Homework 3

In this assignment, you will write a program similar to the UNIX/Linux *wc* utility. If you haven't used *wc*, you can check its introduction here (https://en.wikipedia.org/wiki/Wc_(Unix)). (10 pts)

Basically, you will run your program as this: **`java WC <filename>`**

Your program should count the number of lines and the number of times each word/character is used. A word can be either an identifier or a number.

- An identifier is defined as a letter followed by a sequence of letters or digits ('a'..'z', 'A'..'Z', or '0'..'9'). Identifiers are case insensitive ("AA00", "Aa00", "aA00", and "aa00" are the same). Identifiers are separated by non-letter and non-digit characters.
- A number is defined as a sequence of digits ('0'..'9') that are not in an identifier. Different sequences represent different numbers. For example, number "001" is different from number "1". Numbers are separated by identifiers or non-letter and non-digit characters.

It should first output **the number of lines, words, and characters**. After that, it should output **the five most used characters**, **the five most used numbers**, and **the five most used identifiers** as well as **the number of times these characters/numbers/identifiers** are used.

Since identifiers are case insensitive, the program only outputs identifiers with lower case letters. The characters, numbers and identifiers should be output in the descending order based on the number of times they are used. When two characters happen the same number of times, the character with a smaller Unicode value should be considered as being used more frequently. When two identifiers/numbers happen the same number of times, the identifier/number that occurs earlier in the file should be considered as being used more frequently. When printing characters, invisible ones should be output in Unicode value, except for tab '\t' and newline '\n'.

Please try to create **your own** data structures to keep the information, and **do not** directly use standard Java collections. Inefficient implementations are surely OK (no points will be docked), so that later you may appreciate the use of these collections more.