SENG265
FALL 2020
ASSIGNMENT 1
UNIVERSITY OF VICTORIA

Due: ~~Oct 5, 2020~~ 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 1.2, 1.3 and 1.4 below contain `Specifications` for each of the three programs. Section 3 describe the `Constraints` you should consider in your implementation, and Section 2 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 Senjhalla** (the virtualbox+vagrant virtual machine) using the `-Wall -g` and `-std=c99` compile flags with the `gcc 9.3.0` compiler.

Because a later assignment will test 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 Assignment Package

These instructions assume you have completed Lab 2 and you have cloned your git repository to the home directory in your Senjhalla virtual machine. If you have not yet done so, refer to the Lab 2 video and slides.

Download the `assign1.zip` package from Connex and unzip it into the `a1` folder of your git repository. The folder structure is shown in Fig. 1.

Add your code to the `src` files provided with your solution to parts A, B, and C. You may create additional files in `src` but do not rename `word_count.c`. A suggested template has been provided for you in `word_count.h`.

`cases` hold the test files for parts A, B, and C. The input files have the format `t*.txt` and the corresponding expected output format is `c*.txt`. For example, the expected output for a solution to Part B, using input `t04.txt`, will be found in `cases/B/c04.txt`.

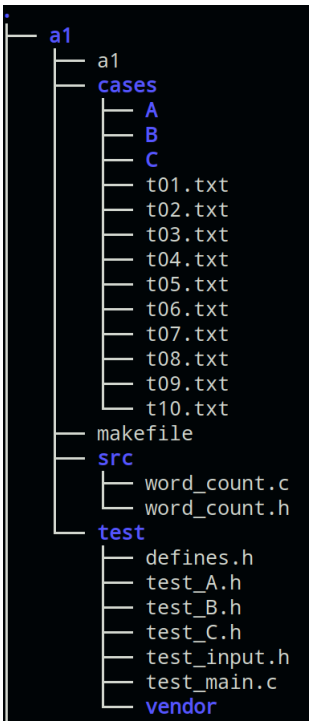

Figure 1: Assignment package

The `test` folder contains the framework that will be used for evaluation. See section 5 for details on how to run the tests. You may add your own tests to the existing tests, but they will not be used in your evaluation.

1.2 Part A. Frequency of words of all lengths

The first part of the assignment is to write a C program, contained in the source files called `word_count.c` and `word_count.h`, which counts the number of words of all lengths. You have been provided skeleton template files in the assignment package.

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;

Tests associated with this part are located in `cases/A` and `test/test_A.h`.

1.3 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. You will also output the **median** word length. In the case of an even number of word lengths, take the average between the two middle bucket word lengths. Do **not** print the median if `--print-words` argument is also provided (as in Part C).

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;
Median word length: 3.5

In the case of a tie (as shown above) then you do a secondary sort based on the word length bucket/bin value. As in the example, Count[02] is a smaller word length than Count[04] so it is sorted above the longer world length.

Tests associated with this part are located in `cases/B` and `test/test_B.h`.

1.4 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: "Tomorrow", "recorded", "syllable" and "tomorrow")
Count[03]=03; (words: "and" and "the")
Count[02]=02; (words: "To", "of" and "to")
Count[04]=02; (words: "last" and "time")

Tests associated with this part are located in `cases/C` and `test/test_C.h`.

2 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.

You have been provided a set of **10** test input files (t01.txt to t10.txt) and **21** (c01.txt to c07.txt) expected output files (7 for each part) located under the `cases` folder. Your code also needs to be able to handle basic user error. See `test/test_input.h` for expected behaviour when handling incorrect command-line arguments. **Do not use exit() to exit the program.**

You have also been provided a testing framework that allows you to run these tests using a `makefile`.

To run the makefile:

```
$ make test
```

This will compile and run the test framework in `test` and create the binary `tests.out`. You can re-run the tests using `./tests.out`. **If you modify you code, you will need to re-compile using make first**.

The output is formatted as follows:

```
TEST_FILE:TEST_FILE_LINE_NUMBER:TEST_FUNCTION:PASS_OR_FAIL_OR_IGNORE
 Expected: 'TEXT_FROM_TEST_CASE';
 Actual  : 'OUTPUT_FROM_YOUR_PROGRAM';. ==> INPUT_FILE
```

Fig.2 shows an example of the test output.

When a test fails, you are provided with the *function that failed*, the *truncated* `Expected` (green) output from the c*.txt file and the `Actual` (red) output from your program, at the point it failed. You can turn tests off while testing by uncommenting `TEST_IGNORE()` in test function(s) located in `test_A.h`, `test_B.h`, `test_C.h`. There are 9 tests that are ignored that will used during the evaluation of your code.

In the provided example, out of the 34 possible tests, 6 tests have failed, and 9 tests ignored.

3 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 as different words (i.e. Tomorrow and tomorrow are printed separately)
- 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.
- In the case of a tie in the word frequency/count, then sort normally by word length.
- Do **NOT** use `exit()` to exit the program on an error.

4 What you must submit

- C source-code name `word_count.c` and other files which contains your solution for Parts A, B and C of Assignment #1 located in `src`
- Ensure your ALL 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.

```
vagrant@ubuntu-bionic:~/adash42/assign1$ make test
Compiling tests.out
gcc -std=c99 -g -Wall -D TEST test/vendor/unity.c test/test_main.c -o  tests.out
test/test_main.c:16:test_no_arguments:PASS
test/test_main.c:17:test_invalid_file:PASS
test/test_main.c:18:test_no_file:PASS
test/test_main.c:19:test_reorder:PASS
======== Part A ===============
test/test_main.c:23:test_empty_file:PASS
test/test_main.c:24:test_single_word:PASS
test/test_main.c:25:test_long_word:PASS
test/test_main.c:26:test_multi_words:PASS
test/test_main.c:27:test_multi_line:PASS
test/test_main.c:28:test_special_characters:PASS
test/test_main.c:29:test_extra_spaces:PASS
test/test_main.c:100:test_long_paragraph:IGNORE: Used during evaluation only
test/test_main.c:113:test_long_paragraphs:IGNORE: Used during evaluation only
test/test_main.c:126:test_super_long:IGNORE: Used during evaluation only
==============================
======== Part B ===============
test/test_main.c:36:test_sort_empty_file:PASS
test/test_main.c:37:test_sort_single_word:PASS
test/test_main.c:38:test_sort_long_word:PASS
test/test_main.c:55:test_sort_multi_words:FAIL:
 Expected 'Count[03]=04;\nCount[05]=03;\nCount[04';
 Actual   'Count[03]=04;\nCount[05]=03;\nCount[06';. => t04.txt
test/test_main.c:68:test_sort_multi_line:FAIL:
 Expected 'Count[03]=05;\nCount[02';
 Actual   'Count[03]=05;\nCount[04';. => t05.txt
test/test_main.c:41:test_sort_special_characters:PASS
test/test_main.c:94:test_sort_tie_breaker:FAIL:
 Expected '\nCount[04]=11;\nCount[05]=07;\nCount[01]=03;\nCount[06';
 Actual   '\nCount[04]=11;\nCount[05]=07;\nCount[01]=03;\nCount[09';. => t07.txt
test/test_main.c:100:test_sort_long_paragraph:IGNORE: Used during evaluation only
test/test_main.c:113:test_sort_long_paragraphs:IGNORE: Used during evaluation only
test/test_main.c:126:test_sort_super_long:IGNORE: Used during evaluation only
==============================
======== Part C ===============
test/test_main.c:49:test_print_empty_file:PASS
test/test_main.c:50:test_print_single_word:PASS
test/test_main.c:51:test_print_long_word:PASS
test/test_main.c:54:test_print_multi_words:FAIL:
 Expected 'Count[05]=03; (words: "brown" and "quick")\nCount[04';
 Actual   'Count[05]=03; (words: "brown" and "quick")\nCount[06';. => t04.txt
test/test_main.c:67:test_print_multi_line:FAIL:
 Expected '05; (words: "aid", "all", "for" and "the")\nCount[02';
 Actual   '05; (words: "aid", "all", "for" and "the")\nCount[04';. => t05.txt
test/test_main.c:54:test_print_special_characters:PASS
test/test_main.c:93:test_print_extra_spaces:FAIL:
 Expected 'there")\nCount[01]=03; (words: "I" and "i")\nCount[06';
 Actual   'there")\nCount[01]=03; (words: "I" and "i")\nCount[09';. => t07.txt
test/test_main.c:99:test_print_long_paragraph:IGNORE: Used during evaluation only
test/test_main.c:112:test_print_long_paragraphs:IGNORE: Used during evaluation only
test/test_main.c:125:test_print_super_long:IGNORE: Used during evaluation only
==============================

-----------------------
34 Tests 6 Failures 9 Ignored
FAIL
vagrant@ubuntu-bionic:~/adash42/assign1$
```

Figure 2: Example of test results with some failing cases.

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.