Problem Set 2: Crypto

due by noon on Thu 9/27

Per the directions at this document's end, submitting this problem set involves submitting source code via `submit50` as well as filling out a Web-based form, which may take a few minutes, so best not to wait until the very last minute, lest you spend a late day unnecessarily.

Goals.

- Better acquaint you with functions and libraries.
- Allow you to dabble in cryptanalysis.

Recommended Reading.

- Sections 11 – 14 and 39 of `http://www.howstuffworks.com/c.htm`.
- Chapters 7, 8, and 10 of *Programming in C*.

diff pset2.pdf hacker2.pdf.

- Hacker Edition challenges you to handle punctuation in inputs.
- Hacker Edition challenges you to swap two variables without a temporary one.
- Hacker Edition challenges you to try out some file I/O.
- Hacker Edition challenges you to crack actual passwords.

Academic Honesty.

All work that you do toward fulfillment of this course's expectations must be your own unless collaboration is explicitly allowed in writing by the course's instructor. Collaboration in the completion of problem sets is not permitted unless otherwise stated by some problem set's specification.

Viewing or copying another individual's work (even if left by a printer, stored in an executable directory, or posted online) or lifting material from a book, website, or other source—even in part—and presenting it as your own constitutes academic dishonesty, as does showing or giving your work, even in part, to another student or soliciting the work of another individual. Similarly is dual submission academic dishonesty: you may not submit the same or similar work to this course that you have submitted or will submit to another. Nor may you provide or make available solutions to problem sets to individuals who take or may take this course in the future. Moreover, submission of any work that you intend to use outside of the course (*e.g.*, for a job) must be approved by the course's instructor or preceptor.

You are welcome to discuss the course's material with others in order to better understand it. You may even discuss problem sets with classmates, but you may not share code. In other words, you may communicate with classmates in English, but you may not communicate in, say, C. If in doubt as to the appropriateness of some discussion, contact the course's instructor or preceptor.

You may turn to the Web for instruction beyond the course's lectures and sections, for references, and for solutions to technical difficulties, but not for outright solutions to problems on problem sets or your own final project. However, failure to cite (as with comments) the origin of any code or technique that you do discover outside of the course's lectures and sections (even while respecting these constraints) and then integrate into your own work may be considered academic dishonesty.

All forms of academic dishonesty are dealt with harshly. If the course refers some matter to the Administrative Board and the outcome for some student is *Admonish*, *Probation*, *Requirement to Withdraw*, or *Recommendation to Dismiss*, the course reserves the right to impose local sanctions on top of that outcome for that student that may include, but not be limited to, a failing grade for work submitted or for the course itself.

Fine Print.

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)?

All students, whether taking the course Pass/Fail or for a letter grade, must ordinarily submit this and all other problem sets to be eligible for a passing grade (*i.e.*, Pass or A to D–) unless granted an exception in writing by the course's instructor or preceptor. No more than one late day may be spent on this, or any other, problem set.

Help!

☐ Surf on over to `cs50.net/discuss` and log in if prompted. Then take a look around!

Henceforth, consider CS50 Discuss *the* place to turn to anytime you have questions. Not only can you post questions of your own, you can also search for or browse answers to questions already asked by others. When posting, just be sure to provide as much detail as possible so that we can assist!

It is expected, of course, that you respect the course's policies on academic honesty. Posting snippets of code about which you have questions is generally fine. Posting entire programs, even if broken, is definitely not. If in doubt, simply flag your discussion as "private to staff," particularly if you need to show us most or all of your code. But the more questions you ask publicly, the more others will benefit as well!

Certainly don't hesitate to post a question because you think that it's "dumb." It is not!

Sanity Check.

☐ You should already have the CS50 Appliance installed, per Problem Set 1. But be sure that you have version 17 (and not 17a, 3-15, or even earlier). To check which version you have, look in the appliance's bottom-right corner, where you should see **17-#**, whereby **#** represents the "release" of version 17 that you have. If you see some number other than **17** (or no number at all), head to `cs50.net/appliance` for instructions on how to upgrade to version 17.

Once sure that you have version 17, update to the latest release by opening a Terminal window and executing the command below (which may take a minute or more).

```
sudo yum -y update
```

If the command appears to fail, restart the appliance, then try again. If it still appears to fail, restart your computer, then try again. If it *still* appears to fail, let us know at `cs50.net/discuss`, and we'll lend a hand!

Tips.

☐ Realize that the CS50 Appliance is a computer, albeit a virtual one. For better or for worse (mostly worse), computers don't like to be forcibly shut down or otherwise interrupted while in the middle of something. Do take care, then, not to quit VMware (or VirtualBox), shutdown your own computer, or even close your laptop's lid while the appliance is in the middle of something (*e.g.*, downloading or installing updates, submitting your work*, etc.*) Best to wait until the appliance isn't doing anything important, then shut it down (as via the green icon in the appliance's bottom-right corner). Bad things can happen, too, if your own computer runs out of disk space, so beware downloading big files on your own computer if you know you're low on disk space while the appliance is running.

☐ When running, the CS50 Appliance "borrows" some of your computer's own RAM and CPU cycles, which can slow down programs on your computer and vice versa. For maximum performance, try to launch VMware (or VirtualBox) and the appliance before launching other programs on your computer, and try to minimize the number of programs running on your computer while the appliance is running.

With that said, if you have lots of RAM (*e.g.*, 4GB) and lots of CPU cycles (*e.g.*, 2GHz), you might not need to give any of this a second thought!

A Section of Questions.

You're welcome to dive into these questions on your own, but know that they'll also be explored in section!

☐ Head to

`https://www.cs50.net/shorts/`

and watch the shorts on loops, scope, and the Caesar Cipher. Be sure you're reasonably comfortable answering the below when it comes time to submit this problem set's form!

- ☐ How does a while loop differ from a do-while loop? When is the latter particularly useful?
- ☐ What does `undeclared identifier` usually indicate if outputted by `clang`?
- ☐ Why is the Caesar Cipher not very secure?

And unrelated to those shorts!

- ☐ What's a function?
- ☐ Why bother writing functions when you can just copy and paste code as needed?

☐ Back when MySpace was cool, it was all the rage to TyPe LiKe ThIs. Maybe it still is? I'm not really sure. In any case, using the CS50 Appliance, CS50 Run (`cs50.net/run`), or CS50 Spaces (`cs50.net/spaces`), write a program that prompts the user for a message, and then outputs the message with its first letter capitalized, with all other letters in alternating case, irrespective of the message's original capitalization, as per the sample output below, wherein boldfaced text represents some user's input.[1][2] Take care to output non-alphabetical characters unchanged. Consider this problem an opportunity to practice; you won't be asked to submit this program.

```
jharvard@run.cs50.net (~): ./a.out
Thanks 4 the add!!!
ThAnKs 4 ThE aDd!!!
```

[1] CS50 Spaces is essentially a fancier version of CS50 Run, complete with code-sharing capabilities and a chat room.

[2] Note that CS50 Run and CS50 Spaces have what appears to be a Terminal window, but you can't actually type commands (*e.g.*, `clang`) in it. That black panel is instead there so that you can see what CS50 Run and CS50 Spaces are doing underneath the hood when you click the ▸ button at top-left. However, if your program prompts for user input (as with `GetString`), you can type user input in that black panel.

☐ Recall from Week 2's second lecture that swapping two variables' values by passing those two variables to a function (even if called `swap`) doesn't exactly work, at least not without "pointers," a topic we'll soon get to. For now, then, let's swap the value of two variables within `main` itself.

Using the CS50 Appliance, CS50 Run (`cs50.net/run`), or CS50 Spaces (`cs50.net/spaces`), write a program that prompts the user for two integers (storing them in variables called `x` and `y`), prints those numbers (one per line), swaps those variables' values without using any other variables, and then re-prints those numbers (one per lined, swapped), as per the sample output below, wherein boldfaced text represents some user's input.[3]

```
jharvard@run.cs50.net (~): ./a.out
x: 1
y: 2
x is 1
y is 2
x is 2
y is 1
```

Consider this problem an opportunity to practice; you won't be asked to submit this program.

☐ We'll visit file I/O (*i.e.*, input and output) later in the term, but have a sneak peek at C's file-handing functions at:

```
https://www.cs50.net/resources/cppreference.com/stdio/
```

Take a look too, as with gedit, at `/usr/share/dict/words` in the CS50 Appliance, a huge list of English words. Then, the CS50 Appliance, CS50 Run (`cs50.net/run`), or CS50 Spaces (`cs50.net/spaces`), write a program that accepts as input a command-line argument (*i.e.*, a `string` in `argv[1]`) and then informs the user whether that argument is in the dictionary by printing `YES` or `NO`, as per the sample outputs below, wherein boldfaced text represents some user's input.

```
jharvard@run.cs50.net (~): ./a.out appetizer
YES

jharvard@run.cs50.net (~): ./a.out appeteaser
NO
```

Consider this problem an opportunity to practice; you won't be asked to submit this program. And don't fret if this one's too challenging for now. We'll revisit the topic eventually!

[3] Cough cough, bitwise operations, cough cough.

Getting Started.

☐ Alright, here we go!

Open a terminal window if not open already (whether by opening gedit via **Menu > Programming > gedit** or by opening Terminal itself via **Menu > Programming > Terminal**). Then execute

```
mkdir ~/Dropbox/hacker2
```

at your prompt in order to make a directory called `hacker2` in your `Dropbox` directory. Take care not to overlook the space between `mkdir` and `~/Dropbox/hacker2` or any other character for that matter! Recall that `~` denotes your home directory, `~/Dropbox` denotes a directory called `Dropbox` therein, and `~/Dropbox/hacker2` denotes a directory called `hacker2` within `~/Dropbox`.

Now execute

```
cd ~/Dropbox/hacker2
```

to move yourself into (*i.e.*, open) that directory. Your prompt should now resemble the below.

```
jharvard@appliance (~/Dropbox/hacker2):
```

If not, retrace your steps and see if you can determine where you went wrong. You can actually execute

```
history
```

at the prompt to see your last several commands in chronological order if you'd like to do some sleuthing. You can also scroll through the same one line at a time by hitting your keyboard's up and down arrows; hit Enter to re-execute any command that you'd like. If still unsure how to fix, remember that `cs50.net/discuss` is your friend!

All of the work that you do for this problem set must ultimately reside in your `hacker2` directory for submission.

Passwords *et cetera*.

☐ On most, if not all, systems running Linux or UNIX is a file called `/etc/passwd`. By design, this file is meant to contain usernames and passwords, along with other account-related details (*e.g.*, paths to users' home directories and shells). Also by (poor) design, this file is typically world-readable. Thankfully, the passwords therein aren't stored "in the clear" but are instead encrypted using a "one-way hash function." When a user logs into these systems by typing a username and password, the latter is encrypted with the very same hash function, and the result is compared

against the username's entry in `/etc/passwd`. If the two ciphertexts match, the user is allowed in. If you've ever forgotten some password, you may have been told that "I can't look up your password, but I can change it for you." It could be that person doesn't know how. But, odds are they just can't if a one-way hash function's involved.[4]

Even though passwords in `/etc/passwd` are encrypted, the crypto involved is not terribly strong. Quite often are adversaries, upon obtaining files like this one, able to guess (and check) users' passwords or crack them using brute force (*i.e.*, trying all possible passwords). Only in recent years have (most) system administrators stopped storing passwords in `/etc/passwd`, instead using `/etc/shadow`, which is (supposed to be) readable only by root.[5] Below, though, are some `username:hash` pairs from some outdated (fake) systems.[6]

```
caesar:50zPJlUFIYY0o
cs50:50gyRGMzn6mi6
jharvard:50yoN9fp966dU
malan:HA610l/.LeOak
nate:50AcIG/VnV3D2
rbowden:50q.zrL5e0Sak
skroob:50Bpa7n/23iug
tmacwilliam:50WZ/Wy2GdA1Y
zamyla:50lMLvy/mlPIE
```

Crack these passwords, each of which has been encrypted with C's DES-based (not MD5-based) `crypt` function. Specifically, write, in `crack.c`, a program that accepts a single command-line argument: an encrypted password.[7] If your program is executed without any command-line arguments or with more than one command-line argument, your program should complain and exit immediately, with `main` returning any non-zero `int` (thereby signifying an error that our own tests can detect). Otherwise, your program must proceed to crack the given password, ideally as quickly as possible, ultimately printing to standard output the password in the clear followed by `\n`, nothing more, nothing less, with `main` returning `0`. The underlying design of this program is entirely up to you, but you must explain each and every one of your design decisions, including any implications for performance and accuracy, with profuse comments throughout your source code. Your program must be designed in such a way that it could crack all of the passwords above, even if said cracking might take quite a while. That is to say, it's okay if your code might take several minutes or days or longer to run. What we demand of you is correctness, not necessarily optimal performance. Your program should certainly work on inputs other than these as well; hard-coding into your program the solutions to the above is not acceptable.

So that we can automate some tests of your code, your program must behave per the below; highlighted in bold is some sample input.

```
jharvard@appliance (~/Dropbox/hacker2): ./crack 50yoN9fp966dU
crimson
```

[4] If you like this stuff, consider taking Computer Science 220r.

[5] Take a look at `/etc/passwd` in the appliance, for instance; wherever you see `x` a password once was.

[6] `http://cdn.cs50.net/2012/fall/psets/2/hacker2/passwd`

[7] In case you test your code with other ciphertexts, know that command-line arguments with certain characters (*e.g.*, `?`) must be enclosed in single or double quotes; those quotation marks will not end up in `argv` itself.

Assume that users' passwords, as plaintext, are no longer than eight characters long. As for their ciphertexts, you'd best pull up the "man page" (*i.e.*, manual) for `crypt` by executing

```
man crypt
```

in a terminal window so that you know how the function works. In particular, make sure you understand its use of a "salt." (According to the man page, a salt "is used to perturb the algorithm in one of 4096 different ways," but why might that be useful?) As implied by that man page, you'll likely want to put

```
#define _XOPEN_SOURCE
#include <unistd.h>
```

at the top of your file and link your program with `-lcrypt`. (If you use `make` to compile your code, that switch will be included automatically.)

You might also want to read up on C's support for file I/O, as there's quite a number of English words in `/usr/share/dict/words` in the appliance that might (or might not) save your program some time.

By design, `/etc/passwd` entrusts the security of passwords to an assumption: that adversaries lack the computational resources with which to crack those passwords. Once upon a time, that may have been true. Perhaps some still do. But when it comes to security, assumptions are dangerous. May that this problem set make that claim all the more real.

We should note that this problem set is no invitation to seek out other passwords to crack.[8] Do not conflate these Hacker Editions with "black hat" editions. We hope, though, that by understanding better the design of today's systems, you might one day build better systems yourself. Besides acquainting you further with C, this problem set urges you to start questioning designs, as vulnerabilities (if not regrets) often result from poor ones.

If you'd like to play with the staff's own implementation of `crack`, well, sorry! :-) Where'd be the fun in that?

[8] In fact, do bear in mind the policies at `http://www.fas-it.fas.harvard.edu/services/student/policies/rules_and_responsibilities`.

How to Submit.

In order to submit this problem set, you must first execute a command in the appliance and then submit a (brief) form online.

☐ Open a terminal window (as via **Menu > Programming > Terminal** or within gedit) then execute

```
sudo yum -y update
```

to ensure you have the latest release of the appliance. Then execute:

```
cd ~/Dropbox/hacker2
```

And then execute:

```
ls
```

At a minimum, you should see `crack.c`. If not, odds are you skipped some step(s) earlier! If you do see those files, you are ready to submit your source code to us. Execute:

```
submit50 ~/Dropbox/hacker2
```

and follow the on-screen instructions. That command will essentially upload your entire `~/Dropbox/hacker2` directory to CS50's servers, where your TF will be able to access it. The command will inform you whether your submission was successful or not. And you may inspect your submission at `cs50.net/submit`.

You may re-submit as many times as you'd like; we'll grade your most recent submission. But take care not to submit after the problem set's deadline, lest you spend a late day unnecessarily or risk rejection entirely.

If you run into any trouble at all, let us know via `cs50.net/discuss` and we'll try to assist! Just take care to seek help well before the problem set's deadline, as we can't always reply right away!

☐ Head to the URL below where a short form awaits:

```
https://www.cs50.net/psets/2/
```

Once you have submitted that form (as well as your source code), you are done!

This was Problem Set 2.