Assignment 01

If any of this is unclear, please get in touch with me ASAP. There are definitely things in the course I want you to struggle with. I absolutely do NOT want you struggling with understanding my expectations.

The goals of this assignment are for you to refresh your skills working with the shell, C, git, and makefiles. Additionally, you will produce some functions that will come in handy throughout the semester.

Due 12pm, Friday, 03 March.

Note that the instructions below assume you will be running a text editor within your virtual machine. This is not a universal solution. Please refer to the "Editing files (and copying files)" section of this course's FreeBSD instructions for more details.

Note also that the default version of `make` available on FreeBSD uses a different syntax to the version we used in Systems Programming, the latter of which is officially "GNU `make`". Rathert than learn a new `Makefile` syntax, the easiest solution is to just install GNU `make` in your virtual machine:

```
$ sudo pkg install gmake
```

Then use `gmake` instead of `make`.

Part I: git repo (core)

First, create a git repository *on weathertop* which will serve as the official version of your code that I will check:

```
laptop$ ssh username@weathertop.cs.middlebury.edu
weathertop$ cd ~
weathertop$ git init --bare cs431
weathertop$ chmod 711 .
```

Second, on the virtual machine on which you will do your development, clone the repo from weathertop:

```
my-vm$ cd /where/I/put/my/schoolwork
my-vm$ git clone ssh://my-username@weathertop.cs.middlebury.edu/home/my-username/cs431
```

You may then add, edit, and commit files on your machine, and push the commits to weathertop, where I will be able to see them.

(Unlike Systems Programming, and because assignments in this course build directly on each other, we will use a single repository for the entire semester's work.)

NOTE On Tuesday, 28 February at 3pm, I will test that I can successfully clone and pull from your git repo, so be sure to set it up before then. I will email you if I have problems; if you don't hear from me by the end of that day, you're good to go.

Part II: utility functions (core)

Inside your new git repo, create a file `util.h` that defines the following function prototypes:

```
char *binary_to_hex(void *data, ssize_t n);
void *hex_to_binary(char *hex);
```

In a file `util.c`, implement those functions according to these specifications:

- `binary_to_hex` should return a pointer to a `malloc(3)`d string that contains the hex representation of the binary data pointed to by `s`, which has length `n`. Each pair of hex digits should be separated by a space; a newline should be inserted after every 16th pair of hex digits and after the final pair of hex digits.

  Examples:

  1. input "AAAA" -> output "41 41 41 41\n"
  2. input "hello" -> output "68 65 6c 6c 6f\n"

- `hex_to_binary` should return a pointer to a `malloc(3)`d buffer that contains the binary representation of the bits hex data pointed to by `hex`, which itself will be NULL-terminated. This function should ignore whitespace. It should support both uppercase and lowercase hex digits, but if any non-hex characters are encountered, it should immediately `free(3)` the buffer and return NULL. If the buffer contains an odd number of hex digits, ignore the final one.

  Examples:

  1. input "41 41 41 41" -> output "AAAA"
  2. input "68 65 6c 6c 6f" -> output "hello"

For both functions, the caller will be responsible for calling `free(3)`.

(If the behavior of the examples is unclear, look up the ASCII table.)

Part III: driver programs (advanced)

You will write two programs `hexdump` and `hexread`. `hexdump` will convert its input from binary to hex; `hexread` will perform the reverse operation. Both programs should operate on files whose names are provided as command-line arguments; if no arguments are provided, both should read from `stdin`. You must include a `Makefile` that builds these programs; its default target should build both.

Style requirements

Beyond the function prototypes as required, none. You are, however, *strongly* encouraged to use good style (comments, appropriate names, etc), as it will benefit you greatly when you need to revisit your code throughout the semester. The idea here is to demonstrate why good style is a good thing, not by forcing you to use it, but by forcing you to deal with the consequences if you don't.

Submission Process

None. I will pull from your git repo at 12pm on Friday, 03 March and consider that your submission.

Assessment

Parts I and II are the core requirements, which you must complete to earn a C.

Part III is the advanced requirement, which you must complete to earn an A. There is no B level for this (very modest) assignment.

(Commentary: in the real world, you'd be kind of crazy to just create functions without also creating the ability to test them.)

Last modified: 02/24/2023 15:37:26