University of Cape Town
Department of Computer Science

CSC3022F
Assignment 1 - Emulating UNIX wc

February 16, 2024

You are required to write a program that provides similar functionality to the Unix utility, `wc`. This application counts lines, words and characters and then displays these counts to standard output. You should write your application to read from standard input, `std::cin`, so that a command like the following will work

```
cat mytestfile.txt | ./myWC
```

assuming your application is called `myWC`. Unlike the regular `wc` utility you won't need to provide command line parameters: you always print out all the counts. Note that unlike wc, we do not count white space when counting characters.

If your text file contains

```
The rain in Spain
falls mainly on the plain.
```

the output will be (line count, word count, character count):

```
2 9 35
```

A second example input

```
Able was I ere I saw Elba.
```

the output will be

```
1 7 19
```

1 Core requirements

NOTE: this assignment is intended as a very simple introduction to the correct structure and layout of a C++ program. You should not define any classes or types (except for a `struct`*, noted in the Mastery Work section) and the focus is on simple I/O and containers. We also want you to use basic C++ constructs, so please pay careful attention to any listed*

constraints. Some C++ containers/algorithms can trivialize this assignment, so you may not use those. You will have many opportunities in the other assignments to explore more complex language features.

The requirements for this assignment are as follows:

Your program must be able to read in text input and produce the line, word and character counts, in that order, and print these to standard output. See the two examples provided above for context.

Note the following

- for the character count, both letters and digits are counted: A-Z, a-z, 0-9 - all other characters should be ignored
- a word is any combination of one of more letters and digits e.g. hello123 is a word
- white space separates words
- the counting process is case insensitive.

Specific constraints for this assignment:

- you must read from standard input
- only basic I/O can be used - no string parsing/tokeninzing/regular expressions
- you must implement you own counting
- you may not use any STL algorithms
- only the `vector<>` and `std::list<>` containers can be used — importantly `std::map` cannot be used.

NOTE: `getline()` can be used to read in an entire line of input into a string. You can use then read words from this string using an input stringstream - include the header file `sstream`. Consult the Notes, slides and the C++ online reference for more information on `getline()` and how to set up and use a stringstream.

Completing the above core work will enable you to score up to 90%. To achieve a higher mark, you should tackle the mastery work outlined below.

2 Mastery work

The remaining 10% will require extending the program as follows:

Extend your program to create and print a character frequency table — output format below.

Additional constraints

- you must a define and use a data type, `CharInfo` which will be an element in your frequency table

```
struct CharInfo {char character; long count; };
```

- note: a struct defines a simple data type where you can access and set members with the '.' operator:

```
CharInfo tabEntry;
tabEntry.character = 'a'; tabEntry.count = 0;
```

- you should not store or report unused character counts
- the character counts should be reported in ASCII order
- you may only use a `std::list<>` or `std::vector<>` to implement your table.

The output will be the original output, augmented by a list of the used character frequency counts. So for the first example above, the output becomes:

```
2 9 35 [a:5 e:2 f:1 h:2 i:3 l:4 m:1 n:4 o:1 p:2 r:1 s:2 t:1 y:1]
```

3 C++ Code structure

Please ensure that you always use your student number as **namespace** when defining methods, structs and classes in this course:

```
/**
*.h file:
*/
#ifndef MY_HDR_H
#define MY_HDR_H

//any includes here

namespace STUDENT_NO {
void myMethod(std::string name ...);
  ...
}
#endif
/**
*.cpp file:
*/
#include "MyHdr.h"

void STUDENT_NO::myMethod(std::string name ...){
  ...
}
```

Remember that you usually have a basic driver file, containing `main()` and other necessary functionality (such as an event loop) and a collection of class source files, along with appropriate header files for other cpp files. In this assignment you are not defining complex classe types, but you must still split your code into an implementation file (which has all your code to manage and report counts), a driver file (which contains `main()` and exits correct, and a header file (which contains any types defined, function prototypes etc).

Additional Notes

1. You may use need to a `std::vector<>` container to hold your data; this will require you to include the header file `vector`. The vector data is not sorted by default, so be sure to add data in the order your desire.
2. Basic console I/O uses `<<` and `>>` for output and input. Be sure to include the `iostream` header file.
3. Be aware that reading from the console requires that the end-of-file condition is correctly detected. This can lead to an extra read operation if you are not careful (look at the Notes/slides).

Please Note:

1. A working Makefile must be submitted. If the tutor cannot compile your program on nightmare.cs by typing make, you will only receive **50%** of your final mark.
2. You must use version control from the get-go. This means that there must be a .git folder alongside the code in your project folder. A **10%** penalty will apply should you fail to include a local repository in your submission.

   With regards to git usage, please note the following:

   -10% - usage of git is absent. This refers to both the absence of a git repo and undeniable evidence that the student used git as a last minute attempt to avoid being penalized.

   -5% - Commit messages are meaningless or lack descriptive clarity. eg: "First", "Second", "Histogram" and "fixed bug" are examples of bad commit messages. A student who is found to have violated this requirement for numerous commits will receive this penalty.

   -5% - frequency of commits. Git practices advocate for frequent commits that are small in scope. Students should ideally be committing their work after a single feature has been added, removed or modified. Tutors will look at the contents of each commit to determine whether this penalty is applicable. A student who commits seemingly unrelated work in large batches on two or more occasions will receive this penalty.

   Please note that all of the git related penalties are cumulative and are capped at -10% (ie: You may not receive more than -10% for git related penalties). The assignment brief has been updated to reflect this new information.

   We cannot provide a definitive number of commits that determine whether or not your git usage is appropriate. It is entirely solution dependent and needs to be

accessed on an individual level. All we are looking for is that a student has actually taken the time to think about what actually constitutes a feature in the context of their solution and applied git best practices accordingly.

3. You must provide a README file explaining what each file submitted does and how it fits into the program as a whole. The README file should **not** explain any theory that you have used. The README be used by the tutors if they encounter any problems.

4. Do **not** hand in any binary files. Do **not** add binaries (.o files and your executable) to your local repository.

5. Please ensure that your tarball works and is not corrupt (you can check this by trying to downloading your submission and extracting the contents of your tarball - make this a habit!). Corrupt or non-working tarballs will not be marked - **no exceptions.**

6. A 10% penalty per day will be incurred for all late submissions. No hand-ins will be accepted if later than 3 days.

7. **DO NOT COPY. All code submitted must be your own.** ***Copying is punishable by 0 and can cause a blotch on your academic record.*** **Scripts will be used to check that code submitted is unique.**