|  |  |
| --- | --- |
| **Assignment 1: KWIC** | **CS3219 Semester 1 2016/17** |

|  |  |  |
| --- | --- | --- |
| **Code Repository URL** | <https://github.com/cs3219ay1617s1-g5/A1_KWIC> | |
| **Student Name** | Teo Hui Wen | Kenneth Ho Chee Chong |
| **Matriculation Number** | A0115034X | A0125364J |

# **Introduction**

The Key Word In Context (KWIC) is an index system where it lists a circular shifts of all lines provided. It is also alphabetized by keyword (sorted and starting character in upper case). There are two implementations for this assignment. One of the it is Abstract Data Type (ADT) which is implemented by Hui Wen and Pipe and Filter which is implemented by Kenneth.

# **Requirements**

## **2.1. Functional Requirements**

Below is the list of functional requirements:

* Sysetm should be able to circularly shift lines by removing the first word and appending it at the end of the line.
* System should be able to alphabetize the outputs.
* System should generate a KWIC index of the input lines.
* System should be able to have a list of words to ignore as keywords.

## **2.2. Non-Functional Requirements**

Below is the list of non-functional requirements:

* The lines must be circularly shifted exhaustively (until the first word comes back as first word)
* Alphabetize is to sort the output in ascending order with the starting letter of the keyword in upper case.
* Keywords must not be in the list of words to ignore.

# **Architectural Designs**

## **3.1. Abstract Data Type (ADT)**

## **3.2. Pipe and Filter**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | KWIC |  |  |  |
|  | | |  | Alphabetizer |  |
|  | | | Reader |  |  |
|  | | |  | Printer |  |
|  | | | CircularShifter |  |  |
|  | | |  | Output Medium |  |
|  | | | Remover |  |  |
|  |  |  | | | | | |
| **Legend** | | | | | | |
|  | Pipe | | | | | |
|  | System I/O | | | | | |
|  | Filter | | | | | |

# **Limitations & Benefits of Selected Designs**

## **4.1. Limitations**

|  |  |
| --- | --- |
| **Abstract Data Type** | **Pipe and Filter** |
| For an object to interact with another, it must know its identity. | Almost impossible to support interactive system.  i.e. Difficult to delete a specific line or allow user input halfway through. |
| When the methods of an object change, so must all other objects that use this method. | Inefficient use of space as data must be copied from filter to filter. May cause a problem if data is too big. |
| Not well suited to enhancements. | Bottleneck may occur if a filter takes a longer time to execute its function compared to the other filters. |
| To add new functions, modules must be modified or added, which could compromise performance. | No centralized data storage. |

## **4.2. Benefits**

|  |  |
| --- | --- |
| **Abstract Data Type** | **Pipe and Filter** |
| Data is not shared directly by the components. | Each filter is able to function on its own.  i.e. Able to use just the shift function or just the alphabetize function |
| Both algorithms and data representation can be changed in individual modules without affecting others. | New functions can be easily added because each filter only does what it is supposed to do then pass on the data. |
| Encourages software reuse. | Modification can be made easily to each filter as each of them is isolated from one another. |
| Encourages decomposition of a problem into a number of interacting components. | Each filter is run as a separate thread, which allows immediate processing of the input. |
| Possible to change implementation of an object without affecting its clients. | Easy to understand. |