COSC2430 HW1: Finding the most frequent word in a document

Instructor: Carlos Ordonez

1 Introduction

You will write a C++ program to find the most frequent word in a text file. You can use any algorithm and data structures that you prefer, as long as the results are correct. It is preferred, but not necessary, that your algorithm is as efficient as possible, both in time to complete as well as memory management.

The purpose of this homework is getting you acquainted with the C++ programming assignments and assess your C++ programming skills. Therefore, **you are not allowed to use the STL Library.**

2 Input

The input is one text file, with at most 500 different words, separated by spaces (repeated spaces should be considered as one space), where words are strings of letters where each line has a line feed (
n) in Unix format (be careful if you develop your program in Windows and you port it later to Unix). You can assume that words will be at most 30 characters long. You should skip numbers (they are not considered words).

Notice that words can be followed by punctuation signs and that some of them can be capitalized (upper case). This should not interfere with your results: a capitalized word is the same as a non-capitalized word for the purposes of this homework. Uppercase letters can be anywhere in the word, for ex.: ”Hungry”, ”hungry” and ”hUngry” should be counted as the same word. Punctuation and separators should be ignored.

3 Program input and output specification

The main program should be called ”topword”. Therefore, your .cpp file should be called topword.cpp.

The input is a plain text file with a sequence of words, where each word is a string of one or more letters. You can create text file with any editor like nano or vi.

Call syntax from the Unix command line:

```
topword inputfile=input.txt
```

Notice that the file name will not necessarily be the same every time. Therefore, your program will have to take that into account. You can use the Command Line Parser that is provided in the TAs' homepage.

The output should be sent to a fixed file “output.txt” in exactly this format (avoid extra spaces or other separators)

```
hungry:2
```

Notice that the output word should always be in lowercase, regardless of how the words appear in the text (i.e. convert to lower case).

Some input files may not have a unique frequent word. If there are two or more frequent words with exactly the same frequency you should display all those frequent words, preferably in alphabetical order,

one per line. Since this is more complicated to program there will be at most 1 test case like this (i.e. make sure your program works for the common case, only one frequent word).

3.1 Example

Example of input file (input.txt):

```
The Hungry, hungry caterpillar
Eric Johnson
123
X10=3.1416
Y=2.718281
hang
ABC XYZ
hungry  yes no
X10
```

Example of result file output.txt (fixed file name in this homework):

```
hungry:3
```

Notice X is a word, but X10 is not a word according to the specification.

4 Requirements

- Your program must compile and run with GNU C++. You can get started with other C++ compilers, but your program will be tested in Linux and GCC.
- Your source code must be individual. Copying or modifying source code from another person is forbidden. Any significant similarity between your source code and source code from another student (including past editions of the course) will be treated as cheating and reported to the Undergrad Director. Any program downloaded or copied from the Internet should be disclosed to avoid false alarms.
- Future homeworks will use similar logic for input/output and testing. Try to write your C++ classes and functions in a way that can be used and maintained in the future.
- **don't forget to comment your code**, especially with a short explanation at the top of each source code file or complicated logic (e.g. pointers, nested loops).
- Follow the call syntax exactly as specified (name of program, input/output parameters). Avoid using other names for the main program or the program parameters since this will make your program fail automated testing.
- Leave the source code in your home folder. Avoid leaving your last version in subfolders. Follow TAs' instructions.
- Test your program well with many files, especially corner cases like empty files, empty lines, many words per line, numbers, strange separators. Your program is not expected to handle every potential case in the input file, but should not fail if there are minor issues in the input file (e.g. the TAs may have a minor mistake).

- Resubmission or fixing code is not allowed after the deadline. Therefore, ask before and test well your program.
- Do not use the STL library. We will indicate in which homeworks and specific requirements STL can be used.
- Your program should write error messages to the screen, not to the output file. Keep in mind that not following the output format will make your program fail test cases.
- Your program should not crash, halt unexpectedly or produce unhandled exceptions. Consider empty input, zeroes and inconsistent information. Each exception will be -10.
- Test cases. Your program will be tested with 10 test cases, going from easy to difficult, where each test case is 10 points. You can assume 80% of test cases will be clean, valid input files.
- A program not finished by the deadline is zero points. A non-working program is worth 10 points. A program that does some computations correctly, but fails several test cases (especially the easy ones) is worth 50 points. Only programs that work correctly with most input files that produce correct results will get a score of 60 or higher. In general, correctness is more important than speed.