wc final report for mbaxsjhc

Overview

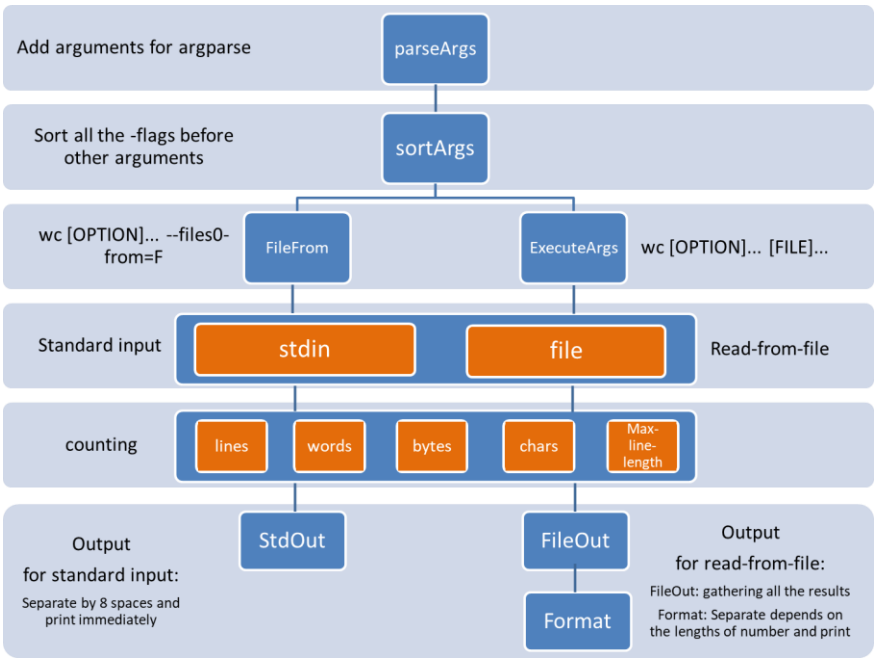
Basically, my wc.py met the majority of the requirements, such as support all the flags of GNU's wc, handle stdin, output the same format as wc and so on. However, the results might be different from wc in some situations, such as:

- the result of '-w', '-m', '-L' in some non-standard text (e.g. Chinese, pdf etc.) is not the same as wc, I will show some examples in "Correctness Testing".
- In "–files0-from=-" (standard input), I handled the output in a different way, for example:

wc	my wc.py
>> wc --files0-from=- >> testinputs/miniwc.py^@testinputs/a.txt^@ 39 109 1036 testinputs/miniwc.py 5 7 34 testinputs/a.txt Keep inputting and total all inputs when exit.	>> python3 wc.py --files0-from=- >> testinputs/miniwc.py^@testinputs/a.txt^@ 39 109 1036 testinputs/miniwc.py 5 7 34 testinputs/a.txt wc: : No such file or directory 44 116 1070 total Total above inputs and start a new total

Product: Design and implementation

- **Architecture**



- **Source code metrics**
  - **"Source Lines Of Code" (SLOC):** Lines:340 words: 1156 bytes: 12158
  - **Number of units (functions, classes, methods):** Classes: 0 Functions: 18 Methods: 3
  - **Stats on those (e.g., longest, cohesion etc.):**
    In my wc.py, some functions are well- cohesion, such as "getLinecount()", "getWordcount()" etc., each function contains only one -flag functionality; however, due to the time limitation, I couldn't fix all the problems of cohesion. For example:

- the longest function in my wc.py is “readFiles()”, which read each file from arguments and total each count and then making them ready to output. This breaks sequential cohesion and I should separate them into different functions.
- The function “output(args, linecount, wordcount, bytecount, charcount, maxline, filename)” breaks procedural cohesion; the parameters are “order dependency”, which is easier to cause problems; therefore, this function should be modified or break into different functions.

- **Correctness Testing**
  - **Unittest:**

    There are 67 tests in my unittest, including the different combinations of flags, a variety types of documents (e.g. doc, pdf) and different languages (e.g. Chinese, Arabic). The result is 60 success and 7 failed, all of the failures are related to the non-standard text, the following is one of the failure reports:

```
======================================================================
FAIL: test_char_with_pdf (__main__.TestChar)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "unittest_wc.py", line 267, in test_char_with_pdf
    self.assertEqual(self.char.count(content), 3094752)
AssertionError: 3093185 != 3094752
======================================================================
```

  - **Coverage:**

    The following is my coverage report:

```
Name            Stmts   Miss  Cover   Missing
---------------------------------------------------------
unittest_wc    352      0   100%
wc             243     43    82%   75-79, 127-131, 312-315, 335-339
---------------------------------------------------------
TOTAL          595     43    93%
```

    The majority of missing line for wc.py is “IsADirectoryError” and “FileNotFoundError” except, since I run coverage for my unittest_wc.py, the error shows as below:

```
======================================================================
ERROR: test_stdf_with_directory (__main__.TestStdF)
----------------------------------------------------------------------
Traceback (most recent call last):
  ……
    except IsADirectoryError:
NameError: global name 'IsADirectoryError' is not defined
======================================================================
```

- **Performance**

  I created the 15 test cases for my efficiency tester, the following shows the “part” of efficiency report and the summary:

Part of my efficiency report

	RealTime	CPUTime	RealTime	CPUTime
Testcase	wc		wc.py	
a.txt	0m0.006s	0m0.005s	0m0.07s	0m0.069s
arabic.txt	0m0.004s	0m0.004s	0m0.062s	0m0.062s
die.java	0m0.004s	0m0.003s	0m0.064s	0m0.063s

summary

RealTime

	Average	Min	Max	Median
wc	0m0.019s	0m0.003s	0m0.201s	0m0.005s
wc.py	0m0.075s	0m0.059s	0m0.186s	0m0.07s

htmlTest.html	0m0.003s	0m0.002s	0m0.063s	0m0.062s
inferno2.txt	0m0.012s	0m0.012s	0m0.078s	0m0.078s
miniwc.py	0m0.004s	0m0.004s	0m0.066s	0m0.065s
1.pdf	*0m0.201s*	*0m0.2s*	*0m0.186s*	*0m0.184s*
1.doc	0m0.008s	0m0.006s	0m0.064s	0m0.063s

CPUTime

	Average	Min	Max	Median
wc	0m0.018s	0m0.002s	0m0.2s	0m0.004s
wc.py	0m0.074s	0m0.059s	0m0.184s	0m0.065s

It is clear to see that the average Time of my wc.py is roughly four times longer than wc; however, the interesting point is that the time consumption of pdf is far more than the other test cases and my wc.py consumed slightly less than wc.

Process

- **Progress though the period**
  - **code growth table**

	miniwc.py	wc.py (cw2)	wc.py (cw3)	wc.py (cw4)
Number	25 lines	71 lines	109 lines	339 lines
Size	2KB	3KB	4KB	12KB
frequency	05 Oct	09 Oct	16 Oct	03 Nov

  - **bugs found and fixed (or managed)**

bugs/difficulties	solutions
Hard to handle arguments(flags)	Use argparse
After refactoring, error message for arguments cannot be customized	Override “print_help” method
In argparse, there is some difficulties for handling arguments. (e.g. "-l file -l file")	preprocess the input list to bubble all - flags to before any filename
“UnicodeDecodeError” error occurred when read a pdf file	Open file with binary mode

Lessons Learned

- **Major challenges or felicities**
  - **What worked, what didn’t**

    Through the coursework, I found that it is hard to fully mimic wc since it is quite complicated; different input arguments cause different results. For example, the format of output for “wc a.txt”(read file) is different from “wc -”(standard input). Thanks to the forum and online sources, I can figure out lots of it; however, I still confuse about how wc count the words and characters for non-standard text (e.g. Chinese, pdf etc.).
  - **Things you were most proud with**

    I had never used python and Linux before this class but now I can write the wc program and create some tests for it. Besides, I know how to refactor my code and make it more readable and maintainable. Compare my first version and last version of wc.py, it is clear to see that my programming skill improved a lot.
  - **Areas to work on**

Although my wc.py is better than before, there are still some aspects I should work on, such as refactoring and solving the problems of cohesion since I mentioned in “Source code metrics”. Both of them are very important for developing software and I realized how important they are during the process of building my wc.py; it is painful to make the big change for my original code since expending functionalities.

- **Personal development**
  - **How have your skills evolved?**

    For programming:

    In the first version of wc.py, I wrote every code in a function and then I tried to break them into more functions since expending functionalities. Until creating unittest, I started to make each function small and concrete. Recently, I focus on refactoring and cohesion in order to improve readability and maintainability.

    For Linux:

    I had never used Linux before; therefore, I learned some basic command, such as “startx”, “cd” etc. in the beginning and then I tried to figure out how “wc” works in Linux. Until now, I can edit files and manage my version control through the terminal.