Software Engineering 265
Software Development Methods
Spring 2019

Assignment 2

Due: Thursday, March 7, 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, specifically when use the `setSENG265` environment. 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 Python programming language (specifically Python 3) to write a second implementation of an file encoder to MTF, and a first implementation of a decoder from MTF into text.
- Use Subversion to manage changes in your source code and annotate the evolution of your solution with "messages" with committed changes.
- Test your code against the ten provided test cases.

This assignment: `coding.py`

For this assignment you are to use the same Move-To-Front encoding scheme as described in the first assignment but this time written in Python. The resulting code will look very different from your solution in C and this is okay. (This is a not-so-subtle hint you should not even consider porting your C solution directly in Python.)

You must also write a decoder for a Move-To-Front (MTF) file—that is, your solution will read through the contents of an MTF file in order to reconstruct the actual text file represented by the encoding. The solution should check to ensure the file given is indeed an MTF file (i.e., first four bytes correspond to the magic numbers for our MTF format), and exit if the magic numbers are missing.

The same test cases used in Assignment #1 will be used in this assignment. There are no extra test cases.

Both the encoder and decoder will be contained in the same Python script file. In order to determine what functionality is to be used, we will take advantage of Python statement obtaining to the name of the Unix command invoked for the script. If we are in the directory currently holding `coding`.py, then the following two commands (performed once):

```
$ ln -s coding.py text2mtf.py
$ ln -s coding.py mtf2text.py
```

will result two symbolic links (same as a "shortcut" in Windows or Mac) referring to the same file. Within `coding.py` we will have an "if" statement to examine the name of the command used to run the script. If the command is:

```
$ ./text2mtf.py tests/test02.txt
```

then Python will see that `coding.py` was used to invoke the script, and the code in `coding.py` will do one thing. If the command is:

```
$ ./mtf2text.py tests/test03.mtf
```

then Python will see that `mtf2text.py` was used to invoke the script, and the code in `coding.py` will do something different than above. (The appendix of this document describes a skeleton script you can use to begin writing `coding`.py, and it takes advantage of the use of symbolic links to `coding.py`.)

Exercises for this assignment

1. Within your git repository ensure you have a subdirectory named a2. Ensure all directories and program files you create are tracked by git. You need not add the directory containing test files to your project unless you wish to do so. Test files are available on the lab-machine filesystem in the directory and are the same as those used for the first assignment.

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, and vice versa);
   - read text input from a file, line by line, and the words within those lines
   - read text char by char from a file
   - write output to a file, char by char and word by word
   - store, retrieve and locate words in a list

3. Do not use classes or regular expression in the assignment.

4. 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 *cmp* to compare your MTF files and *diff* to compare text files.

5. 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.

6. Write two test scripts, i.e., Unix scripts that go through all of the ten provided test cases (one for encoding, one for decoding) and reports which test cases fail.

What you must submit

- A single Python script named `coding.py` within your subversion repository containing a solution to Assignment #2.
- Two Unix scripts (i.e, one for testing encoding functionality, one for testing decoding functionality).

Evaluation

Our grading scheme is relatively simple.

- "A" grade: An exceptional submission demonstrating creativity and initiative. The code within `coding.py` runs without any problems. Test scripts are provided. The program is clearly written and is structured in a way that also uses functions appropriate.
- "B" grade: A submission completing the requirements of the assignment. The code within `coding.py` runs without any problems. Test scripts are provided. The program is clearly written.
- "C" grade: A submission completing most of the requirements of the assignment. The code within `coding.py` runs with some problems. Test scripts might be missing.
- "D" grade: A serious attempt at completing requirements for the assignment. The code within `coding.py` runs with quite a few problems. At least one or two tests pass. Test scripts might be missing.
- "F" grade: Submission either represents very little work or cannot be executed for testing.

Appendix

Skeleton for `coding.py`

```
#!/usr/bin/env python3  # For use on ELW B238 machines
# -*- coding: utf-8 -*-

# See https://www.python.org/dev/peps/pep-0263/ about encoding

import os
import sys


def encode_main():
    print("In encode_main")


def decode_main():
    print("In decode_main")


command = os.path.basename(__file__)
if __name__ == "__main__" and command == "mtfencode.py":
    encode_main()
elif __name__ == "__main__" and command == "mtfdecode.py":
    decode_main()
```