Assignment 6: Word, Number, and Character Usage Statistics

Objectives:

Practice selecting and making use of appropriate STL containers and algorithms to perform tasks.

Statement of Work:

Implement a program that collects the statistics of word, number, and character usage in a file (redirected as the standard input).

Requirements:

1. Write a program that will read input (from standard input) until end of input is reached ("end of file"), which will count the number of times each word, number, and character appears in the input.
    - A word is defined as a consecutive sequence one or more letters ('a'..'z' or 'A'..'Z').
    - Words are not case insensitive ("AA", "Aa", "aA", and "aa" are the same).
    - A number is defined as any consecutive sequence of digits ('0'..'9').
    - Note that both words and numbers can be of length of 1, that is, contain one letter or one digit, respectively.
    - Different sequences represent different numbers. For example, number "001" is different from number "1".
    - Words are separated by any non-letter characters.
    - Characters are all printable characters plus the space, tab `'\t'` , and new line `'\n'`.
    - Numbers are separated by any non-digit characters. We will not complicate this by having you distinguish between integers and floating point numbers with decimal places.

Output specifications:

- Your program should track the number of times each word, number, and character appears.
- The program should then output the ten most used characters, the ten most used numbers, and the ten most used words, along with the number of times each of these characters/numbers/words are used.
- Since words are case insensitive, the program should only output the words in lower case.
- The characters, numbers and words should be printed in descending order based on the number of times they are used.
- Breaking ties (for the "Top Tens"):
  - When two characters occur the same number of times, the character with the smaller ASCII value should be considered as being used more frequently.
  - When two words (or numbers) occur the same number of times, the word (or number) that occurs earlier in the input should be considered as being used more frequently.

2. An example executable code of the program is provided to you (see below). You should make the outputs of your program match this sample executable. When printing characters, use `'\t'` for tab and `'\n'` for newline. and `'space'` for a space or blank character. All other characters should be printed normally.
3. Write a makefile for your project that compiles an executable called proj6.x
4. Make use of any appropriate C++ STL containers and algorithms. You should also use C++ string class instead of default c-strings. Here are a few good reference links for the library lookups:
   - http://www.cplusplus.com/references/stl/
   - http://www.cplusplus.com/string/string/

Note that you should select whatever container(s) will make YOUR program's algorithms the most efficient in terms of growth rate (i.e. "Big-O complexity analysis").

5. In a file called `analysis.txt`, write up your analysis of the complexity analysis of the important algorithms and procedures in your program. Note that your analyses will be based on not only the code YOU write, but also on the STL containers you choose for managing your data. Your analyses need to include analysis of at least (but not limited to) each of these necessary tasks:

- Reading the input set
- Storing the characters / words / numbers in your chosen containers, and setting their tracking values
- Looking up the final tracking info on your character / word / number frequencies
- Deciding on (and accessing for output) your "Top Ten" most frequent list for each case
- Any other important algorithm/tasks you perform to complete the job

6. While not a program requirement for submission, it is *recommended* that you verify your analysis of your program elements by testing larger input sets and also by measuring the actual run time speed of those test runs. You can do this in a program easily by using the `ctime` library and capturing the returns from the `clock()` function before and after an algorithm, then subtract the two clock times to see the difference. Convert the number to seconds by dividing by the constant `CLOCKS_PER_SEC`. On linprog, you can look up more details at the manual page for clock (`man clock`).

Example executable, some test cases

Submission

Tar all of your source code, as well as your makefile and your analysis.txt file, into a single tar archive and submit online via blackboard, using the Assignment 6 link.

Your tar file should be named in this format, all lowercase:

lastname_firstname_p6.tar

Example: My tar file would be: **Gaitros_Davd_proj6.tar**

Verify your file has been submitted properly.

Note that in addition to the provided test cases, we will also test your program using additional test files. Your program must be able to pass all the test cases in order to obtain a full score for the corresponding components. Part of the grading will take your choice of STL containers and your complexity analyses into account, and testing will involve some larger sets of inputs.