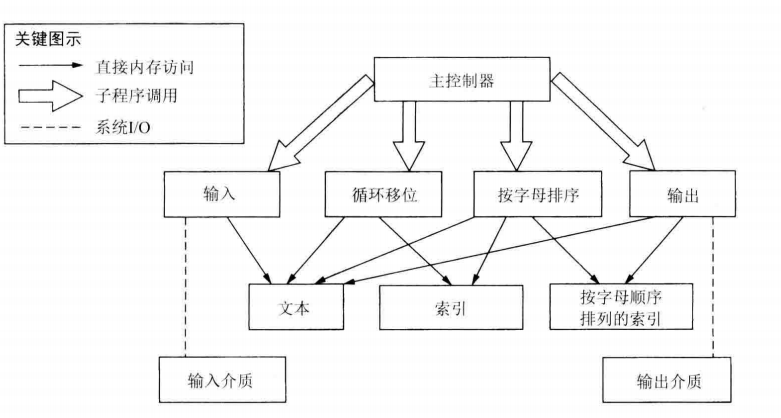
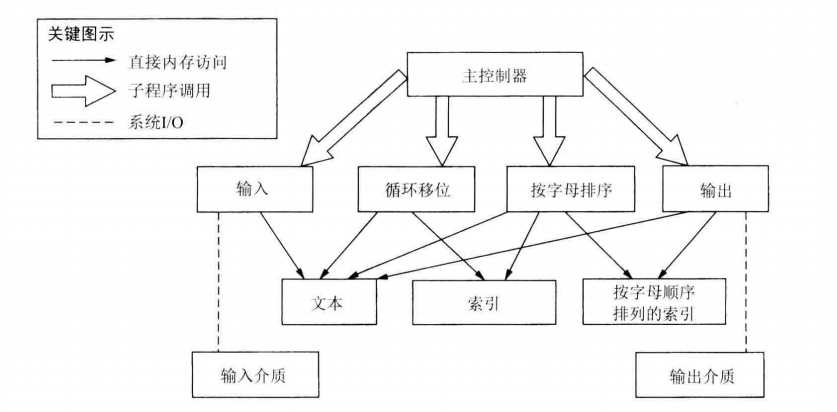
方案一



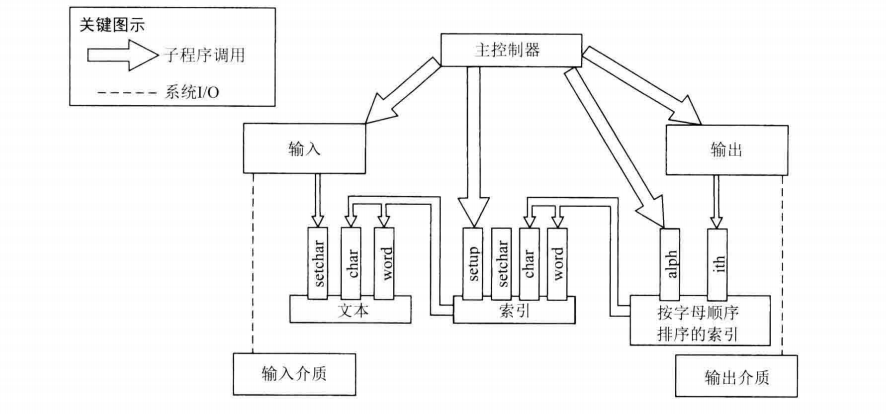
方案一将问题分解成4个部分的功能:输入、循环移位、按字母排序和输出。因此，系统的功能被分解和模块化。主程序通过顺序调用这4个模块来进行协调。数据集中存放在本地，由不同的构件分别执行计算。但是这种设计是难以进行变更的，因为计算模块是通过读写操作直接访问或操纵数据的，任何对数据和数据格式的改变都将会影响到所有模块。再者，设计中没有一个元素是明显可复用的。

方案二



方案二类似地把系统功能性地分解成若干部分，然后通过主程序顺序调用它们执行，但是，计算模块计算得到的数据就存储在该模块本身中。例如循环移位模块负责文本中关键字的索引，而按字母顺序排序的模块负责存储排好序（按字母顺序）的索引，模块之间通过访问方法进行访问，而不是直接访问，因此这些模块就形成了数据抽象。在数据抽象中，方法的接口没有提供任何关于模块数据或数据表达的提示，这使得修改与数据相关的设计决策变得更加容易，且不会影响到其他模块。因为数据抽象模块同时涵盖了数据和数据上的操作，所以这些模块比起第一种设计中的模块更加易于复用。而在底部，修改系统就不那么容易了，因为功能和数据之间有着很高的耦合度。例如，忽略那些循环移位以干扰单词开始的索引则意味着，(1)改善现有的模块使其更加复杂，更加依赖于上下文，更加不利于复用;或者(2）插入一个新模块用来删除那些建立起来但效率低下的无用的索引，并且修改现有的模块来调用新的模块。

方案三



方案三中，数据存储在ADT中，它管理泛化数据类型，比如，行和索引集，而不是基于KWIC的数据抽象。由于ADT的一般性，它们甚至比数据抽象有更高的可复用性。而且，数据和数据上的操作都封装在ADT模块中，所以数据表达形式的改变只局限在模块内部。这种设计和第一种设计类似，因为它们都将系统按功能性分解并且将它们模块化成可计算的模块。但是，在这种设计中，很多计算模块是由事件触发的，而不是显式的过程调用，例如，当一行新文本输入时循环移位模块就被激活。此外，通过增加新的可由系统事件触发执行的计算模块，该设计可以很容易地被扩展，而并不需要对现有模块做修改就可以将新模块集成到系统中去。

主程序/子程序风格将系统组织成层次结构，包括一个主程序和一系列子程序。主程序是系统的控制器，负责调度各子程序的执行。各子程序又是一个局部的控制器，调度其子程序的执行。分为四个基本功能：输入，移位，排序，输出，主程序按次序调用这四个模块，通过共享的数据存储和无约束的读写协议在模块见进行数据交换。