Software Engineering 265
Software Development Methods
Summer 2017

Assignment 2

Due: Tuesday, June 27, 11:55 pm by "git push"
(Late submissions **not** accepted)

Programming environment

For this assignment you must ensure your work executes correctly on *linux.csc.uvic.ca*. You can use *git push* and *git pull* to move files back and forth between your account on the UVic CSC filesystem and your computer. Bugs in the kind of programming done this term tend to be platform specific, and so something that works perfectly on your own machine may end up either on *linux.csc* or behaving differently because of system defaults. The actual coding fault is very rarely that of *linux.csc*'s configuration. "It worked on my machine!" will be treated as the equivalent of "The dog ate my homework!"

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.) The URLs of significant code fragments you have found and used in your solution must be cited in comments just before where such code has been used.

Objectives of this assignment

- Understand a problem description, along with the role played by sample input and output for providing such a description.
- Use the Python programming language (specifically Python 3) to write *phase1.py*, which will be a second implementation of the forward and backward transform described in Assignment #1.
- Use *git* to track changes in your source code and annotate the evolution of your solution with "messages" provided during commits.
- Test your code against the twenty provided test cases.

This assignment: *phase1.py*

For this assignment you are to use the same forward- and backward-transform scheme as described in the first assignment but this time written in Python 3. The resulting code will look *very different* from your solution in C and this is okay. (Another way of stating this: do not even consider porting your C solution into Python as it simply won't work!)

We will extend the implementation slightly, however, in permitting *blocksize* to be specified at the command line. That is, for A#1 *blocksize* was fixed at 20, but in A#2 this can vary. The *blocksize* used to perform the forward transform for a PH1 file is always stored in the file itself (i.e., an integer written in bytes 5 to 8).

The test cases used in Assignment #1 (tests 01 through to 10) will be used for Assignment #2. As in A#1, these ten tests have *blocksize* of 20. Another ten tests have been added (tests 11 through to 20) to */home/zastre/seng265/tests*. In that directory is *README.md* which describes the *blocksize* to for the forward transform taking a text file to its PH1 file. For example, the forward-direction of test 18 will take the contents of *t18.t*xt and use a block size of 100 to create *t18.ph1*. Note that the *blocksize* need only be specified at the command-line for the forward-transform only, i.e., to go from a PH1 file back to a text file, the *blocksize* in the PH1 file must be used to perform the backwards transform.

The same format for magic number / *blocksize* information stored at the start of a PH1 file in A#1 will be used for A#2.

Running the program

The direction of the transform (--*forward* or --*backward*), plus the input (--*infile*) and output (--*outfile*) filenames, must be provided at the command line. Furthermore if the direction is forward, then the *blocksize* must also be specified (i.e., --*blocksize*) Here are some example invocations assuming *phase1.py* is available in the current directory. The backslash ("\") is used in *bash* to indicate a command continues on the next line.

```
% ./phase1.py --forward --infile ~zastre/seng265/tests/t18.txt \
    --outfile a.ph1  --blocksize 100
```

Here we check if the output is correct by comparing it with the expected output:

```
% diff a.ph1 ~zastre/seng265/tests/t18.ph1
```

And now we testing the program to see if it can correctly perform the backwards direction of the transform:

```
% ./phase1.py --backward --infile ~zastre/seng265/tests/t18.ph1 --outfile a.txt

% diff a.txt ~zastre/seng265/tests/t18.txt
```

In both uses of *diff*, I would know the files are identical if *diff* itself prints no messages.

Exercises for this assignment

1. Write your program *phase1.py* program in the *a2/* directory within your *git* repository.
   - Examine command-line arguments to determine what operations, and which file, the program will be processing.
   - Open a file for input.
   - Read text input from a file, block by block.
   - Implement the forward and backward direction of the transform as described earlier.
   - Open a file for output, write PH1 header info, write blocks/strings to the file, and then close the file.

2. Keep all of your code in one file for this assignment. In later assignments we will use the multiple modules in Python.

3. **Code restrictions: Use the script structure described in lectures (i.e., all code is defined in functions, with the script involving the *main()* function via an *if* statement). The only global variables permitted are those for constants (i.e., the magic number, the end-of-block symbol). Do not write your own Python classes.**

4. Use the test files in */home/zastre/seng265/tests* to guide your implementation effort. Start with simple cases (for example, the one described 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". There are ten pairs of test files.

5. 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 will specify error-handling as part of the assignment.

6. 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.

What you must submit

- A single Python script named *phase1.py* within your git repository containing a solution to Assignment #1. Ensure your work is **committed** to your local repository **and pushed** to the remote **before the due date/time**. (You may keep extra files used during development within the repository.)

Evaluation

Students will demonstrate their work to a member of the course's teaching team. Sign-up sheets for demos will be provided a few days before the due-date; each demo will require around 10 minutes.

Our grading scheme is relatively simple.

- "A" grade: An exceptional submission demonstrating creativity and initiative. *phase1.py* runs without any problems. All ten tests pass. The program is clearly written and uses functions appropriately (i.e., is well structured).
- "B" grade: A submission completing the requirements of the assignment. *phase1.py* runs without any problems; all ten tests pass. The program is clearly written.
- "C" grade: A submission completing most of the requirements of the assignment. *phase1.py* runs with some problems; some tests do not pass.
- "D" grade: A serious attempt at completing requirements for the assignment. *phase1.py* runs with quite a few problems; most tests do not pass.
- "F" grade: Either no submission given, or submission represents very little work.