KWIC Search Engine

Team LPNX

Xinhe Chen

YuWei Pai

Keith Nguyen

Hengbo Liu

SE 6362.001

Preliminary Project Management Plan

Team Lead (Rotated):

Team lead will be rotated by week.

Team Members:

Xinhe Chen (xxc170630)

YuWei Pai (yxp170730)

Keith Nguyen (kxn161730)

Hengbo Liu (hxl170010)

Roles:

Xinhe Chen	Architectural Specification Specification of Program Manual Backend Java code
YuWei Pai	Non-functional requirement
Keith Nguyen	Introduction Functional Requirements
Hengbo Liu	Frontend, Database, Connection code

Team Website:

https://github.com/xinhechen/KWIC

Introduction

The KWIC system stands for Key Word In Context, the most popular format for concordance lines. This system sorts and align the keywords in a title so that each word can be searchable in the index. This excludes the stop words. When the keyword is searched, a KWIC index uses a wide layout to provide context for each word.

Requirement Specification

Functional Requirement

Below is the described functional requirements for the KWIC system based on the information given by Dr. Chung. This section will contain the (1) components; (2) input; (3) operation; (4) output.

Issued ID	FR1
Name	Must include the required components below
Description	● The system shall have a web interface. The system shall be able to run on Chrome and Firefox. ● The system shall implement ADT architecture and functions.

Features	● The KWIC system has a website. ● The website runs the same way on Chrome and Firefox. ● The system implements the ADT architectures and functions as shown in the diagram.

Issued ID	FR2
Name	Inputs
Description	System shall accept the listed inputs..
Features	● The system shall accept an ordered set of line. ● Each line is an ordered set of words. ● Each word is an ordered set of characters. ● The system shall be able to accept multi-line input.

Issued ID	FR3
Name	Operation
Description	The system shall repeatedly remove the first word and appending it at end of line.

Features	● The getchar, setchar, CS-char, CS-word functions shall perform the actions listed above.

Issued ID	FR4
Name	Output
Description	The output shall list all circular shifts of all lines in ascending alphabetical order.
Features	● The system shall sort the output in ascending alphabetical order and list the output.

Non-Functional Requirement

Described non-functional requirements for KWIC system as below. The following non-functional requirements are presented. There are five NFR soft goals that the system should accomplish: (1) Good Performance; (2) Portability; (4) User-Friendly; (5) Open Source; (6)Reusability. Each of the NFR softgoal is described in details.

Issued ID	NFR1
Name	Good Performance
Description	The system shall be accomplished with limited response

	time, and sufficient memory space for adding additional users or data calculation.
Type	Softgoal
Affecting NFRs / Operationalizations	● Manageable Space ● Low Response Time ● Increased storage capacity ● Use uncompressed/compressed format data ● Process and retrieve data in limited time

Issued ID	NFR2
Name	Portability
Description	The system shall consider portability issue while developing the system.
Type	Softgoal
Affecting NFRs / Operationalizations	● Platform independence- web browser ● Standard programming language ● Operating system

Issued ID	NFR3

Name	Enhanceability
Description	The system shall able to add additions of system function.
Type	Softgoal
Affecting NFRs / Operationalizations	● Noise word eliminator

Issued ID	NFR4
Name	User-Friendly Access
Description	The system shall consider user-friendly access, and easy understanding manual is necessary.
Type	Softgoal
Affecting NFRs / Operationalizations	● Easy understanding user interface ● User manual

Issued ID	NFR5
Name	Open Source
Description	The system shall be open source, can encourage more developer to join in, and enhance the system quality and performance.

Type	Softgoal
Affecting NFRs / Operationalizations	● Completeness ● Availability ● Maintenance ● Online website ● Programming language

Issued ID	NFR6
Name	Reusability
Description	The components shall serve as reusable entitles.
Type	Softgoal
Affecting NFRs / Operationalizations	● Circular shift ● Alphabetizer

Architectural Specification

To meet all the functional requirements and non-functional requirements. 4 architectures were considered for the KWIC system. Shared data, Abstract Data Type(ADT), Implicit Invocation, Pipe and Filter. All architecture meet the functional requirements. The key to choosing an architecture to implement lies in how each architecture can meet the non functional requirements specified above.

Shared data

Structure

In the Shared data architecture, each component shares the data with all the other components. The Input will store all the characters in Characters. Circular Shift will access the Characters storage to retrieve everything and perform circular shift. The results will be stored as indexes. Alphabetizer will access the Characters storage to retrieve the sentences and also the Index storage to retrieve the indexes created by Circular Shift. Alphabetizer will then create a set of the alphabetized Index. Output will access the Characters and Alphabetized Index to recreate the sentences and output the results. Figure 1 shows the basic architecture of the Shared Data Structure.

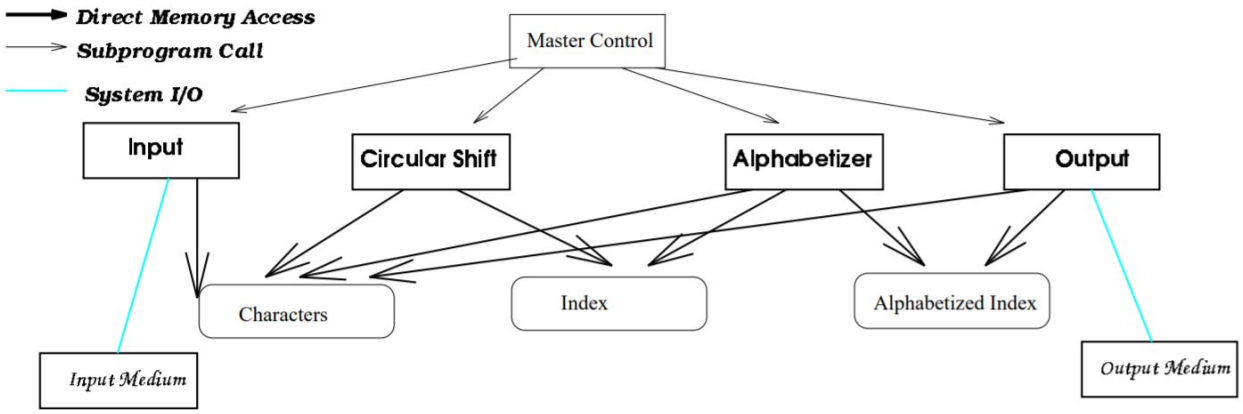


Figure 1. Shared Data

Non-Functional Requirements

Modifiability - changes in processing algorithms

The Shared Data architecture requires a large amount of changes if changes are made in processing algorithms. For example, changing between batch and incremental alphabetizer will affect Circular shift.

Modifiability - changes in data representation

Modifying data representation will also be a huge problem for this architecture. Since the data is shared by most modules, changing the data representation may result in need to change every single module.

Enhanceability - additions of systems functions

Adding system functions such as noise eliminators will also be difficult for Shared Data. Putting it between Circular Shift and Alphabetizer will create blanks in the Index data storage. Thus alphabetizer will need to be changed to accommodate the addition of a noise eliminator.

Performance - Space and time

Shared data is great for saving space and time. Data is shared by all the modules so no data needs to be stored multiple times. Time is also conserved since modules do not need to copy data.

Reusability

The reusability of this architecture is low. It depends on shared data and format.

The Shared Data architecture only meets the Performance NFR. It is very difficult to modify, enhance or reuse which is why this architecture will not be pursued for the KWIC system.

Abstract Data Type

Structure 1

In an ADT structure data is not directly shared between each process component. Instead, each module provides an interface, and other modules may access its data by invoking the interface. This allows each module to only supply data it would like to share or hide its info. This also prevents other modules from changing its data. Figure 2 shows an ADT architecture.

The Master Control invokes the input, which invokes the setchar in Line Storage. Line Storage takes in the input and stores the inputted sentence. Master Control then invokes the setup in Circular Shift. Setup will retrieve all the data from Line Storage by using the char and word interfaces and then perform the circular shift. CS-setchar will be invoked to store the circular shifted sentences. Master Control will then invoke the alpha in Alphabetic Shifts, which uses CS-char and CS-word to retrieve the circular shift data. It will perform the alphabetization and then store it. The Master Control lastly invokes the Output which

uses i-th from Alphabetic Shift to retrieve the alphabetized sentences and outputs them.

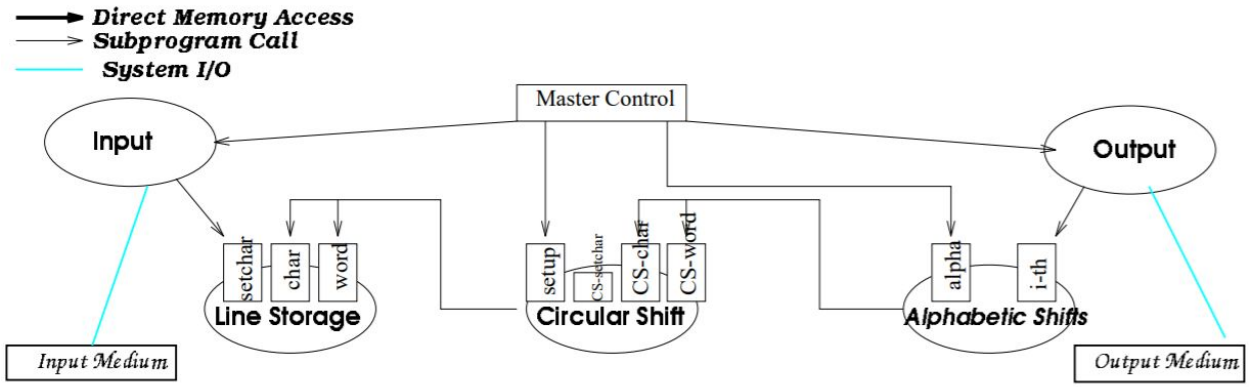


Figure 2. ADT1

Non-Functional Requirements

Modifiability - changes in processing algorithms

ADT provides great modifiability. Each module is independent and is not affected by changes in other modules. Changing between batch and incremental alphabetizing does not affect anything.

Modifiability - changes in data representation

ADT provides great modifiability. Each module is independent and is not affected by changes in other modules. Changing a data representation in one component does not affect other modules as long as the module being changed returns the same format data when its interface is invoked.

Enhanceability - additions of systems functions

Little change is required when adding system functions to ADT. for example, if Noise Eliminator is added, the only thing that needs to be changed would be where Alphabetic Shifts gets its data.

Performance - Space and time

Usage of space is more inefficient since each component needs their own copy of the data (Circular Shift and Alphabetizer).

Time usage will also be more inefficient because each component will need to reconstruct the sentences.

Reusability

ADT is more reusable than shared data. Modules make fewer assumptions about others they interact with.

Although ADT has performance issues compared to shared data, it is a good tradeoff for the modifiability, enhanceability and reusability it provides. The KWIC system is only a basis for a search engine and may require a lot of modifying, enhancing, and reusing. Thus, ADT will be chosen over Shared Data.

Structure 2

Another ADT architecture style is considered as shown in figure 3. In this architecture, a few things are changed to make the structure more uniform and understandable.

The Master Control invokes the Input component and the input component invokes setup in line storage. Setup stores the data in Line Storage using setchar. Master Control then invokes the setup in Circular Shift. Setup retrieves data from line storage using char and word and performs the circular shift. Setchar stores the circularly shifted words in circular shift. Master Control then invokes the setup in alphabetic shift which again retrieves information using char and word from circular shift. It then performs the alphabetization and uses setchar to store the alphabetized data. Lastly, Master Control invokes Output to retrieve the alphabetic shift data using char and word and outputs the results.

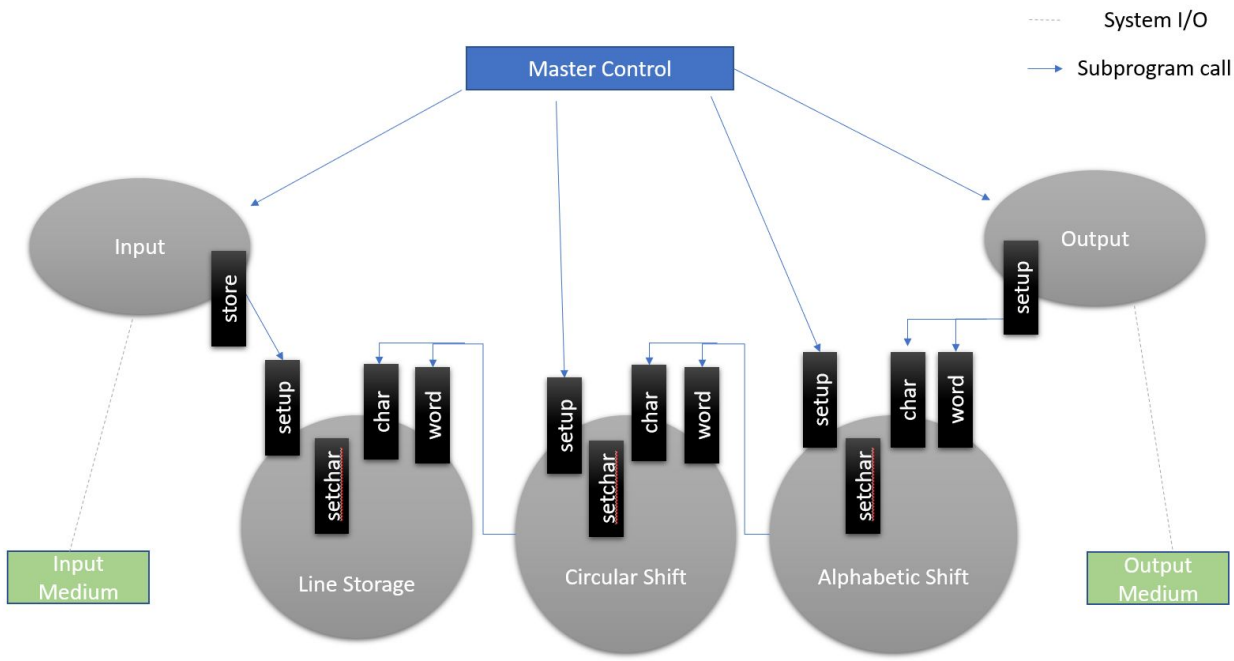


Figure 3. ADT2

Non-Functional Requirements

Modifiability - changes in processing algorithms

Same as ADT1

Modifiability - changes in data representation

Same as ADT1

Enhanceability - additions of systems functions

Same as ADT1. But it has the advantage of modeling a new component after the existing components. It will be easier to create the new system function since the structure of each component is uniform throughout, and the new function can use the same structure.

Performance - Space and time

Same as ADT1

Reusability

Same as ADT1. However, each component is more reusable since they are modeled with the same structure.

The NFR mentioned in ADT1 is the same as of ADT2. However, in this architecture each component in this structure is very similar to the others. This makes it more understandable, once you understand one component, the other components are also understood. It also makes it easier to enhance. If a Noise Eliminator is to be added, the Noise Eliminator can be modeled after the components already present. Due to the better understandability, enhanceability, this structure will be chosen over the first ADT structure.

Implicit Invocation

Structure

Implicit invocation is another structure considered for the KWIC system. The component integration is based on shared data, but interface to data is more abstract. The storage format are not exposed to computing modules. Figure 4 shows the architecture of implicit invocation. Master control invokes Input which invokes operation insert to create I- Lines. It then implicitly invokes Circular Shift which uses i-th to read in I-Lines creates the shifted lines and create a new CS - Lines line buffer. After this, Alphabetizer is implicitly invoked by i-th and reads in the CS- Lines, alphabetizes everything and creates a line buffer A-Lines. The output then access all the A- lines through the 2nd i-th and outputs them.

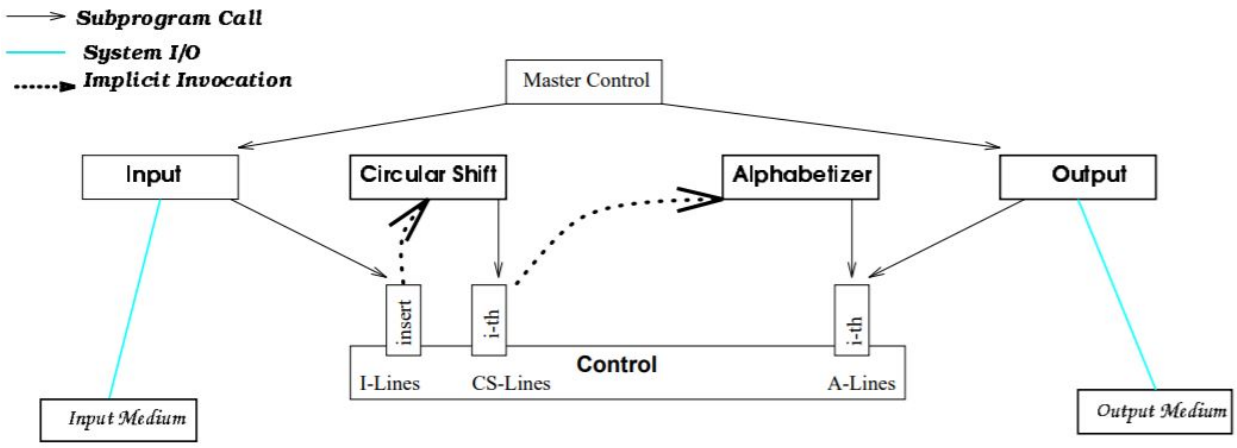


Figure 4. Implicit Invocation.

Non-Functional Requirements

Modifiability - changes in processing algorithms

Each module is independent and is not affected by changes in other modules. Changing between batch and incremental alphabetizing does not affect anything.

Modifiability - changes in data representation

Data access is abstract and does not affect others.

Enhanceability - additions of systems functions

Little change is needed when adding system functions to implicit invocation. The additions are invisible to other modules and do not affect them.

Performance - Space and time

Implicit invocation uses more space due to natural representation. It is also very time costly due to implicit invocations. Detection of implicit invocation is very costly.

Reusability

This architecture is highly reusable since implicitly invocated modules rely only on existence of certain externally triggered events.

Implicit Invocation meets most NFR requirements including modifiability, enhanceability and reusability. However it suffers from performance issues. It uses a lot of space to store the lines, and detection is too costly. Thus we will not be considering this architecture for the KWIC system.

Pipe and Filter

Structure

The pipe and filter architecture is the last structure to be considered for the KWIC system. Figure 5 shows the architecture. Each module is a filter that processes the input data and produces outputs data. Input takes in the input from the System I/O and outputs it. Circular Shift takes in the output of Input, performs circular shift, and outputs the results. Alphabetizer takes in the results of Circular Shift, alphabetizes everything, and outputs it to Output which displays the results.

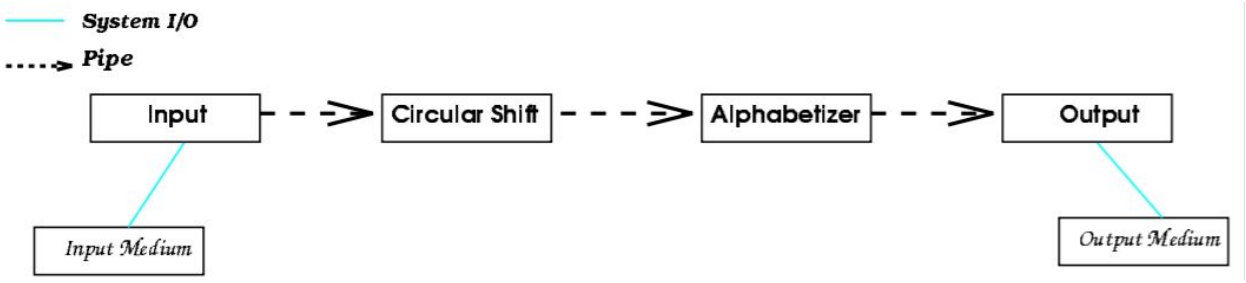


Figure 5. Pipe and Filter

Non-Functional Requirements

Modifiability - changes in processing algorithms

Because each filter only needs an input, and does not access any of the other filters, it is process independent. Modifying processing algorithms will not affect other processes.

Modifiability - changes in data representation

Because each filter only needs an input, and does not access any of the other filters, it is process independent. Modifying data representation will not affect other processes.

Enhanceability - additions of systems functions

Adding new systems functions will be very easy, since a filter can be added anywhere in the process. However, it is difficult to support an interactive system if the user tries to delete from the original lines or shifted lines.

Performance - Space and time

Pipe and Filter usage of space and time is inefficient. Each filter copies the input data to its output part. Data is replicated repeatedly which wastes time.

Reusability

Pipe and Filter is highly reusable since it is process independent.

Pipe and Filter is reusable, highly modifiable but it suffers from performance and interactive system enhancements. Therefore Pipe and Filter will not be considered for the KWIC system.

Conclusion

4 different architectures were considered for the KWIC system. Shared data, ADT, Implicit Invocation and Pipe and Filter. After comparing the NFRs of all the architecture. We find ADT to be the best match. It is highly modifiable, enhanceable and reusable. Although it suffers in the performance area. It is a good tradeoff for the other NFRs. A second ADT architecture is considered to enhance the original ADT architecture. It is more intuitive than the first ADT architecture and the KWIC system will be implemented using this architecture.

Specification of program

Program Objectives

The KWIC system is a web based program where users can type in a sentence, click enter and the website will display the circularly shifted results and alphabeticalized results of this sentence as shown in Figure 6. If the user enters another sentence and presses enter, the KWIC system will perform Circular Shift on the second sentence, add it below the first sentence and display it. The system will alphabetize the first and second sentence together and display it as shown in Figure 7.

Figure 6. Input sentence 1.

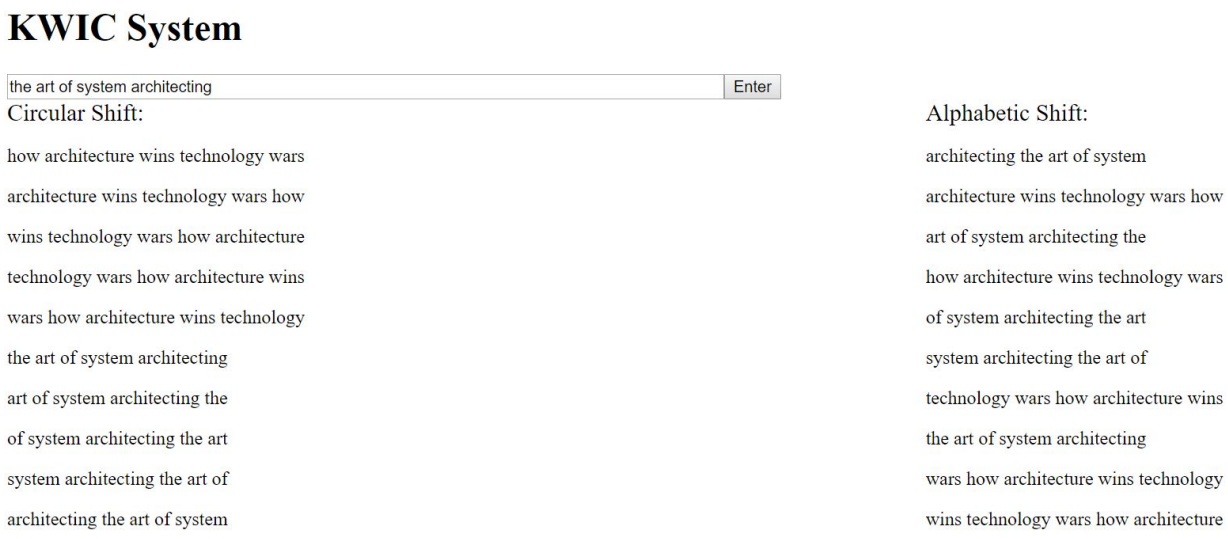


Figure 7. Input sentence 2.

Output Specification

The output of the program is all the circularly shifted sentences and all the alphabetized sentences the user enters and any previous inputs the user has entered before.

Required Input

The input shall be through a website where users can type in sentences. The input shall be one sentence with no punctuations. The system incrementally processes each sentence. Lower and Upper case do not matter. They will all be converted to lowercase for storage.

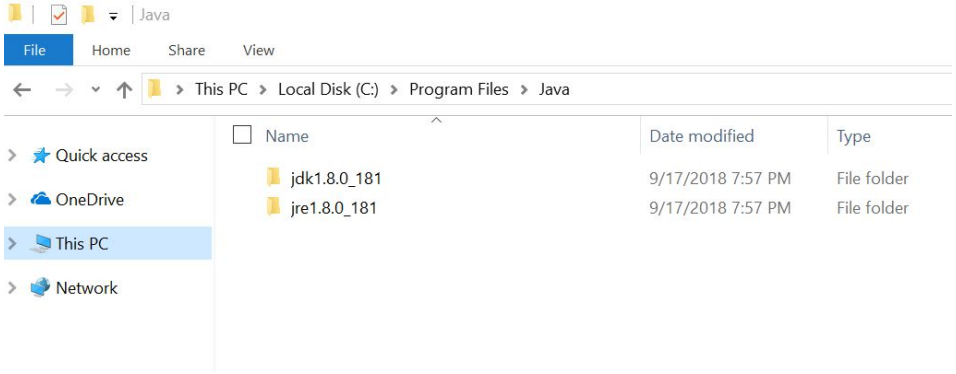
Processing Requirements

The program uses java as a backend to process sentences input through the frontend website. The backend takes in a sentence, performs circular shift, noise elimination and alphabetization using the ADT architecture specified in the architecture section and stores the results in database. The website can then retrieve everything from the database and outputs the results.

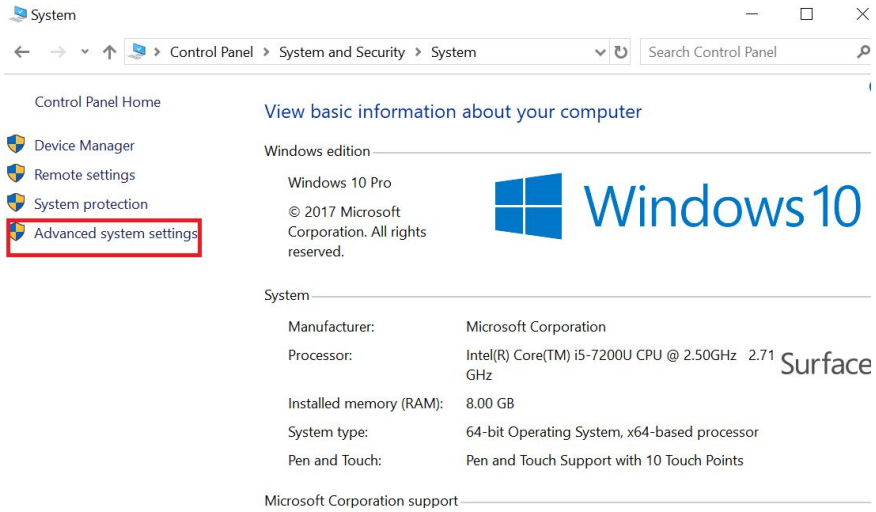
Manual (How to install and run an example)

The source code for the KWIC system can be found at https://github.com/xinhechen/KWIC
To run the code, JDK8, Eclipse, Apache-Tomcat, MySQL, XAMPP will be needed. The following will be a step by step guide on how to download, install and run the program.

1. JDK8
    1) Go to https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html and download the correct version of jdk8 for your computer.
    2) Run the installer which will install both the JDK and JRE
    3) Go to C:\Program Files\Java to take note of your JDK installed directory. In this diagram it is "C:\Program Files\Java\jdk1.8.0_181"

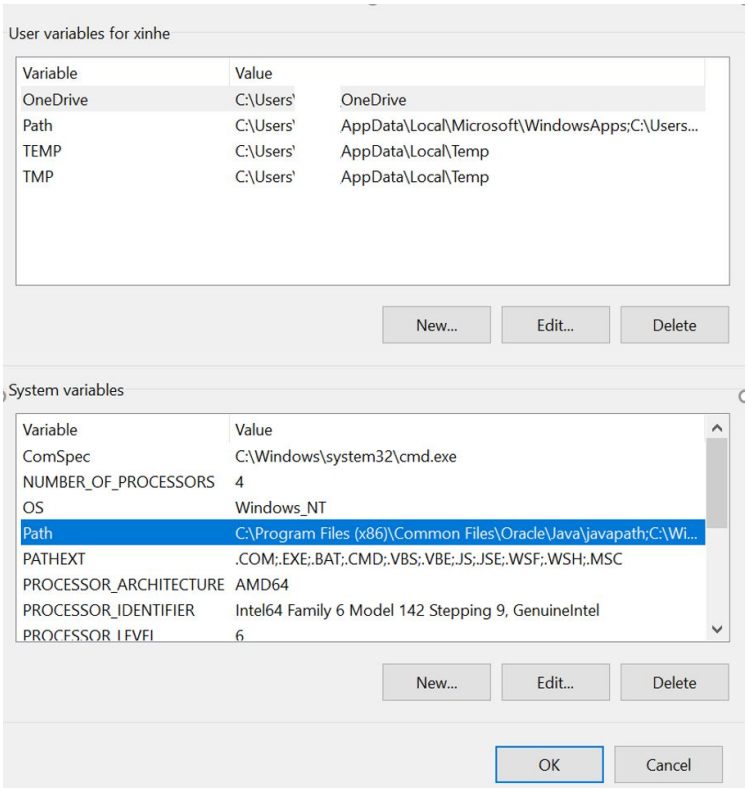


    4) Go to Control Panel - System and Security - System - Advanced system settings



    5) Go to Advanced tab -Environmental Variables.

6) Under System Variables, scroll down Path and select it. Click Edit...



7) Click new and add your jdk bin directory (C:\Program Files\Java\jdk1.8.0_181\bin for the computer shown in graph)

2. Eclipse

1) Download eclipse at https://www.eclipse.org/downloads/
2) Install eclipse

3.Apache Tomcat

1) Go to https://tomcat.apache.org/download-90.cgi and download the binary core installer
2) Run and install Apache Tomcat

4. MySQL

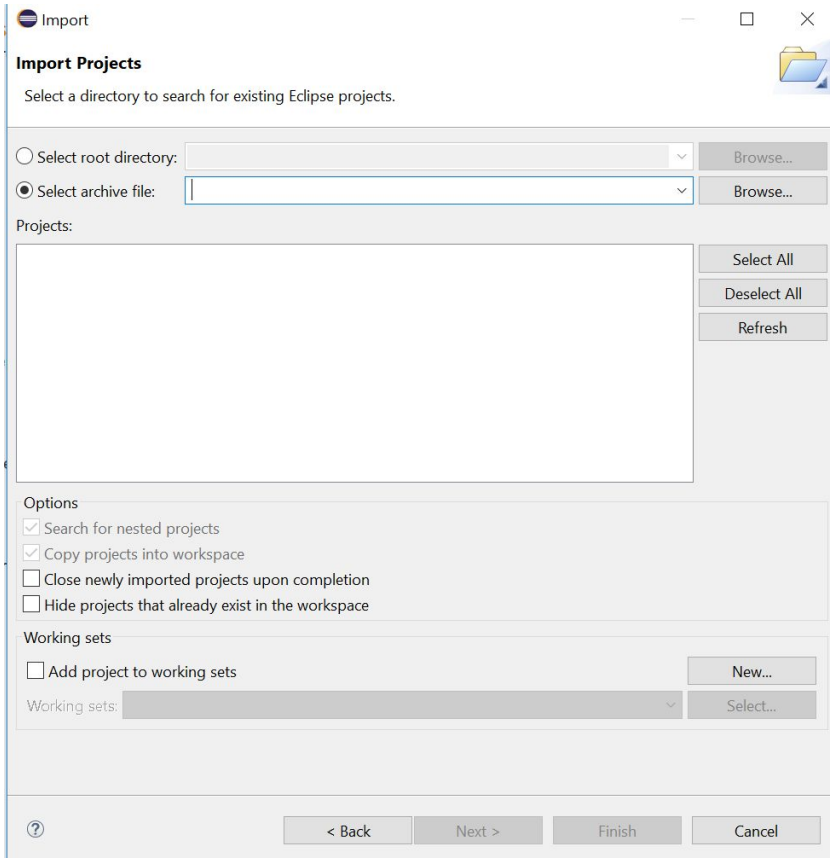
1) Go to https://dev.mysql.com/downloads/installer/ to download MySQL
2) Run the program to install MySQL
3) Make the root password “root”

5. XAMPP

1) Go to https://www.apachefriends.org/download.html and download the appropriate installer for your computer
2) Run the installer to install XAMPP

6. Import code to eclipse

1) Download the sfarch file from https://github.com/xinhechen/KWIC as zip
2) Launch Eclipse select File -> Import ->General->Existing Project into Workspace -> Next
3) Choose Select archive file, select the zip file downloaded and click finish



4) username =root password= root url:
5) Run XAMPP to turn on MySQL
6) Run As Tomcat server at localhost

7. Go to the webpage: https://localhost:8080/sfarch/input.jsp to see the working program.