COSC6340:
HW0: Counting words with an aggregation query

1 Introduction

The goal is to compute word frequencies. You will write a C++ program to compute a GROUP BY query on a document with words. You can use merge sort or quicksort algorithms and data structures that you prefer, as long as the results are correct and you do not load the input file in RAM. It is preferred, but not necessary, that your algorithm is as efficient as possible, both in time to complete as well as memory management. **You are not allowed to use the STL Library.**

SQL motivation: Assuming you have a simple table with ONE column the goal is to compute a GROUP BY query on that table. You can think of having a hypothetical table and an aggregation query based on this table as shown below. You can try to create such table in any DBMS (e.g. SQL Server, MySQL) and run this aggregation query.

```
CREATE TABLE T(keyword char(20),primary key(keyword));

SELECT keyword,count(*)
FROM T
GROUP BY keyword;
```

2 Input

The input is a large text file, potentially with thousands of words. You cannot assume the file can be loaded in RAM. Words consist of letters only (ignore numbers!). You can assume words have up to 20 characters. You can assume the separator is only space and end of line. You should ignore separators and punctuation symbols. Words should be converted to uppercase.

Example of input and result file:

```
------------------------------------------------------------
Z
Abc
ABC
DE
Z
DEF  DE
abc
Z
z
------------------------------------------------------------
```

3 Program and input specification

The main program should be called ”groupby.cpp”. DO NOT CHANGE this cpp file name! The program has two parameters.

Call Syntax:

```
groupby input=whatever.txt;output=freq.txt
```

Note that the file name will not necessarily be the same every time, so your program will have to take that into account. You can use the Command Line Parser that is provided in the T.A.'s homepage. The output for the example above should be sent to an output file of the form

```
ABC 3
DE 2
DEF 1
Z 4
```

Please note that the output should always be in UPPER case, regardless of how the words appear in the text. One word per line, do not include any other information as this will hinder automated testing. The frequency should be separated by ONE space.

4 Requirements

- Homework is individual. Cheating is strongly discouraged. We use tools to detect cheating comparing source code (even detecting changing variable names or reformatting source code).
- Download the Borland C++ compiler. Links to the download page as well as instructions on its usage are on the TA homepage. It is recommended that you do a couple of short programs with the compiler in order to familiarize yourself with its usage.
- you should not load the input text file in RAM. YOu can read the text file multiple times. You can write any temporary files for processing.
- You have two choices for words: (1) assume all distinct words cannot fit in RAM; (2) assume the words can fit in RAM. YOu need to indicate your choice.
- It is preferred you solve the homework with a fast sorting algorithm (quicksort, merge sort), but you can use hashing. However, the output must be sorted anyway.
- You should indicate clearly if you downloaded code from any Internet source. If the TA detects you downloaded code your grade will be decreased 50% at least.
- You will submit a zip file including a README.TXT file and all your source code. DO NOT include executable or compiled code like DLLs. Detailed Instructions on TA web page.
- The README file should explain your basic programming approach. Also, include short comments on each source code file (no need to comment every class/function).
- Do not use the STL library.
- Your program should write error messages to the screen, not the output file. 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. You can assume 80% of test cases will be clean, valid input files. If your program fails an easy test case 10-20 points will be deducted. A medium difficulty test case is 10 points off. Difficult cases with specific input issues or complex algorithmic aspects are worth 5 points.
- A program not submitted 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 80 or higher. In general, correctness is more important than speed.