Project 3: Custom Word Count

Submission deadline: 9pm on 11/17, 2022
(9% of the total grade + 1% extra credit)

Write a program that reads a stream of English sentences from stdin and print the frequency of each word in the sorted order (from the largest to the n-th largest). The program takes ‘n’ as an optional command line argument. So, if n is 5 (as below), your program need to print the top 5 words by their frequency in the sorted order. If n is missing, your program should print all words from the largest frequency to the smallest frequency.

```
$ ./count 5 < file1
5436 the
3038 I
2884 and
2791 to
2738 of
```

Requirements & Information:

- Write the code in C – your code should compile with gcc installed on eelab5 or eelab6
- The input text consists of an arbitrary number of English sentences (a stream of English words). For simplicity (for ease of parsing), we define a word as one or more sequence of alphabets delimited by non-alphabets. Here are some of the examples for parsing:

  I say hello! -> three words (I, say, hello)
  I can’t say no. -> five words (I, can, t, say, no)
  “Do you know me?” -> four words (Do, you, know, me)

- For word parsing, you can refer to token.c attached to the homepage.
- Check the correctness of your program by comparing the result of “./token < file1 | sort | uniq –c | sort –k 1 –r | more”
- We provide a few text files for testing. Please see the attached files.
- You can assume that the same word does not show up for more than 2 billion times.
- You can use standard C runtime libraries (including qsort()) but you cannot use any library that implements basic/advanced data structures (like a hash table, a priority queue/heap, etc.)
- Name your source code file as count.c.
- You will get **ZERO** point if the source code doesn’t compile (due to compiling errors). Make sure that your code compiles well before submission.
- Suppress all warnings at compiling by turning on the “-W -Wall” options in the CFLAGS in Makefile. You will get **some penalty** if gcc produces *any* warnings.

Collaboration policy:

- You can discuss algorithms and implementation strategies with other students, but you should write the code on your own. That is, you should NOT look at anyone else’ code.

Submission:

- You need to submit three files – “count.c”, “Makefile”, and “readme” in a tarred gzipped file with the name as YourID.tar.gz. If your student ID is 20211234, then 20211234.tar.gz should be the file name.
- ‘make’ should build the binary, “count”.
- “readme” (a text file with “readme” as the file name) should describe how you implement the code. (1) Explain the data structure you chose to implement as well as the overall algorithm. What is the worst-case running time of your algorithm in terms of Big-O notation? (2) Explain how you checked the correctness of your program (3) Add some timing results with the text files we provide. (4) **(extra credit up to 1%)** Explain why your choice of the data structure and algorithm is optimal in terms of running time? You can assume that the optional argument ‘n’ is missing for this analysis.

Grading:

- We will evaluate the correctness of your program.
- We will give penalty to the program that is way too slow (like bottom 10%) – if you chose the right data structure/implementation/algorithm, it should work fine.
- We will evaluate your readme file.
- We will evaluate the clarity of your code – coding style, comments, etc.