Software Engineering 265
Software Development Methods
Spring 2019

Assignment 1

Due: Tuesday, February 12, 4:30 pm by submission via git
(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 BSEng 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 to write the first implementation of a file encoder named “text2mtf” (and do this without using dynamic memory).
- Learn more about the ASCII encoding of text.
- Use Unix commands such as “diff” and “hexdump” to support your testing and coding.
- Use git to manage changes in your source code and annotate the evolution of your solution with “messages” given to commits.
- Test your code against the provided test cases.

This assignment: “text2mtf.c”

Move-to-Front Coding (MTF) is an adaptive coding scheme sometimes used for lossless compression of text. Words appearing frequently in a source file are encoded with integers that are smaller than encodings for words appearing less frequently. (An insight here is that fewer bits are needed to represent smaller integers than larger integers, and we could exploit this as a form of compression.) However, for this assignment all of our codes will be of the same size. If compression is achieved then it will be the result of substituting a one-byte code for a multiple-character word. (In later assignments we will experiment with differing code lengths.) In this assignment we are simply exploring MTF coding; if the resulting files are smaller then that is a happy result.

MTF behaves in a manner similar to the way we might use a vertical stack of books. If we need a book from the middle of the pile, we retrieve it and when finished put the book back on top of the pile. The figure below shows a sequence of input words (taken from Dr. Seuss’s “The Cat in the Hat”), along with the order in which words appear in the word list after each input word is processed, and finally the output generated by the encoding. Notice the cases when a word is moved from a position within the word list to the top of that list.

input words	**i**	**sat**	**there**	**with**	**sally**	**we**	**sat**	**there**	**we**	**two**
word list	i	sat	there	with	sally	we	sat	there	we	two
		i	sat	there	with	sally	we	sat	there	we
			i	sat	there	with	sally	we	sat	there
				i	sat	there	with	sally	sally	sat
					i	sat	there	with	with	sally
						i	i	i	i	with
										i
output coding	*1 i*	*2 sat*	*3 there*	*4 with*	*5 sally*	*6 we*	*5*	*5*	*3*	*7 two*

When a new word appears in the input we (a) output the code corresponding to the first unused position in the list and (b) follow this with characters in the word. A decoder can read the output and not only build the word list on the fly, but also use codes (i.e., the “5 5 3” sequence above) to retrieve a word already in the list. (Such a code represents the position of the word in the list before it is moved to the top.)

Codes vs. characters?

The output in the figure above is a mixture of codes and characters. For this assignment, however, we will make these assumptions:

- Only ASCII codes less than or equal to 127 can make up input words.
- ASCII codes greater than or equal to 128 will indicate output codes.
- An output code of *i* will be represented by ASCII character *128 + i*.

- Code numbering starts at 1 (i.e., the code 0 is never used) and the maximum code value will be 120. This means our scheme will only be able to encode files having at most 120 unique words (although those words may be repeated throughout the text).
- No input word will be greater than 20 characters in length.

As the C language permits us to treats character values as integers, we can easily perform arithmetic on ASCII codes.

Spaces and newlines?

The output in the figure is silent on what happens to spaces and newlines. For this assignment we make these two assumptions:

- Spaces between words within the same line are implied in the output coding. Therefore all inputs used for testing will only have single spaces between words.
- New lines (i.e., the '\n' character or ASCII 10 or 0x0a) appearing in the input will be written to the output without any encoding.

Magic numbers

Although your assignment solution is only meant to provide an MTF encoding for a test file (i.e., we will write a decoder in Assignment #2) we should somehow indicate that our output file is special. We will do this with a *magic number*. This is a byte sequence at the start of a file used to indicate that file's type. (The Unix command "file" utilizes magic numbers.) The four-byte sequence for our MTF magic number files is 0xba 0x5e 0xba 0x11.

Our actual MTF encoding

The words in our example (i.e., the figure above) are contained in */home/zastre/seng265/a1/tests/test00.txt* and the MTF encoding following our assumptions and descriptions at that same directory in *test00.mtf*. What follows is a representation of the contents of test00.mtf (using the Unix utility *hexdump*):

```
$ ./text2mtf tests/test00.txt

$ hexdump -C tests/test00.mtf
00000000  ba 5e ba 11 81 69 82 73  61 74 83 74 68 65 72 65  |.....i.sat.there|
00000010  84 77 69 74 68 85 73 61  6c 6c 79 86 77 65 85 85  |.with.sally.we..|
00000020  83 87 74 77 6f 0a                                 |..two.|
00000026
```

The 0x81 value appearing at byte 5 in the dump is the same as the decimal value 129, i.e., a code value of 1 to which was added 128. (One ASCII char equals one byte.) At byte 5 is the ASCII character for "i". The seventh byte is the code value for 2, following which are the ASCII codes for "s", "a" and "t". Then follows more MTF codes and ASCII chars with values less than 128. Our example has only one line and at byte 38 we see the newline character (ASCII 0x0a); other text files could have multiple lines, and their resulting encodings reflect this (i.e., 0x0a at several places in the *hexdump* output for that file's MTF encoding).

Exercises for this assignment

1. If you have not already done so, ensure your git project is checked out from the repository. Within the project create an "a1" subdirectory. Ensure all directories and program files you create are placed under git control. (You need not add the test directory to git control unless you wish to do so.) Test files are available on ELW B238 machines in the directory */home/zastre/seng265/a1/tests*.

2. Write your program. Amongst other tasks you will need to:
   - obtain a filename argument from the command line;
   - create a new filename based on the old file name (i.e., replace ".txt" ending the input filename with ".mtf" ending the output filename);
   - read text input from a file, line by line, and the words within those lines
   - write output to a file, char by char
   - store words in a statically-allocated string table

3. **Do not use "malloc", "calloc" or any of the dynamic memory functions.** For this assignment you can assume that the longest input line will have 80 characters, and no input file will have more than 100 lines. No word will be longer than 20 characters. There is no need for you to first read in the whole input file before generating an encoding (but you should probably perform the encoding input-line-by-input-line.)

4. Keep all of your code in one file for this assignment. In later assignments we will use separable compilation available in C.

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. (You may want to avoid test00 until you have significant functionality already completed.) **Refrain from writing the program all at once, and budget time to anticipate when things go wrong!** Use the Unix command *diff* to compare your output with what is expected.

6. For this assignment you can assume all test inputs will be well-formed (i.e., our teaching assistant 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.

What you must submit

- A single C source file named “text2mtf.c” within your git remote repository (in the “a1” subdirectory) containing a solution to Assignment #1.
- **No dynamic memory-allocation routines are to be used for Assignment #1.**

Evaluation

For this first assignment students will demonstrate their work to a member of the course’s teaching team. Instructions on demo-slot signup will be provided a few days before the due-date; each demo will require from 10 to 15 minutes.

Our grading scheme is relatively simple.

- “A” grade: An exceptional submission demonstrating creativity and initiative. “text2mtf” runs without any problems. The program is clearly written and uses functions appropriately (i.e., is well structured).
- “B” grade: A submission completing the requirements of the assignment. “text2mtf” runs without any problems. The program is clearly written.
- “C” grade: A submission completing most of the requirements of the assignment. “text2mtf” runs with some problems.
- “D” grade: A serious attempt at completing requirements for the assignment. “text2mtf” runs with quite a few problems.
- “F” grade: Either no submission given, or submission represents very little work.