CSE 303, Spring 2009
Homework 3: Web Sites / Digits (regexes, C), 50 points
Due Friday, April 24, 2009, 11:30 PM

This two-part assignment focuses on using regular expressions (and related commands such as `sed` and `egrep`) in `bash` shell scripts, and on writing basic C programs. Part 1 is inspired by previous UW 303 instructors such as Dan Grossman, Hal Perkins, and Magdalena Balazinska. Part 2 is inspired by a similar assignment from Prof. Steve Wolfman of UBC.

Turn in two files, one named `web.sh` and one named `digits.c`, from the Homework section of the course web site. You will also want the various input support files from the Homework section of the course web site.

Part 1 of 2: Top Web Sites (`web.sh`):

In this part of the assignment, you will write a shell script to look at the relationship between the popularity of a web site and the size of its main page. There is a list of 100 popular web sites available at http://100bestwebsites.org/ .

You will write a script `web.sh` that reads the text from that URL, and examines it looking for URLs of top web sites. You will look through the HTML text to discover the links to the top 100 sites, then you will examine each of those web pages to see how long they are in bytes. Your script's output will be the following: (abbreviated; see web site)

```
$ ./web.sh
1 yahoo.com 9490
2 google.com 4462
3 amazon.com 96979
4 about.com 88354
...
100 gutenberg.net 15590
```

The above 100bestwebsites.org page lists its top 100 sites as a series of links in its HTML code. You do not need to know much about HTML for this assignment, but you should know that a link has the following form:

```
<a href="http://www.google.com/">Click here to search Google</a>
```

HTML is loose about whitespace and allows other attributes to be specified, so the following is also a legal link:

```
< a id="link4" href = "http://www.bigempire.com/filthy/watchmen.html" > Click! </a >
```

You may assume that every relevant link URL on the page begins with `http://` .

Some links on the page refer to the 100bestwebsites.org site itself. You should filter out any such links from your results. You may assume that the 100 links of interest are the first 100 on the page that do not mention 100bestwebsites.org.

Note that the actual link URLs in the 100bestwebsites HTML code contain full addresses such as http://www.google.com/ or http://www.usatoday.com/news/states/ns1.htm . Your eventual output should trim off the http:// and optional www. from these URLs, as well as any trailing / character at the end of the URL. You may not want to do this until last, because you'll still need to fetch these URLs using their full paths to compute the page sizes.

Your script should accept an optional **command-line argument** specifying how many URLs to display. If no argument is passed, display all 100. You may assume the value is a positive integer and that the page contains at least that many links. Your script should be efficient and should not download the contents of all 100 web sites if it is not going to display them all. The following is an example log of execution with an argument passed:

```
$ ./web.sh 3
1 yahoo.com 9490
2 google.com 4462
3 amazon.com 96979
```

Fetch the contents of a URL using the `curl` command. Some URLs in the top 100 list (notably Craigslist) will return 0-byte results to `curl` to indicate that they have moved to a different location (HTTP error code 302). You can fix this by passing an appropriate argument to `curl` to tell it reattempt the get the URL from the new place. See the `curl man` or `info` page. You may assume that every web page exists and will be able to be fetched successfully by `curl`. (If any page does not exist or is not reachable, `curl` will return an empty page and your script will likely output 0. This is fine.)

Development Strategy:

We suggest that you solve smaller parts of this assignment one by one and then combine them into an overall script:

1. Type commands into a `bash` terminal to figure out a one-line command that will grab the top 100 web sites from the 100bestwebsites URL and output those full URLs one per line, such as:
   ```
   http://www.yahoo.com/
   http://www.google.com/
   http://www.amazon.com/
   ```
2. Type commands into a `bash` terminal to figure out a one-line command that will grab a single web site and report its length in bytes, such as 9490 for http://www.yahoo.com/ .
3. Create a shell script that glues together the previous commands in appropriate ways to produce output such as:
   ```
   http://www.yahoo.com/ 9490
   http://www.google.com/ 4462
   ```
4. Add features such as numbering links, cleaning up URLs to shorter forms, and the command-line argument.

Your script should not leave behind any temporary files after it is done executing.

You may want to test your script by temporarily changing your URL from 100bestwebsites.org to the following URL, which is tougher to examine because it has more spaces and extraneous text inserted into it. Your program should produce the same output when used with this URL as with the previous 100bestwebsites.org URL.

- http://www.cs.washington.edu/education/courses/cse303/09sp/homework/3/top100test.html

Grading:

In terms of grading, most of the points will come from the correctness of your scripts. Some points will also come from the style, design, and "internal correctness" of your script code. You should not use a more complex command or control structure when a more simple one would achieve the same result. You should avoid redundancy as much as possible in the limited context of the shell programming language. If a particular value is important to your program as a whole, you should store that value in a well-named variable near the top of the code and use that variable throughout your code, rather than referring directly to the value throughout your code. (This is analogous to the idea of declaring class constants for important values in Java.)

Each Linux/Unix box can be slightly different; for full credit, your commands must be able to work properly either on `attu`, or on the basement lab computers, or on a fresh Ubuntu installation (with Java installed).

The answers to all questions in this assignment can be found entirely using commands shown in the lecture notes from the first three weeks. You may use other commands if you like, but you should constrain yourself to those from lecture or from the *Linux Pocket Guide* textbook. Ask the instructor if you are unsure whether a particular command is allowed.

You should format your code nicely with proper indentation, spacing, and clear variable names. You should place a descriptive comment heading at the top of your program as well as brief comments throughout your script explaining what each part of the script is doing. For reference, our solution to Part 1 is around 25 lines long (18 lines long if you exclude blank lines and comments), though you do not need to match this exactly to get full credit; it is just a rough guideline.

(continued on next page)

Part 2 of 2, Digit Counter (`digits.c`):

For the second part of this assignment, you will write a C program `digits.c` to explore a phenomenon called Benford's Law, which describes a surprising pattern in the frequency of occurrence of the digits 1-9 as the first digits of natural data. (For example, in the number 328905, the first digit is 3.) You might expect each digit to occur with equal frequency in arbitrary data. Indeed, in truly random data over appropriate ranges, each digit does appear with equal frequency. However, a substantial amount of data from diverse sources does not exhibit a uniform distribution. If you are curious about the distribution, Benford's Law, and the process by which it was discovered, check out the Wikipedia page:

- http://en.wikipedia.org/wiki/Benford's_law

You will write a C program `digits.c` that examines integer data and counts the first digits of those integers, then outputs statistics. Your program will read input from the console (standard input) using the `scanf` function. You will continually read integers, one per line, until you see the value `-1` at which point your program will output its statistics and exit. For full credit, you should appropriately use an array to help you perform the counting of first digits.

Though your program reads from standard input, for testing we will redirect in the input so that it actually reads from files. The following is an example file `enrollment.txt` that shows number of students enrolled at various major universities. (UW is the first data point with 28,570.) Notice that the file ends with `-1` to signify the end of the input.

```
28570
12176
5476
543
3490
24892
28619
2595
603
2527
1465
1858
-1
```

If your program were compiled into an executable program named `digits` and run with this file, the output would be:

```
$ ./digits < enrollment.txt
Digit   Count   Percent   Histogram
    1       3    25.00%   ************
    2       5    41.67%   ********************
    3       1     8.33%   ****
    4       0     0.00%
    5       2    16.67%   ********
    6       1     8.33%   ****
    7       0     0.00%
    8       0     0.00%
    9       0     0.00%
TOTAL      12
```

You must match the above output format exactly. The columns above are separated by exactly 3 spaces. The 'Digit' column is exactly 5 spaces wide. The 'Count' column is exactly 5 spaces wide, with each count value right-aligned within the column. The 'Percent' column is exactly 7 spaces wide, with each percentage value right-aligned within the column and displaying exactly 2 digits after the decimal point. The 'Histogram' column should show exactly one `*` star character for every full 2% for that row's integer. For example, since the digit 2 above has 41.67%, there are 20 stars.

You may assume valid input, that the input to your program will consist entirely of positive integers until -1. You may not make any assumption about the number of integers to read. It could be very few, none at all, or a very large number. Several other input and expected output files are posted to the course web site to help you verify your program.

Grading:

Your program should produce no errors or warnings from `gcc -Wall` . You should not use pointers on this assignment. Follow similar guidelines as in Part 1 about clear code, naming, spacing, and redundancy. Put descriptive comments at the top of your program and on major sections of code. Our solution is 40 lines long (25 w/o blanks/comments).