CSCI 1312 (Introduction to Programming for Engineering), Fall 2015

Homework 6

Credit: 20 points.

1 Reading

Be sure you have read (or at least skimmed) the assigned readings from chapter 7.

2 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu`, with each file as an attachment. Please use a subject line that mentions the course and the assignment (e.g., "csci 1312 homework 6" or "CS1 hw6"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

Yes, this writeup is long. But I think the code you write need not be, and it's an interesting problem!

You may have heard claims that E is the most frequently-used character in English text, followed by T, and so forth. Your mission for this assignment is to write two programs that together will allow you to find out how well this claim holds up in practice (and, okay, to give you some practice working with files in C):

- The first program analyzes a single file, counting occurrences of each character in the Roman alphabet and writing results to an output file. (It also counts total "text" characters, defined here to be ones that are printable but not "whitespace", i.e., tabs, spaces, or newline characters.)
- The second program merges one or more files output from the first program and writes results to standard output.

(Why two programs? Mostly pedagogical reasons.) This is not a trivial assignment but I am providing a function and an example program that I think will make it doable, and if you capture output of the second program in a file (perhaps using Linux output redirection!) you can use the Linux command

```
sort -n -r filename
```

to display the results in a way that shows the most-often-used letter first, etc. One place to look for interesting (or at least non-trivial) text files is Project Gutenberg[1], though you should be careful to get the plain-text version of whatever book(s) you select (really-plain-text, not UTF8).

Note: For pedagogical reasons I do want you to write two programs; I don't think you'll learn quite as much writing only one. Also for pedagogical reasons — and because I think it works out better

[1] `http://www.gutenberg.org`

for you and for me — please go along with the part of the writeup that says that these programs get filenames from command-line arguments rather than by prompting the user as you've done in previous programs.

1. (10 points) The first program (to analyze a single file) analyzes a single input file and produces an output file. It should get their names from command-line arguments. The output file should have the following format:

   - A line giving total "text" characters (ones that are "printable" as defined by the library function `isprint` and not "whitespace" as defined by library function `isspace`).
   - A line for each character in the Roman alphabet that occurs at least once in the input, with how many times it occurs in the file (upper or lower case) and its representation.

   This is probably easiest to understand with examples: If the input file looks like this:

   ```
   testing 1 2 3 4?

   TESTING 4 3 2 1!
   ```

   the output file should look like this:

   ```
   24 total text characters
   2 e
   2 g
   2 i
   2 n
   2 s
   4 t
   ```

   I recommend that you use an array of 26 counters, one for each character of the Roman alphabet. To help(?) you, the file alpha_index.c[2] contains a function that examines a character read from the input file and either returns its index into the alphabet (0 for "a" or "A", 1 for "b", or "B", etc.), or -1 if the character is not alphabetic. You can either copy and paste the function from this file into your program, or you can put the file in your directory and use the line `#include "alpha_index.c"` in your program to have the compiler include it with your code.[3]

   Of course(?), the program should check that the user supplied two command-line arguments and that the input and output files could be opened.

   Hints:

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS1312_2015fall/Homeworks/HW06/Problems/alpha_index.c`

[3] I suspect that many programmers would instead use the library function `isalpha` and do something that directly works with the characters as small integers. I do it this way because the fine print for `isalpha` suggests that in some circumstances it might recognize as "alphabetic" characters I don't mean to include, and while it's true that, these days, in the overwhelming majority of C implementations the representations of the characters of the Roman alphabet are contiguous (e.g., `'a'` is `'z'+25`), that's not guaranteed by the standard, and my function would work even if it weren't. Also I think it's kind of a clever use of pointer arithmetic!

- I think your best best for this program is to read the input file one character at a time, as in the "copy file" sample program `copy-file.c`[4], and to write its output file using `fprintf`.
- Function `count_chars` in sample program `countchars.c`[5] shows one way to count "text" characters in a file.
- Function `alpha_index()` shows a way to transform a character into an index from 0 to 25, or -1. To transform the index for an alphabetic character back to its character representation, an easy way compatible with the character-to-index transformation would be to use the global variable `alphabet`; it's an array of characters, after all(?), with element 0 equal to `'a'`, etc.

2. (10 points) The second program (to merge results from one or more executions of the first program) simply combines results in the obvious(?) way. Given the following two input files (output of the first program):

   - The output file from the previous example:

     ```
     24 total text characters
     2 e
     2 g
     2 i
     2 n
     2 s
     4 t
     ```

   - The following file:

     ```
     56 total text characters
     3 a
     1 c
     2 d
     6 e
     2 f
     1 g
     3 h
     4 i
     2 l
     2 m
     2 n
     9 o
     2 p
     4 r
     3 s
     7 t
     1 w
     1 y
     ```

     which, for the curious, is the output from processing this input:

[4] `http://www.cs.trinity.edu/~bmassing/Classes/CS1312_2015fall/SamplePrograms/Programs/copy-file.c`

[5] `http://www.cs.trinity.edu/~bmassing/Classes/CS1312_2015fall/SamplePrograms/Programs/countchars.c`

```
Now is the time for all good persons
to come to the aid of their party!
```

output should look something like the following:

```
processing input file sample1-step1-out.txt
  24 total text characters
processing input file sample2-step1-out.txt
  56 total text characters

summary:

3 a (3.7500%)
1 c (1.2500%)
2 d (2.5000%)
8 e (10.0000%)
2 f (2.5000%)
3 g (3.7500%)
3 h (3.7500%)
6 i (7.5000%)
2 l (2.5000%)
2 m (2.5000%)
4 n (5.0000%)
9 o (11.2500%)
2 p (2.5000%)
4 r (5.0000%)
5 s (6.2500%)
11 t (13.7500%)
1 w (1.2500%)
1 y (1.2500%)

11 non-alphabetic text characters (13.7500%)
```

Of course(?), the program should check that the user supplied at least one command-line argument and that all of the input files could be opened. It probably should also check, as it reads through the input files, that each one is in the right format (output of the first program), since otherwise the program might easily crash. It doesn't have to do this all at once: it's probably simpler to process the input files one at a time, and it's okay if the program starts processing and producing output and then bails out if it encounters an error.

Hints:

- I think your best best for this program is to read the input files mostly using `fscanf`. But for some things (e..g., reading that first line) you may find it more useful to use `fgetc`, since for that line you want to just get the count and discard the rest of the line. The following line of code will discard input (from `infile`) up through the next newline:

  ```
  while (fgetc(infile) != '\n');
  ```

  Notice that in processing the single letter read from the "how many occurrences of this letter" lines you could once again use my `alpha)` function.

- The main program in `countchars.c`[6] shows one way to process one or more input files whose names are given via command-line arguments.
- You can bail out of a C program at any time — i.e., from a function other than `main()` — by calling the library function `exit(n)` where `n` is the same kind of value you've been returning from `main()` — 0 for "everything okay" and anything else for an error.

[6] `http://www.cs.trinity.edu/~bmassing/Classes/CS1312_2015fall/SamplePrograms/Programs/countchars.c`