`wc` final report for mbaxtvt2

Overview

The wc program is a work-alike reimplementation of GNU's wc tool and it meets the specifications of CW4 coursework. It takes an arbitrary number of file names and prints newline, word and byte counts for each one, while also displaying the total number of counts (if the input files where more than one). It can also take optional flags, in order to print specific counts (ex. Lines only) or provide a "help" and "version" message. The available flags – options are the following:

Flag (short)	Flag (long)	Description
-c	--bytes	Print the byte counts
-m	--chars	Print the character counts
-l	--lines	Print the byte newline counts
-L	--max-line-length	Print the length of the longest line
-w	--words	Print the word counts
N/A	--help	Display a help message and exit
N/A	--version	Display version information and exit

The program also provides an alternative way to specify input files, using a seperate file which contains the file names, terminated by NULL characters or the stdin if the seperate file's name is '-'. This option is enabled using the --files0-from flag.
Flags and file operands can be combined in any order with the exception of –files0-from, which cannot be combined with file operands.

Product: Design and implementation

Architecture:

'Wc' is structured using the four following functions. The MVC model is followed, using different functions to handle the Model (file counting), View (terminal output), Controller (argument handling)

def arg_parse(inp):

Input: A list which contains flags and file operands
Output: A list of boolean values which contains count options and a list which contains the file operands, or an error message if a flag was unknown.
Description: This function uses Python's argparse module to parse the input list and generate the count options depending on the flags present. It also returns the file operands in a seperate list.

def counter(f,is_stdin):

Input: A file descriptor or data from stdin input (f) and a boolean value which specifies if stdin mode is on.
Output: A collection of all the possble counts from the file contents or stdin input

Description: This function executes the counting part of the program. If is_stdin is True, then line count is reduced by one (to simulate real wc's behaviour), else it is returned as it is, accompained by all the other counts.

def arg_handling(inp):

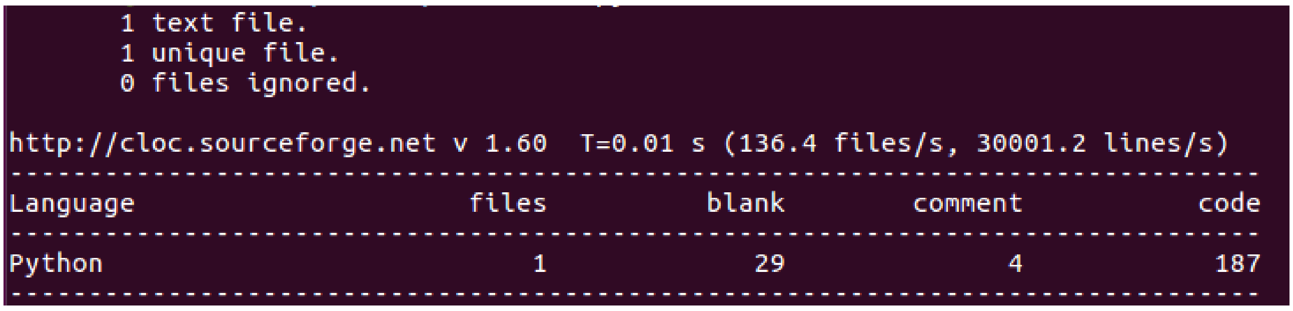
Input: The raw input from sys.argv.
Output: A list of boolean values which contains count options and a list which contains the file operands, or an error message if an error happened(for example flag was unknown).
Description: This function preprocesses the raw input from sys.argv and handles the –files0-from=F option of the program, getting all the file operands from the F file. Then, It calls arg_parse() to get the count options.

def print_counts(flags, files):

Input: A list of boolean values which contains count options and a list which contains the file operands. (output of arg_handling)
Output: A wc-like formatted string with the counts specified
Description: This function creates a formatted wc-like message, depending on the file operands and count options.

Source code metrics:

- Source lines of code (SLOC) are shown below. The tool "cloc" was used to get the results.



```
       1 text file.
       1 unique file.
       0 files ignored.

http://cloc.sourceforge.net v 1.60  T=0.01 s (136.4 files/s, 30001.2 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                           1             29              4            187
-------------------------------------------------------------------------------
```

- There are **four different units** / functions in the application, which are described above. The **longest** unit is the one that preprocesses the input (**arg_handling()** function), having **83 lines** of code.

Correctness testing:

A doctest with 27 test cases was used in order to test the correctness of the program (system level testing). Also, unit tests were executed to test the unit responsible for the counting (counter() function) for a wide variety of input files and the arguments, as well as to test the longest unit, arg_handling(). Finally, ad-hoc testing was executed to test the correctness of the program with input from the stdin (files0-from=- function or '-' file operands).

Performance:

The performance of the 'Wc' program is compared to the GNU's 'Wc' in the table below. Both programs were executed using one file as an input. The file has one line, 2680 words and its size is 34016 bytes:

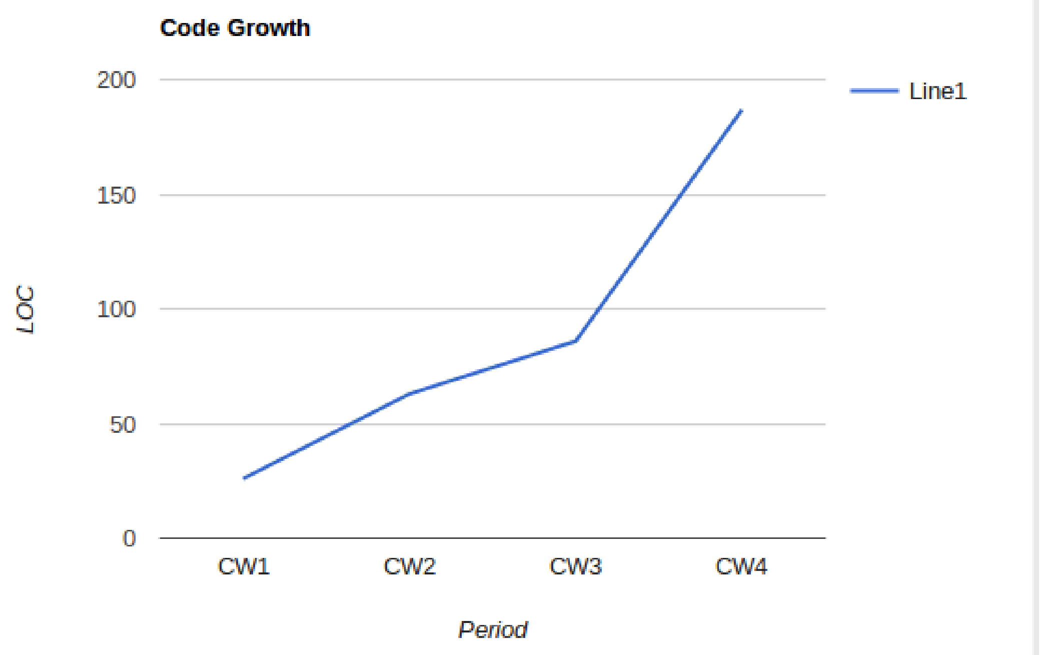
Time/Program	wc.py	GNU wc
Real time	0.062	0.006
User time	0.044	0.004
System time	0.016	0.000

It is clear that wc.py is much slower (by one order of magnitude) in terms of performance than the GNU version. wc.py is even slower when input files have multiple lines, because the counter() function reads the contents of each file, one line at a time, thus executing multiple accesses to the hard drive.

Process

Progress through the period:

In the following diagram, the code growth of the program though the period is visible:



It is clear that the growth is bigger when the functionality specifications of the program where extended (ex. CW3 to CW4). It should also be noted that the code grew during the process of argparse refactoring (CW2 to CW3), but in a small scale. New bugs were always found from CW2's period to CW4's.

Lessons Learned

Developing wc.py was a very good way to start learning Python. Having no previous experience in Python programming, I learned the core concepts quite quick and could write code easily and quickly. Also, I learned core Software Engineering concepts, like internal qualities and Unit testing, and how they are implemented in a real project (I was not familiar with testing methods). I learned to write code wth higher quality overall, and how to quicken the processes of refactoring and testing an application. Finally, reverse engineering a real application (or even a simple command line tool as GNU wc) is a very hard but rewarding experience and it improved by far my analytical and problem solving skills