

On the Criteria To Be Used in Decomposing Systems into Modules

By D.L. Parnas

Outline

- Introduction
- Perspective
- Approaches
- Example System
- Modularizations
- Comparison
- Conclusion

Introduction

What is modular programming?

"A well-defined segmentation of the project effort ensures system modularity. Each task forms a separate, distinct program module. At implementation time each module and its inputs and outputs are well-defined, there is no confusion in the intended interface with other system modules. At checkout time the integrity of the module is tested independently; there are few scheduling problems in synchronizing the completion of several tasks before checkout can begin. Finally, the system is maintained in modular fashion; system errors and deficiencies can be traced to specific modules, thus limiting the scope of detailed error searching."

Designing Systems Programs, 1970

Introduction

Important Points:

- Each task is created as a separate module
- Each module includes well-defined input and output
- Each module is tested individually
- The system is maintained in a modular fashion

Introduction

Important Points:

- Each task is created as a separate module
- Each module includes well-defined input and output
- Each module is tested individually
- The system is maintained in a modular fashion

Introduction

Important Points:

- Each task is created as a separate module

This was the conventional approach during the early 1970s

What was the state of the art at the time?

Perspective

Important Events:

1971:

- UNIX operating system is released

1972:

- Dennis Ritchie at Bell Labs creates the C programming language
- Atari introduces 'Pong' video game system
- "Decomposing Systems into Modules" published

1973:

- Bob Metcalfe at Xerox PARC invents Ethernet

Approaches

The functional approach:

- Referred to as "conventional"
- The system design can be generated from a flowchart
- Each step in the flowchart becomes a separate module

Approaches

The information hiding approach:

- Referred to as "unconventional"
- Each module has certain knowledge about the system
- The interface of each module is designed to hide the inner workings

Example System

KWIC (Key Words In Context)

- The KWIC system accepts an ordered set of lines
- Each line is an ordered set of words
- Each word is an ordered set of characters

Example System

KWIC (Key Words In Context)

- Any line may be "circularly shifted" by repeatedly removing the first word and appending it at the end of the line
- The KWIC system outputs a listing of all circular shifts of all lines in alphabetical order

Modularizations

How is this broken down?

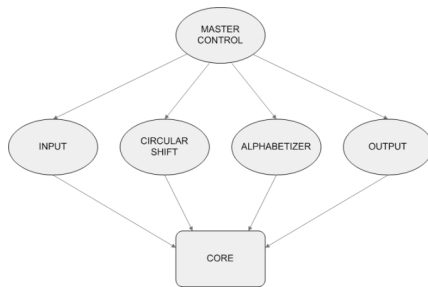
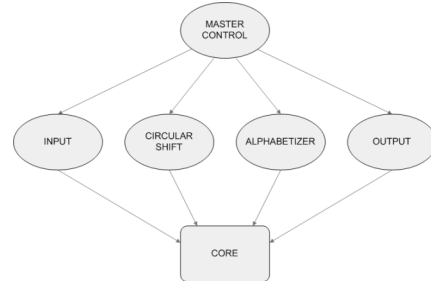
- Functional Modularization
- Information Hiding Modularization

Functional Modularization

Modules:

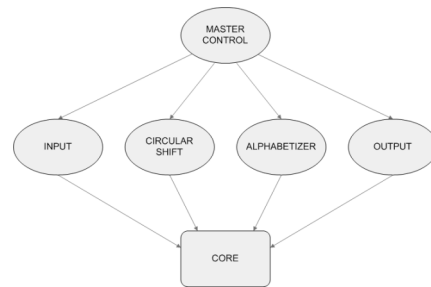
- Input
- Circular Shift
- Alphabetizing
- Output
- Master Control

Functional Modularization



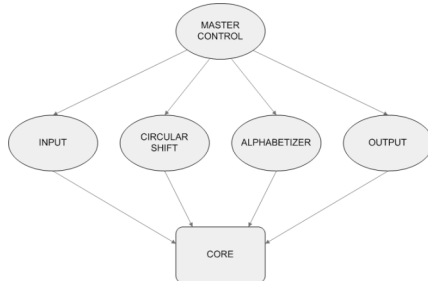
INPUT Module:

- Reads the data lines from the input medium and stores them in core
- Packs characters four to a word
- Creates an index for the starting address of each line



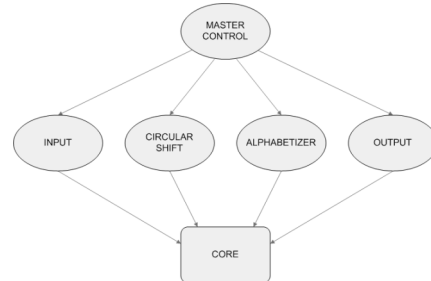
CIRCULAR SHIFT Module:

- Creates an index for the address of the first character of each circular shift
- Outputs words in pairs to core



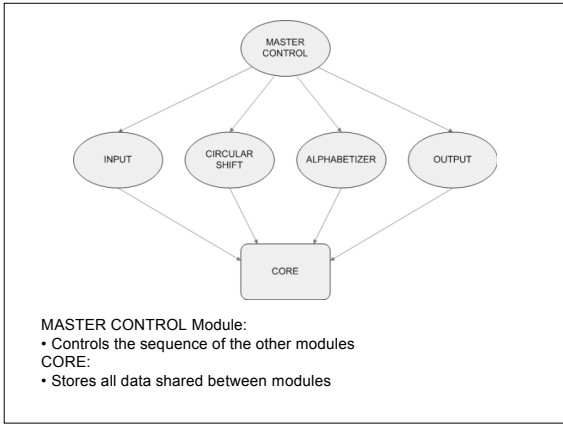
ALPHABETIZING Module:

- Uses data created by Input and Circular Shift modules
- Produces an alphabetized array in the same format as Circular Shift



OUTPUT Module:

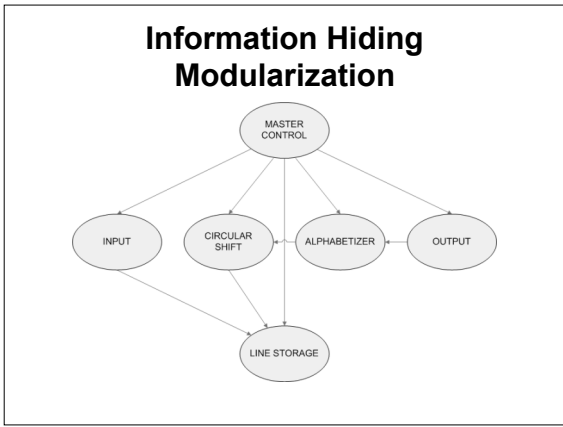
- Uses arrays created by Input and Alphabetizing modules
- Outputs list of all circular shifts



Information Hiding Modularization

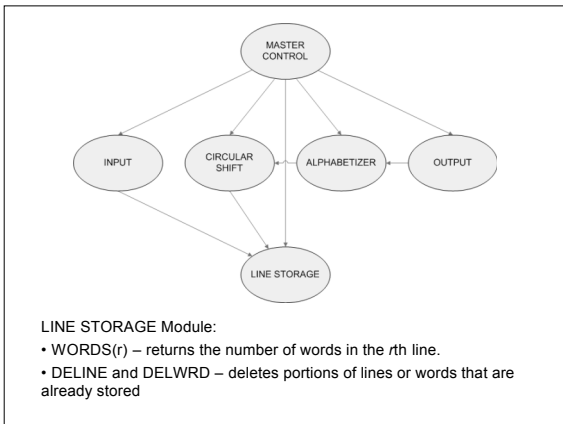
Modules:

- Line Storage
- Input
- Circular Shifter
- Alphabetizer
- Output
- Master Control



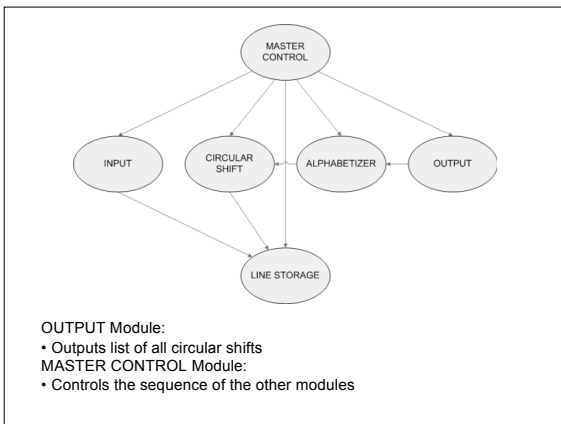
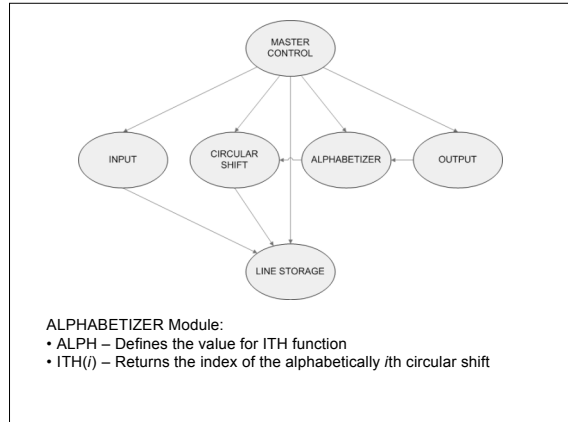
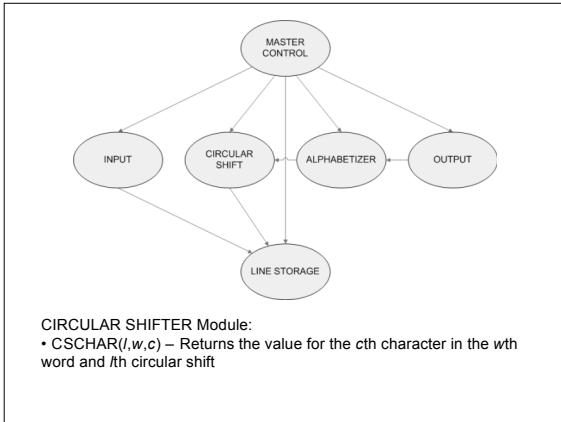
LINE STORAGE Module:

- $CHAR(r, w, c)$ – returns the value for the c th character of the r th line and w th word
- $SETCHAR(r, w, c, d)$ – sets the value for the c th character of the r th line and w th word to the character value for d



INPUT Module:

- Reads the data lines from the input medium and calls the Line Storage module to store them



Comparison

Similarities:

- The two modularizations both work and both reduce the problem into manageable parts
- The algorithms for each may be the same
- The resulting programs may function the same

Comparison

Differences:

- Changeability
- Comprehensibility

Changeability

What if one of the design choices changed?

- Input format
- Storing all lines in core
- Packing characters four to a word
- Using an index for circular shift versus changing stored data for shifts
- Alphabetizing the list once

Changeability

1. Input format –
Functional: Input
Information Hiding: Input
2. Storing all lines in core –
Functional: All Modules
Information Hiding: Line Storage

Changeability

3. Packing characters four to a word –
Functional: All Modules
Information Hiding: Line Storage
4. Changing stored data for shifts –
Functional:
Alphabetizer, Circular Shift, and Output
Information Hiding: Circular Shift

Changeability

5. Alphabetizing the list once –
Functional: Alphabetizer, Output
Information Hiding: Alphabetizer

Changeability

Functional: most changes affect all modules
Information Hiding: each change affects only one module

Comprehensibility

- Functional Modularization:
- The formats are complicated
 - This makes design decisions more difficult
 - Interactions between module developers is important

Comprehensibility

- Information Hiding Modularization:
- The interfaces are primarily abstract
 - Interaction between module developers is minimized
 - Independent development can begin earlier

Comprehensibility

Functional Modularization:

- Modules are difficult to understand without knowing the inner workings of related modules

Information Hiding Modularization:

- Modules can be understood primarily without knowledge of other modules

Conclusion

- The functional approach is obsolete
- It is nearly always incorrect to modularize based on a flowchart
- The information hiding approach has a number of advantages in terms of changeability, independent development, and comprehensibility

Resources

- Chung, Lawrence. "Modular Decomposition Issues." www.utdallas.edu/~chung/SA/2.4md2.pdf
- Computer Hope. "Computer History: History for 1960-1980." www.computerhope.com/history/196080.htm
- Liu, Ling. "Decomposing Systems into Modules." <http://www.cs.wm.edu/~coppit/csci780-fall2003/presentations/05-criteria-for-decomposing.pdf>