SENG265
FALL 2017
ASSIGNMENT 1
UNIVERSITY OF VICTORIA

Due: Oct 16, 2017 by 10:00 am, by "git push".
(Late submissions **not** accepted)

1 Assignment Overview

This assignment involves writing a command line application to process streams of text from a file. The program will read lines of text from a given file, compute the frequency of words of certain length from the file and print these frequencies to standard output.

The overall goal of this assignment is to introduce C programming in a unix setting, with particular emphasis on C array and string processing. Your eventual submission will consist of the source files for the program, accompanied by test cases. There are three parts to this assignment and Sections 1.1, 1.2 and 1.3 below contain `Specifications` for each of the three programs. Section 2 describe the `Constraints` you should consider in your implementation, and Section 3 describes the `Testing` component. `What you should Submit` is outlined in Section 4 and the `Evaluation` scheme is given in Section 5.

Your code is expected to compile without warnings **in the course lab (ECS 342)** or **linux.csc.uvic.ca** using the `-Wall -g` and `-std=c99` flags with the `gcc 4.8` compiler. To save time, you can also create a *Makefile* as discussed in the lab. Because the next assignment tests your knowledge of dynamic memory allocation, you are asked to not use dynamic memory allocation for this assignment, as it is useful to learn how to rely exclusively on automatic allocation.

1.1 Part A. Frequency of words of all lengths

The first part of the assignment is to write a C program, contained in a source file called `word_count.c`, which counts the number of words of all lengths. The program must compile, with no warnings, and run using the following commands:

```
$ gcc -Wall -std=c99 -o word_count word_count.c
```

```
$ ./word_count --infile <input_file>
```

After compiling, a correct implementation will take the name of a word list file as a command line argument and output the frequency of words of all lengths in that file, e.g in the form of a function Count[arg] where arg is the length of the word. For example, consider the following as input file `input_file.txt`:

```
Tomorrow, and tomorrow, and tomorrow,
To the last syllable of recorded time;
```

There are 2 words of length 2: `to` and `of`, and so Count[2]=2;

There are 3 words of length 3: `and` (twice) and `the`, and so Count[3]=3;

There are 2 words of length 4: `last` and `time`, and so Count[4]=2;

And finally there are 5 words of length 8: `tomorrow` (3 times), `syllable` and `recorded`, and so Count[8]=5;

Therefore the complete output of the program should be:

Count[02]=02;
Count[03]=03;
Count[04]=02;
Count[08]=05;

A note on output formatting: For all three parts of the assignment the outputs could be rendered as single 0 padding, e.g.

Correct: Count[08]=05; or
Incorrect: Count[8] = 5;
Incorrect: Count[08]= 5;
or even
Incorrect: Count[8] = 0005;

1.2 Part B. SORTED Frequency of words of all lengths

The second part of the assignment implements the same program as in Part A but the output is sorted by frequency of words, and outputted in descending order of frequency. Add an additional optional argument to your Part A code (i.e. do not create a new C source file), that will be run as shown. You **cannot** assume that the arguments will be run in this order.

```
$ ./word_count --sort --infile <input_file>
```

For example, for the same input file as above, the output should be:

Count[08]=05;
Count[03]=03;
Count[02]=02;
Count[04]=02;

1.3 Part C. SORTED Frequency of words of all lengths with Words information

The third part of the assignment adds in the option to display the unique words found for each word length in alphanumeric order. Add an additional optional argument to your Part A & B code (i.e. do not create a new C source file), that will

be run as shown. You **cannot** assume that the arguments will be run in this order.

```
$ ./word_count --sort --print-words --infile <input_file>
```

For example, for the same input file as above, the output should be:

```
Count[08]=05; (words: "recorded", "syllable" and "tomorrow")
Count[03]=03; (words: "and" and "the")
Count[02]=02; (words: "of" and "to")
Count[04]=02; (words: "last" and "time")
```

2 Constraints

You are **NOT** allowed to use *malloc/realloc* for this assignment. Therefore, you can make the following assumptions in your code:

- The maximum file size is 5000 bytes
- The maximum words in a file is 750
- The maximum word size is 34 bytes
- Lower case and upper case words should be treated the same (i.e. Tomorrow and tomorrow both go in the same bucket)
- The only allowed special characters to be included in the input file are .,;(). No other special characters are expected to be included in the input file.

3 Test Inputs

You should test all of your programs with a variety of test inputs, covering as many different use cases as possible, not just the test input provided. You should ensure that your programs handle error cases (such as files which do not exist) appropriately and do not produce errors on valid inputs. Since thorough testing is an integral part of the software engineering process, you will be expected to submit one test input.

For the `word_count` program, submit a file `count_readme.md` that describes **what use case or scenario** you are testing and a file `input_file.txt` containing input text. Your submitted test case is expected to be a valid input, and therefore must obey all of the constraints on input given in Section 2. You will not receive any marks for your test case if it violates any of the input constraints or you do not submit both files.

Due to file size constraints for electronic submission, your test files may be at most 10kb in size.

4 What you must submit

- C source-code name `word_count.c` which contains your solution for Parts A, B and C of Assignment 1.
- A text file `input_file.txt` that contains **your** submitted test input file.
- A text `count_readme.md` file that explains what use case your `input_file.txt` input file tests.
- Ensure your work is **committed** to your local repository **and pushed** to the remote **before the due date/time**. (You may keep extra files used during development within the repository.)

5 Evaluation

The teaching staff will primarily mark solutions based on the input files provided for this assignment. Students must adhere to the software requirements (command execution and output formatting) outlined in this assignment. For each assignment, some students will be randomly selected to demo their code to the course markers. Each student will have this opportunity to demo at least one assignment during the semester. Sign-up procedure will be discussed in class.

Our grading scheme is relatively simple.

- "A" grade: A submission completing ALL Parts and all requirements of the assignment and all tests pass. The `word_count` programs runs without any problems.
- "B+" grade: A submission that completes part A & B of the assignment and all tests pass. The `word_count` programs runs without any problems.
- "B-/B" grade: A submission that completes part A of the assignment and all tests pass. The `word_count` programs runs without any problems.
- "C" grade: A submission that completes part A of the assignment and passes some tests. The `word_count` programs runs with some problems.
- "D" grade: A serious attempt at completing requirements for the assignment. The `word_count` program compiles and runs with some problems.
- "F" grade: Either no submission given (or did not attend demo); submission represents very little work or understanding of the assignment.