CS 3370

Program 5 – Cross-Reference Generator

Write a *cross-reference generator*, which is a program that reads a text file, extracting word-like tokens (containing only letters, hyphens, and apostrophes; no numerals), and prints a listing of the line numbers where each token appears, along with the number of occurrences in that line in the format **line-number:count**. To illustrate, here is an excerpt from the output of this assignment (the order of identical keys ignoring case is immaterial; i.e., "A" can immediately precede *or* follow "a", etc.):

```
-                   : 79:1
-c                  : 130:1
A                   : 48:1
a                   : 9:1, 10:1, 12:2, 14:1, 17:2, 19:1, 26:1, 27:1, 28:2,
                    : 39:1, 41:1, 43:1, 45:2, 46:2, 49:1, 50:2, 51:1, 56:3,
                    : 81:1, 82:1, 94:1, 111:1, 112:1, 114:1, 117:1, 132:1, 135:1,
                    : 138:1, 142:2, 143:1, 144:1, 152:1, 156:1, 161:2, 163:1, 164:1,
                    : 167:1, 169:1, 175:1, 182:2, 190:1, 192:1
about               : 16:1, 29:1, 166:1, 190:1, 191:1
above               : 137:1
accompanied         : 6:1
across              : 26:1
admit               : 20:1
advancing           : 170:1
After               : 166:1
after               : 130:1
again               : 155:1
algorithm           : 166:1
all                 : 44:1, 104:1, 135:1
allocate            : 135:1
allocating          : 132:1
allocation          : 16:1
almost              : 30:1
alphanumeric        : 174:1
also                : 50:1, 79:1, 81:1, 151:1
altered             : 178:1
Although            : 92:1
altogether          : 139:1
an                  : 93:1, 137:1, 157:1, 165:1, 175:1, 178:1
And                 : 19:1, 31:3, 155:1, 178:1
and                 : 14:1, 18:1, 19:1, 29:1, 34:2, 38:1, 39:1, 40:1, 41:1,
                    : 43:1, 54:1, 55:1, 82:2, 83:1, 92:1, 94:1, 103:2, 109:1,
                    : 110:1, 130:1, 135:1, 138:1, 147:2, 165:1, 168:1, 189:2, 192:1
another             : 25:1, 181:1
…
byte-code           : 130:1
…
you'll              : 192:1
zero-based          : 78:1
```

Notice that the tokens are alphabetized *ignoring case* (but do *not* alter any key). The colons are lined up so that the longest word in the list, when printed, will leave one space before and after the colon. Duplicate line numbers are not printed. Instead, the number of times a word appears on the line is printed after a colon (e.g., the word "a" appears twice on line 28). If there are more than nine line numbers for a token, you wrap to the next line, as illustrated above. (Don't hard-code the 9, though; make it a defined constant.)

Use the file **Strings.txt**, which accompanies this assignment, as the input file. Write your main so that it reads the file name to process from the command-line (`argv[1]`). If no command line argument is present, then process standard input (`cin`). You should only make one pass through the file. Submit your output in a separate text file (zip it along with your source code).

As you might expect, this assignment is not hard at all if you use the proper containers and algorithms (~56 lines of code). Let the C++ library do the heavy lifting for you.

Assessment Rubric

Competency ↓	**Emerging →**	**Proficient →**	**Exemplary**
Efficiency		Use effective data structures	
Clean Code		No repeated code (refactor); No unnecessary code	Simplest possible logic to fulfill program requirements; intelligent use of a comparator function object to implement a strict weak order; use a regular expression and **regex_iterator** to extract words from each line
Other	Use range-based **for** when applicable	Proper use of command-line arguments as assigned	