Harvey Mudd College
Computer Science 60
Spring 2010

Assignment 2
KWIC Index API
Due. 11:59 p.m., Wed. 15 Sept. 2010

You may pair-program for this assignment. If you intend to do so, please send me the name of your partner by Friday. If you don't have a partner, I'll try to help you find one, so let me know early.

The problems in this assignment are to be done in a purely functional style using the Racket language, except for input/output, which I will provide. The input procedure will return the titles as a list of strings. The output procedure will display the index that you create, in the manner shown. Submit a single file named a02.rkt. As always, document your functions clearly. **Be sure to spell each function name in the API exactly as given**, otherwise the automatic grading script might not give you credit for the function.

A "kwic" index (**K**ey**W**ord-**I**n-**C**ontext index) is an index that alphabetizes words in a list of sentences or titles, so that titles can be looked up based on the occurrence of individual words in them. A given title may appear multiple times in the index, once for each non-noise word in the title. Accompanying each entry in the index is a citation number that enables the user to find more information about the title.

Then a kwic index for a list of titles, might appear as follows (the italic words are not part of the output):

```
                                 gutter                  reference number
                                   | |                                  |
   My opinions may have changed, but not the fact that I am rig : 6
   want less corruption, or more chance to participate in it.   : 5
            My opinions may have changed, but not the fact that : 6
              I either want less corruption, or more chance to  : 5
                       No good deed goes unpunished.            : 4
( … abridged for brevity … )
                No good deed goes unpunished.                   : 4
   one's principles than to live up to them.                    : 0
  le are the ones you don't know very well.                     : 1
                      I either want less corruption, or more    : 5
  e the ones you don't know very well.                          : 1
  nly normal people are the ones you don't know very well.      : 1
                                   | |
reference number              title
0 : It is easier to fight for one's principles than to live up to them.
1 : The only normal people are the ones you don't know very well.
2 : No good deed goes unpunished.
3 : I either want less corruption, or more chance to participate in it.
4 : My opinions may have changed, but not the fact that I am right.
```

Notice the whitespace *gutter* down the middle of the index. The words to the right of the gutter are those on which alphabetization has occurred. Surrounding the word on either

side is the *context*, the rest of the title, up to the amount of space provided. On the rightmost end, after the colon is the reference number, which in this case is just the position of the title in the original list, counting from 0. The index is followed by the titles with the reference number preceding them. The reference numbers in our case are generated by the program. The index above was produced from the following list of aphorisms one per line:

```
It is easier to fight for one's principles than to live up to them.
The only normal people are the ones you don't know very well.
No good deed goes unpunished.
I either want less corruption, or more chance to participate in it.
My opinions may have changed, but not the fact that I am right.
```

A required argument to the index-producing function is a list of ***noise words***, words on which we do not want to index. Case should be ignored when comparing input words to these. In the current example, the list of noise words was specified as the following list of symbols:

```
'(a am an and are by for if in is of the those to)
```

Requirements:

Two functions are required to be submitted:

a. [75 points] Function **`(kwic Noise Titles)`** returns a raw list of triples, each containing:

- a single string of words from the *right* of the keyword onward
- a single string of words to the *left* of the keyword
- the reference number for the title

(Note that we swap the position of left and right to make visual debugging simpler, because sorting occurs on the words to the right rather than the left. The proper order will be restored by the **`format`** function below.) For purposes of this assignment, space in the titles is the only delimiter of words, not punctuation marks. Any multiple spaces between words are the same as a single space. Any punctuation marks are to be considered part of the word to which the punctuation is connected.

b. [25 points] Function **`(format Left Right Triples)`** formats the output of function **`kwic`** into a list of single strings, each of which forms a line of printable index. The variables **`Left`** and **`Right`** indicates the number of characters on the left and right of the gutter, respectively. **`Triples`** is a list of triples (lists of three elements), such as produced by **`kwic`**. Function **`format`** does not put in newline characters explicitly. That is left to the ultimate print procedure, about which we do not worry in this assignment.

One reason for separating out **`kwic`** as a function is that we might make further use of the index contents in list form rather than string form, for example in developing a more comprehensive application. The **`kwic`** and **`format`** functions give us an "API"

(Application Programming Interface) for the problem. They should not contain any external input/output, as that is provided separately. For example, they should not do any reading or printing (except for debugging).

Racket built-in functions that could be useful for this assignment

- `map`
- `foldl`
- `sort` (second argument is element-comparison function)
- `reverse`
- `append`
- `string-append`
- `string-length`
- `string-ci<?` (compares strings case-insensitively for <)
- `string-ci=?` (compares strings case-insensitively for =)
- `string->list`
- `substring`
- `list->string`
- `number->string` (to enable numbers to be concatenated with strings)
- `lambda` (a form that creates a function)

Concepts that might be useful for this assignment

- functions as arguments
- anonymous functions
- recursion, accumulator arguments

Design and Development Hints

1. Think about how you are going to decompose the problem before starting to code. A suggested decomposition is given here, but it is not the only one possible.

2. Structure your functions as layered applications of simpler functions that do specific things.

3. Test the simpler functions independently. This makes it much easier to find errors than figuring out what went wrong in the final composition.

4. Test built-in functions when you use them the first time, to make sure that they do what you think they do.

5. **One possible decomposition** for the **`kwic`** function is:

    a. For each title string, create a list of words from the string and attach a number indicating the ordinal position in the list (counting from 0).

    b. Form all possible splits of the numbered titles into two lists of words (words before and after the gutter), removing blanks originally between words. Retain

the number along with each split. This results in a list of triples (after-gutter, before-gutter, number) from each title.

c. Filter out triples that begin with noise words.

d. Sort the remaining triples on the list of words before the gutter.

e. Convert the lists of words back into strings to form the final triples.

6. To help you understand this possible decomposition, I give below a data-flow diagram for steps a-d.