Exercise 7: Concordances

October 16, 2015

Introduction

Computers play an important role in the analysis of texts. Concordances are such an example. A concordance is a list of all (or at least the most important) words in a text including their context, frequency, and so on, to allow careful study of these texts. Before the age of the computer, these concordances were produced manually, which is a time-consuming process, and were conducted only for important texts. In this assignment you design and implement a console program that reads a text file and stores all words in the order of appearance, after which the user can ask questions about the text such as word frequency, word occurrence, and word contexts.

Learning objectives

After doing this assignment you are able to:

- work with arrays;
- work with console I/O that handles user queries and displays information;
- work with text file I/O;
- work with regular expressions

Remember that *every* function you hand in should include preconditions (using `asserts`) and postconditions using JavaDoc.

Part 1a: Basic User Interaction

Design and implement a console program that allows the user to enter commands:

1. `enter` *filename* : the program attempts to open the text file with file name *filename* and reads and stores all words in order of appearance in an array. If successful, the number of words read is reported to the user. If the operation fails, then this information is told to the user as well. For reading words, use a simple version of a `Scanner`, as explained in the case study in the lecture.

2. `content` : the program displays all currently stored words in order of appearance.

3. `stop` : the program terminates.

Scenario:

- Structure your console program in the same way as explained in the lecture. Pay specific attention to the use of switch statements, and functions to handle the separate commands.

Part 1b: Reading words, more complex

The simple version of the Scanner can obtain strange character combinations. For instance, in the novel "Alices Adventures In Wonderland.txt", provided on Blackboard (the underlined parts should not belong to a word):

- "Wonderland,"
- "whatsoever."
- "do:"
- "think–"
- "(for,"
- "distance–but" (should be two words, viz. "distance" and "but")
- "Australia?"

Check for yourself which strange 'words' your program generates. It is a known hard problem to invent a solution that works for all conceivable texts. For this reason, we restrict ourselves to creating a regular expression as delimiter for our `Scanner` which solves the most obvious cases (like the ones mentioned above). Note that "waistcoat", "rabbit-hole", and "one-two-threes" are words, but "distance–but" has to be read as two words. Scenario:

- Start with setting a delimiter pattern that produces the default behaviour
- Step by step extend the pattern to include more delimiters, and check if your new pattern is really an improvement

Part 2. Word Occurrences

Extend the program created in part 1 with the following user commands:

4. `count` *word* : the program reports the number of occurrences of *word*. This number is shown to the user, together with the total number of words, and the percentage of occurrences of word with respect to the total number of words

Part 3. More challenges

Extend the program created in part 2 with the following user commands:

5. `where` *word* : the program displays all index-positions corresponding to occurrences of *word* in the text, followed by the total number of occurrences found.

5. `context` *m* *word* : the program displays all occurrences of *word* including *m* words immediately before and *m* words immediately after that occurrence. The output is terminated with the total number of occurrences found.

Note that in some cases it is not possible to display *m* words before and/or after the occurrence of the given word.