Computer Science 381
Programming Unix in C
The College of Saint Rose
Winter Immersion 2014

Lab 10: String Processing

Recommended Due Date: Wednesday, January 15, 2014

In this lab, you will focus on string processing – first with a few useful Unix commands, then using C.

Unix Text Processing Utilities

As mentioned in the previous lab, much of the real power of the Unix command line comes from the ability to construct a pipeline of relatively simple commands to perform complex tasks. This lab, you will look at two powerful building blocks: `awk` and `sed`.

We will only touch on the basics – these commands can do far more. There are too many tutorials, examples, and other online resources to mention. Many of those found on the first couple pages of appropriate web search results are quite good.

`awk`

`awk` is actually a programming language in itself, but is frequently used to extract or transform lines of a text file, often as part of a pipeline of commands. Its name comes from the names of its original authors: Aho, Weinberger, and Kernighan. It is also comes in closely-related variants `nawk` and `gawk`.

As a simple example, if you have a file of names called `namelist` in the format:

```
John Smith
Mary Jones
Alan Williams
Josephine Adams
```

you could print only the last names (*i.e.*, the second column of words) with the command:

```
awk '{ print $2 }' namelist
```

or in "Last, First" format with the command:

```
awk '{ printf $2 ", " $1 "\n"}' namelist
```

If such a list of names were to be generated by a program, we could transform "First Last" output to the "Last, First" format, then sort alphabetically by last name in a pipeline:

```
prog | awk '{ printf $2 ", " $1 "\n"}' | sort
```

Lab Question 1:
Write a Unix pipeline that uses an `awk` command to transform the output of "`ls -l`" as described below. (5 points)

The output on mogul in `/home/cs381/labs/ooc` should be:

```
intqueuetest is owned by terescoj and has size 6920
README is owned by terescoj and has size 254
```

Note: you will have to pipe the output of "`ls -l`" through the command `grep -v "^total"` to omit the "total" line that is produced by `ls -l`.

The above `awk` commands generate output for each input line. `awk` also has special keywords `BEGIN` and `END` that specify output at the start or end of the output. Back to the name list example:

```
awk 'BEGIN { printf "Last, First\n" } \
 { printf $2 ", " $1 "\n"} \
 END { printf "The end.\n" } ' namelist
```

would produce:

```
Last, First
Smith, John
Jones, Mary
Williams, Alan
Adams, Josephine
The end.
```

Lab Question 2:
Write a Unix pipeline that uses an `awk` command to compute the total bytes of all files in a directory starting with `ls -l`. (5 points)

Hints: print the 5th field of each line of the output of `ls -l` preceded by (or succeeded by) a `+`, then use a `BEGIN` (or `END`) statement in your `awk` program to produce an output like:

```
0+452+1234+93
```

then pipe that output to the `bc` command to do the arithmetic and produce the answer.

`sed`

Next, `sed`, the "stream editor", is used to edit an input stream according to some rules to produce output.

For example, the following command would change all instances of the word "John" to "Bob" in the file `namelist` and send the modified output to standard out.

```
sed "s/John/Bob/g" namelist
```

Note that this does not change the contents of the input file. But it prints out:

```
Bob Smith
Mary Jones
Alan Williams
Josephine Adams
```

In the command, the first parameter to `sed` specified the rule to apply. Here, we **s**earch for "John", replace it with "Bob", **g**lobally (as opposed to a single instance).

The big caveat here is that this will replace *all* instances of "John" with "Bob", even if it's not a first name. Suppose one of the last names is "Johnson" – it would become "Bobson".

Another useful `sed` option is "`p`", which prints. For example, to print lines 3 through 7 of a file:

```
sed -n '3,7p' somefile
```

More on `find`, `grep`, `sed`, and `awk`

Check out this blog for lots of excellent examples of how to use these powerful Unix tools.

Strings in the C Standard Library

We have used several of the functions from the C standard library from the `string.h` header file. Take a look at the list in Section B3 of K&R, and the man page for `string` on your favorite Unix system.

Programming Assignment

There is an online dictionary `/usr/share/dict/words` on mogul that is used by utilities such as the `aspell` command. You know that a *palindrome* is a word or phrase that reads the same in either direction, *i.e.*, if you reverse all the letters you get the same word or phrase. A word is *palindromic* if its reverse is also in the dictionary. For example, "noon" is palindromic, because it is a palindrome and hence its reverse is trivially in the dictionary. A word like "draw" is palindromic because "ward" is also in the dictionary.

Your task is write a C program to find all palindromic words in the dictionary.

Your program should have the following phases:

- Read the dictionary file `/usr/share/dict/words` into an array of strings. You can determine the size of this array ahead of time by running `wc` on the dictionary to see how many words it contains. You may assume that each word is at most 45 characters long.

  Aside: I figured this out with the command

  ```
  awk ' { if (length($1) > max) max = length($0) } \
    END { print max } ' words
  ```

  and the longest word is "pneumonoultramicroscopicsilicovolcanoconiosis".

- For each word, compute its reverse and do a linear search of the dictionary to see if the reverse of the word is in the dictionary. If so, mark the word and increment your counter of the total number of palindromic words. To improve efficiency, you might want to keep track of the positions in your array that store the first word beginning with each letter of the alphabet. Then you can search only within that letter's range when searching for a word's reverse.

- Print the total number of palindromic words to the screen (`stdout`).

The first several words in the dictionary start with numbers and many throughout start with punctuation such as `-`; you should ignore them. In fact, ignore any word that contains any character other than lowercase letters.

The program should be fairly straightforward, but it will likely take longer than you expect to implement. Please ask if you run into trouble!

You may use the `timer.c` and `timer.h` code from the shared area for this lab. Start the clock (take a timer reading) after loading the dictionary and just before you start finding palindromic words; read the timer again as soon as you have finished. Write the elapsed time for the compute phase to `stdout`.

To summarize, your program should have the following output:

- total number of palindromic words (to `stdout`)
- the elapsed time for the compute phase (to `stdout`)

Notes:

- Use the `-O` flag to `gcc` to generate optimized code when you're ready to run timings.
- Of the 355,536 words that contain only lowercase characters, my program finds 2670 palindromic matches.
- This takes about 32 seconds on my Mac.

This program and its `Makefile` is worth 40 points as broken down below.

Submission

Please submit all required files as email attachments to *terescoj@strose.edu*. You are recommended to do so by Wednesday, January 15, 2014. Be sure to check that you have used the correct file names and that your submission matches all of the submission guidelines listed on the course home page.

Grading

Grading Breakdown	
Lab questions	10 points
Program correctness	25 points
Program error checking	2 points
Program memory management	2 points
Program documentation	5 points
Program efficiency	5 points
`Makefile`s	1 point