Table of Contents

Crack

tl;dr

Implement a program that cracks passwords, per the below.

```
$ ./crack 50fkUxYHbnXGw
rofl
```

Background

On most systems running Linux these days is a file called `/etc/shadow`, which contains usernames and passwords. Fortunately, 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/shadow`. If the two hashes match, the user is allowed in. If you've ever forgotten some password, you might have been told that tech support can't look up your password but can change it for you. Odds are that's because tech support can only see, if anything at all, a hash of your password, not your password itself. But they can create a new hash for you.

Even though passwords in `/etc/shadow` are hashed, the hash function is not always that strong. Quite often are adversaries, upon obtaining that file somehow, able to guess (and check) users' passwords or crack them using brute force (i.e., trying all possible passwords). Below is what `/etc/shadow` might look like, albeit simplified, wherein each line is formatted as `username:hash`.

```
anushree:50xcIMJ0y.RXo
brian:50mjprEcqC/ts
bjbrown:50GApilQSG3E2
lloyd:50n0AAUD.pL8g
malan:50CcfIk1QrPr6
maria:509nVI8B9VfuA
natmelo:50JIIyhDORqMU
rob:50JGnXUgaafgc
stelios:51u8F0dkeDSbY
zamyla:50cI2vYkF0YU2
```

Specification

Design and implement a program, `crack`, that cracks passwords.

- Implement your program in a file called `crack.c` in a directory called `crack`.
- Your program should accept a single command-line argument: a hashed password.
- If your program is executed without any command-line arguments or with more than one command-line argument, your program should print an error (of your choice) and exit immediately, with `main` returning `1` (thereby signifying an error).
- Otherwise, your program must proceed to crack the given password, ideally as quickly as possible, ultimately printing the password in the clear followed by `\n`, nothing more, nothing less, with `main` returning `0`.
- Assume that each password has been hashed with C's DES-based (not MD5-based) `crypt` function.
- Assume that each password is no longer than five (5) characters. Gasp!
- Assume that each password is composed entirely of alphabetical characters (uppercase and/or lowercase).

Walkthrough

Note: The walkthrough video incorrectly states that passwords are no longer than four (4) characters. Instead, per the specification, make sure to handle passwords that are up to five (5) characters.

Usage

Your program should behave per the examples below. Assumed that the underlined text is what some user has typed.

```
$ ./crack
Usage: ./crack hash
```

```
$ ./crack 50JGnXUgaafgc
ROFL
```

Testing

Correctness

No `check50` for this one! But odds are, if you can crack all ten passwords above, you're in good shape!

Style

```
style50 crack.c
```

Hints

Be sure to read up on `crypt`, taking particular note of its mention of "salt":

```
man crypt
```

Per that man page, you'll likely want to put

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

at the top of your file in order to use `crypt`.