing to cite (as with comments) the origins of code, writing, or techniques that you discover outside of the course's own lessons and integrate into your own work, even while respecting this policy's other constraints.
- Giving or showing to a classmate a solution to a problem when it is he or she, and not you, who is struggling to solve it.
- Looking at another individual's work during a quiz or test.
- Paying or offering to pay an individual for work that you may submit as (part of) your own.
- Providing or making available solutions to problems to individuals who might take this course in the future.
- Searching for, soliciting, or viewing a quiz's questions or answers prior to taking the quiz.
- Searching for or soliciting outright solutions to problems online or elsewhere.

- Splitting a problem's workload with another individual and combining your work (unless explicitly authorized by the problem itself).
- Submitting (after possibly modifying) the work of another individual beyond allowed snippets.
- Submitting the same or similar work to this course that you have submitted or will submit to another.
- Using resources during a quiz beyond those explicitly allowed in the quiz's instructions.
- Viewing another's solution to a problem and basing your own solution on it.

Assessment

Your work on this problem set will be evaluated along four axes primarily.

Scope
: To what extent does your code implement the features required by our specification?

Correctness
: To what extent is your code consistent with our specifications and free of bugs?

Design
: To what extent is your code written well (i.e., clearly, efficiently, elegantly, and/or logically)?

Style
: To what extent is your code readable (i.e., commented and indented with variables aptly named)?

To obtain a passing grade in this course, all students must ordinarily submit all assigned problems unless granted an exception in writing by the instructor.

Getting Ready

Have a more in-depth look at debugging techniques from Dan. (Odds are these 23 minutes with Dan will save you hours over the course of the term, since GDB is a far better tool than `printf` in many cases!)

https://www.youtube.com/watch?v=-G_klBQLgdc

On Your Mark, Get Set…

Unlike any other problem you've worked on up through this point in the course, this problem has been split into multiple parts. And that means you need to carry your work from the previous problem to this one. Maybe that's a good thing… perhaps you enjoyed working on Part 1 of the Game of Fifteen and can't wait to conclude it here. But perhaps you didn't do well on it, and dread the thought.

It is certainly not our goal to let the struggles you encountered in the past cascade and create even more trouble for you in the future. And to that end, any time you encounter a split problem like this in CS50 AP (or indeed any time prior work is necessary in a subsequent problem) you'll always have the opportunity for a clean slate by using a partial staff solution to get you going.

Below are two options for getting started with this problem. The first option is for those who wish to start with the staff's implementation of `init` and `draw` from Problem 3-1[5] already implemented for them. The second option is for those who wish to complete their own implementation of the Game of Fifteen, relying on their own solution to Problem 3-1 as their foundation. **Only choose one of the below two options**, though if you choose Option 2 you may still be interested in reading up on what your classmates who choose Option 1 are doing, particularly as it is an early loo

Then, after having chosen your option and followed all the steps therein, pick up at "The Game Concludes".

But first, in either case, log into cs50.io[6] and execute

```
update50
```

within a terminal window to make sure your workspace is up-to-date.

Option 1: Start from a Clean Slate

Remember all that stuff about Makefiles that we talked about in the last problem? And how the Makefile for the `fifteen` program was pretty simple, all things considered? Well,

[5] http://cdn.cs50.net/ap/1516/problems/3/1/3-1.html
[6] https://cs50.io/

you're about to encounter a slightly more complex `Makefile` which will in turn help you learn more about them, because you'll be downloading the staff's **object code** (remember what that is?), not our source code, and putting those two files together to create a single executable. How exciting!

To begin, in your terminal window, execute

```
cd ~/workspace/unit3
```

at your prompt to ensure that you're inside of `unit3` (which is inside of `workspace` which is inside of your home directory). Then execute

```
wget http://cdn.cs50.net/ap/1516/problems/3/2/fifteen2.zip
```

to download a ZIP of this problem's distro into your workspace (with a command-line program called `wget`). You should see a bunch of output followed by:

```
'fifteen2.zip' saved
```

Confirm that you've indeed downloaded `fifteen2.zip` by executing

```
ls
```

and then run

```
unzip fifteen2.zip
```

to unzip the file. If you then run `ls` again, you should see that you have a newly unzipped directory called `fifteen2` as well. You can now delete the ZIP, with:

```
rm fifteen2.zip
```

confirming your intent to delete that file, then proceed to execute

```
cd fifteen2
```

followed by

```
ls
```

and you should see that the directory contains four files:

```
Makefile  fifteen.c  fifteen.h  staff.o
```

`staff.o` is the aforementioned object file. **Take care not to delete this file**, as it contains the staff's implementations of `draw`, and `init` as the 0s and 1s of binary that your computer understands (but not the C code you've been writing).

`fifteen.h` is a user-made "header file", similar in spirit to some of the header files with which you're familiar by this point, such as `stdio.h` or `ctype.h`. Because of wanting to give you access to some staff code, we had to change the way we organize information in the Game of Fifteen. If you open up `fifteen.h`, you'll see that all of our `#include` and `#define` lines, as well as our function and global variable declarations have moved here. If you declare any new functions or global variables in completing this problem, make certain to place those declarations in `fifteen.h`.

`fifteen.c`, you'll note, looks a bit different, too. It includes `fifteen.h` only, the comments have been deliberately removed from `main` to make the code slightly shorter, and space for the implementations for `draw` and `init` have been removed (the resulting object code we wrote for those implementations now lives in `staff.o`. Now only the `TODO` placeholders for `move` and `won` remain.

Pop open `Makefile` (remember how), and notice how it is slightly different than the `Makefile` you saw in the last problem. Now, our `fifteen` executable has three dependencies:

```
fifteen: fifteen.c fifteen.h staff.o
 clang -ggdb3 -O0 -std=c11 -Wall -Werror -o fifteen fifteen.c staff.o -lcs50
 -lm
```

Thus notice that this time, `make` will only attempt to recompile `fifteen` if either `fifteen.c` changes (it very probably will!), `fifteen.h` changes (it might) `staff.o` changes (it hopefully won't!)

We also removed the option to `make clean` from this `Makefile`, to minimize the likelihood that you might accidentally delete `staff.o`, as recall that one of the things `make clean` does is to forcibly (without double-checking) delete all files with the `.o` extension.

Note that if you elect to use Option 1, you're obligated to adhere to some of the design choices we made when solving this problem. In particular, know that we decided that the board's blank space is implemented in `board` as `0` (and so you must now do the same) and we index into `board` a la `board[row][column]`, not `board[column][row]`.

Option 2: Extend Your Game

Execute

```
cd ~/workspace/unit3/fifteen
```

which should have been created in the last problem. Next type

```
ls
```

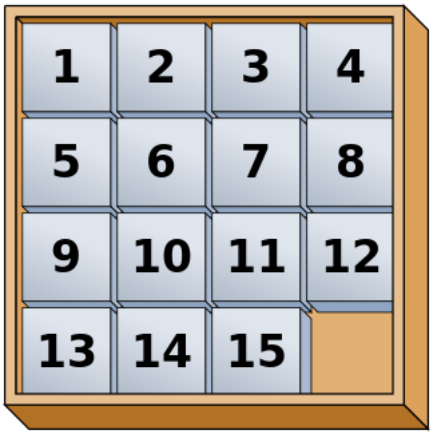
and you should see that the directory contains three (or four, depending on whether your executable remains in the directory from the first half of this problem) files.

```
Makefile  fifteen*  fifteen.c  questions.txt
```

The `fifteen` executable file may not be present, and that's okay. But the other three should be. `questions.txt`, in fact, should be completely filled in and `fifteen.c` should have its `init` and `draw` functions implemented.

The Game Continues

Recall that the Game of Fifteen is a puzzle played on a square, two-dimensional board with numbered tiles that slide. The goal of this puzzle is to arrange the board's tiles from smallest to largest, left to right, top to bottom, with an empty space in board's bottom-right corner, as in the below.



Sliding any tile that borders the board's empty space in that space constitutes a "move." Although the configuration above depicts a game already won, notice how the tile numbered 12 or the tile numbered 15 could be slid into the empty space. Tiles may not be moved diagonally, though, or forcibly removed from the board.

Your objective here is to complete the implementation of `fifteen` by implementing two functions: `move`, which simulates the action of sliding a tile around the board; and `won`, which determines if the game is in a winning state.

To test your implementation of `fifteen`, you can certainly try playing it. (Know that you can force your program to quit by hitting ctrl-c.) Be sure that you (and we) cannot crash your program, as by providing bogus tile numbers. And know that it is possible for you to automate execution of this game. In fact, in `~cs50/unit3` are `3x3.txt` and `4x4.txt`, winning sequences of moves for a 3 × 3 board and a 4 × 4 board, respectively. To test your program with, say, the first of those inputs, execute the below.

```
./fifteen 3 < ~cs50/pset3/3x3.txt
```

Feel free to tweak the appropriate argument(s) to `usleep` to speed up animation for testing, if you wish. In fact, you're welcome to alter the aesthetics of the game. For (optional) fun with "ANSI escape sequences," including color, take a look at our implementation of `clear` and check out http://isthe.com/chongo/tech/comp/ansi_escapes.html for more tricks.

You're welcome to write your own functions and even change the prototypes of functions we wrote (though, if using the staff solution as your starting point, it would not be wise to modify the prototypes of `init` or `draw`!). But we ask that you not alter the flow of logic

in `main` itself so that we can automate some tests of your program once submitted. In particular, `main` must only returns `0` if and when the user has actually won the game; non-zero values should be returned in any cases of error, as implied by our distribution code. If in doubt as to whether some design decision of yours might run counter these wishes, simply reach out to your teacher.

Finally, here are some tips from Zamyla. First, for `move`:

https://www.youtube.com/watch?v=gxMHcoBMiq4
And also for `won`:

https://www.youtube.com/watch?v=6KSq4JUfhIk
If you'd like to play with the staff's own implementation of the full `fifteen` game, you may execute the below.

```
~cs50/unit3/fifteen
```

If you'd like to see an even fancier version, one so good that it can play itself, try out the below.

```
~cs50/unit3/fifteen-solver
```

Instead of typing a number at the game's prompt in the latter, type `GOD` (named for the so-called "God Mode" implemented in many games of this sort, where the computer plays the game itself) instead. Neat, eh?

This was Problem 3-2.