Fetching URLs

You will write a simple HTTP client program, `get`, which fetches a list of URLs over the web. This will exercise reading from and writing to files, simple string parsing and manipulation, and networking.

Expect one command-line argument, the name of a file. Open that file and read from it space-separated URLs; valid URLs can not contain whitespace, and you may assume that no URL in your input is longer than 255 characters, so a good way to read them is with `scanf`, a 256-character buffer, and the format `"%255s"`. For each URL, parse it into the following pieces:

hostname From after the `http://` to before the next slash; in the URL `http://www.cs.cornell.edu/courses/cs2022/2010sp/index.html`, the hostname is `www.cs.cornell.edu`.

path Starting from the slash after the hostname, to the end of the URL; in the example, the path is `/courses/cs2022/2010sp/index.html`.

filename From after the last slash in the path to the end of the URL; in the example, the filename is `index.html`. If the URL ends in a slash, and thus the filename is the empty string (starts with the null character), it defaults to `index.html`.

The **hostname** will always start at index 7 in the URL, and extend to the next slash (you can loop manually to find it or use `strchr`). You will need it to be null-terminated, so you must `malloc` enough space for the entire **hostname** plus one character for the null, copy it in, and stick a null (`'\0'`) at the end. (Remember to `free` it when you don't need it anymore!) The same slash that comes after the hostname is also the beginning of the **path**, and the **filename** begins just after the last slash in the URL (think `strrchr`); neither of these need the special treatment to be null-terminated, since they already run to the end of the original URL.

Connect (using `getaddrinfo`, `socket`, `connect`, etc.) as a client to **hostname**, on port 80, and issue an HTTP request for the path. The best way to prepare such a request is to use `sprintf` with the format `"GET %s HTTP/1.0\r\n\r\n"` and **path** as argument. After writing the request to the server with `send`, read the entire response one buffer at a time with `recv` and write what you read into a file named **filename** (using `fopen`, `fwrite`, and `fclose`).

Thus, for example, if there is a file named `urls`, with the following contents:

urls
```
http://www.cs.cornell.edu/courses/cs2022/2010sp/01-quick.pdf
http://www.cs.cornell.edu/courses/cs2022/2010sp/
http://www.cs.cornell.edu/courses/cs2022/2010sp/foo/nosuchfile.html
```

Then after running:

```
$ ./get urls
```

There will be three new files in the working directory, named `index.html`, `01-quick.pdf`, and `nosuchfile.html`. Each one will begin with an HTTP status response and some headers, followed by a blank line, followed by the contents of the file requested. Verify it by deleting the headers from `01-quick.pdf` and opening it up to read the first lecture notes! Because there is no file named `/courses/cs2022/2010sp/foo/nosuchfile.html` at the web server `www.cs.cornell.edu`, the output from that request will be a standard Error 404 Page-Not-Found message from the server.

Because this program will just consist of `main` and its helper functions, simply put all of your code in `get.c`, and submit it on CMS. You may decide for yourself how to structure your code, but, as a hint, here are the functions found in the solution:

`int opensock(char *host)` Given a null-terminated hostname, go through all of the `getaddrinfo`, `socket`, and `connect` dance to connect to the given host on port 80, and return the new socket.

`void get(char *url)` Given a null-terminated URL, parse it as described above, call `opensock` on the hostname part to get a connected socket, `fopen` the filename with mode `"w"`, set up the request with `sprintf`, write it with `send` (in a loop to ensure that every byte is sent even if there is a short return value), and then loop on `recv`, writing into the output file with `fwrite`. When the connection closes (`recv` returns 0), close the output file with `fclose` and return.

`int main(int argc, char **argv)` Open the file specified by `argv[1]`, with `fopen` with mode `"r"`, read URLs using `fscanf` and call `get` on each. When there are no more URLs to read (`fscanf` returns 0), return success.