Problem 3-1: Fifteen (Part 1)

This is CS50. Harvard University. Fall 2014.

Table of Contents

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.
- Begin to implement a party favor.

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

Recommended Reading

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

Getting Started

Recall that, for almost all of Units 1 and 2, you started writing programs from scratch, creating your own `unit1` and `unit2` directories with `mkdir`. For this problem, you'll instead download some "distribution code" (otherwise known as a "distro"), written by us, and add your own lines of code to it. You'll first need to read and understand our code, though, so this problem set is as much about learning to read someone else's code as it is about writing your own!

Let's get you started. Log into cs50.io[5] and execute

```
update50
```

within a terminal window to make sure your workspace is up-to-date. If you somehow closed your terminal window (and can't find it!), make sure that **Console** is checked under the **View** menu, then click the green, circled plus (+) in CS50 IDE's bottom half, then select **New Terminal**.

Next, 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/3/1/fifteen.zip
```

[5] https://cs50.io/

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:

```
'fifteen.zip' saved
```

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

```
ls
```

and then run

```
unzip fifteen.zip
```

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

```
rm fifteen.zip
```

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

```
cd fifteen
```

followed by

```
ls
```

and you should see that the directory contains two files:

```
fifteen.c  Makefile
```

Off we go!

Making Things Up

To kick things off, just

```
make fifteen
```

You shouldn't have touched anything yet, so this program should compile with no trouble.

Recall that `make` automates compilation of your code so that you don't have to execute `clang` manually along with a whole bunch of switches. Notice, in fact, how `make` just executed a pretty long command for you, per the tool's output. However, as your programs grow in size, `make` won't be able to infer from context anymore how to compile your code; you'll need to start telling `make` how to compile your program, particularly when they involve multiple source (i.e., `.c`) files. And so we'll start relying on "Makefiles," configuration files that tell `make` exactly what to do.

Go ahead and look at the file called `Makefile`. This `Makefile` is essentially a list of rules that we wrote for you that tells `make` how to build `fifteen` from `fifteen.c` for you. The relevant lines appear below.

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

The first line tells `make` that the "target" called `fifteen` should be built by invoking the second line's command. Moreover, that first line tells `make` that `fifteen` is dependent on `fifteen.c`, the implication of which is that `make` will only re-build `fifteen` on subsequent runs if that file was modified since `make` last built `fifteen`. Neat time-saving trick, eh? In fact, go ahead and execute the command below again, assuming you haven't modified `fifteen.c`.

```
make fifteen
```

You should be informed that `fifteen` is already up-to-date. Incidentally, know that the leading whitespace on that second line is not a sequence of spaces but, rather, a tab. Unfortunately, `make` requires that commands be preceded by tabs, so be careful not to change them to spaces, else you may encounter strange errors! The `-Werror` flag, recall, tells `clang` to treat warnings (bad) as though they're errors (worse) so that you're forced (in a good, instructive way!) to fix them.

Let's finish looking at that `Makefile`. Notice the line below.

```
all: fifteen
```

This target implies that you can build `fifteen` simply by executing the below.

```
make all
```

Even better, the below is equivalent (because `make` builds a `Makefile`'s first target by default).

```
make
```

If only you could whittle this whole problem set down to a single command! Finally, notice these last lines in `Makefile`:

```
clean:
    rm -f *.o a.out core fifteen log.txt
```

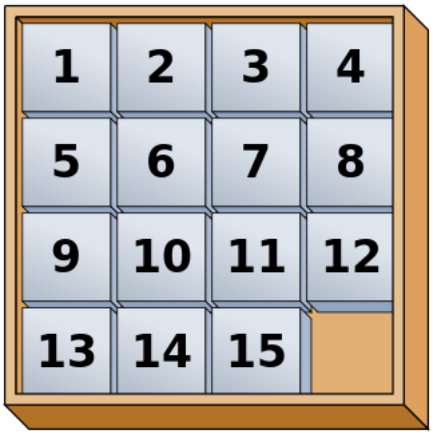
This target allows you to delete all files ending in `.o` or called `core` (more on that soon!), `fifteen`, or `log.txt` simply by executing the command below.

```
make clean
```

Be careful not to add, say, `*.c` to that last line in `Makefile`! (Why?) Any line, incidentally, that begins with `#` is just a comment. This Makefile is not terribly complex… but soon enough the principles we've just covered will make (pun intended!) our programming lives much simpler.

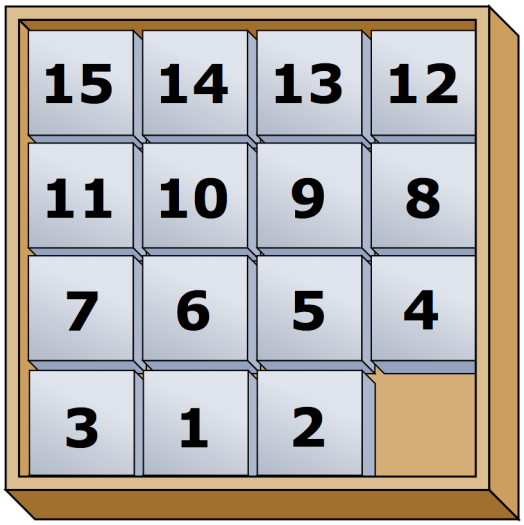
The Game Begins

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.

Although other configurations are possible, we shall assume that this game begins with the board's tiles in reverse order, from largest to smallest, left to right, top to bottom, with an empty space in the board's bottom-right corner. **If, however, and only if the board contains an odd number of tiles (i.e., the height and width of the board are even), the positions of tiles numbered 1 and 2 must be swapped, as in the below.** The puzzle is solvable from this configuration.



Okay, navigate your way to `~/workspace/unit3/fifteen`, and take a look at `fifteen.c`. Within this file is an entire framework for the Game of Fifteen. The challenge up next is to complete this game's implementation.

But first go ahead and compile the framework. (Can you figure out how?) And, even though it's not yet finished, go ahead and run the game. (Can you figure out how?) Odds are you'll want to run it in a larger terminal window than usual, which you can open clicking the green plus (+) next to one of your code tabs and clicking **New Terminal**. Alternatively, you can full-screen the terminal window toward the bottom of CS50 IDE's UI (within the UI's "console") by clicking the **Maximize** icon in the console's top-right corner.

Anyhow, it appears that the game is at least partly functional. Granted, it's not much of a game yet. But that's where you come in!

Commentary

You'll notice, if you have a look at the distro, that our `fifteen.c` file is woefully lacking in comments. As we said upfront, part of this assignment is beginning to read and understand code others have written for you, and so your first objective is to replace all of the `TODO`s you see scattered about `fifteen.c` with actual comments that explain what is happening in the game. Don't treat this portion of the problem set lightly, and hopefully the fact that we don't provide comments here on some stuff that may not be immediately apparent to you at first glance is a good reminder of just how important it is to provide high-quality comments in your own code!

questions

Read over the code and comments that you've just prepared in `fifteen.c` and then answer the questions below in `questions.txt`, which is a (nearly empty) text file that we included for you inside of the distro's `fifteen` directory. No worries if you're not quite sure how `fprintf` or `fflush` work; we'll simply be using those to automate some testing.

1. Besides 4 × 4 (which are Game of Fifteen's dimensions), what other dimensions does the framework allow?
2. With what sort of data structure is the game's board represented?
3. What function is called to greet the player at game's start?
4. What functions do you apparently need to implement?

fifteen

Alright, get to it, implement this game!

Err… maybe not at all of it. At least not yet. In this problem, you only need to implement two of the functions: `init` and `draw`. (We'll leave `move` and `won` alone for now.)

Any design decisions not explicitly prescribed herein (e.g., how much space you should leave between numbers when printing the board) are intentionally left to you. Presumably the board, when printed, should look something like the below, but we leave it to you to implement your own vision.

```
15 14 13 12

11 10  9  8

 7  6  5  4

 3  1  2  _
```

Incidentally, recall that the positions of tiles numbered 1 and 2 should only start off swapped (as they are in the 4 × 4 example above) if the board has an odd number of tiles (as does the 4 × 4 example above). If the board has an even number of tiles, those positions should not start off swapped. And so they do not in the 3 × 3 example below:

```
8  7  6

5  4  3

2  1  _
```

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.) But by the time you've completed this portion of the problem, there won't be terribly much you'll be able to actually play. After all, you're only initializing and drawing the board here.

You're welcome to write your own functions and even change the prototypes of functions we wrote. 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. If in doubt as to whether some design decision of yours might run counter to the staff's wishes, simply reach out to your teacher.

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, type `GOD` (for the so-called "God Mode" implemented in many games of this sort, where the computer plays the game itself) instead. Neat, eh?

And if you'd like to check the correctness of your program officially with `check50`, you may execute the below. **Note that `check50` assumes that your board's blank space is implemented in `board` as `0`; if you've chosen some other value, best to change to `0` for `check50`'s sake. Also note that `check50` assumes that you're indexing into `board` a la `board[row][column]`, not `board[column][row]`.**

```
check50 1516.unit3.fifteen1 fifteen.c
```

Notice that `1` in `fifteen1`… it's not a typo!

This was Problem 3-1.