Software Engineering 265
Software Development Methods
Spring 2018

Assignment 1

Due: Thursday, February 8, 11:55 pm by "git push"
(Late submissions **not** accepted)

Programming environment

For this assignment you must ensure your work executes correctly on *linux.csc.uvic.ca*. You can use *git push* and *git pull* to move files back and forth between your account on the UVic CSC filesystem and your computer. Bugs in the kind of programming done this term tend to be platform specific, and so something that works perfectly on your own machine may end up crashing on *linux.csc*. The fault is very rarely that of *linux.csc*'s configuration. "It worked on my machine!" will be treated as the equivalent of "The dog ate my homework!"

Individual work

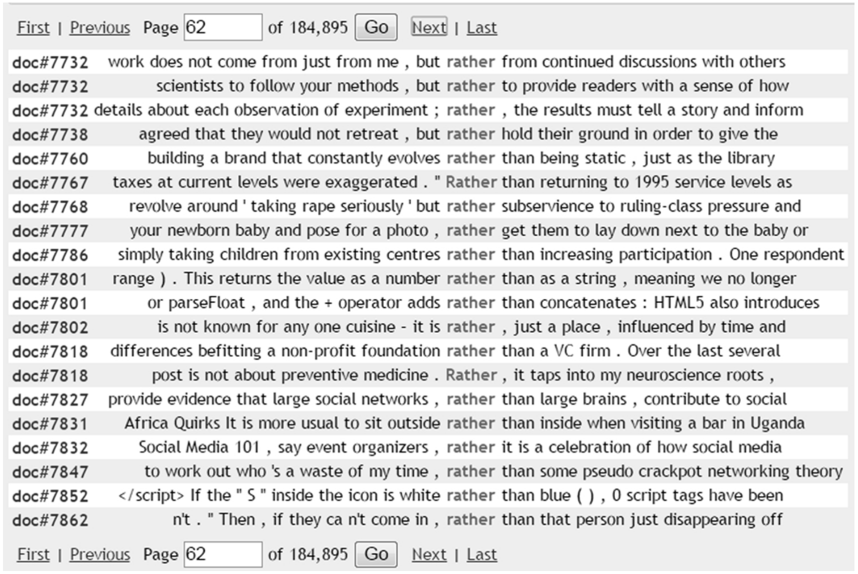
This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussion is encouraged. **However, sharing of code fragments is strictly forbidden without the express written permission of the course instructor (Zastre).** If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact the course instructor as soon as possible. (Code-similarity analysis tools will be used to examine submitted work.) The URLs of significant code fragments you have found online and used in your solution must be cited in comments just before where such code has been used.

Objectives of this assignment

- Understand a problem description, along with the role played by sample input and output for providing such a description.
- Use the C programming language to write the first phase of a "keyword-in-context" utility. The program will be named (rather unimaginatively) *kwic1.c*.
- Use *git* to track changes in your source code and annotate the evolution of your solution with "messages" provided during commits.
- Test your code against the ten provided test cases.

This assignment: *kwic1.c*

Throughout this term the assignments will focus on a particular kind of text indexing known as “keyword in context” (or KWIC). This is an indexing style used in some books and websites. The idea is that reading or seeing a word *that has been indexed in its original context* will aid the reader to decide if the reference is indeed what they want or need. Sometimes this kind of index is used as construct a *concordance*. Shown below is a snapshot of concordance prepared using KWIC (taken from http://bit.ly/2DBK3vM).



doc#7732	work does not come from just from me , but	rather	from continued discussions with others
doc#7732	scientists to follow your methods , but	rather	to provide readers with a sense of how
doc#7732	details about each observation of experiment ;	rather	, the results must tell a story and inform
doc#7738	agreed that they would not retreat , but	rather	hold their ground in order to give the
doc#7760	building a brand that constantly evolves	rather	than being static , just as the library
doc#7767	taxes at current levels were exaggerated . "	Rather	than returning to 1995 service levels as
doc#7768	revolve around ' taking rape seriously ' but	rather	subservience to ruling-class pressure and
doc#7777	your newborn baby and pose for a photo ,	rather	get them to lay down next to the baby or
doc#7786	simply taking children from existing centres	rather	than increasing participation . One respondent
doc#7801	range) . This returns the value as a number	rather	than as a string , meaning we no longer
doc#7801	or parseFloat , and the + operator adds	rather	than concatenates : HTML5 also introduces
doc#7802	is not known for any one cuisine - it is	rather	, just a place , influenced by time and
doc#7818	differences befitting a non-profit foundation	rather	than a VC firm . Over the last several
doc#7818	post is not about preventive medicine .	Rather	, it taps into my neuroscience roots ,
doc#7827	provide evidence that large social networks ,	rather	than large brains , contribute to social
doc#7831	Africa Quirks It is more usual to sit outside	rather	than inside when visiting a bar in Uganda
doc#7832	Social Media 101 , say event organizers ,	rather	it is a celebration of how social media
doc#7847	to work out who 's a waste of my time ,	rather	than some pseudo crackpot networking theory
doc#7852	</script> If the " S " inside the icon is white	rather	than blue ( ) , 0 script tags have been
doc#7862	n't . " Then , if they ca n't come in ,	rather	than that person just disappearing off

In the image the occurrences of the word “rather” are shown in the context of sentences in which that word appears: at the left of each line is a document number indicating where the line’s sentence can be found. Rather than a document number, the reference could be a page number, or a line number, or some other kind of reference.

In this first assignment we will begin to write a utility which can build a KWIC index. However, to keep coding simple, in this first assignment we focus on content that should appear in lines but without the left-hand column reference items. (We will add this latter piece in later assignments.)

Building such an index requires, amongst other things, knowing which words must *not* be indexed. Definite and indefinite articles (*the*, *that*, *a*, *an*), conjunctions or modifiers (*and*, *but*, *or*, *not*), even prepositions (*on*, *at*, *under*, *between*, etc.) are examples of what need not be indexed.

In essence, this first assignment will require to transform some input text into associated KWIC output text. We will, however, include as part of the input those words to be excluded from the index listed. Consider the next page displaying an input file and the expected *kwic1* output for that file. (These correspond to *in07.txt* and *out07.txt* from the test files provided to you for this assignment.)

```
1
::
of
and
the
too
on
who
to
that
::
that fortune
sense and sensibility
life of robert browning
the man who knew too much
legend of montrose
visit to iceland
orthodoxy
the mountains
on the track
ward of king canute
```

```
life of robert BROWNING
ward of king CANUTE
that FORTUNE
visit to ICELAND
ward of KING canute
the man who KNEW too much
LEGEND of montrose
LIFE of robert browning
the MAN who knew too much
legend of MONTROSE
the MOUNTAINS
the man who knew too MUCH
ORTHODOXY
life of ROBERT browning
SENSE and sensibility
sense and SENSIBILITY
on the TRACK
VISIT to iceland
WARD of king canute
```

The input file has a version number (for this assignment the version is 1), followed by a line with "::". There then follow the list of exclusion words, one word per line, which list it itself ended with another "::". Lastly the remainder of the file is made up of the lines-for-indexing of which the KWIC index will constructed for that file.

Each line in the output contains text based on one line-for-indexing of the input with exactly one word capitalized. When reading the output line-by-line (i.e., from top to the bottom) the capitalized words are in alphabetical order. Some lines-for-indexing *appear more than once in the output* as they have more than one indexed word. And as mentioned earlier in the assignment, the problem for A#1 has been simplified in that no line-number references appear in the output.

Your task it to write *kwic1.c* which constructs such output from such input.

Running the program

Your program will be run from the Unix command line. Input is expected from *stdin*, and output is expected at *stdout*. ***You must not provide filenames to the program, nor hardcode input and output file names.***

For example, assuming your current directory contains your executable version of *kwic1.c*, (i.e., named *kwic*), and a *tests/* directory containing the assignment's test files is also in the current directory, then the command to transform the previous page's input into required output will be:

```
% cat tests/in07.txt | ./kwic1
```

In the command above, output will appear on the console. You may want to capture the output to a temporary file, and then compare it with the expected output:

```
% cat tests/in07.txt | ./kwic1 > temp.txt
% diff tests/out07.txt temp.txt
```

The same thing (i.e., producing output and comparing it with the expected output) can be combined into a one-liner:

```
% cat tests/in07.txt | ./kwic1 | diff tests/out07.txt -
```

The ending hyphen/dash informs *diff* that it must compare the contents of *tests/out07.txt* with the input piped into *diff's* stdin.

Exercises for this assignment

1. Write your program *kwic1.c* program in the *a1* directory within your git repository.
   - Read *stdin* line by line and processing each line appropriately.
   - Read and store exclusion words.
   - Read and store lines-for-indexing
   - Create the output index by using
   - Create and use arrays in a non-trivial array.
   - Use the "-std=c99" flag when compiling to ensure your code meets the ISO/IEC 9899:1999 standard for C.

2. **Do not use *malloc*, *calloc* or any of the dynamic memory functions. Do not use regular expressions. AND DO NOT HARDCODE FILENAMES INTO YOUR PROGRAM! SORRY FOR SHOUTING!** See the "Simplifying Assumptions" section below for more supportive and positive guidance to help you with your implementation.

3. Keep all of your code in one file for this assignment. In later assignments we will use the separable compilation features of C.

4. Use the test files in */home/zastre/seng265/tests* (i.e., on the CSC filesystem) guide your implementation effort. Start with simple cases (for example, the one described in this writeup). In general, lower-numbered tests are simpler than higher-numbered tests. Refrain from writing the program all at once, and budget time to anticipate for when "things go wrong". There are ten pairs of test files.

5. For this assignment you can assume all test inputs will be well-formed (i.e., the teaching team will not test your submission for handling of errors in the input). Later assignments will specify error-handling as part of the assignment.

6. Use *git add* and *git commit* appropriately. While you are not required to use *git push* during the development of your program, you **must** use *git push* in order to submit your assignment.

What you must submit

- A *single* C source file named *kwic1.c* within your git repository containing a solution to Assignment #1. Ensure your work is **committed** to your local repository **and pushed** to the remote **before the due date/time**. (You may keep extra files used during development within the repository.)
- No dynamic memory-allocation routines are permitted for Assignment #1 (i.e., do not use anything in the *malloc* family of functions).

Simplifying Assumptions for Assignment #1

a) All input files will be KWIC version 1 files.
b) At most 20 exclusion words will appear in an input file, and each word will be no more than 20 characters in length. It is possible that there are no exclusion words. Ignore any duplication in exclusion words.
c) At most 100 lines-for-indexing will appear in an input file, and each line is less than 70 characters in length.

d) Amongst all lines-for-indexing, there will be at most 100 distinct indexed words (i.e., if you count capitalized words an eliminate duplicates, there will be at most 100 such words).
e) Indexed words will appear at most once in any line-for-indexing.
f) All lines-for-indexing will be in lower case; words are separated by single spaces; and there is no punctuation.
g) Decimal numbers are treated as indexed words; use lexicographic (i.e., alphabetical) ordering for determining where line-for-indexing containing such a word should appear in the output.
h) If an indexed word appears in more than one line-for-indexing, print the lines in the order they appear in the input.

Evaluation

Our grading scheme is relatively simple and is out of 10 points. We may award grades within the categories below.

- 10/10: *kwic1* runs without problems and compiles without warnings. All ten tests pass. The program is clearly written and uses functions appropriately (i.e., is well structured).
- 8/10: *kwic1* runs without any problems and compiles with any warnings. All ten tests pass.
- 7/10: A submission completing most of the requirements of the assignment. *kwic1* runs with some problems; some tests do not pass.
- 5/10: A serious attempt at completing requirements for the assignment. *kwic1* runs with quite a few problems; most tests do not pass.
- 4/10 or lower: Either no submission given, or submission represents very little work.

Through the term we will also request each student demonstrate at least one assignment to the teaching team. More about this process will be described closer to the due date of this assignment. (A#1 demos will take place the week after reading-break week.)