Software Engineering 265
Software Development Methods
Spring 2019

Assignment 4

Due: Sunday, April 7, 11:30 pm by submission via git push
(no late submissions accepted)

Programming environment

For this assignment please ensure your work executes correctly on the Linux machines in ELW B238. You are welcome to use your own laptops and desktops for much of your programming; if you do this, give yourself a few days before the due date to iron out any bugs in the C program you have uploaded to the lab machines. (Bugs in this kind of programming tend to be platform specific, and something that works perfectly at home may end up crashing on a different hardware configuration.)

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 programs.)

Objectives of this assignment

- Understand a problem description, along with the role used by sample input and output for providing such a description.
- Use the C programming language, separable compilation and dynamic memory to write a C version of the enhanced MTF decoder. **(*Note: You will write only the decoder – and <u>not</u> the encoder – for this assignment.)***
- Use git to manage changes in your source code and annotate the evolution of your solution with "messages" with committed changes.
- Test your code against the provided test cases.

This assignment: "coding2.c"

In our final MTF coding exercise we will re-visit the C programming language. Using the same MTF coding specification (and test files!) from assignment #3, you are to:

- Use separate compilation to combine modules using MTF coding operations with modules that invoke that functionality.
- Use C's dynamic memory mechanism to respond, at runtime, to the size of the input (i.e., number of unique words in the test case).

*For this assignment you will **write the decoder only** (i.e., you are not required to implement the encoder).*

Separable compilation

In assignments #2 and #3 you wrote encoding and decoding logic in the same file. This reduces code duplication as structures and routines are shared by encoding and decoding algorithms. We use this idea in A#4 by splitting our C program into separate files.

- *coding2.c*: Contains top-level functions *encode()* and *decode()* each of which are passed file pointers as arguments. All of the support functions needed to accomplish the work of these top-level functions will be contained in *coding2.c*. Note that *coding2.c* ***does not contain a main() function***. Better solutions will keep file- and program-scope variables to a minimum.
- *coding2.h*: Function prototypes for functions within *coding2.c* that will be called by C statements that are outside of *coding2.c*.
- *mtf2text2*.c: Logic to obtain the filename argument from the command line, create the name of the output file name (i.e., a "txt" file), invokes *decode()*, and responds if *decode()* indicates an error occurred during decoding.
- *text2mtf2.c*: Logic to obtain the filename argument from the command line, create the name of the output file name (i.e., an "mtf" file), invokes *encode()*, and responds if *encode()* indicates an error occurred during encoding. *Note that this file is provided only for completeness, i.e., you do not need to test for encoding in A#4.*
- *makefile*: Configuration data needed by *make* to invoke *gcc* such that the binaries *mtf2text2* and *text2mtf2* are created. (Note that *text2mtf2* will, in effect, do nothing.)

Of these five files, only <u>coding2.c</u> is to be modified (or equivalently, modifying any of the other files requires the express written permission of the instructor). However, you should include all of these files in your *a4* directory and as part of your Git commit.

Dynamic memory

In assignment #1 we specified some compile-time constants meant to specify maximum sizes of data structures at compile time. **For this assignment the only compile-time constant permitted** for such a maximum size is MAX_FILENAME_LEN (used in *mtf2text2.c* and *text2mtf2.c*).

Exercises for this assignment

1. Within your git project create an *a4* subdirectory. Ensure all directories and program files you create during development are placed under Git control. You need not add the directory containing test files to your git unless you wish to do so. Test files to be used are the same as that for A#3.

2. All files described on page 2 are within the */home/zastre/seng265/a4* directory. ***You are required to change coding2.c.*** *No other change to any other file is permitted without express written permission from the instructor.*

3. Write your program (which for this assignment will only perform decoding). The decoder will be invoked in the manner similar to Assignment #1, albeit with Assignment #3 test files.

4. You are welcome to use regular expressions if you wish but I seriously doubt you will need them.

5. Use the test files to guide your implementation effort. Start with the simple example in test 01 and move onto 02, 03, etc. in order. (Note that test19 is particularly punishing.) **Refrain from writing the program all at once, and budget time to anticipate when things go wrong!** Use the Unix command *diff* to compare text files.

6. For this assignment you can assume all test inputs will be well-formed (i.e., our marking team will not test your submission for handling of input or for arguments containing errors). Later assignments might specify error-handling as part of their requirements.

7. Write one test script, i.e., a Unix script that goes through all of the provided test cases (i.e., for decoding) and reports which test cases fail.

What you must submit

- A single C file named *coding2.c* within your git repository containing a solution to Assignment #4. However, ensure all five files are committed and pushed. (Therefore submission means committing your modifications to *coding2.c,* and all of the other files *without* modifications.)

- One Unix script (i.e, for testing your decoding functionality).

Evaluation

Our grading scheme is relatively simple.

- "A" grade: An exceptional submission demonstrating creativity and initiative. The code within `coding2.c` runs without any problems. Test script is provided. The program is clearly written and is structured in a way that also uses functions appropriately.
- "B" grade: A submission completing the requirements of the assignment. The code within `coding2.c` runs without any problems. Test script is provided. The program is clearly written.
- "C" grade: A submission completing most of the requirements of the assignment. The code within `coding2.c` runs with some problems. Additional compile-time constants have been introduced into the code. Test script might be missing.
- "D" grade: A serious attempt at completing requirements for the assignment. The code within `coding2.c` runs with quite a few problems. At least a handful of tests pass. Test script might be missing.
- "F" grade: Submission either represents very little work or cannot be executed for testing.