KWIC Exercise

- On a subsequent slide, you will be given the description of a simple program for which you will be asked to devise two architectures. For the purposes of this exercise, you should imagine that it is instead a complex software system. That is, you should consider how to divide it into pieces and how the pieces should interact
- The architectures that you devise should be expressed with *box-and-arrow diagrams*, as described on the next slide
- Source: D. Parnas. "On the Criteria to be Used in Decomposing Systems into Modules." *Communications of the ACM,* 15(12):1053-1058, December 1972.

Box and Arrow Diagrams

- The simplest and most popular way of describing a software architecture is with a box and arrow diagram
- Each box corresponds to a component. Components can be active or passive. Active components are units of computation; that is, they actively compute. Passive components are data repositories
- Directed lines (*arrows*) between boxes denote connections or dependencies. They may use formatting (dashes, arrowhead styles, etc.) to denote specific types of interactions. For example, these diagrams often distinguish between the flow of data and the flow of control

Key Word in Context (KWIC)

The KWIC index system accepts as input an ordered set of lines, each line is an ordered set of words, and each word is an ordered set of characters. Any line may be “circularly shifted” by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order of the keyword used to shift the line

– Common (*stop*) words may be omitted

Circular Shifts

- Original title
  - `Gone with the Wind`
- Circular shifts (key words underlined)
  - `Gone with the Wind`
  - `with the Wind Gone`
  - `the Wind Gone with`
  - `Wind Gone with the`
- Stop word removal
  - `Gone with the Wind`
  - `Wind Gone with the`

Example With Multiple Titles

- `Gone with the Wind`
- `War and Remembrances`
- `The Winds of War`

KWIC Example

```
           Gone            with the Wind
and        Remembrances                 War
Winds of   War                          The
           War             and Remembrances
with the   Wind                        Gone
     The   Winds           of War
```

Exercise

- Assume that you had to implement KWIC. Decide how you would break it into approximately six pieces (even if you don't think it is big enough to have pieces!)
  - Use a box for each piece and give it a label
  - Ignore stop-word removal
- Decide how the pieces communicate
  - Use a line between two boxes to indicate some form of communication
  - Label the line to indicate the type of communication
- Come up with at least two solutions
  - For each, give circumstances when that solution would be advantageous

Shared Data Decomposition

- Break into components based on functions computed
  - Also Known As: *Functional Decomposition*
- All components share access to data in memory
- Usually contain a *master-controller* routine
  - Dependencies realized with function calls

Modules

- Input: reads titles, stores in memory
- Circular shifter: constructs array of pairs <line index; word index>
- Alphabetizer: batch sorting of circular shifts
- Output
- Master controller

Shared Data

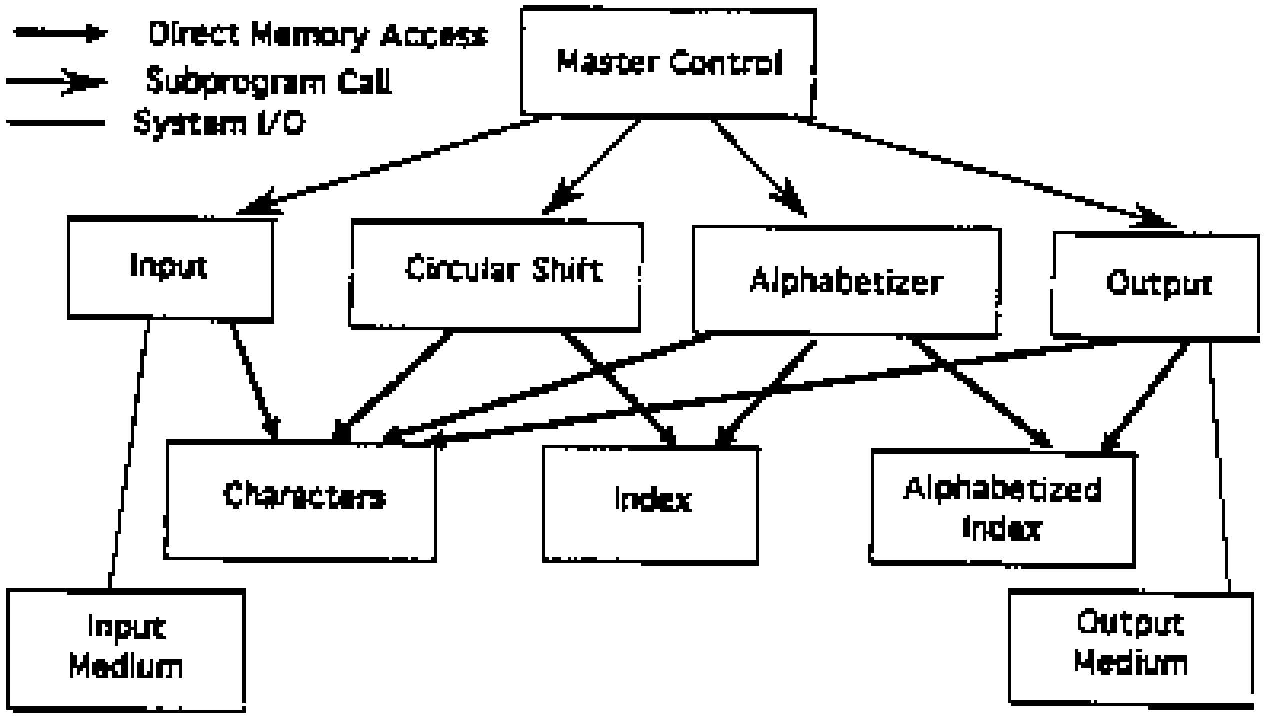

Figure 6: KWIC – Shared Data Solution

Abstract Data Types (ADT) Decomposition

- Organization based on data structures
- Representation hiding
- Modules holding data structures also provide services
- Basis for object orientation

Modules

- Line: ADT
- Input: reads titles, hands to Line
- Characters: ADT
- Circular shifter: ADT
- Alphabetic Shifts: ADT
  - I-th circular shift
- Output
- Master controller

Abstract Data Type

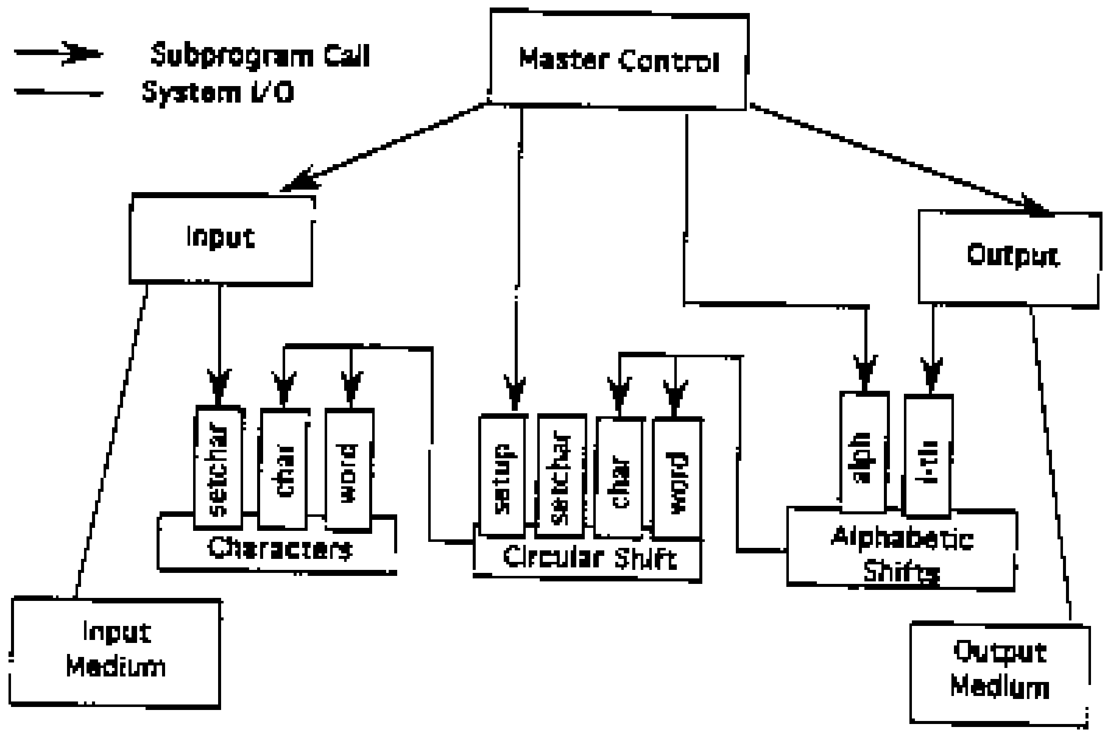

Figure 7: KWIC - Abstract Data Type Solution

Implicit Invocation

- Client modules express interest in state changes in server modules (*registration*)
- Server announces changes to all registered listeners (*broadcast*)
- Server does not know identity of clients
- Unit of notification is the *event*

Modules

- Input of new line
- Update line data structure
- Invoke circular shifter
- Update (second) line data structure
- Invoke alphabetizer

Implicit Invocation

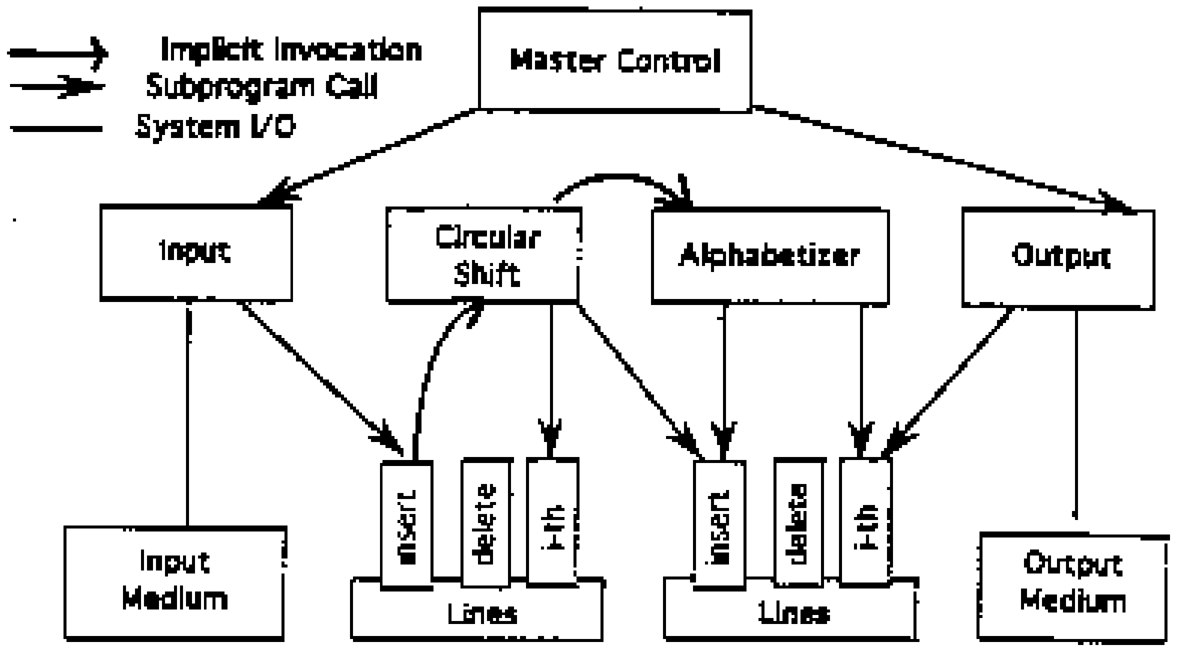

Figure 8: KWIC – Implicit Invocation Solution

Pipe And Filter

- System organized into independent programs (*filters*)
- Each filter takes in input (`stdin`) and produces output (`stdout`)
- Filters are connected together via FIFO queues (*pipes*)
- Common assumption of sequential files containing lines of ASCII characters
  - Can you think of a non-ASCII example?

Modules

- Filters for circular shift and alphabetizer
- Pipe connections with files of FIFOs
- No common data storage

Pipe And Filter

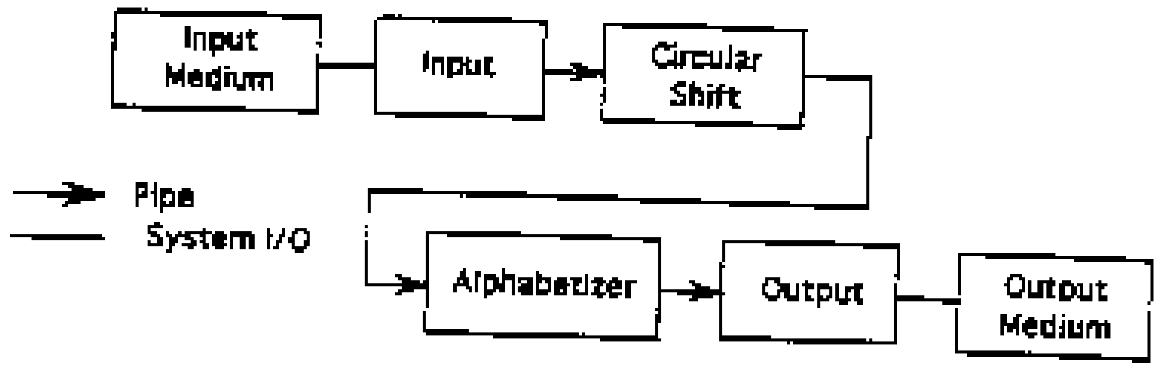

Figure 9: KWIC - Pipe and Filter Solution

Evaluation

- Shared data
  - +
  - –
- ADT
  - +
  - –
- Implicit invocation (active data)
  - +
  - –
- Pipe and filter
  - +
  - –

Evaluation

- Shared data
  - + Intuitive, efficient access
  - – Brittle to changes in representation
- ADT
  - + Maintainability, reuse
  - – Performance
- Implicit invocation (active data)
  - + Enhancements, data representation changes, reuse
  - – Difficult to control/think about
- Pipe and filter
  - + Intuitive, reuse
  - – Interactivity, space efficiency

Exercise Continued

- List at least three possible enhancements / improvements that could be made to KWIC
- For each of the four styles (shared data, ADT, implicit invocation, pipe and filter) indicate how well it would cope with each of your changes
  - Use the categories (*brittle, moderate, easy*) to label your choices

KWIC Changes

- Input format
- Reuse
- Processing algorithm
  - Shift each line as it is read
  - Shift lines after all are read
  - Shift lines on demand
  - Incremental versus batch sort

- New functionality
  - Stop-word elimination
  - Interactive line deletion
  - Use of external storage
  - Performance optimization
- Data representation
  - Line storage
  - Explicit circular shifts
  - Index/offset circular shift

Lessons

- Hide design decisions, particularly where representation decisions are concerned: *information hiding*
- Organize modules around data

Quiz

1. Critique the use of UML in describing software architectures? List a set of desirable characteristics for a software architecture document. For each, determine whether or not UML provides a suitable means for describing that characteristic. If so, indicate which UML diagram or diagrams are appropriate. If not, indicate what alternative you would use.

Quiz - 2

2. Sketch an algorithm for selecting an architectural style. The input to the algorithm is a situation, including a systems functional and non-functional requirements, any existing components or previous versions that are available, and any other information you deem relevant to making the choice. Output is the selected style. How would you evaluate the quality of your algorithm? How would you improve it over time?

Homework

- Imagine the following scenario. In many publicly accessible area, screens of various sizes exist capable of displaying computer generated material. Also, each screen has a communications port capable of receiving display requests from client devices that are within a limited vicinity of the screen. A protocol exists by which a client device can communicate with the screen and request permission to use it. Once permission is granted, the device may supply content for display. This exercise is concerned with the software that runs on the screen server that interacts with clients and controls the display
- List a set of non-functional concerns that must be addressed by the server software. For each, indicate what sort of computational approach should be taken to address it
- List and describe a set of server components
- Provide a box-and-arrows diagram description of the system's architecture
- For each connecting line, indicate what kind of connector you would choose
- You may do this exercise alone or in groups of two or three