INF 212 Analysis of Programming Languages
Project 7 – Decomposition and Modularity
Due date: 5/27
Demo date: 6/4

Goal: Understand decomposition options and how languages support/encourage (or not) those decompositions.

For this homework, you are going to write a simple program from scratch. In fact, not just one but four. The function of these four programs is exactly the same: they implement a KeyWords In Context (KWIC) indexing scheme. The input and the output should be exactly the same in all cases. The internal decompositions, however, should be completely different.

You can implement this homework in whatever language you want. If you want, you can implement the different programs in different languages.

Brief description of KWIC indexing

For information about this kind of indexing, see the Wikipedia page, also known as "ptx" (permuted index) Unix command. For the purposes of this homework, we're going to make some simplifications. Here is an input/output example. Given an input file with these lines:

White tigers live mostly in India.
Wild lions live mostly in Africa.

The output of your program should be this:

Africa
in Africa
in India
India
lions live mostly in Africa
live mostly in Africa
live mostly in India
mostly in Africa
mostly in India
tigers live mostly in India
White tigers live mostly in India
Wild lions live mostly in Africa

In other words, KWIC (in this simplified version) outputs a listing of all word shifts, of all lines, in alphabetical order.

For the purposes of this homework you can make all sorts of simplifying assumptions regarding the words -- normalize to lower case, or not, assume alphanumeric characters, etc. None of those details matter for the topic at hand.

Decomposition of the problem

Independent of the decompositions, we can model the KWIC indexing problem, generally, as having the following subproblems:

Reading the file
Shifting the lines
Sorting
Displaying the result

Task: Implement KWIC using 4 different decompositions

Program 1: Shared data decomposition

Your first program should use an imperative, shared data style, similar to what you might do in a low-level programming language. Read the file and hold the lines in a flat array in memory (don't lose the EOL character). Construct another array containing the indeces of the beginnings of the words. Construct yet another array (or change the previous) containing indeces of words, but sorted. Display the output.

Program 2: Functional decomposition

Your second program should use functional programming style, i.e. no state should ever be held beyond the scope of functions, and state is immutable. Decompose the problem into pure functional steps (verbs) that take inputs and produce outputs that are then consumed by the next function.

Program 3: OOP decomposition

Your third program should use object-oriented style, i.e. identify the main nouns of the problem, identify their operations, and decompose the problem around those objects.

Program 4: Event-based decomposition

Your fourth program should use an event-based, pub/sub style, i.e. identify the agents involved, the events that should be raised, and decompose the problem around those.

The discussion during the demo will revolve around your understanding of the different styles, the strengths and weaknesses of these styles, the information that each style exposes/hides, how the language(s) you chose encourages or discourages the styles, and how possible requirements changes will affect each of the styles.