

Lab Assignment #2

Description

This lab assignment will help you learn some important techniques for completing your homeworks. It is graded on the scale of **credit** or **no-credit**, and you may work in groups of 2. Please submit Lab #2 online. Read the README at `/usr/class/cs193i/bin/README-submit`, and follow those directions. Remember to include the user name of each teammate in the correct format (on separate lines) as specified by the README-submit.

This lab assignment is due **Wednesday, July 14, 2004 at 11:59pm**.

Learning Perl

If you are not comfortable with Perl, then read O'Reilly's [Perl in a Nutshell](#), Second Edition, on the Safari techbooks site. You might also want to browse through [Learning Perl](#), also from O'Reilly.

Perl In a Nutshell

<http://proquest.safaribooksonline.com/?XmlId=0-596-00241-6>

Learning Perl

<http://proquest.safaribooksonline.com/?XmlId=0-596-00132-0>

The best chapter to start from is [Perl in a Nutshell](#), Chapter 4. In particular, pay attention to the section on Regular Expressions.

Telnet

Telnet is a protocol (and program) based on TCP that allows you to connect to a host on the Internet to send/receive bytes. It was used in the past to allow users to connect to remote systems and interact on the command line as if they were sitting at the console. Recently, this use has been made obsolete by SSH, which provides encryption of data.

However, it is still useful for several reasons. Try this: log into two cluster machines (elaine, saga, etc...). Go to `/usr/class/cs193i/hw/examples/` and run `servexample.pl` on one of them. It should say "Server ready." Then, go to your second shell, and type

```
telnet <the first machine's host name> 3456
```

This should connect you to the server's port 3456. Try typing something, and switch back and forth between the shell windows to see what both the server and client say to each other.

So, you can see that we can use telnet to debug our server programs. You can run your server, then use telnet to act as a client—feeding bytes through to see if your server is working fine and dandy.

One fun thing to do is to surf the web, um... manually. Try typing this:

```
telnet www.yahoo.com 80
```

This means that telnet will connect to www.yahoo.com's port 80, the web server. The web server accepts Hypertext Transfer Protocol (HTTP) commands (see <http://en.wikipedia.org/wiki/http>). At the prompt, type:

```
GET /index.html HTTP/1.0
```

And then press enter twice. The web server should send you back your file, the Yahoo.com home page. If you read that file carefully, you will see HTML (hypertext markup language), the way to transmit web pages over the web. Occasionally you will see some text of the form ``. These are web links. If you then telnet to yahoo again, and issue the GET on the text between the quotes, you will have "clicked" on a link.

Try manually surfing Google.com. You will see that Google's home page is a lot shorter (fewer bytes) than Yahoo's.

HTTP is defined in RFC 2616. Go here <http://www.rfc-editor.org/rfc/rfc2616.txt> to read it. Actually, don't. It's a really, really long RFC. Luckily, the RFC you will read for HW#1 is about 10 times shorter. But in the mean time, go into RFC2616 and look for information about the ubiquitous 404 error. It's cool to see the official definition of what these server errors mean. You can also skim quickly the section about caching web pages.

Perl Regular Expressions

Now, we will write a few Perl programs to learn more about regular expressions. First get a section of Shakespeare's *Midsummer Night's Dream* by doing the following:

```
telnet the-tech.mit.edu 80 > output.txt
GET /Shakespeare/midsummer/midsummer.2.1.html HTTP/1.0
<PRESS ENTER ONCE MORE>
```

This should output the server's response to output.txt. Open that file in your favorite editor... and examine the top. The first 13 lines or so are not part of the HTML file. The 10 or so lines starting with HTTP/1.1 200 OK are the HTTP Response Headers, which tell your browser something about the file or the server. Delete these 13 or so lines. The first line of the file should now read `<!DOCTYPE HTML` and so on.

Now, if you rename the `output.txt` to `midsummer.html`, you will have an html file you can open! But, instead of doing that, we will do something a lot more geeky.

Question 1: How many times does the string “night” appear in your `midsummer.html`?

The string search should be case insensitive, and allow “night” to be part of other words, such as fortnight. A solution to this can be written in under 15 lines of code. To answer Question 1, we’d like you to write a Perl program named **`nightcount.pl`**. Place the numerical answer in a file called `Lab2.txt`, and submit it with your other files.

This Perl program might follow the following design (in pseudocode).

```
# keep a counter
# build a regular expression pattern that will match the word in question
# while there are still lines in the file
    # read in a line... and if it matches... add one to the count!

# print the count at the end
```

Include your Perl program in your directory for the Lab #2 submit.

But of course, you can do many other things with Regular Expressions. For example, you could replace every “p” in the document with an “f” – actually, nevermind. Or you could replace every “her” with “him” and every “him” with “her” ...

Why don’t we try that?

Question 2: Swap every instance of “her” with “him,” and vice-versa.

It is okay if “her” is part of a word like “there”... it’ll just turn into “thime”... Note that there is something tricky about this question. It is ok to write your result to a new file, `midsummer.out.html`. Name your file **`her2him.pl`**, and submit it along with the other answers. A naïve solution to this can take about 20 lines of Perl code.