Cyber Miner KWIC Indexing System

Final Phase I

Yu Bai yxb121530
Marco Mereu mxm168730
Caroline Wigginton caw150730

http://utdallas.edu/~yxb121530/

Table of Contents

1. Introduction

This section provides an overview of the information provided by this document by detailing the project's purpose, scope, and providing a list of definitions.

1.1 Purpose

The purpose of this document is to describe the requirements and architecture styles of a KWIC indexing system. This document also serves to further the development team's understanding of the problem domain and facilitate the development of said system.

1.2 Scope

Cyber Miner is a KWIC indexing system which builds indices for a future implementation of a web search engine. Cyber Miner currently receives input from a text file of strings separated by a dollar sign ('$') and prints the indices to the console after the file is circularly shifted and alphabetized.

1.3 Definitions, Acronyms, and Abbreviations

- KWIC: Key Word in Context
- UML: Unified Modeling Language

1.4 Project Deliverables

- Phase I
    - Phase 1.1: Interim Project I
        - Deliverables: Preliminary Project Specification and Presentation
        - Due Date: June 14
        - Team Leader: Yu Bai
    - Phase 1.2: Final Part I
        - Deliverables: Finalized Project Specification and working indexing system
        - Due Date: June 25
        - Team Leader: Caroline Wigginton
- Phase II
    - Phase 2.1 Interim Project II
        - Deliverables: Project Specification and working search engine system
        - Due Date: July 12
        - Team Leader: Marco Mereu
    - Phase 2.2 Final Project
        - Deliverables: Presentation and system demo
        - Due Date: July 26
        - Team Leader: Caroline Wigginton

2. Process Architecture

This section describes the team composition and each team member's assigned roles. *Table 1* details the roles of each member.

Team Member	Roles
Yu Bai	Architect Requirements Engineer
Marco Mereu	Project Manager Quality Assurance Engineer
Caroline Wigginton	Architect Development Engineer

Table 1. Team Roles.

- **Requirements Enginee**r: Responsible for gathering and defining the functional and nonfunctional requirements
- **Architect**: Responsible for delivering the system's conceptual architecture design for implementation
- **Development Engineer**: Responsible for software development
- **Quality Assurance Engineer**: Responsible for verifying the architectural design matches the requirements specification. This is a continuous process that ends only when the system is fully developed and shipped
- **Project Manager**: Responsible for keeping track of all the technical and team related development issues

3. Requirements Specification

This section describes the requirements and features of the Cyber Miner Indexing System to illustrate the system's processes and its actors to the reader.

3.1.1 Functional Requirements

- FR1.0 – The system shall provide a text file to accept an ordered set of lines.
- FR2.0 – The system shall accept an ordered set of lines.
    - FR2.1 – Each ordered set of lines shall be an ordered set of words.
    - FR2.2 – Each ordered set of words shall be an ordered set of characters.
- FR3.0 – The system shall circularly shift each line by repeatedly removing the first word and appending it to the end of the line.
- FR4.0 – The system shall output a list of all circular shifts of lines in ascending alphabetical order.

3.1.2 Non-Functional Requirements

- NFR1.0 – Modifiability: *capable of partial or minor changes*

    - Classes are easy to understand and can be changed
- NFR2.0 – Maintainability: *ease at which components can be added or replaced*
    - Modular classes with minimal or no coupling
- NFR3.0 – Performance: *short response time*
    - Input lines take < 3 seconds to be output
- NFR4.0 – Reusability: *extent of possible component reuse*
    - Several methods were able to be slightly modified and reused in other classes
- NFR5.0 – Intuitiveness: *easily understood*
    - Simple user interface with very few options

4. Architectural Specification

This section lists and describes three styles considered for the KWIC Indexing System.

4.1 Pipe and Filter Architectural Style

The KWIC architecture using the pipe and filter style has four components: input, circular shift, alphabetizer, and output. Each component is a "filter" and data is sent between filters through a "pipe". Each filter acts independently and is unaware of the existence of other filters.

The output of each filter becomes the input for the following filter. Input, the first filter, reads data from the input medium which is then sent to circular shift. Circular Shifter generates all the circular shifts from input and sends its output to Alphabetizer which then sorts its input into alphabetical order. The alphabetized circular shifts are then written to the output medium. This process is outlined in *Figure 1*.

- *Advantages:*
    - Simple implementation
    - Allows for reuse as each component operates in isolation
    - Easy to maintain and enhance

- *Disadvantages:*
    - Processing limited due to batch style
    - Inefficient use of space
    - Difficult to implement systems that require user interaction

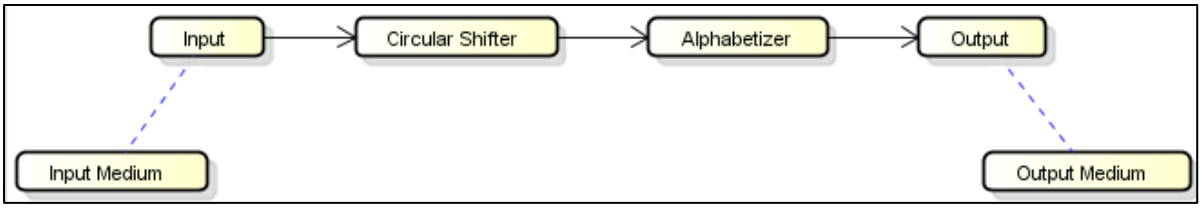

Figure 1. Pipe and Filter Architectural Style

4.2 Implicit Invocation Architectural Style

Implicit invocation has master control and control components in addition to it other components: input, circular shift, alphabetizer, and output. Input reads from the input medium which passes data to insert on the control module which stores the input lines. Control then implicitly invokes setup on circular shift which then retrieves the input lines from control.

Once circularly shifted, the lines are then passed back to the control module. Control then implicitly invokes setup on Alphabetizer which retrieves the circularly shifted lines. Alphabetizer then sends the alphabetized and circularly shifted lines back to control. Control implicitly invokes setup on the output component and output retrieves the alphabetized lines and sends them to the output medium. Master control explicitly invokes the input and output components. This process is illustrated in *Figure 2.*

- *Advantages:*
  - Reusable as components that are invoked implicitly only rely on trigger events.
  - Loose coupling
  - Easy to enhance and add new components
  - Modifiable as trigger events can be changed (e.g. triggering after one line is read or when all lines have been read)

- *Disadvantages:*
  - Difficult to understand
  - Can require more space performance due to the buffers in control

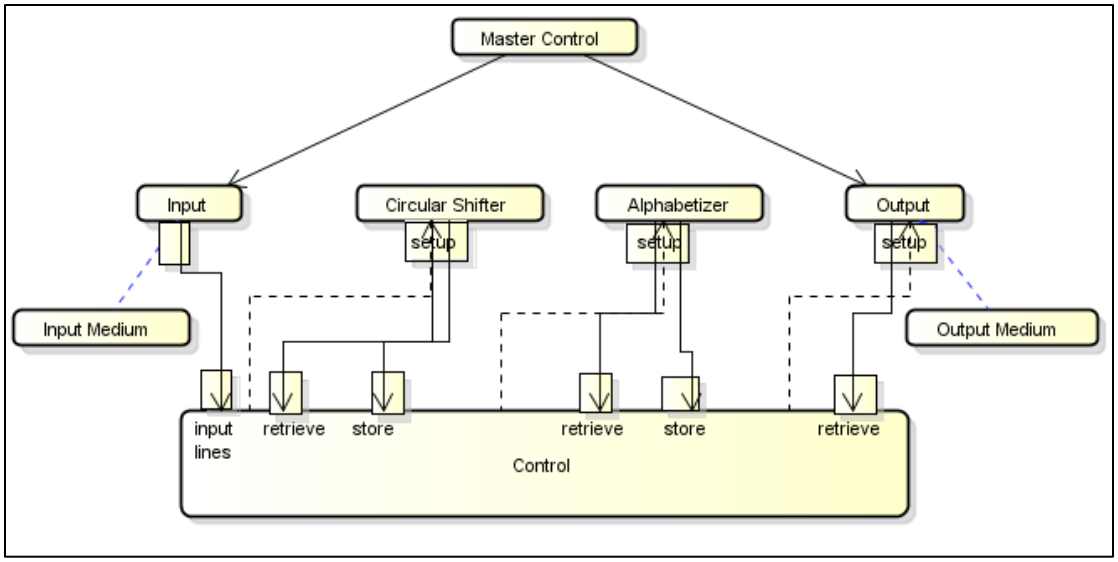

Figure 2. Implicit Invocation Architectural Style.

4.3 Abstract Data Type Architectural Style

The abstract data type was chosen for the KWIC indexing system as it has the following advantages:

- Allows for operations on data
- Independence between components/modular
- Supports reuse with loose coupling

ADT style allows for operations to be encapsulated in their respective classes and each object provides interfaces for other parts of the system. This style contains 5 modules: master control, input, line storage, circular shift, alphabetizer, and output.

Input reads from the input medium and stores the lines by calling the SetChar method of LineStorage. After input has passed control to LineStorage, then LineStorage modifies and/or accesses words and lines to prepare for the following class, CircularShift.

CircularShift begins with the Setup method which gets lines from the Char and Word methods of LineStorage. CircularShift's own Setup and SetChar methods then generate the circular shifts and control is then passed to the Alphabetizer through the interfaces Char and Word. Alphabetizer then creates the circularly shifted, alphabetized lines using the Char and Word methods from CircularShift. The output module then accesses the i-th method on the Alphabetizer and sends the output to the output medium. This process is depicted in *Figure 3.*

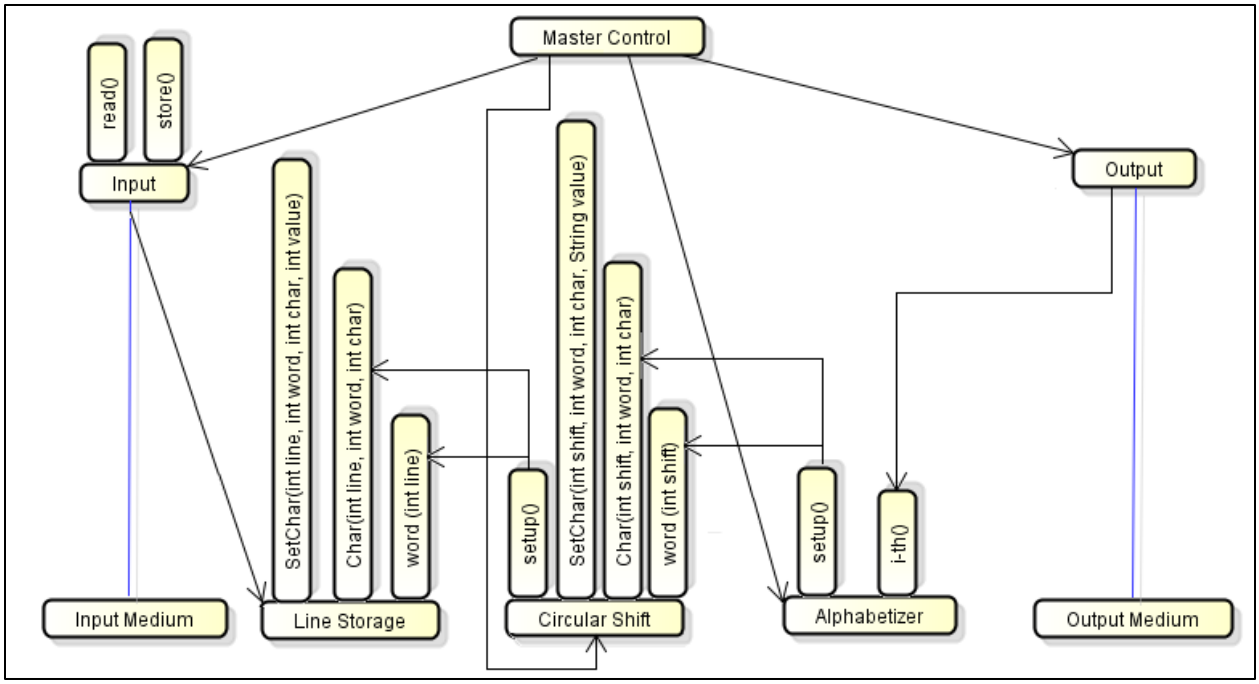

Figure 3. Abstract Data Type Architectural Style.

4.4 Tradeoff Analysis

Four architectural styles in *Table 2* are rated from 1 to 5 based on five different nonfunctional requirements. Each nonfunctional requirement has a multiplier depending on the importance of that nonfunctional requirement to the system. The total score of each style varies between 5 (worst) and 75 (best).

	Shared Data	**Abstract Data Type**	**Implicit Invocation**	**Pipe and Filter**
Modifiability × 1	2	4	4	5
Maintainability × 3	3	4	5	3
Performance × 2	4	4	2	1
Reusability × 4	1	4	4	5
Intuitiveness × 5	2	4	3	4
Total	**33**	**60**	**54**	**56**

Table 2. Tradeoff Analysis.

5. Prototype Screenshots

This section shows sample input and output of the Cyber Miner KWIC Indexing System. *Figure 4* is a tested text file used as an input medium for the system. *Figure 5* is a list of the circularly shifted lines from the input in *Figure 4*. *Figure 6* is the circularly shifted lines from *Figure 5* alphabetized through the alphabetizer module.

```
hello$reading from the file$wow!$10/10$cool$sup dawg$A B C D$i like cats
```

Figure 4. Sample Input Text File.

```
---------CIRCULAR SHIFTS----------
hello
reading from the file
from the file reading
the file reading from
file reading from the
wow!
10/10
cool
sup dawg
dawg sup
A B C D
B C D A
C D A B
D A B C
i like cats
like cats i
cats i like
```

Figure 5. Circular Shift Output.

```
---------ALPHABETIZED SHIFTS----------
10/10
A B C D
B C D A
C D A B
cats i like
cool
D A B C
dawg sup
file reading from the
from the file reading
hello
i like cats
like cats i
reading from the file
sup dawg
the file reading from
wow!
```

Figure 6. Alphabetizer Output.

6. User Manual

The source code for our KWIC Indexing System is available on our team website and can be run in Visual Studio with the .NET framework installed.

Step 1. Open the text file and input a string with each line separated by a '$' delimiter as shown in *Figure 7.*

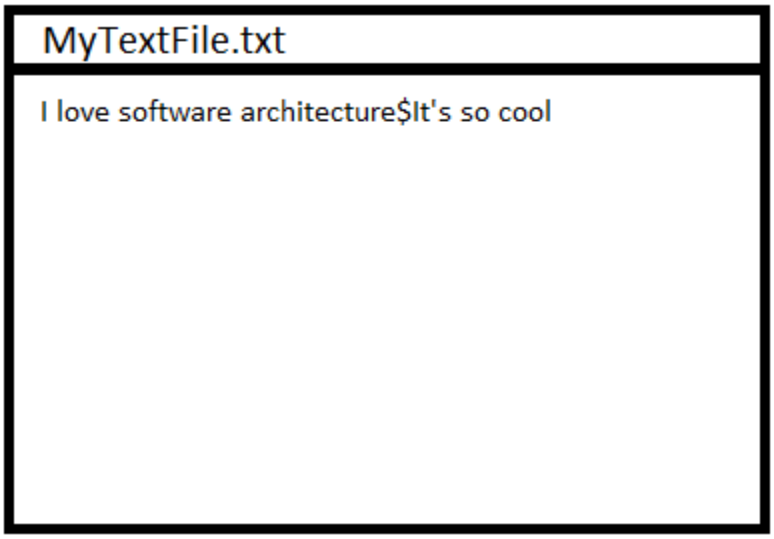

Figure 7. Text File Mock-up.

Step 2. After entering the desired text, run the program by hitting F5 or pressing the Start button shown in *Figure 8.*

Figure 8. Visual Studio Start Option.

Step 3. A console window will open displaying the circularly shifted and alphabetized lines as shown in *Figure 9.*

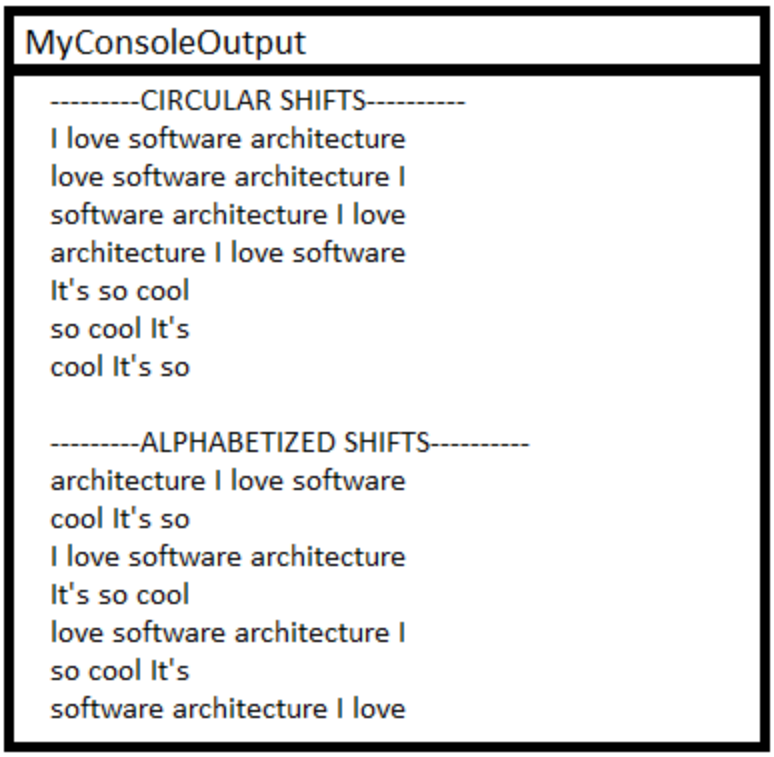

Figure 9. Console Output Mock-Up.

7. References

[1] Mary Shaw, David Garlan. “Software Architecture: Perspectives on an Emerging Discipline.” Upper Saddle River, New Jersey: Simon & Schuster, 1996.

[2] Course SE 4352 material (https://www.utdallas.edu/~chung/CS4352/syllabus.htm)