CSSE 132 – Introduction to Systems Programming
Rose-Hulman Institute of Technology

Final Exam Coding Practice

On your Linux, pull your Git repository, and look for the `finalexampractice` directory.

Part 1

1. Complete the ARM Assembly function in `part1.s`. Compile with `make test_asm`, run tests with `make run`, and debug with `make run-gdb`.

2. Complete the C functions in `part1.c`. Compile with `make test`, run tests with `./test`, and debug with `gdb ./test`.

Part 2

In this part, you'll write a program that does File I/O.

1. In your `practice` directory, create a new C file called `part2.c`.

2. Do not include any of your code from `part1.c`. We provide a correct implementation of `part1`. To use the functions in `part1`, add this exact line to the top of your `part2.c` file:
   `#include "support/part1.h"`

3. Add a main function that takes arguments (`int main(int argc, char** argv)`).

4. The code should take a single command-line argument, which is a filename such as `sample.txt`. It should first open this file.

   (continued on back)

5. Once the file is open, the code should loop, presenting the user with a command-line such as the following.

```
Devowel which line of sample.txt?
```

6. When the user inputs a number at stdin, the code should "devowel" (i.e., run `removeVowels` from part 1) on that line of the file, where 0 is the index of the initial line of the file, 1 is the next, etc; and print the result to standard out. It should then return to the command line. Remember to avoid memory leaks!

7. If a blank (just `\n`) is written at the prompt, the code should end the prompt, close the file, and return.

8. Example behavior:

```
$ ./part2 sample.txt
Devowel which line of sample.txt? 0
Fr scr nd svn yrs g r fthrs brght frth n ths
Devowel which line of sample.txt? 7
fld, s  fnl rstng plc fr ths wh hr gv thr lvs tht
Devowel which line of sample.txt? 1
cntnnt,  nw ntn, cncvd n Lbrty, nd ddctd t th
Devowel which line of sample.txt?
$
```

9. Assume that the user will only type a valid line number at the prompt.

10. Hint: the `fgets()` function is helpful for reading the file as well as stdin, as it only reads up to a `\n`.

11. Hint: the `atoi()` function is helpful, but it will only work if you give it a string with numerical characters only.

12. To compile your code, use `make part2`.

13. Use `./test_part2.sh` to test your code for the expected behavior. In addition, check:

    - Does it avoid memory leaks? (Especially consider use of `removeVowels`, which malloced some space.)
    - Does it properly null-terminate strings?
    - Does it close the file?

Part 3

In this part, you need to complete the program `./part3.c` to implement a **client** to communicate with an existing server (`./example_server` is provided in the repo).

The server tells knock-knock jokes, if the client gives the correct responses. Here are some examples. Note: The brackets [ ] are not part of the messages.

```
Server -----[Knock, knock.]----------------------> Client
Server <-----------------------[Who's there?]----- Client
Server -----[Hawaii.]---------------------------> Client
Server <------------------------[Hawaii who?]----- Client
Server -----[I'm fine. Hawaii you?]--------------> Client
```

```
Server -----[Knock, knock.]----------------------> Client
Server <-----------------------[Who's there?]----- Client
Server -----[Little old lady.]-------------------> Client
Server <---------------[Little old lady who?]----- Client
Server -----[I didn't know you could yodel!]-----> Client
```

The specific communication protocol is as follows:

1. The client will first connect to the server with IP address `127.0.0.1` and port number `20000`.
2. Once connected, the server will send the message `Knock, knock.` to the client. The client replies with the message `Who's there?` and the server replies with a message `X.` where `X` is some name, word, or phrase. The client should respond with a message of the form `X who?`. Finally, the server sends the punchline, and the connection is closed.
3. The client should print every message of the exchange to `stdout`, with a newline after each message. Note: the server will not null-terminate its messages. (The client can optionally null-terminate its messages to the server.)

   For example (`$` is the Linux command prompt):

   ```
   $ ./part3
   Knock, knock.
   Who's there?
   Little old lady.
   Little old lady who?
   I didn't know you could yodel!
   $
   ```

4. Note that no keyboard input (e.g., reading from `stdin`) should be involved in this program. The protocol should be automated by the code.

Your program must satisfy the following specifications:

- You must implement this in a new file called `part3.c`.
- To run your code, first compile the code by running `make part3`. Then open an additional SSH terminal to run `./example_server`. Once the server is running, you can run your client. When testing your client, check what is printed to stdout on the server side—it can help debug your client.
- To test the code, please run `./test_part3.sh` to validate your implementation.