Software Architecture Project 1 Documentation
KWIC Software for Web-based Search Engine

Group 20 - Jason Switzer, Cliff Halasz

1. Requirement Specification

1.1 Functional Requirements

The KWIC index system accepts a set of lines each a set of words, where each word is a set of ordered characters. The system will then output all possible circular shifts on this set of lines, where a circular shift is defined as all the resultant lines from taking a line and appending the first word on the end until you return to the original line. A line should be able to be removed by the user retroactively, with the system updating the list by removing all circular shifts of that line.

1.2 Non-Functional Requirements

A primary goal of this package is future development and maintenance. The system should be capable of adding new features quickly and easily as customers demand them; such as a feature to remove simple words (I, am, a, the, etc.) from the lines before shifting, or changing the saved index format. The system needs scale in performance, and even for large data sets, the end user should be kept aware that time consuming calculations are being made. Space efficiency is not emphasized. Internal design should be for reusable components so that any part of the system can be quickly reused elsewhere or replaced by a similar component.

The system is also designed for portability. It's developed as a Java 1.6 Web Start application, which means that it will run on any client machine with Java installed and is deliverable over the Internet. The Web Start system is a part of the standard Java JRE. The user needs only to associate the .jnlp file, accessible here, to the javaws binary in the JRE bin folder. Alternatively, the system can be run as a local application if downloaded and compiled (see the README).

Lastly, while not explicitly requested, the system will provide a means to save the indexed data. The format of the output will be as follows:

<index>|<indexed string, circular shift result>

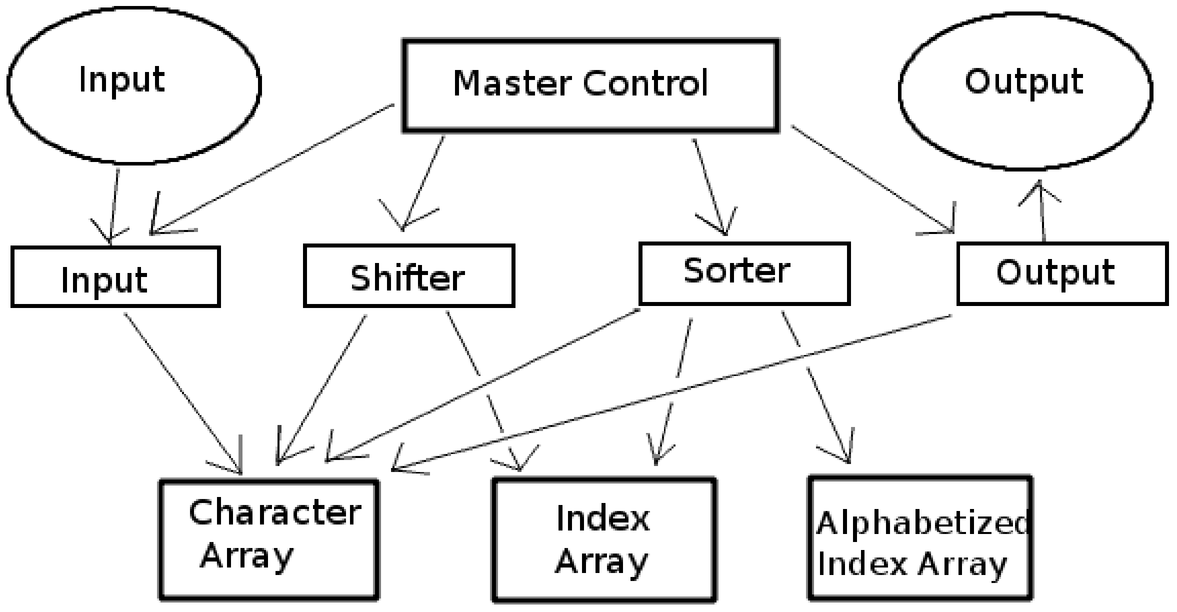
Also, the ability to remove input and indexed records is another functionality that was not explicitly requested. This was added as a proof that our system is extensible and powerful.

2. Architecture Specification

2.1 Shared Data

2.1.1 Description

The Shared Data architecture utilizes a Master Control module which makes subprogram calls to several modules in order, each of which manipulates a section of shared data structures. The Input module reads in the data and stores an array of characters. Then the Shifter is called, which operates on the same array and creates an array of indices which are the character indexes of each starting shifted string from the parent string. The Sorter uses these arrays to produce an array of alphabetized indices stored in a third array. Finally, the Output module uses the alphabetized indices and the character array to print the list in alphabetical order.



2.1.2 Advantages

The Shared Data architecture is by far the most efficient in both time and space complexity. Since data is shared, it is only stored in one location. In addition, by storing indices you avoid even copying the data for the shifts. Not having to copy data also makes this the fastest approach possible.

2.1.3 Disadvantages

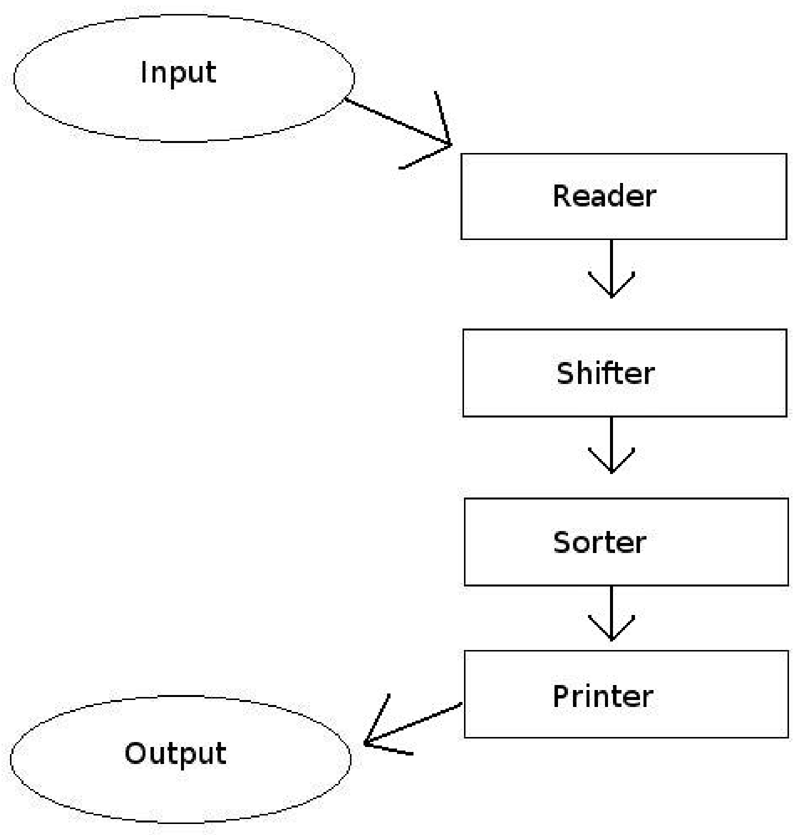
The Shared Data approach would have to be custom-tailored to the exact situation and data structures, leaving little room for reuse of code. Any change to the data structure would require changes in all modules. Any changes to the shifting algorithm would effect both the shifter, alphabetizer and the output. Additionally, it is going to be the most unintuitive solution, making development and maintenance far more difficult and time consuming. Adding support for additional features such as removal is possible, but will begin to complicate an already complicated design.

2.2 Pipe and Filter

2.2.1 Description

The Pipe and Filter architecture includes 4 separate modules, each of which processes an input and then passes the result on to the next module as input. The modules include a Reader, a Shifter, a Sorter, and a Printer. Each module will be completely separate and as long as each maintains the same outline then modification is easy. These 4 modules are the major components of the system, called Filters, and the connections between them are known as Pipes.

Pipe and Filter Architecture



The Reader takes in the input and splits it into an array of strings to pass to the Shifter. The Shifter then performs circular shifting on all the strings, and then passes that to the Sorter. The Sorter reads through the list and arranges it in alphabetical order. Then it is passed to the Printer, which reads through line by line and prints the outputs for the user.

2.2.2 Advantages

The Pipe and Filter system is very simple to understand and implement, easy to maintain, and supports some code reuse among modules. Additional features could be implemented easily by adding new filters, though this will be an inefficient design.

2.2.3 Disadvantages

The Pipe and Filter architecture is unfortunately very space inefficient as all the data has to be recopied over at each pipe transition. Also, Pipe and Filter modules cannot process in parallel because each Filter expects the full input from the Pipe at the beginning of its execution.

2.3 Hybrid Architecture

2.3.1 Description

The Hybrid architecture aims to combine the best aspects of the ADT/OO and Implicit Invocation architectures discussed in class. This approach uses a generalized object system that builds upon the standard Java library for high re-usability and maintainability. It also uses the standard Java GUI DataModel architecture and background threads to update the interface seamlessly and quickly. Due to the use of the DataModel architecture, an implicit invocation occurs when an item is added to the list which triggers an event that the list widget (JList) registered for in order to update itself.

The system is designed to operator similar to Iterators: each module in the system is iterated over and each result is passed to the next module down the line for processing. Once the Reader reads a line, it is added to the GUI (input record list) and added to the Output list immediately. The Output list will pass the line to the Circular Shifter, and then iterate over each circular shifted result, adding it to itself (the Output list). Once the Output list is created with the fully indexed output, it can also be iterated over, with each result added to the GUI (indexed record list) directly.

2.3.2 Advantages

This design maintains high reusability and is also highly extensible. Each step can be separated and operate independently, making it possible to perform the steps in parallel. Because each module is only aware of what it expects as input and which step is next in line, the coupling is kept low, though not as high as implicit invocation. The DataModel architecture makes it easy to keep the user interface updated as frequently as we want with very custom code necessary.

2.3.3 Disadvantages

The coupling is not as low as implicit invocation because the IndexedList has to be aware of how to perform the shifting, though this is done only as an abstraction from the GUI, which acts as the main controlling unit. Since the GUI only cares about the results in the Output list, it was decided that this is a comfortable medium. The DataModel architecture is not designed for rapid updates to the underlying data model, resulting in quirky GUI behavior in certain circumstances (loading large files causes unwanted flicker). Space is not empasized, so no attempt to reduce object usage is taken; there are situations in which too many objects are created, though this is likely limited because of the iterator design (those objects are temporaries and are short lived, which is good for garbage collection).

2.4 Decision

The Pipe and Filter system would be much easier to understand and more reusable than the Shared Data architecture. Time efficiency only becomes a problem on very large sets of data, beyond the scope of the requirements, and space efficiency is not a requirement. Development time would be shorter giving us more room to ensure quality and work on additional features. Since we will need to use this system for future work, there is a high importance on extendability and re-usability. We decided to implement the hybrid architecture so as to best meet the functional and non-functional requirements.

3. Program Specification

Main extends SingleFrameApplication and sets up the MainWindow, which controls the GUI interface and performs the tasks of the Printer. This can essentially be considered the Main Control module of the program. When the user selects a file, MainWindow passes the file path to InputReader which performs the work of the Reader module. The InputReader is iterated over to obtain the input lines, which is stored in an IndexedString.

The IndexedString objects are passed directly to the IndexedList, which acts as an abstraction to the Alphabetizer. The IndexList also provides an abstraction to the CircularShifter (see the IndexList.add function), which performs the role of the Shifter by creating new IndexedString objects for all of the circular shifts of the IndexedStrings it was passed. The IndexList will iterate over the CircularShifter and add each circularly shifted result to itself. The underlying data type of the IndexList was chosen for sorted inserts (TreeSet), so no further work is needed.

Once the InputReader terminates the iterator, the MainWindow (Main Control module) will iterate over the IndexList as well, adding each IndexedString directly to the user interface, which represents the Output module. The GUI uses DataModels, which will implicitly fire events to trigger the update of the appropriate widgets. This is all abstracted and provided by the Java JRE.

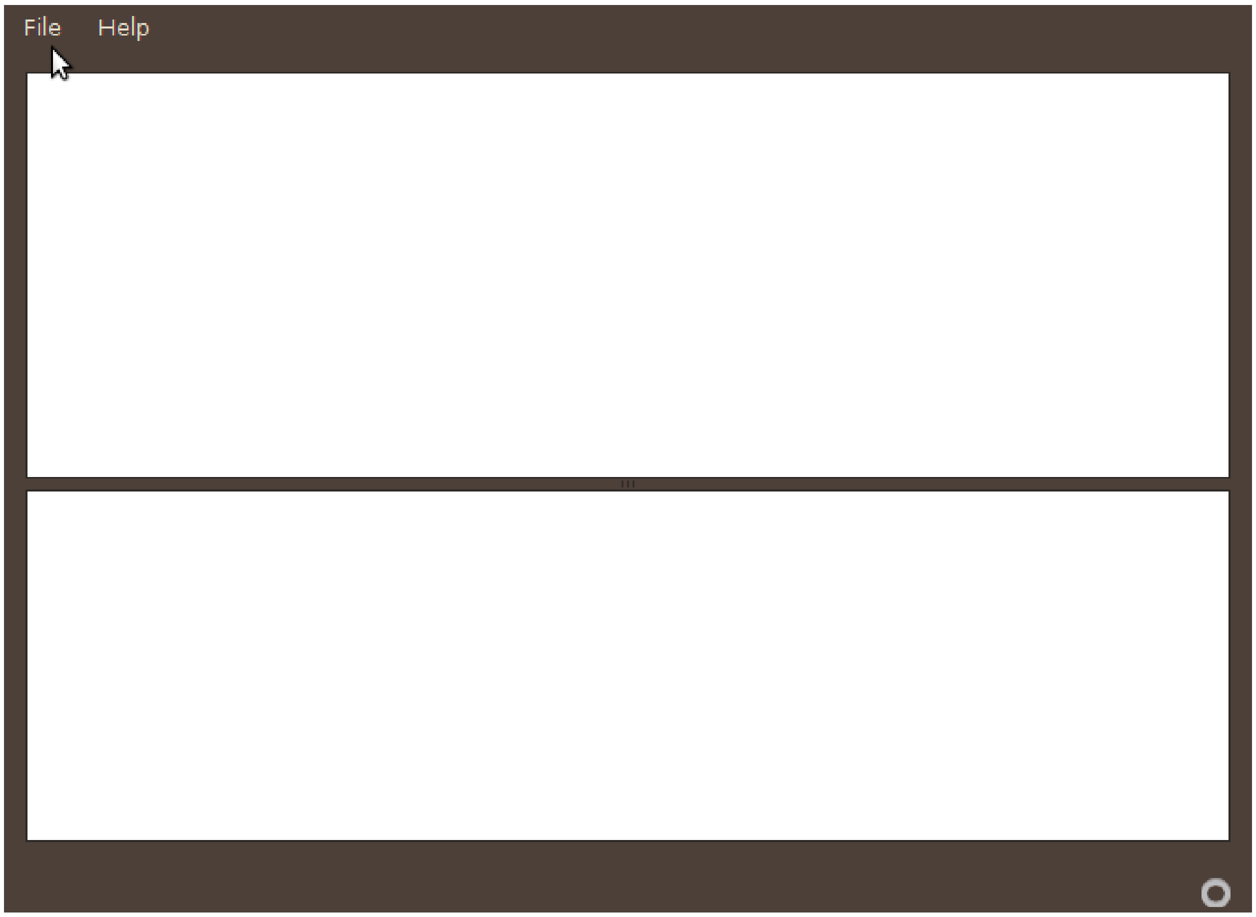
4. User Manual

The KWIC indexing system is capable of loading input data and generating the indexed output. The following describes how a user interacts with the indexing program to generate and save indexed data.

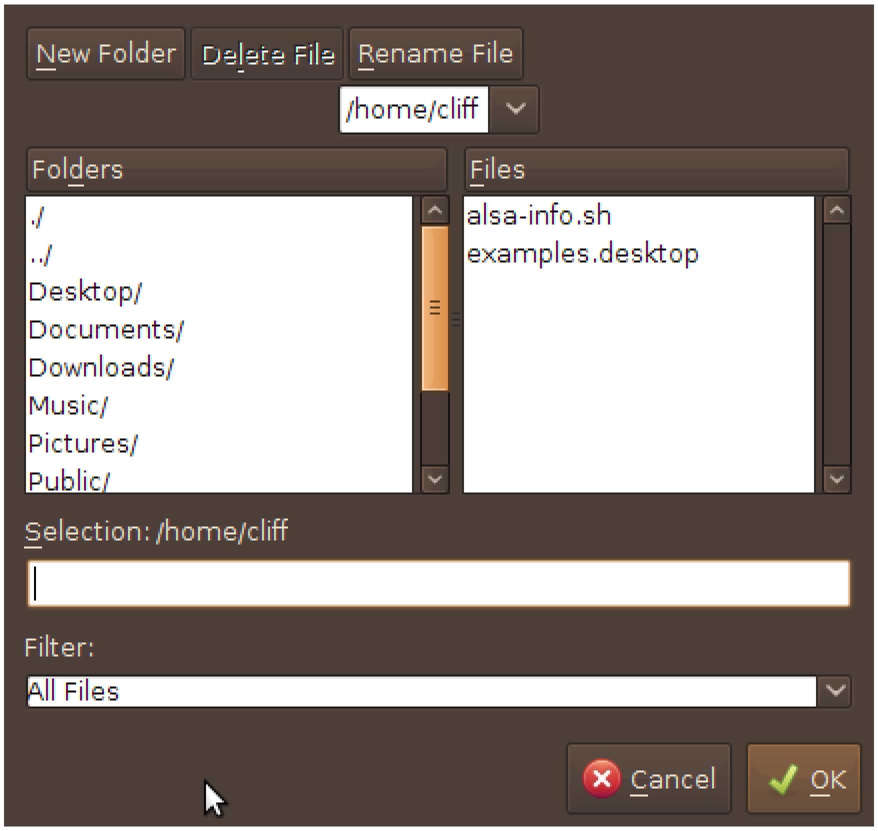
4.1 Loading an input file

To begin editing text, you must load an input file first. Each line from the input file must conform to the formatting requirements, otherwise, an error will be raised. Currently, this error is only raised to the console. The line will be skipped, thus not present in the input record list and index data will also not be present in the index record list.

Start by expanding the *File* menu and selecting the *Load...* menu item:



A file selection dialog will open, allowing the user to select any accessible file:



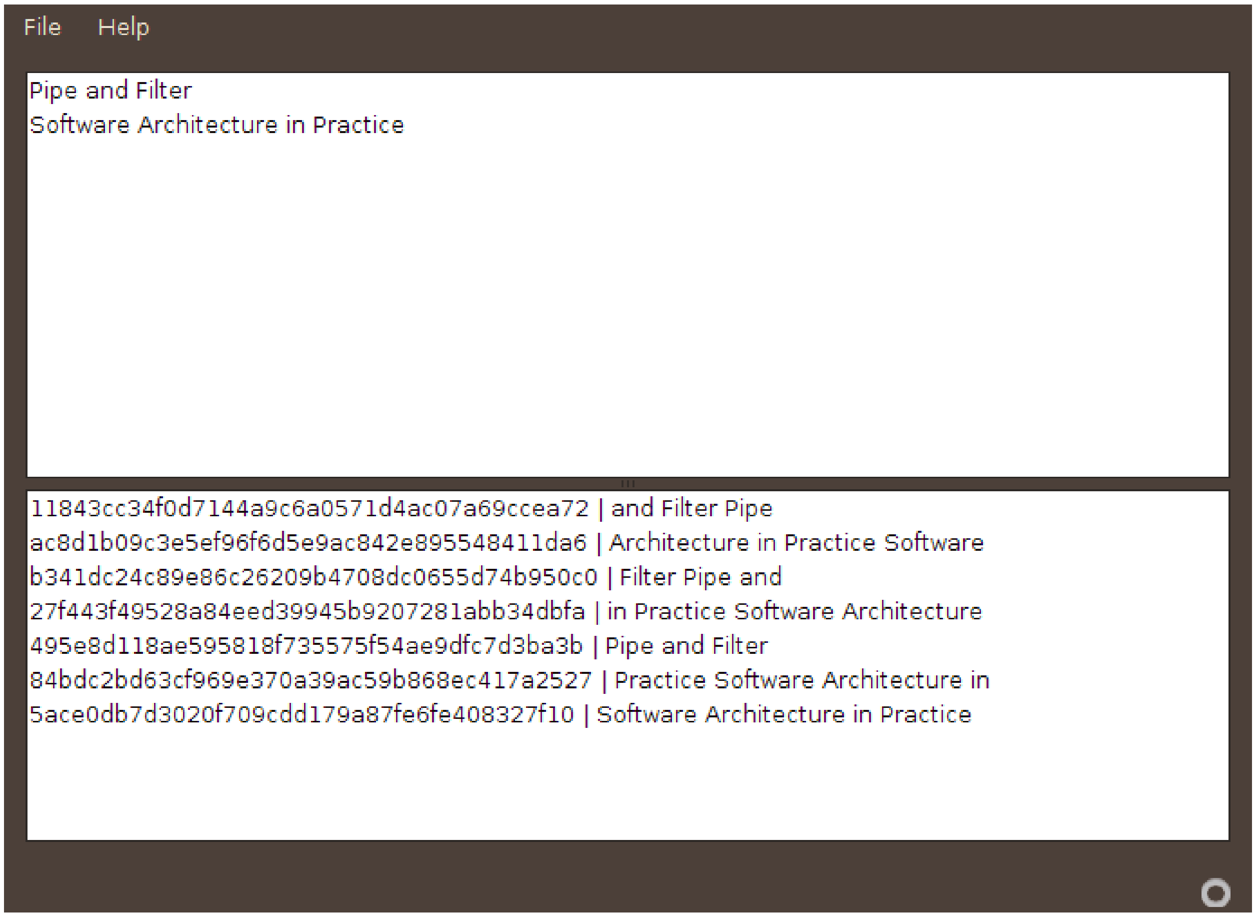
./
../
Desktop/
Documents/
Downloads/
Music/
Pictures/
Public/

alsa-info.sh
examples.desktop

Selection: /home/cliff

Filter: All Files

Once the file is selected, click on the *OK* button, triggering the load of the file. This will begin loading the file and indexing each properly formatted line into the input record list. Once the data has been loaded and indexed, the index data will be loaded into the index record list.

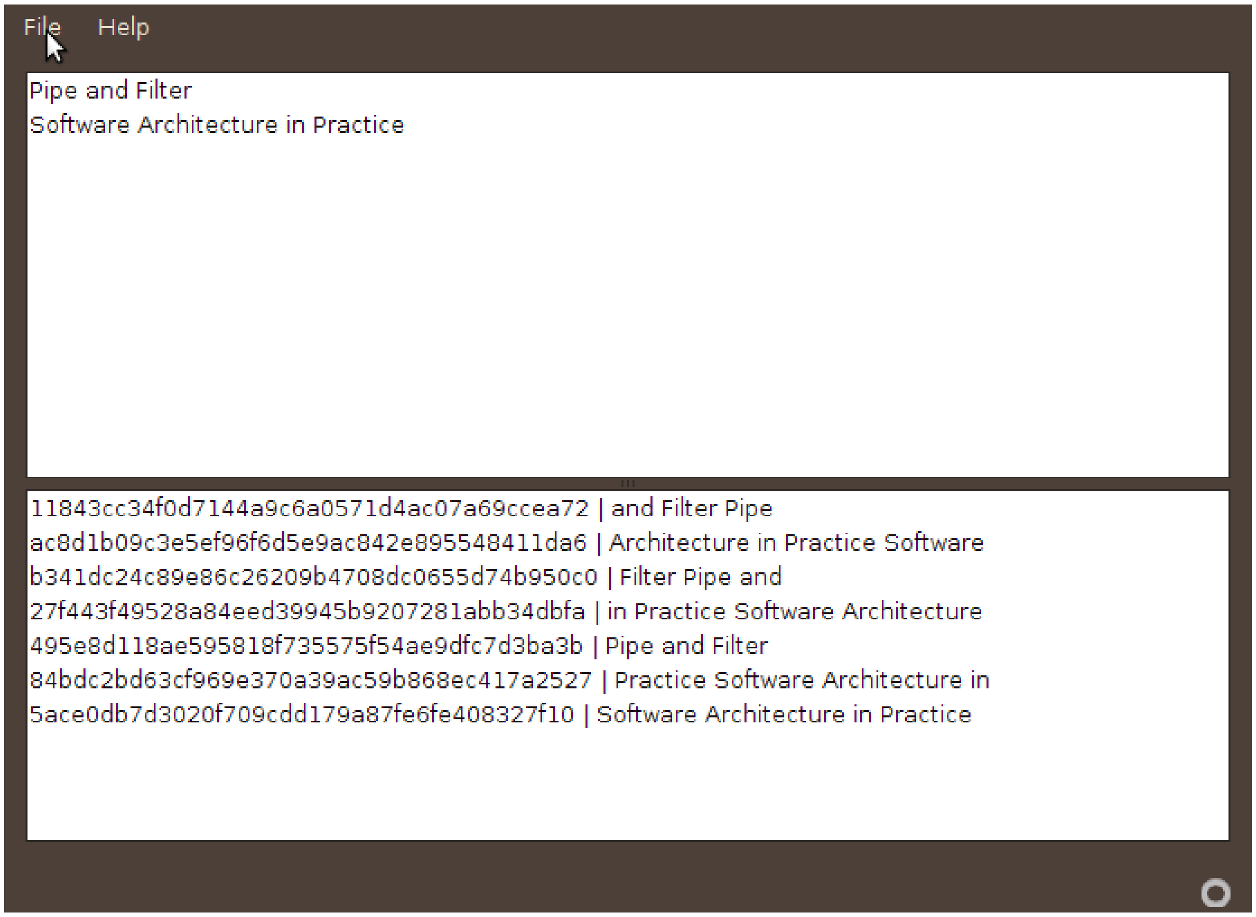


```
Pipe and Filter
Software Architecture in Practice
```

```
11843cc34f0d7144a9c6a0571d4ac07a69ccea72 | and Filter Pipe
ac8d1b09c3e5ef96f6d5e9ac842e895548411da6 | Architecture in Practice Software
b341dc24c89e86c26209b4708dc0655d74b950c0 | Filter Pipe and
27f443f49528a84eed39945b9207281abb34dbfa | in Practice Software Architecture
495e8d118ae595818f735575f54ae9dfc7d3ba3b | Pipe and Filter
84bdc2bd63cf969e370a39ac59b868ec417a2527 | Practice Software Architecture in
5ace0db7d3020f709cdd179a87fe6fe408327f10 | Software Architecture in Practice
```

4.2 Saving the indexed data

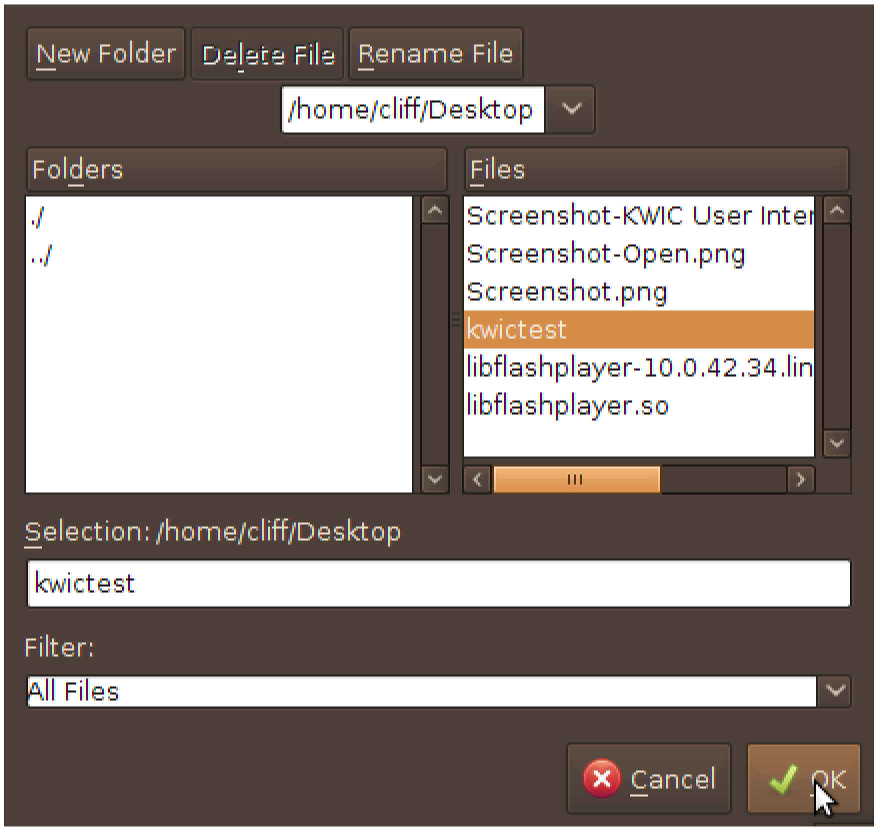
Once input data has been loaded and indexed. The user has the option to save the indexed data to a file. The format of this indexed data will match the specification. To begin saving of the indexed data, expand the *File* menu and select the *Save...* menu item:



```
Pipe and Filter
Software Architecture in Practice
```

```
11843cc34f0d7144a9c6a0571d4ac07a69ccea72 | and Filter Pipe
ac8d1b09c3e5ef96f6d5e9ac842e895548411da6 | Architecture in Practice Software
b341dc24c89e86c26209b4708dc0655d74b950c0 | Filter Pipe and
27f443f49528a84eed39945b9207281abb34dbfa | in Practice Software Architecture
495e8d118ae595818f735575f54ae9dfc7d3ba3b | Pipe and Filter
84bdc2bd63cf969e370a39ac59b868ec417a2527 | Practice Software Architecture in
5ace0db7d3020f709cdd179a87fe6fe408327f10 | Software Architecture in Practice
```

A file selection dialog will open, allowing the user to either select an existing file or enter a new file name:



Selection: /home/cliff/Desktop

kwictest

Filter:

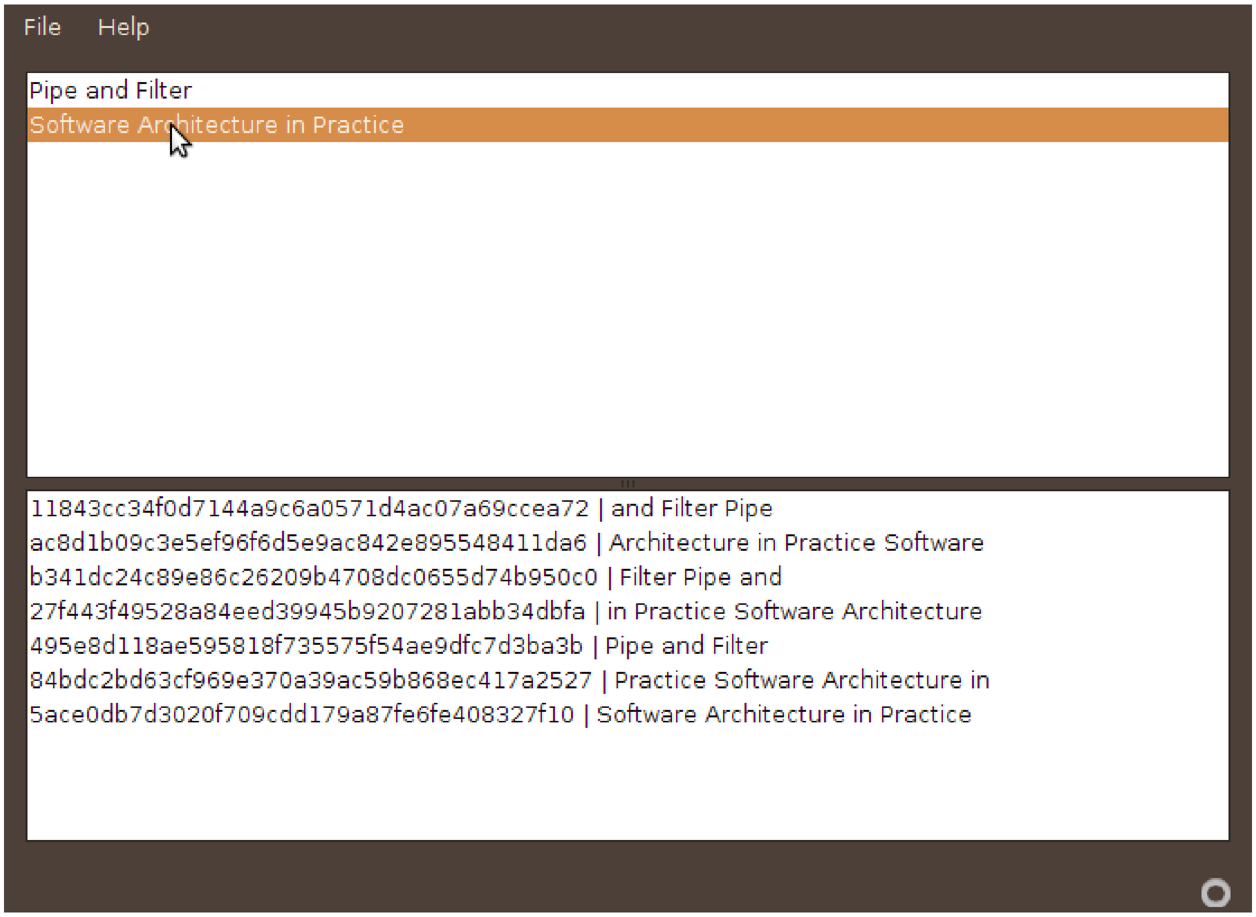
All Files

Once the file is selected, click on the OK button, triggering the save of the indexed data.

4.3 Removing input lines

After an input data file has been loaded, the user can remove input entries. Removing the input entries will also remove their corresponding indexed records. The only way to recall an input line is to load the original file again.

Start by selecting one or more items in the input record list. Select multiple items by holding the Control key to add individual items to the selection or Shift to add a group of items:



Pipe and Filter
Software Architecture in Practice

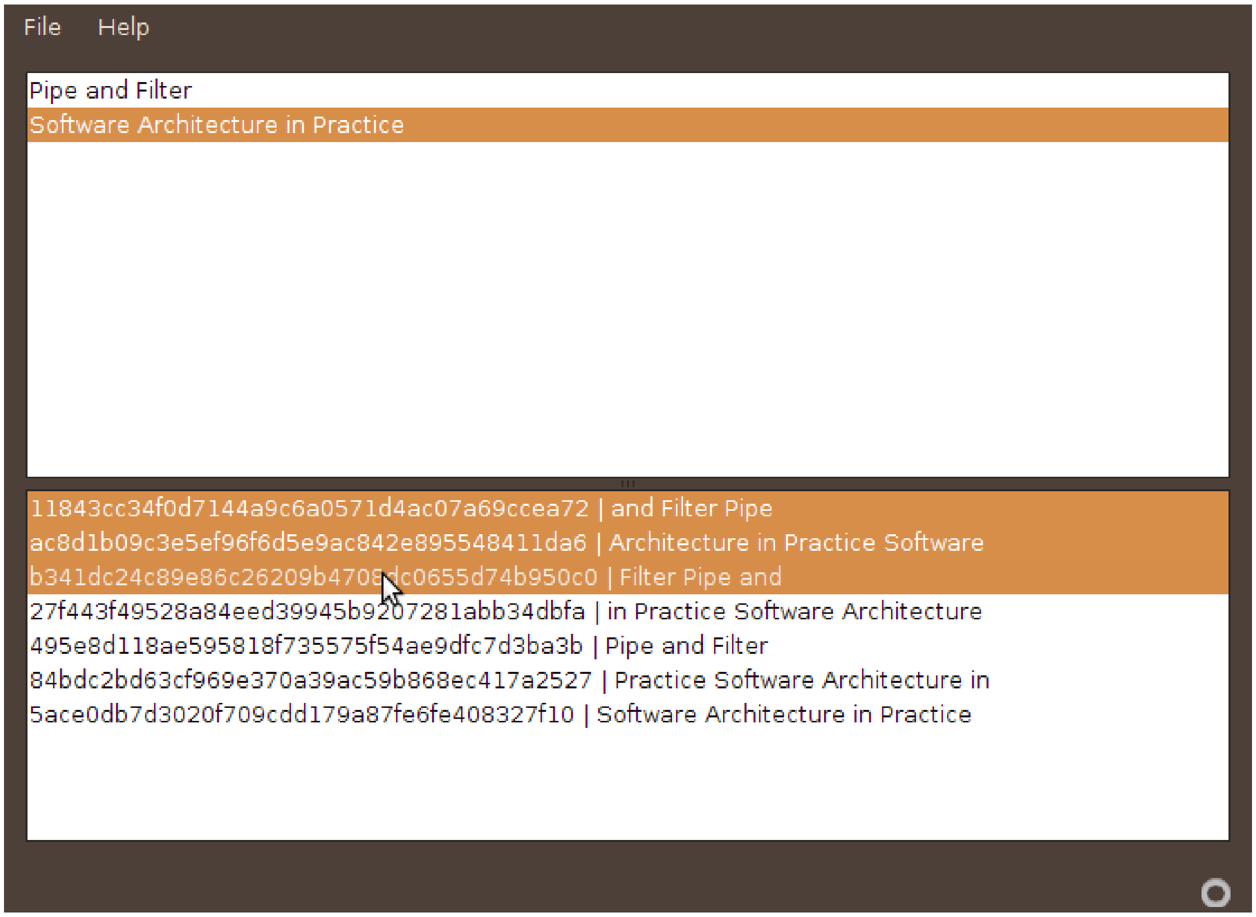
```
11843cc34f0d7144a9c6a0571d4ac07a69ccea72 | and Filter Pipe
ac8d1b09c3e5ef96f6d5e9ac842e895548411da6 | Architecture in Practice Software
b341dc24c89e86c26209b4708dc0655d74b950c0 | Filter Pipe and
27f443f49528a84eed39945b9207281abb34dbfa | in Practice Software Architecture
495e8d118ae595818f735575f54ae9dfc7d3ba3b | Pipe and Filter
84bdc2bd63cf969e370a39ac59b868ec417a2527 | Practice Software Architecture in
5ace0db7d3020f709cdd179a87fe6fe408327f10 | Software Architecture in Practice
```

Right click on the selection and select the *Remove...* menu item. Once removed, the KWIC GUI will begin removing the input records and the corresponding indexed records.

4.4 Removing indexed data

After an input data file has been loaded, the user can remove indexed entries. Removing the indexed entries will not remove their corresponding input records or other indexed records corresponding to the original input record. The only way to recall an indexed line is to load the original input file again.

Start by selecting one or more items in the indexed record list. Select multiple items by holding the Control key to add individual items to the selection or Shift to add a group of items:



```
Pipe and Filter
Software Architecture in Practice
```

```
11843cc34f0d7144a9c6a0571d4ac07a69ccea72 | and Filter Pipe
ac8d1b09c3e5ef96f6d5e9ac842e895548411da6 | Architecture in Practice Software
b341dc24c89e86c26209b4708dc0655d74b950c0 | Filter Pipe and
27f443f49528a84eed39945b9207281abb34dbfa | in Practice Software Architecture
495e8d118ae595818f735575f54ae9dfc7d3ba3b | Pipe and Filter
84bdc2bd63cf969e370a39ac59b868ec417a2527 | Practice Software Architecture in
5ace0db7d3020f709cdd179a87fe6fe408327f10 | Software Architecture in Practice
```

Right click on the selection and select the *Remove...* menu item. Once removed, the KWIC GUI will begin removing the indexed records.