Contents

[Part 2 – Mini-Project to Evaluate Architectural Patterns (Styles) 3](#_heading=h.1fob9te)

[Total Points and Due Date 3](#_heading=h.3znysh7)

[Which of the three architectural styles better facilitate a particular change 4](#_heading=h.2et92p0)

[Architectural Changes 5](#_heading=h.tyjcwt)

[Main program and subroutines – architecture for part 1 5](#_heading=h.3dy6vkm)

[Main program and subroutines – architecture for part 2 5](#_heading=h.1t3h5sf)

[Object-oriented style – architecture for part 1 6](#_heading=h.4d34og8)

[Object-oriented style – architecture for part 2 7](#_heading=h.2s8eyo1)

[Pipe-and-filter – architecture for part 1 8](#_heading=h.3rdcrjn)

[Pipe-and-filter – architecture for part 2 9](#_heading=h.26in1rg)

[Conclusions 10](#_heading=h.2jxsxqh)

[Summary 12](#_heading=h.2p2csry)

# 

# Part 2 – Mini-Project to Evaluate Architectural Patterns (Styles)

For this project, you need to make certain changes to the architectures of the earlier versions of the project and to describe which of the three architectural styles better facilitate a particular change. The proposed changes are as follows:

* Make the system interactive: it interactively inputs a line at a time and upon demand outputs an alphabetized list of its current circular shifts. The interactive prompt should provide for an option line like:

Add (A/a), Display (D/d), Quit (Q,q):

When D is chosen, all output lines should be displayed.

* Rather than simply outputting an alphabetic list of all the circular shifts, the output routine eliminates articles and trivial words such as “a”, “an”, “the,”, “and,” or capitalized version of these words.

In your presentation, you need to more precisely address the earlier questions and be able to support and justify your answers.

# Total Points and Due Date

Total Points: 100 (70 for the design and algorithms; 30 for presentation)

Tentative Due Date: November 29, 2021

# Which of the three architectural styles better facilitate a particular change

This project will compare three architectural styles on an interactive Key Word in Context (KWIC) effort. It will accept a line of input from a user, eliminate trivial words such as “a”, “an”, “the” and “and”, as well as capitalized versions of these words. The application will use a file containing these trivial words, allowing easy maintenance of the trivial word list.

The application will then process the stripped line, first by outputting it in its original format, then shifting the last word to the beginning of the text line and outputting the adjusted text line. This process is repeated until all non-trivial words have been shifted to the beginning position.

Three commands will be available to the user. “A” or “a”, known as the “Add Command”, will accept the input line into the application. “D” or “d”, known as the “Display Command”, will display all lines. The entire set of text lines will be alphabetized providing an alphabetized list of all words in the first position of the line, making it easy to identify all words and phrases in the text file. The “Q” or “q”, known as the “Quit Command, will exit out of the application.

Summary of functionality changes:

* Make application interactive (Add/Display/Quit)
* Eliminate articles and trivial words (Elimination Set).

This project will use the following three architectural styles:

* Main program and subroutines
* Object-oriented style
* Pipe-and-filter.

It will compare these styles across six categories:

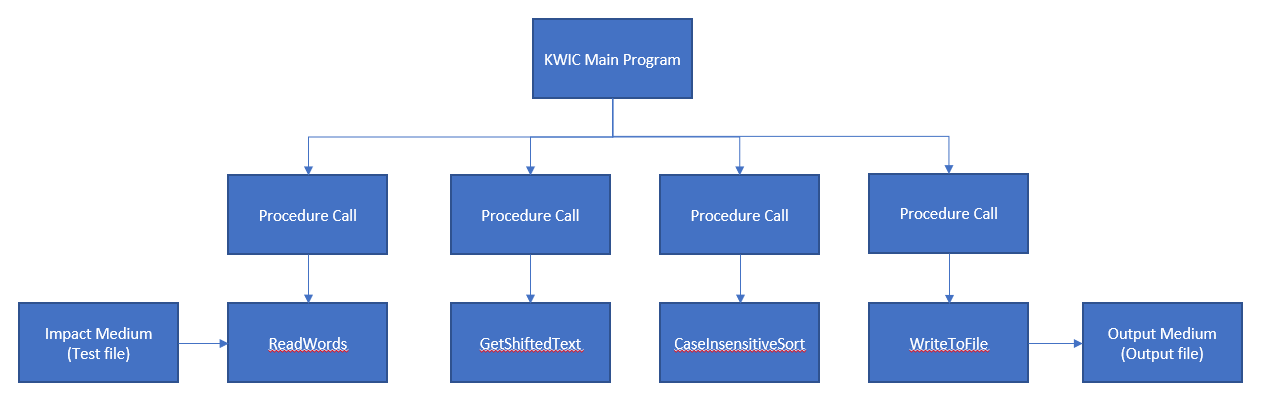
* Change in algorithm
* Change in data representations
* Change in functionality
* Performance in time
* Performance in space
* Reusability.

# Architectural Changes

Below are graphic representations of the three architectural styles as created in the original “part 1” project and updated for “part 2”.

## Main program and subroutines – architecture for part 1

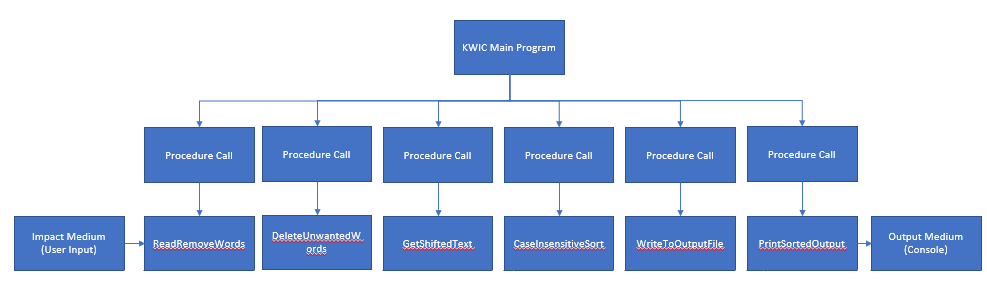
The architectural style of main program and subroutines is a traditional language-influenced style that is the result of using older, traditional programming languages, such as Basic, COBOL, C, C++, Java and Pascal. In this architecture, the functionality was divided into different modules which were called through a procedure call from the main program. The input was provided from the text file, once the sentences were shifted and sorted, they were written to output text file.



## Main program and subroutines – architecture for part 2

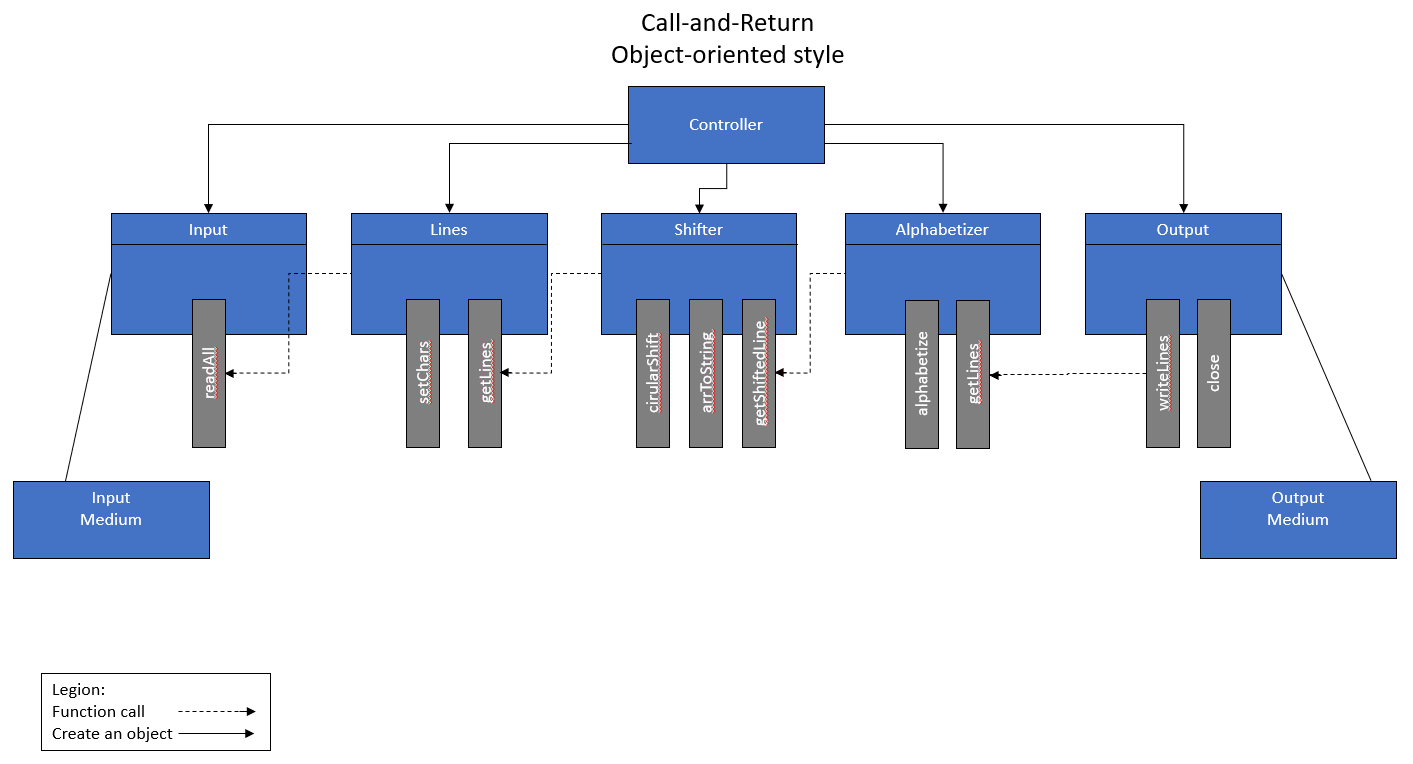
The new architecture of the main program and subroutines architectural style  
needed the inputs from the user instead of reading it from the text file. Hence to  
accommodate the required functionality based upon add, display, and quit options  
that were given to the users, new changes were added in the main program. The  
text was shifted and sorted when a user enters option ‘a/A’, the sorted output was  
displayed on the console when the user enters option ‘d/D’, and program quits once  
the user enters option ‘q/Q’ and before quitting writes the sorted words to the  
output file.  
There was also a need to remove some unnecessary words from the sorted text   
before they are displayed hence to implement that feature two modules or   
procedures were created in the program which was called from the main function.

Another important change was to display the output on the console window rather   
than adding the sorted output to the output text file. Hence even for that change, a   
new module was also added and called from the main once the user enters the   
display option.



## Object-oriented style – architecture for part 1

The architectural style of object-oriented is a style in which the state of an object is strongly encapsulated with the functions that interact with the state of the objects. Typical programming languages would include C++ and Java.

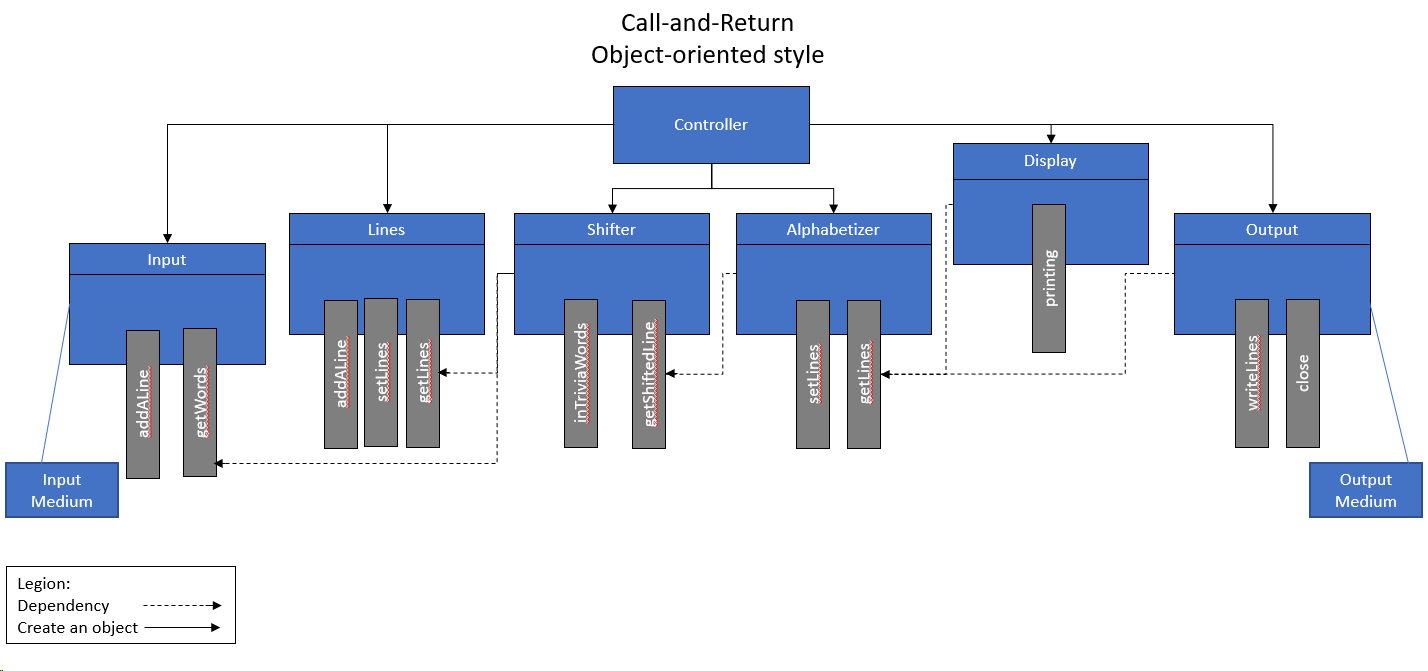


## Object-oriented style – architecture for part 2

The new architecture required a new class to help present the output to the user. It takes the sorted lines from alphabetizer objects and prints them into the console. Also, a new function has been added in the Shifter class to check for trivia words and delete them. Input class has been reused to read the trivia words file. The interactive feature has been added to the controller class, it has a new private function to print the menu to the user, also the main function has conditional sections depending on the user selection.

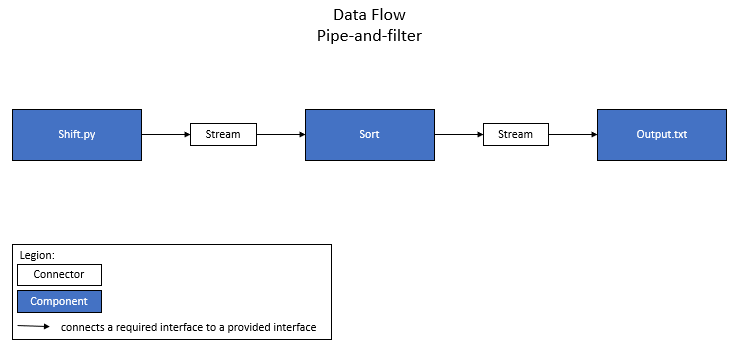
I want to talk more about the elimination of trivial words. I was thinking of two choices, either I add a new module to iterate over the line words and remove the trivial words before passing the line to the "Shifter" class or add a new function in the "Shifter" class to do that.

The first option will be better for reusing purposes but since we already have iteration over the words in the "Shifter" class so the second option of adding a new function in the “Shifter” class will provide better performance. I chose the second way because we are not expecting extra work on this project so performance is more important than reusability.



## Pipe-and-filter – architecture for part 1

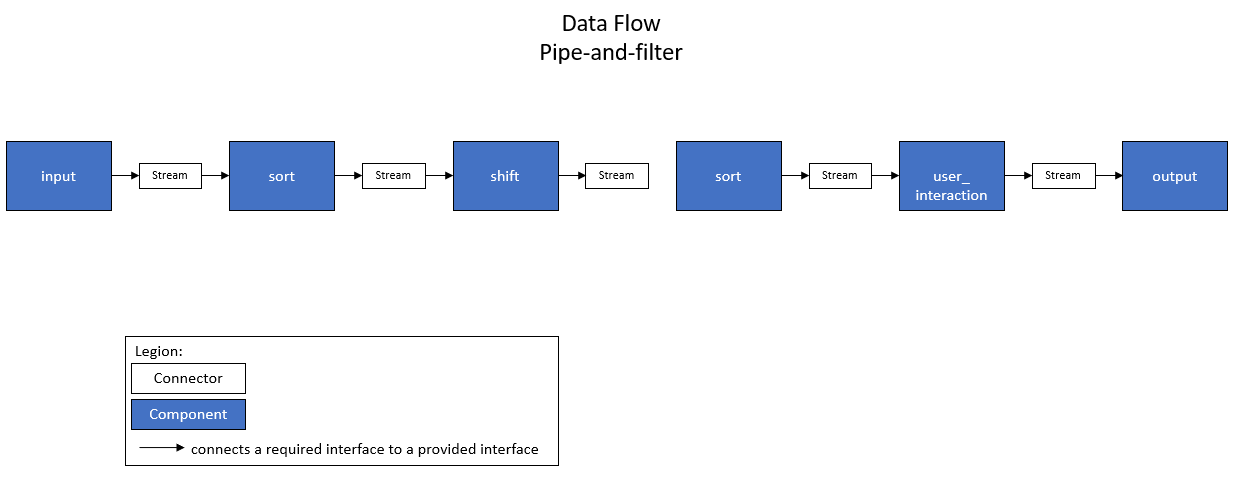
The architectural style of pipe-and-filter allows separate programs to executed potentially concurrently while data is being passed as a stream from the initial program to next program, both executing at the same time. This style is often used in operating systems and Unix shells. It uses mutually independent filtering processes to incrementally alter incoming data into output. This style is often used to couple two and more filtering process together and can run as a concurrent process.



## Pipe-and-filter – architecture for part 2

The new architecture required two new filters. One needed to be written, which was called user\_interaction.py. This one added the functionality of user interaction with the system. The second filter was adding another sort filter, which added the functionality of removing a predefined dictionary of words from the original text file.

Command to run with the new program: sort -u dictionary.txt input.txt | python3 shift.py | sort -f > output3.txt | python3 user\_interaction.py. The new architecture for this can be found in the diagram below.



The first sort filter deletes the “unimportant” words from the text file, then saves this to a new file. The shift filter shifts the sentences according to the KWIC requirements and saves this to a file for testing purposes. The second sort filter sorts this new output from the shift output. Finally, the user interaction filter adds the user interaction element of the program where the user is given the choice to add, display, or quit.

However, this new architecture returns an EOF error that the developers have not found a way to resolve through their research. All of the architecture, until the final filter, works as designed. But, the final filter works on its own, but it does not work when added to the larger program. The error is being handled, but a way to resolve the error has not been found.

# Conclusions

Main program and subroutines are easy to maintain and efficient with their time and space.

*Change in algorithm:* Change in the algorithm was not simple. To accommodate the   
algorithm changes, additional code was added to the main program. As well as   
some old code i.e., subroutine definition and subroutine call were removed from the  
main function.

*Change in data representation:* Change in the data representation was also not   
easy. To implement the changes in data representation, three new modules were   
added which actually increased the size of the source code.

*Change in functionality:* Change in functionality was not impacted much as the final   
output was showing the shifted and sorted text which is the same as the old   
program.

*Reusability*: It is difficult to reuse this code as all the implementation is in the same   
file and there are no separate modules. If the new program that wants to reuse this   
program has identical requirements of this project it can reuse the source file, else   
it can’t.

Object-orientedprovides an ease of changing data and structures but are not necessarily efficient in use of resources.

*Change in algorithm:* Changing the algorithm was simple and easy to perform. In this exercise, the interactive feature has been simply added to the controller class by adding some conditional sections to execute the user needs. Also, a new function to “Lines” had to be modified to add a line at a time instead of reading all the lines.

*Change in data representation:* Changing in data representation is also easy to handle. It can be performed with some modification in one affected module, and the other can work fine. For example, if we need to write the output into a JSON file instead of a text file, we can modify the “Output” class, and that is the only part that could be affected.

*Change in functionality:* Changing in functionality is not very simple and it requires some extra work. In this example, it was hard to be added to an existing module so we needed one more module to print the output into the console, which was pretty simple but it will affect the performance.

*Reusability:* Reusability is supported in object-oriented and some modules of our object have been reused depending on the new requirements we got.

Pipe-and-filters are simple and easy to maintain but are limited when it comes to complex application needs.

*Change in algorithm:* Changing the algorithm was simple and easy to perform. All that needed to be done was adding more filters. For example, in this exercise, two filters needed to be added to the architecture. One of the filters was already written, making this change easier.

*Change in data representation:* Changing the data representation would only require a new filter, making it a simple task. As long as the data can be converted into the required format, the filter can be added to make this change as needed.

*Change in functionality:* Changing the functionality would be simply adding a new filter or modifying an existing filter. For example, the functionality of this program was easily changed by adding two filters, one of which was already written.

*Reusability:* As can be seen in this example, this architecture allows for a high level of reusability. The developers were able to use the sort function twice to complete two different tasks at different times in the execution of the program by utilizing the different built-in options available with the sort function.

Performance Comparison

| **Performance Evaluations** | **Main and subroutines** | **OO** | **Pipe-and-filters** |
| --- | --- | --- | --- |
| Performance: |  |  |  |
| Size of object code | 703 KB | 104 KB | 11 KB |
| Execution Time1  (3A)+D+Q | 46 ms | 29 ms | X |
| Execution Time 2  (3A)+D+Q | 52 ms | 33ms | X |
| Execution Time 3  (3A)+D+Q | 43 ms | 30 ms | X |
| Average Exec Time  (Execution Time 1 + Execution Time 2 + Execution Time 3) / 3 | 47 ms | 30.7 ms | X |
| Memory | 7.5 MB | 39.4 MB | X |

# Summary

The researchers found the pipe-and-filter architecture to be a bad choice for adding user interaction functionality. This architecture does not work well because the interaction filter expects input from the user, but because of the nature of pipe-and-filter, the information being sent through the pipe is used as the interaction element. Because the architecture is linear and not modular and the first filters must be run before the interaction element, the interaction portion cannot be used in the beginning of the architecture. The researchers found no way around this issue.

The researchers also found that the object-oriented performed better than the main-subroutine in all three areas: size of the object code, the execution time and memory.

