KWIC Refactoring Drill

Dataflow Patterns: 1.1.

Question

There are two pieces of code that compute the word associated with a circular shift. Find them.

Answer

First (in csword): `shift_idx = (fwno + wordno) % len(lines[lno])` Second (in csline): `return [lines[lno][(i+fwno) % wrd_cnt] for i in range(wrd_cnt)]`

Dataflow Patterns: 1.2.

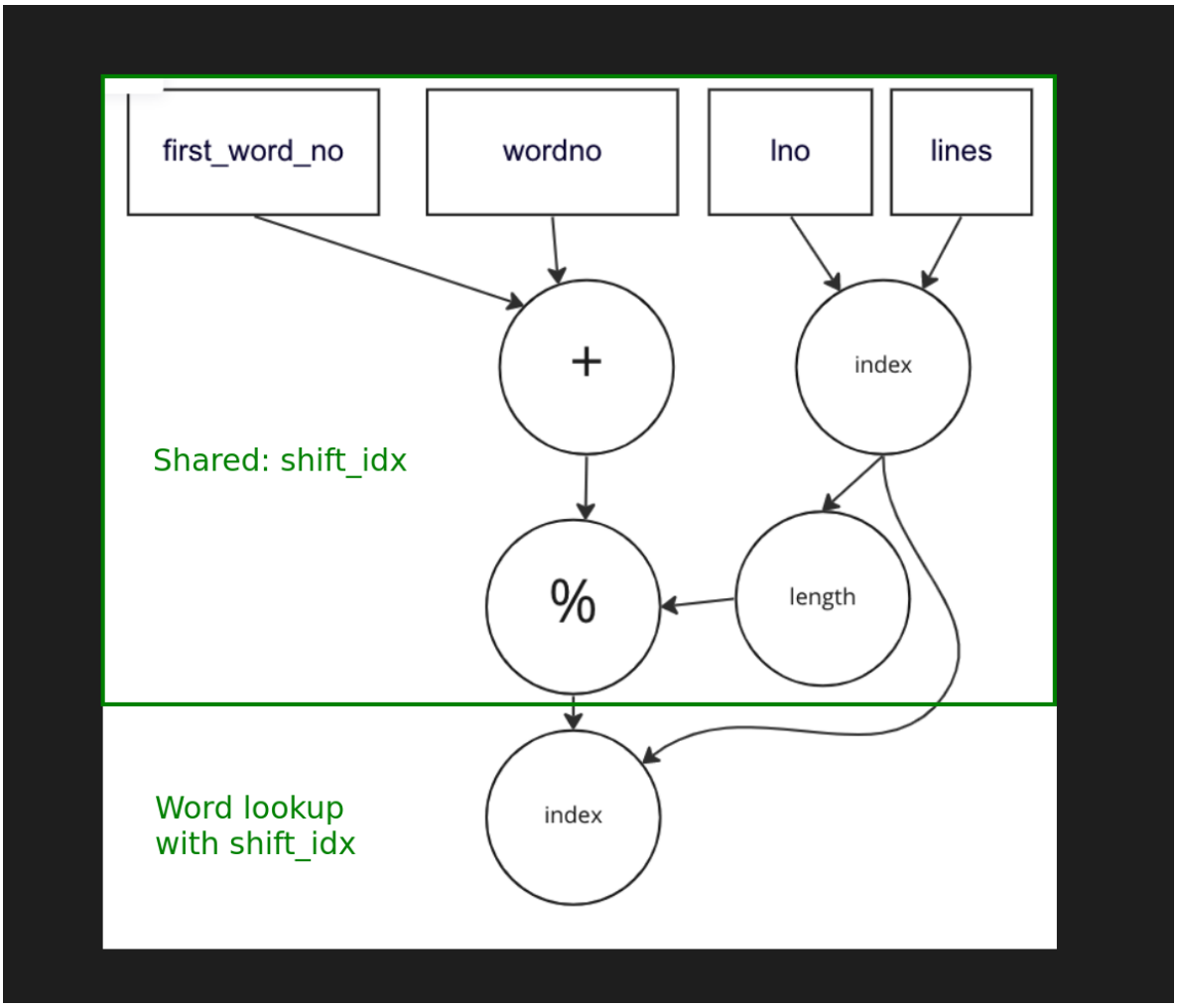
Question

These two snippets of code have the same dataflow graph. What is it?

Answer

`(fwno + i) % len(lines[x])`

The previous student Björn Carlsson illustrated this.



Dataflow Patterns: 1.3.

Question

How would you go about finding code in other programs that follows the circular-shift pattern?

Answer

Scan for modulus operators; see if they fit the (const+i)%len pattern.

Rationale: First, the modulus operator (%) has a funny shape and stands out. Second, it is near universal when this pattern occurs (in information-retrieval parlance, high term frequency) and very uncommon in other code (in information-retrieval parlance, high inverse document frequency). Multiplying the term frequency and inverse document frequency together gives a number called the tf-idf; since the tf-idf is high, it is a good thing to search for.

Data-Centric Refactoring: 1.1.

Question

For each module of the code: What secret is it hiding? That is, what design decisions are contained in that module, where changing the design decision could not change any module.

Answer

The short answer is ”nothing interesting” for all modules except for the sorting algorithm. Exact answers may vary depending on what non-interesting secrets you notice and find mention-able. It also varies depending on exactly how you imagine the spec, and what restrictions you believe all client code should follow.

Input module: None Circular shifter: Iteration order (but not output order) Alphabetizing module: Sorting algorithm, sort order (if rest of code is written in restricted fashion) Output module: Whether the output is buffered. The output destination (possibly). Master control: The input source.

Data-Centric Refactoring: 1.2.

Question

Consider each of the following design changes. What code would need to change for each?

- Use of persistent storage (not in-memory) for the line storage
- Using on-demand instead of up-front alphabetization. (I.e.: Using a selection algorithm instead of a sorting algorithm.)
- Storing shifts in byte-packed arrays

Answer

- Use of persistent storage (not in-memory) for the line storage
    - Input, CS, alphabetize, and output modules.
- Using on-demand instead of up-front alphabetization. (I.e.: Using a selection algorithm instead of a sorting algorithm.)
    - Alphabetize, output, and master control modules
- Storing shifts in byte-packed arrays
    - CS, alphabetize, and output modules.

Data-Centric Refactoring: 1.3.

Question

Separate each of lines_storage, circ_index, alph_index into their own ”module,” with an abstracted accessor interface. Each should not be accessed from outside that module.

Answer

Here’s one possible design. Following the original, it keeps everything in the top level rather than localizing state with objects.

Input module:

```
def initialize(lines)
def get_line(idx)
def line_count()
```

Circ shifts:

Directly depends on Input module as data source

```
def get_word(shift_idx, word_idx)
def word_count(shift_idx)
def shift_counts()
```

Alph indexes:

```
def alph_idx(alpha_idx)
def alph_count()
```

Data-Centric Refactoring: 1.4.

Question

Refactor the code so that (1) the storage format of the input, and (2) the format of circular shifts are both now secrets. In particular, there should now be only one instance of the pattern you identified in question 2 of the previous exercise.

Answer

Actually, the above modularization already suffices.

Data-Centric Refactoring: 1.5.

Question

Repeat question 1 for this refactored version.

Answer

Input module: Hides line representation Circ shifts: Hides shift representation, when shifts are computed Alph indexes: Hides sorting algorithm and order, hides when alphabetization is performed Output: Still hides buffering behavior and output destination/format Master control: Hides input source

Data-Centric Refactoring: 1.6.

Question

Repeat question 2 for this refactored version.

Answer

- Use of persistent storage (not in-memory) for the line storage
  - Just line storage
- Using on-demand instead of up-front alphabetization
  - Just alphabetization
- Storing shifts in byte-packed arrays
  - Just the circular shifter