Software Engineering 265
Software Development Methods
Summer 2018

Assignment 4

Due: Friday, August 3rd, 11:55 pm via a push to your Git remote repository.

(no late submissions accepted)

Programming environment

For this assignment you must ensure your work executes correctly on the machines in ECS B238 (i.e., these have gcc 4.8.5 installed). You are free to do some of your programming on your own computers; if you do this, give yourself a few days before the due date to iron out any bugs in the C solution you have pushed to your remote repository.

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussion is encouraged. **However, sharing of code fragments is strictly forbidden without the express written permission of the course instructor (Zastre).** If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact me as soon as possible. (Code-similarity analysis tools may be used to examine submitted programs.)

Objectives of this assignment

- Use the dynamic-memory support available in C (i.e., *malloc()*, *realloc()*, etc.).
- Use some of the separable-compilation features of C.
- Use a *makefile*.
- Use git to manage changes in your source code and annotate the evolution of your solution with "messages" provided during commits.
- Test your code against the provided test cases.

This assignment: *uvroff2.c* and *formatter.c*

At this point in the semester you are probably weary our formatting problem. (That written, however, you may enjoy reading the article on troff at Wikipedia – head to `https://en.wikipedia.org/wiki/Troff`). However, on the bright side you

thoroughly understand what output is expected for the given inputs. Now you can concentrate on what is perhaps the most difficult aspect of programming in C: dynamic memory. For this assignment you will be wrapping a solution to the formatting problem in a C file / module. The module provides different access functions depending on whether a file's contents are to be formatted or an array of strings is to be formatted. The formatted output will be returned from the called function as a dynamically-allocated array of strings.

- *uvroff2.c* will accept a filename as an argument and will call the appropriate routine in *formatter.c*. The strings in the returned string array will be output to stdout.

- If no filename is provided to *uvroff2*, then the contents of *stdin* will be formatted. (Please: do not prompt the user for this!)

- *driver.c* already creates a statically-allocated array of strings and passes this to the *format_lines()* function in *formatter.c*. As with *uvroff2.c*, the array of strings returned from the routine are to be output to *stdout*. Please note that it is your code's responsibility to allocate memory for both the result array and the strings accessible from the result array.

- The specifications from the second assignment are to be used for this assignment.

- Several files have already been provided for you in */home/zastre/seng265/assign4*. You are to use these files when starting your work. Ensure they are placed in the *assign4* directory of your project.

- With your completed *uvroff2* program, the input would be transformed into the output (here redirected to a file) via one of the two following UNIX commands:

```
% ./uvroff2 /home/zastre/seng265/assign2/tests/in11.txt > \
     ./myout11.txt

% cat ~zastre/seng265/assign2/tests/in11.txt |./uvroff2 > \
    ./myout11.txt
```

where the file "myout11.txt" would be placed in your current directory.

Exercises for this assignment

1. Within your git repository confirm there is an *a4* subdirectory. Use the test files in */home/zastre/seng265/assign2*. (Files *in01.txt* through to *in20.txt* are the same as those used for the second and third assignment.) Your completed *uvroff2.c* and *formatter.c* files (plus *driver.c* and *formatter.h*) must be located in this directory. Ensure that your *a4* directory and all files within it are added and committed to git.

2. **Your solution cannot assume a maximum length for an input line, nor can you assume a maximum number of lines. Use *malloc()* to allocate memory for strings and arrays of strings.**

3. Reasonable run-time performance of your program is expected. None of the test cases should take longer than 15 seconds to complete on a machine in ELW B238.

What you must submit

- Three C files named *uvroff2.c, driver.c* and *formatter.c* within your git repository containing a solution to Assignment #4.

- If you have modified *makefile* or *formatter.h*, then ensure these files are also in the Git repository. Failure to include any such modifications may mean we cannot build your submission (i.e., no tests will pass).

Evaluation

Our grading scheme is relatively simple and is out of 10 points. We will award grades within the categories below.

- 10/10: *uvroff2* and hence the code in *formatter.c* run without any problems. *driver.c* runs without any problems. The program is clearly written and uses functions appropriately (i.e., is well structured).

- 8/10: *uvroff2* and hence the code in *formatter.c* run without any problems. *driver.c* runs without any problems.

- 7/10: A submission completing most of the requirements of the assignment. *uvroff2* and hence the code in *formatter.c* run with some significant problems. *driver.c* might or might not run without any problems.
- 5/10: A serious attempt at completing requirements for the assignment. *uvroff2* and hence *formatter.c* run with several serious problems.

- 4/10 or lower: Either no submission given, or submission represents very little work.