CMSC 223 Systems Programming
Lab 7: Words, Words, Words!

Here is a problem: Given a text, what is its vocabulary? What are the top 25 most frequently used words?

That is, how many words are used. For example, the entire text of the children's book, *Green Eggs and Ham* by Dr. Seuss, contains only 50 words. How many words are there in Lewis Carroll's *Alice's Adventures in Wonderland*?

In this lab we will try and answer two questions about texts:

1. How many words are used in a given text?
2. What are the twenty-five most frequent words used in a given text?

To answer these questions, we will write some C programs (so we can exercise our understanding and use of arrays and strings) and, along the way, learn to solve problems by combining small programs with some of the Linux commands and utilities.

1. How many words in a given text?

Whenever one must process text, the tricky question is dealing with punctuations. For example, in the sentence below:

```
O'er the ramparts we watched, were so gallantly, streaming?
```

The apostrophe in the word, `O'er`, could be replaced by a 'v'. But what about words like `it's`, `didn't`, etc.? Properly addressing these comprehensively requires deep knowledge of syntax rules which can then be incorporated in a program. Additionally, if we're only interested in counting words, we must normalize words spelled in upper/lowercase letters. For example, a "The" at the start of a sentence is the same word as "`the`" elsewhere in a sentence.

To simplify issues surrounding punctuations and capitalizations, we will make the following decision: before processing any text, we will **replace all punctuations with a blank character** (thus "`O'er`" will become "`O er`"), and **we will convert all uppercase letters into lowercase**. This will obviously result in an approximation but will nevertheless get us started without too much loss, as you will soon see.

Linux has some very useful commands that come in handy when solving problems. With a good knowledge of the command set, you do not always have to write programs to address all processing tasks. The commands enable us to process data much more efficiently and correctly.

`tr – translate or delete characters`

The `tr` command has the following incantation:

```
tr [OPTION]… SET1 [SET2]
```

The command takes input from standard input and translates any occurrence of the pattern specified in `SET1` in the input into the pattern specific in `SET2` (the presence of `SET2` is optional, as indicated by the square brackets `[SET2]`). For example, the command:

```
tr A-Z a-z
```

converts all uppercase letters in the input, into lowercase letters. Go ahead and enter the command in your shell and enter some text to see how it works.

Let's take the *Green Eggs and Ham* text (in ~dkumar/cs223/Lab7/ham.txt) as our test input. First, examine its contents (using cat/more/less). Next, try the command:

```
tr A-Z a-z < ~dkumar/cs223/Lab7/ham.txt
```

You will see that all the text is now converted into lowercase letters. Next try the command:

```
tr [:punct:] " " < ~dkumar/cs223/Lab7/ham.txt
```

All punctuations disappear and are replaced by a space character (" "). Thus, to eliminate all punctuations and convert all letters to lowercase, we can use the Linux pipe in a single command:

```
tr A-Z a-z < ~dkumar/cs223/Lab7/ham.txt | tr [:punct:] " "
```

Nice. We accomplished all this without writing any programs!

For the next task, to count the number of words, we can again, instead of solving the entire problem, write a very simple program that does the following:

Program: words
Usage: words < textFile
Output: Prints out all the words in textFile so that only one word appears on each line.

For example, the first two lines of *Green Eggs and Ham*:

```
I am Sam
I am Sam
```

Will be printed as:

```
I
am
Sam
I
am
Sam
```

Using the command:

```
./words < ~dkumar/cs223/Lab7/ham.txt
```

And you can also cascade the earlier commands to give you the output below:

```
i
am
sam
i
am
sam
```

Here is the main program:

```
#define MAX_WORD_LEN 30 // Assume no word will be more than 30 chars
int main(void) {
   char word[MAX_WORD_LEN+1];

   while ((scanf("%s", word)) != EOF)
      printf("%s\n", word);
   return 0;
} // main()
```

The program above reads a word at a time, using `scanf()`, and then prints it out to standard output.
Question: Why is the word array defined to be one more than the `MAX_WORD_LEN`?

Go ahead, complete and enter the program above in a file called `words.c` and make sure you can run it on the `ham.txt` file as shown above.

Next, use the command

```
tr A-Z a-z < ~dkumar/cs223/Lab7/ham.txt | tr [:punct:] " " | ./words
```

You will see that each word is now output on a single line. Our next task is to make sure that this word list has no repeated words. To do this, you can use a combination of two Linux commands: `sort`, `uniq`

```
tr A-Z a-z < ~dkumar/cs223/Lab7/ham.txt | tr [:punct:] " " | ./words | sort | uniq
```

The `sort` command sorts the input words so that all duplicates will now appear next to each other (you can run the command above without the "| `uniq`". Finally, the `uniq` command eliminates all duplicate lines that are next to each other. Look at their man pages for more information.

Lastly, all you must do is count how many words are in the output. You can do that using the `wc` command.

Did you get 50? Legend has it that someone challenged Dr. Seuss that he couldn't write a story using just 50 words. His book, *Cat In The Hat*, uses only 236 words. How many words does *Alice's Adventures In Wonderland* use? Go ahead and try it. It is in `~dkumar/cs223/Lab7/Alice.txt`

You now have a chain of commands to accomplish the task. Linux allows you to define a command of your own that will carry out all the commands in the chain as needed. This is called a script. To create a simple script, create a new file named, **`getwords`**, with the following contents:

```
#!/bin/bash
tr A-Z a-z < ~dkumar/cs223/Lab7/ham.txt | tr [:punct:] " " | ./words | sort | uniq
```

Make sure the above file is executable:

```
chmod 755 getwords
```

You can now run the entire set of commands by entering just the name of the script file:

```
./getWords
```

This is your first Linux script. We will experiment more with scripts later in the course.

2. What are the twenty-five most frequent words used in a given text?

We can write a program that processes the output from (1) to count words. Make a copy of words.c. into countWords.c. Here is the start of an algorithm to count words:

```
Initialize wordlist, an empty array of words
While there is a word in the input
    add the word to the wordlist, only if it isn't already there
```

You can use the following definition of wordlist:

```
char wordlist[MAX_WORDS][MAX_WORD_LEN+1];
```

You can define MAX_WORDS to be some large number, sufficient for the text: say 10,000. The code for the above algorithm may look like this:

```
int nw = 0; // Number of words stored so far
while ((scanf("%s", word)) != EOF)
      int p = search(wordlist, nw, word); // Look for word in wordlist

      if (p < 0) {  // search returns -1 if word not in wordlist, index o/w
           strcpy(wordlist[nw], word);    // copy word into wordlist
           nW = nw + 1;
       }
    }
```

The search() function can be written as below:

```
int search(char words[][MAX_WORD_LEN+1], int n, char w[]) {
    // Look for word, w in words.
    // Return -1 if not found, o/w return index such that words[index]=w

    for (int index=o; I < n; i++) {
        if (strcmp(words[index], w) == 0)
            return index;
    }
} // search()
```

Complete, write and test your program so that it confirms that you have created the word list. You will need to use the command:

```
tr A-Z a-z < ~dkumar/cs223/Lab7/ham.txt | tr [:punct:] " " | ./countWords
```

Next, create an array of counts and add the commands in boldface shown below:

```
int counts[MAX_Words] = {0};

int nw = 0; // Number of words stored so far
while ((scanf("%s", word)) != EOF)
      int p = search(wordlist, nw, word); // Look for word in wordlist

      if (p < 0) {    // Search returns -1 if word not in wordlist, index o/w
           strcpy(wordlist[nw], word);    // copy word into wordlist
           counts[nw++]++;
      }
      else
           counts[p]++;
  }
```

Now, you have two parallel arrays: one that has all the words, and the second recording how many times a word has been seen. That is:

```
Wordlist = i      counts = 85
           am               16
           sam              19
            …                …
```

Next, you need to sort the two arrays, by count, and print out the top twenty five most frequent words. Here are the commands you can add to the end of the above code:

```
sort(counts, wordlist, nw);

for (int i = 0; i < 25; i++)
    printf("%3d: %s\n", counts[i], wordlist[i]);
```

You can use any sort method you like.

Once done, send an email to your instructor with the list of 5 most frequent words (and their counts) in `Alice.txt`