The **packaging and submission procedure** for this assignment is similar to that for the previous one, replacing `hw02` with `hw03`. As well, the **guidelines** on allowable use of external resources and other details remain unchanged. **It is important to use the class newsgroup for clarification and elaboration.**

The **main task** addressed by this assignment is that of *partitioning a set of words (strings) into maximal subsets of anagrams*, and presenting the results in lexicographic order (on strings, sets, and partitions). In more detail, two strings are said to be anagrams of one another if rearranging the characters in one yields the other (e.g., fire and rife). For this assignment, we will ignore character case when comparing characters. Thus Maine, anime, and amine are anagrams of each other despite the uppercase M in the first and the lowercase m in the others. The "is an anagram of" relation on strings is an *equivalence relation*[1] (reflexive, symmetric, and transitive) and may be used to partition a set of strings into a collection of disjoint subsets such that each string in the original set belongs to exactly one of the disjoint subsets.

The **input** to the program (provided to the program on its standard input) is a collection of strings using the UTF-8 encoding (as used in the previous assignment) in no predefined order, and potentially with duplicates, with one string per line. Thus the strings cannot contain a newline. However, non-newline white space (e.g., spaces and tabs) *is* significant. As well, the strings may contain punctuation (e.g., quotes and dashes of various kinds) and other assorted characters.

The desired **output** of the program (to be written by the program to its standard output) is the partition (collection of subsets) of the input as described earlier, written in sorted order of the subsets, as used in the previous assignment. (You should be able to reuse some of the code from that assignment, although it is not required to do so.) The words in each set are written one per line and there is a single blank line separating adjacent sets.

1. (50 pts.) Write a program that reads a set of strings on its standard input, one per line, as described above, and outputs a single integer to its standard output, followed by a newline. That integer should be the number of partitions when the input set of strings is partitioned as described above. Reminder: Use the proper names for your executables, as in the previous assignment. For instance, this one would be `hw03q01`.

2. (50 pts.) Write a program to that reads the same kind of input as the previous question, but that outputs (instead of the integer above) the first five (in lexicographic order) partitions formatted as described above. If there are fewer than five partitions of the input set, then the output is limited to those.

3. (50 pts.) Write a series of tests that may be used to check the correctness of code for the earlier questions. Using a combination of suitable comments and the associated report (below), document your tests and explain why they are both correct and interesting (i.e., test for likely errors, boundary conditions, tricky cases, etc.).

[1] cf. `https://en.wikipedia.org/wiki/Equivalence_relation`

4. (50 pts.) Write a brief report (two pages suggested length, five pages maximum length) documenting your program and, in particular, highlighting the parts that are interesting, parts that are or incomplete or buggy, aspects that were easy, difficult, etc. (Documented bugs will diminish their negative impact on the score.)

Reminders

- Start work early.
- Use the proper naming convention for all your files, including submitted package, source files, produced executables, etc.
- Verify that your submission is a proper tarred gzipped package.
- Ensure that your programs read from standard input and write to standard output and that they produce exactly the output described above, without any additional prompts or diagnostics. (Standard error may be used for those if desired.)
- Use the newsgroup for additional details, hints, etc.
- Test your submission on `aturing`.
- Pay attention to good testing. It should be very hard for an incorrect program (yours or another) to pass all the tests you provide.