Software Engineering 265
Software Development Methods
Summer 2020

Assignment 3

Due: Monday, July 20, 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 virtual machines you installed as part of Assignment #0 (which I have taken to calling `Senjhalla`). This is our "reference platform". This same environment will also be used by the course instructor and the rest of the teaching team when evaluating submitted work from students.

Any programming done outside of this reference platform might not work correctly. Such work may receive a failing grade.

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 will be used to examine submitted work.)

Objectives of this assignment

- Use the dynamic-memory support available in C (i.e., `malloc()`, `realloc()`, etc.) through the use of provided linked-list routines.
- 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: `senjify3.c`

For this third assignment you will revisit the problem by using the same formatting-behavior specification as was given for the second assignment. That is, there are no additional formatting commands or features required for this present assignment, and therefore you may also use all tests cases from A#2.

This also now means that `senjify3` must accept a filename as an argument for input. If none is provided, input is to be from `stdin`. All output is, as before, to `stdout`. The starter `senjify3.c` and other support files needed can be found on the lab machines in:

```
/home/zastre/seng265/assign3
```

Note that I've already provided the logic in `senjify3.c` for distinguishing between input from a named file and input from `stdin`.

You are required to use the linked-list routines provided to you in this assignment. This linked-list implementation is very close to the one described in lectures. Also provided to you is the `emalloc.c` module which contains the function that will also be described in lectures.

Assuming you have copied the tests inputs and outputs from the SENG servers (i.e., tests from the previous assignments as found in `/home/zastre/seng265/assign2`) into `Senjhalla`, and that this subdirectory is named `tests/` and is located within the directory you are using for writing `senjify3.c`, then the input would be transformed into the output via the following command:

```
% cat tests/in04.txt | ./senjify3 > testout04.txt
```

and the file `testout04.txt` produced by re-directing the output from the program appears here in the same directory as `senjify3`. To compare the output produced by `senjify3` with what is expected, you can use the Unix `diff` command as shown below (assuming you're still in the same directory as when you executed the command above):

```
% diff testout04.txt tests/out04.txt
```

In you wish, you can combine both the formatting and the testing into one longer command by using only Unix pipes:

```
% cat tests/in04.txt | ./senjify3 | diff – tests/out04.txt
```

Also notice it is now possible to provide the input filename on the command line rather than piping input to `stdin`. This is shown in the following which repeats the two commands above using `senjify3`:

```
% ./senjify3 tests/in04.txt > testout04.txt
```

```
% ./senjify3 tests/in04.txt | diff - tests/out04.txt
```

Your program must support both styles of usage (i.e., input specified as a filename to the program, or input provided as a stream of characters from `stdin`).

Exercises for this assignment

1. Write your program in the `a3` directory within your local repo for SENG 265. Amongst other tasks you will need to:
   - read text input from a `stdin` or f file, line by line;
   - write output to `stdout`;
   - extract substrings from lines produced when reading a file;
   - use the `-std=c99 -O0` flags when compiling to ensure your code is compliant with the 1999 C programming language standard.

2. **Your solution cannot assume a maximum length for an input line, nor can you assume a maximum number of lines. Use the `getline()` function to help with dealing with the length of an input line.**

3. Use the test files to guide your implementation effort. Start with simple cases (such as those given in this writeup). In general, lower-numbered tests are simpler than higher-numbered tests. Refrain from writing the program all at once, and budget time to anticipate for when "things go wrong".

4. For this assignment you can assume all test inputs will be well-formed (i.e., the teaching team will not test your submission for handling of errors in the input). Later assignments may error-handling as part of the assignment.

5. Use `git add` and `git commit` appropriately. While you are not required to use `git push` during the development of your program, you **must** use `git push` in order to submit your assignment.

6. Reasonable run-time performance of your program is expected. None of the test cases should take longer than 15 seconds.

What you must submit

- Three C files named `senjify3.c`, `list.c` and `list.h` within your git repository containing a solution to Assignment #3.

- If you have modified `makefile` or `emalloc.h`, `emalloc.c`, then ensure these files are also in your git project. Failure to include any such modifications may mean the teaching team cannot build your submission (i.e., no tests will pass).

Evaluation

Our grading scheme is relatively simple.

- "A" grade: A submission completing the requirements of the assignment. Submitted code is well-structured and very clearly written. Global variables are kept to a minimum. `senjify3` runs without any problems; that is, all tests pass and therefore no extraneous output is produced. If `valgrind` produces a clean report for the largest test cases, then that may also raise the assignment mark.
- "B" grade: A submission completing the requirements of the assignment. `senjify3` runs without any problems; that is, all tests pass and therefore no extraneous output is produced. The program is clearly written.
- "C" grade: A submission completing most of the requirements of the assignment. `senjify3` runs with some problems.
- "D" grade: A serious attempt at completing requirements for the assignment. `senjify3` runs with quite a few problems. *Most tests do not pass, although some tests do pass.*
- "F" grade: Either no submission given, or submission represents very little work. *No tests pass.*