CS242: Modularization

- Prepared by Charlie Meyer, May 2009
- Updated October 2009
- Examples from ”On The Criteria To Be Used In Decomposing Systems Into Modules” by D.L. Parnas

What is a Module?

- A collection of data and functions
- Elements of a program that logically fit together
- Elements of a program that depend on each other, but do not depend much on other elements (high cohesion, low coupling)
- Elements of a program that hide their implementation details but expose and interface for performing a task (in a well designed, loosely coupled module at least)

What is modularization?

- A mechanism for improving the flexibility, testability, reliability, and readibility of a system
- Shortens development time
- A system must first be planned out then implemented
- A system should be decomposed into modules based off of specific criteria

Benefits of having a modular system

- **Development Time:** since individual modules are independent, they can be developed concurrently
- **Flexability:** individual modules should be relatively independent of others (low coupling), so a change to one module will not impact the entire system
- **Testability:** each module can be independently tested to ensure that it performs its functionality correctly
- **Readability:** each module should be easy to understand (since they only do one thing), so the system should then be easy to understand by understanding each of its modules

Example: KWIC Index System

- Reads in an ordered set of lines, each line is an ordered set of words, each word is an ordered set of characers
- Lines may be circularly shifted by removing the first word and appending it to the end of the line
- The KWIC system outputs a listing of all circular shifts of all lines in alphabetical order

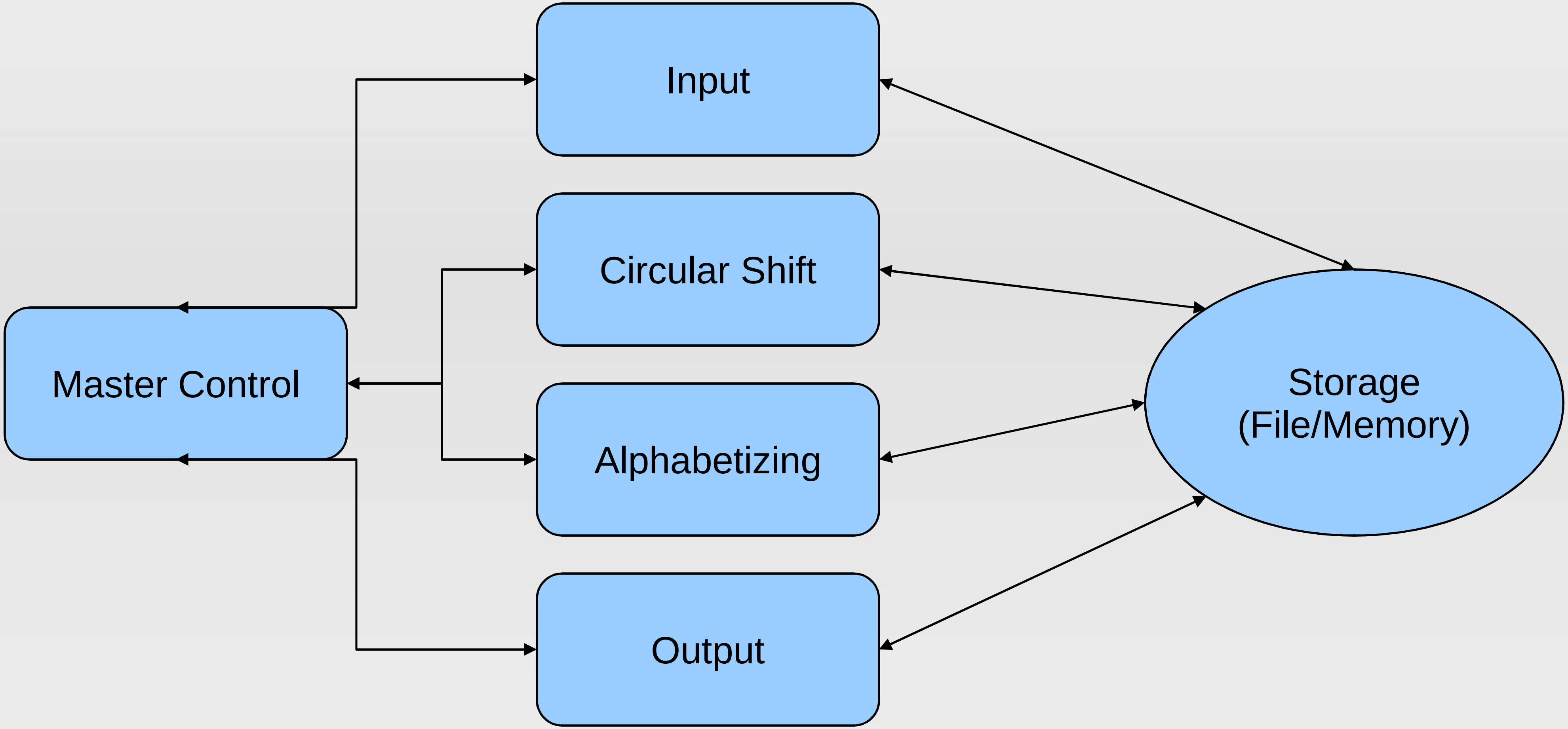
Example: KWIC Index System

- Input:
  - Designing Software for Ease of Construction
  - Cookies are Tasty
- Output:
  - are Tasty Cookies
  - for Ease of Construction Designing Software
  - of Construction Designing Software for Ease
  - Construction Designing Software for Ease of
  - Cookies are Tasty
  - .....

Modularization 1: Classical

- **Input** – reads in data and stores it for use by other modules
- **Circular Shift** – creates a data structure of indicies of the first character of each circular shift for each line and stores it
- **Alphabetizer** – creates a data structure similar to the circular shift module, but this time all the indicies are in alphabetical order
- **Output** – using the data structures from input and alphabetizer, this module outputs the data to console
- **Master Control** – this module controls the sequencing of each of the above modules

Modularization 1: Classical



Modularization 1: Clasical

- Based on a flow chart design, that is, each module does its task after the previous one has completed
- Each module leaves data in a format that the next module accepts and processes
- Each step in processing is its own module
- This has been shown to be the most common modularization for this type of system

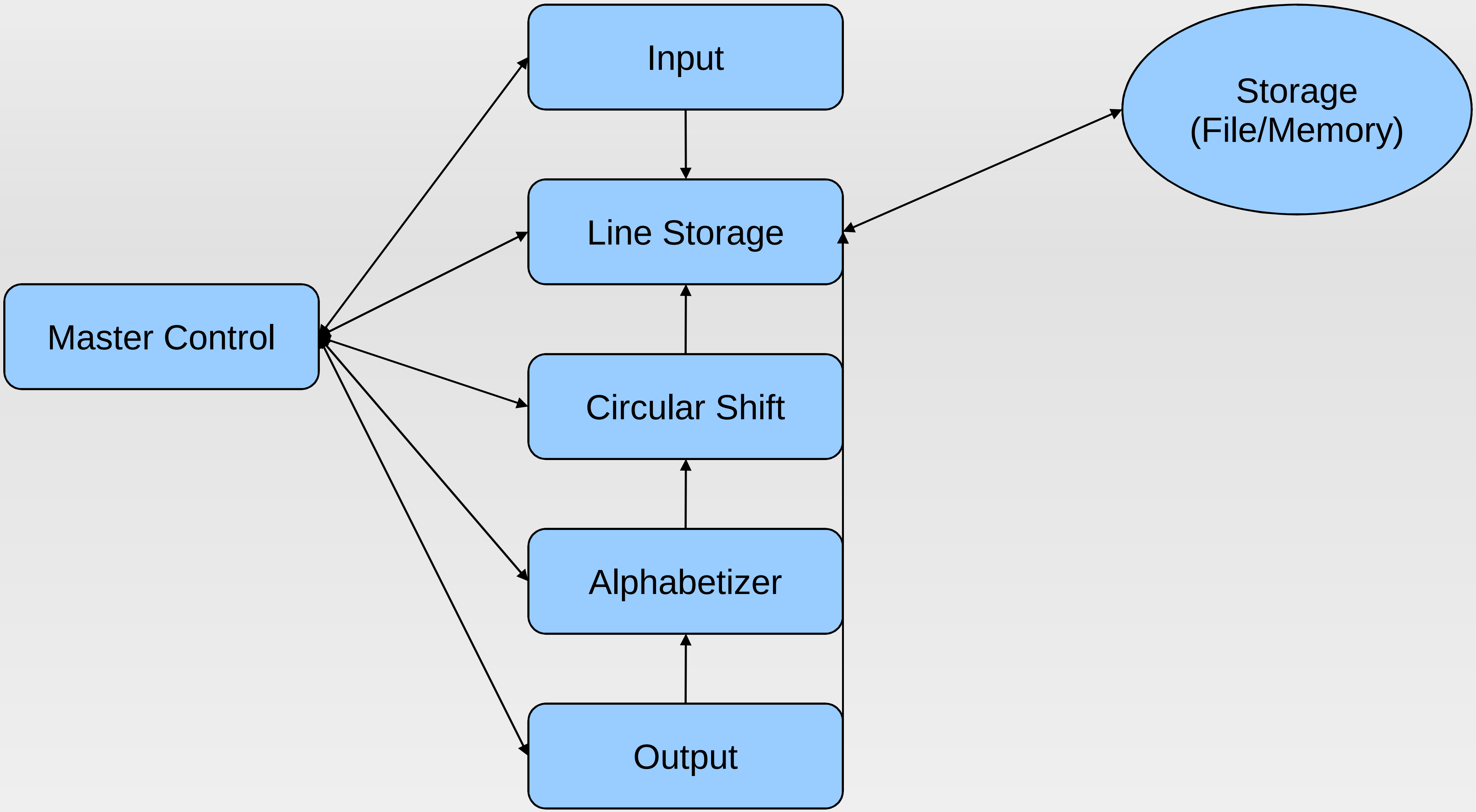
Modularization 2: Information Hiding

- **Line Storage** – contains functionality to store individual characters and retrieve individual characters. Also, has functionality to retrieve the number of words in a given line.
- **Input** – reads lines in and calls the functions of the line storage module
- **Circular Shift –** has a setup function to initialize the module, then a CSCHAR(*i*,*w*,*c*) which provides the cth character of the wth word of the ith circular shift.

Modularization 2: Information Hiding

- **Alphabetizer** – contains a setup method, then a ITH(*i*) which returns the index of the circular shift which comes in the ith alphabetical ordering
- **Output** – prints output to the screen
- **Master Control** – controls the sequencing of the other modules

Modularization 2: Information Hiding



Modularization 2: Information Hiding

- Each module only exposes an interface to the other modules, does not expose its inner workings or data structures
- Arguably easier to understand when first reading code, since all modules have very low coupling and high cohesion

What if we changed the specifications?

- Specification that may change (# *modules impacted by modularization 1, # modules impacted by modularization 2)*:
  - Input format (1,1)
  - All lines stored in memory vs disk (all modules, 1)
  - Each word is 4 characters (all modules, 1)
  - Store each circular shift in memory vs creating indicies for them (3, 1)
- Modularization 2 is much more adaptable to change!

Concurrent Development

- Modularization 1
  - All data structures must be designed before work can proceed since all data structures are shared between modules.
  - Detailed descriptions of each structure must be made available to other developers working on other modules
- Modularization 2
  - Module interfaces must be designed before concurrent work can begin
  - Simple documentation about each function is needed

Summary

- Having a modular system is a good thing (and is a pretty common sense concept with most OO languages)
- Plan out your system before you begin coding, and try to make your design as flexible as possible
- Modules should be loosely coupled, that is, a change to the internals of one module should not greatly impact the operation of other modules in your system

In Class Exercise

- Suppose you had to write a program to solve the 8 queens problem
  - Generate all possible locations of 8 queens on an 8x8 chess board such that no queen can capture any other queen using standard chess moves
- Design all of your modules on paper with people sitting around you
  - Consider
    - Interfaces exposed
    - Data structures used

More things to consider when designing your modules

- What if we wanted to generalize to N queens on a NxN board?
- What if we wanted to store our data on disk vs in memory?
- What if we wanted to add a UI to the system to display the results rather than writing them to disk?
- What if we kept an 8x8 board, but instead of queens we placed 32 knights, 14 bishops, 16 kings, or 8 rooks such that no 2 pieces could capture each other?