CSC2100B Data Structures List, Stack, and Queue

Irwin King

king@cse.cuhk.edu.hk
http://www.cse.cuhk.edu.hk/~king

Department of Computer Science & Engineering
The Chinese University of Hong Kong

Introduction

- Look at a VCR
  - PLAY, FFW, REW, REC, etc.
  - Instruction manual tells what it should do without how it is implemented inside.
- Why do we use data abstraction?
  - Simplification of software development
    - It facilitates the decomposition of the complex task of developing a software system
  - Reusability
  - Modifications to the representation of a data type

Abstract Data Types (ADTs)

- Data Encapsulation or Information Hiding is the concealing of the implementation of a data object from the outside world.
- Data Abstraction is the separation between the specification of a data object and its implementation.
- A data type is a collection of objects and a set of operations that act on those objects.
- An abstract data type (ADT) is a data type that is organized in such a way that the specification of the objects and the specification of the operations on the objects is separated from the representation of the objects and the implementation of the operations.

Abstract Data Types (ADTs)

- An abstract data type (ADT) is a set of operations.
- Abstract data types are mathematical abstractions.
- Nowhere in an ADT's definition is there any mention of how the set of operations is implemented.

ADT

- Objects such as list, sets, and graphs, along with their operations, can be viewed as abstract data types, just as integers, reals, and booleans are data types.
- There is no rule telling us which operations must be supported for each ADT; this is a design decision.
- Error handling and tie breaking (where appropriate) are also generally up to the program designer.

ADTs

- We will see how each can be implemented in several ways.
- If they are done correctly, the programs that use them will not need to know which implementation was used.

ADT Example

- For the Set ADT, we might have such operations as union, intersection, size, and complement.
- Alternately, we might only want the two operations union and find, which would define a different ADT on the set.

Arrays

- Axiomatization
- Ordered Lists
- (MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY)
- Matrix Operations

The List ADT

- We will deal with a general list of the form $a_1, a_2, \ldots, a_n$.
- We say that the size of this list is n.
- We will call the special list of size 0 a null list.
- For any list except the null list, we say that a_{i+1} follows (or succeeds) a_i ($i < n$) and that a_{i-1} precedes a_i ($i > 1$).

The List ADT

- The first element of the list is a_1, and the last element is a_n.
- The predecessor of a_1 and the successor of a_n is not defined.
- The position of element a_i in a list is i.

List Definition

- A list of elements of type T is a finite sequence of elements of T together with the following operations:
  - Create the list, and make it empty.
  - Determine whether the list is empty or not.
  - Determine whether the list is full or not.
  - Find the size of the list.

List Definition

- Retrieve any entry from the list, provided that the list is not empty.
- Store a new entry replacing the entry at any position in the list, provided that the list is not empty.
- Insert a new entry into the list at any position, provided that the list is not full.
- Delete any entry from the list, provided that the list is not empty.
- Clear the list to make it empty.

Definition for a Queue

- A queue of elements of type T is a finite sequence of elements of T together with the following operations:
  - Create the queue, and make it empty.
  - Determine if the queue is empty or not.
  - Determine if the queue is full or not.
  - Determine the number of entries in the queue.

Queue Definition

- ***Insert*** a new entry after the last entry in the queue, if it is not full.
- ***Retrieve*** the first entry in the queue, if it is not empty.
- ***Serve*** (delete) the first entry in the queue, if it is not empty.
- ***Clear*** the queue to make it empty.

Operations on Lists

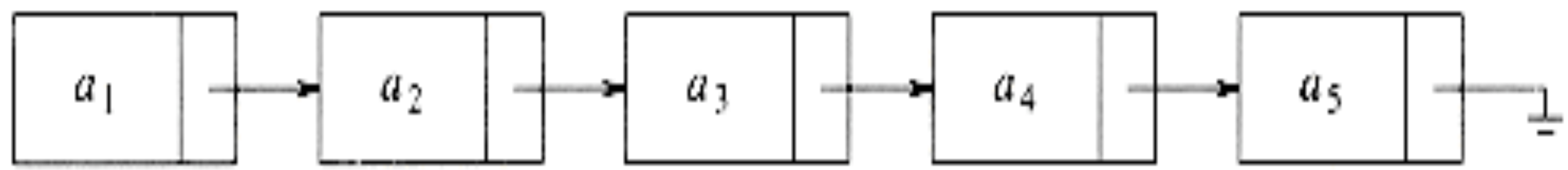
- Associated with these "definitions" is a set of operations that we would like to perform on the list ADT.

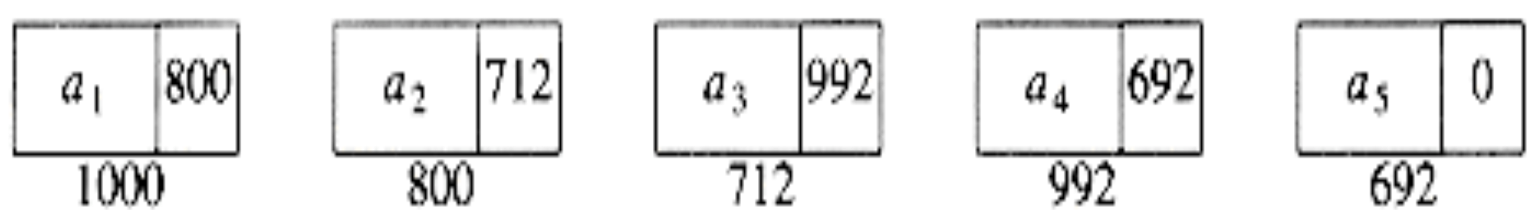
- One may want to perform a `print_list`, `make_null`, `find`, `insert`, `delete`, and `find_kth`.

Simple Array Implementation of Lists

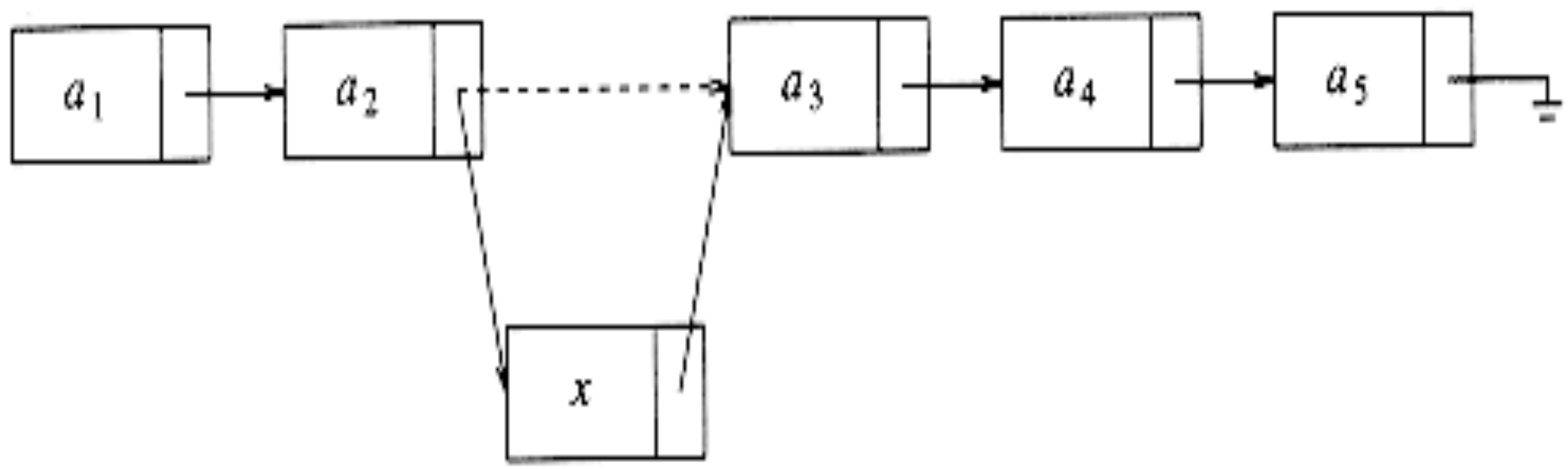
- Linear Time
  - `print_list`
  - `make_null`
  - `find`
  - `find_kth`
- What happens to insert and delete?
- How much room do you need in the beginning?
- Dynamically allocated?

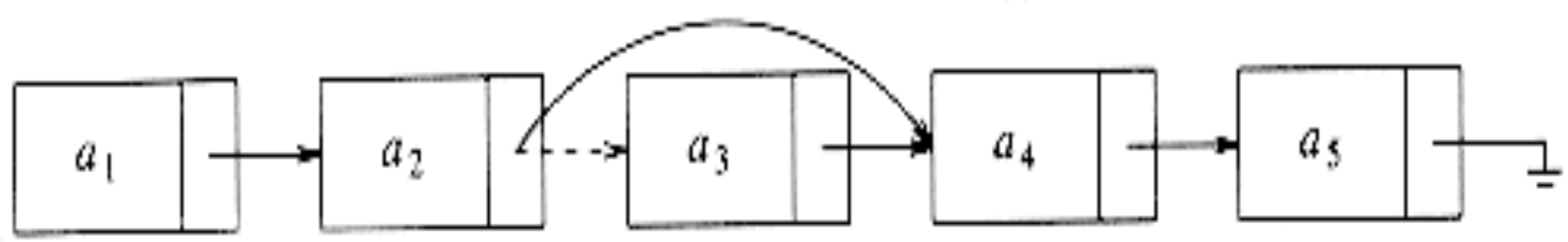
A Linked List



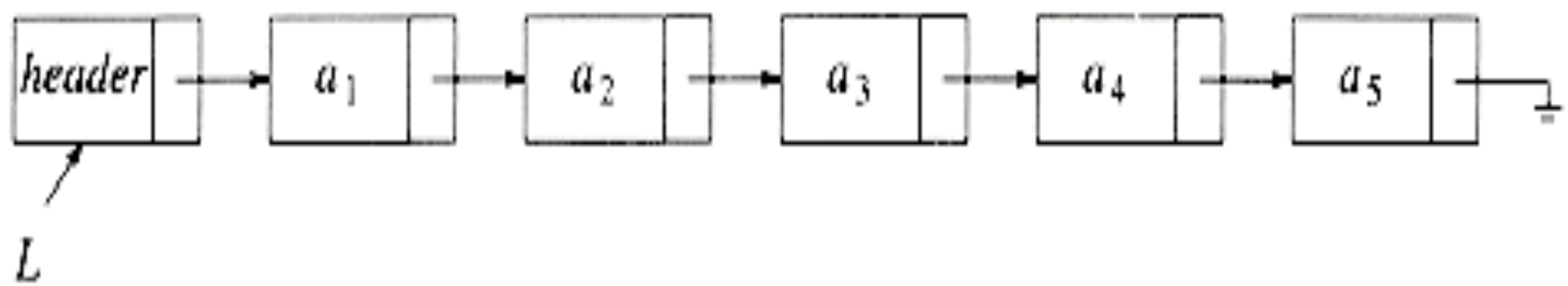


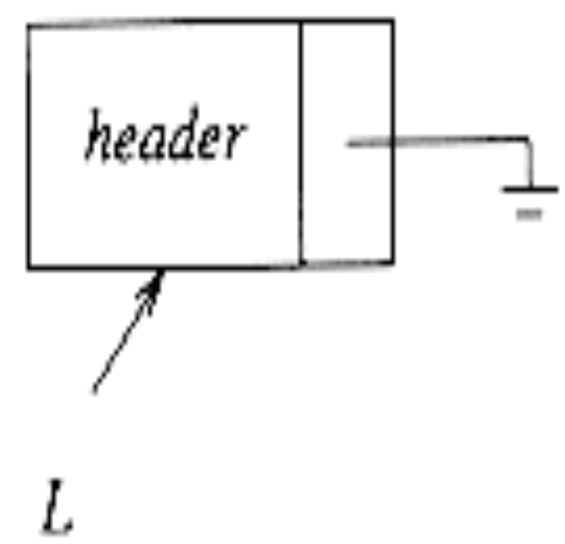
Insertion and Deletion





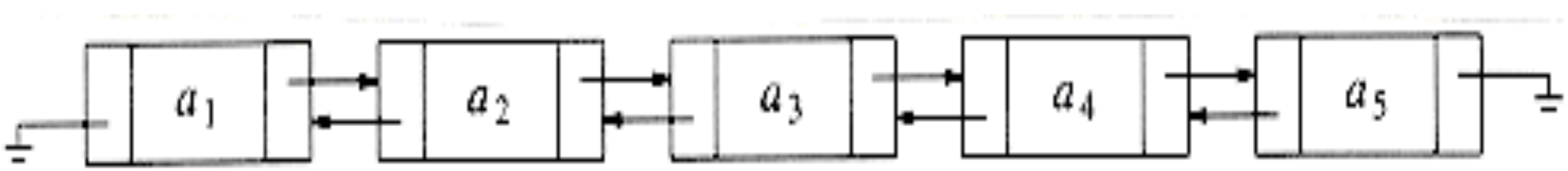
Linked List with a Header





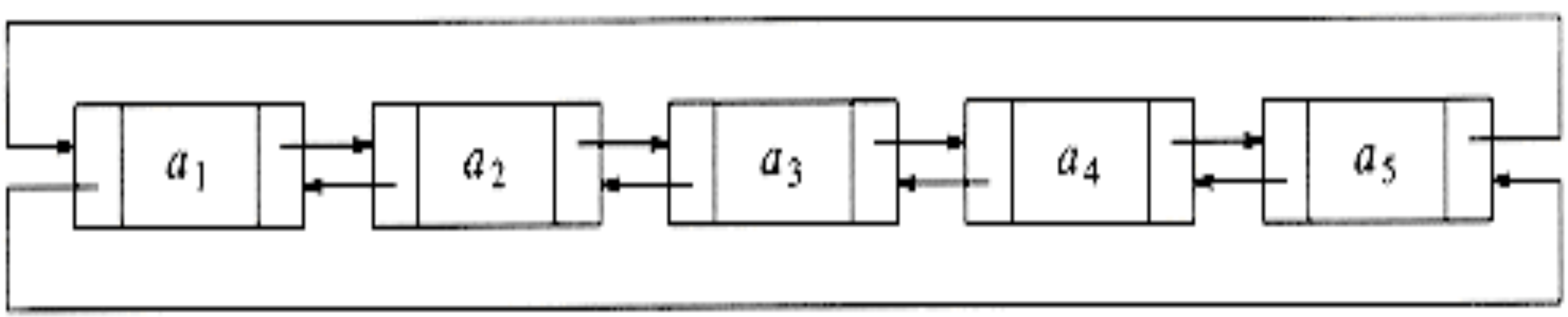
Doubly Linked List

- Why use doubly linked lists?
- What does it do to the storage complexity?
- Which operations will be simpler?



Circularly Double Linked Lists

- Why do we need to use this data structure?



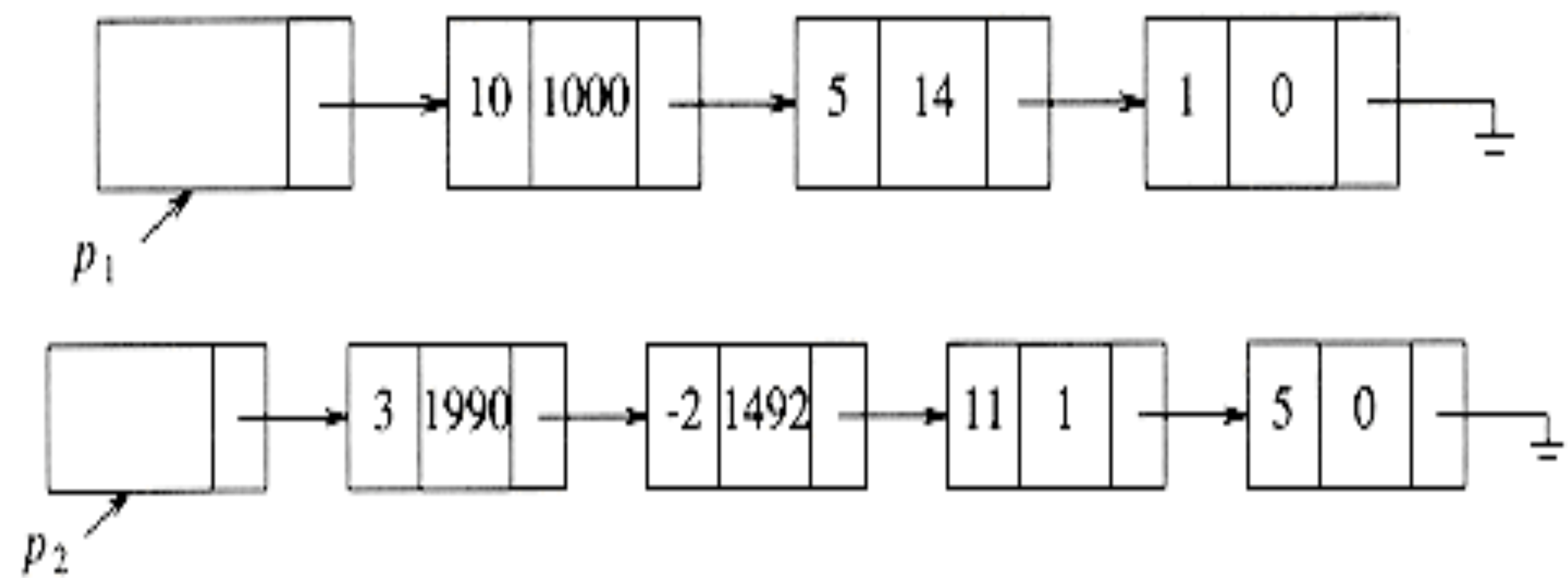
Example

- Representation of the Polynomial ADT

$$F(X) = \sum_{i=0}^{N} A_i X^i$$

- Example: P(X) = $4X^3+2X^2+5X+1$
  - Addition
  - Subtraction
  - Multiplication
  - Differentiation
- Can array data structure be used?

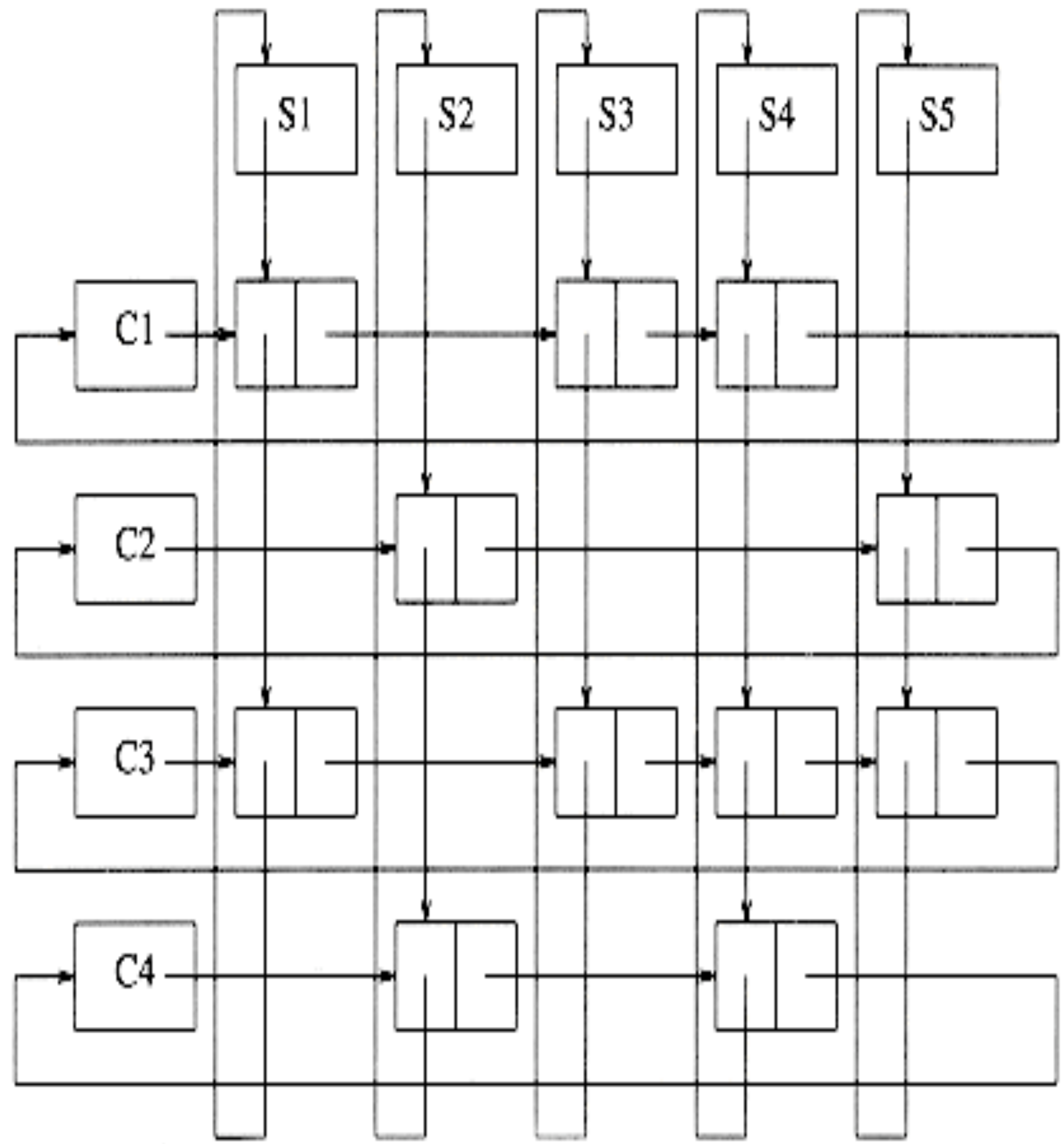
Example



Example

- Problem
  - CUHK has 12,000 students
  - CUHK has 1,000 courses
  - Two types of reports
    - Registration for each class
    - Classes that each student is registered for

Example



Notes

- Circular list saves space but not time
- It is used when
  - Few courses per student
  - Few students per course

Review Questions for Lists

- What is the difference between an array and a list?
- Which of the operations specified for general lists can also be done for queues? For stacks?
- List three operations possible for general lists that are not allowed for either stacks or queues.
- What is list traversal?

CSC2100B Data Structures
Stack

Irwin King

king@cse.cuhk.edu.hk

http://www.cse.cuhk.edu.hk/~king

Department of Computer Science & Engineering

The Chinese University of Hong Kong

Definition of a Stack

- A stack of elements of type T is a finite sequence of elements of T together with the following operations:
  - Create the stack, and make it empty.
  - Determine if the stack is empty or not.
  - Determine if the stack is full or not.

Definition of a Stack

- Determine the number of entries in the stack.
- If the stack is not full, then insert a new entry at one end of the stack, call its top.
- If the stack is not empty, then retrieve the entry at its top.
- If the stack is not empty, then delete the entry at its top.
- Clear the stack to make it empty.

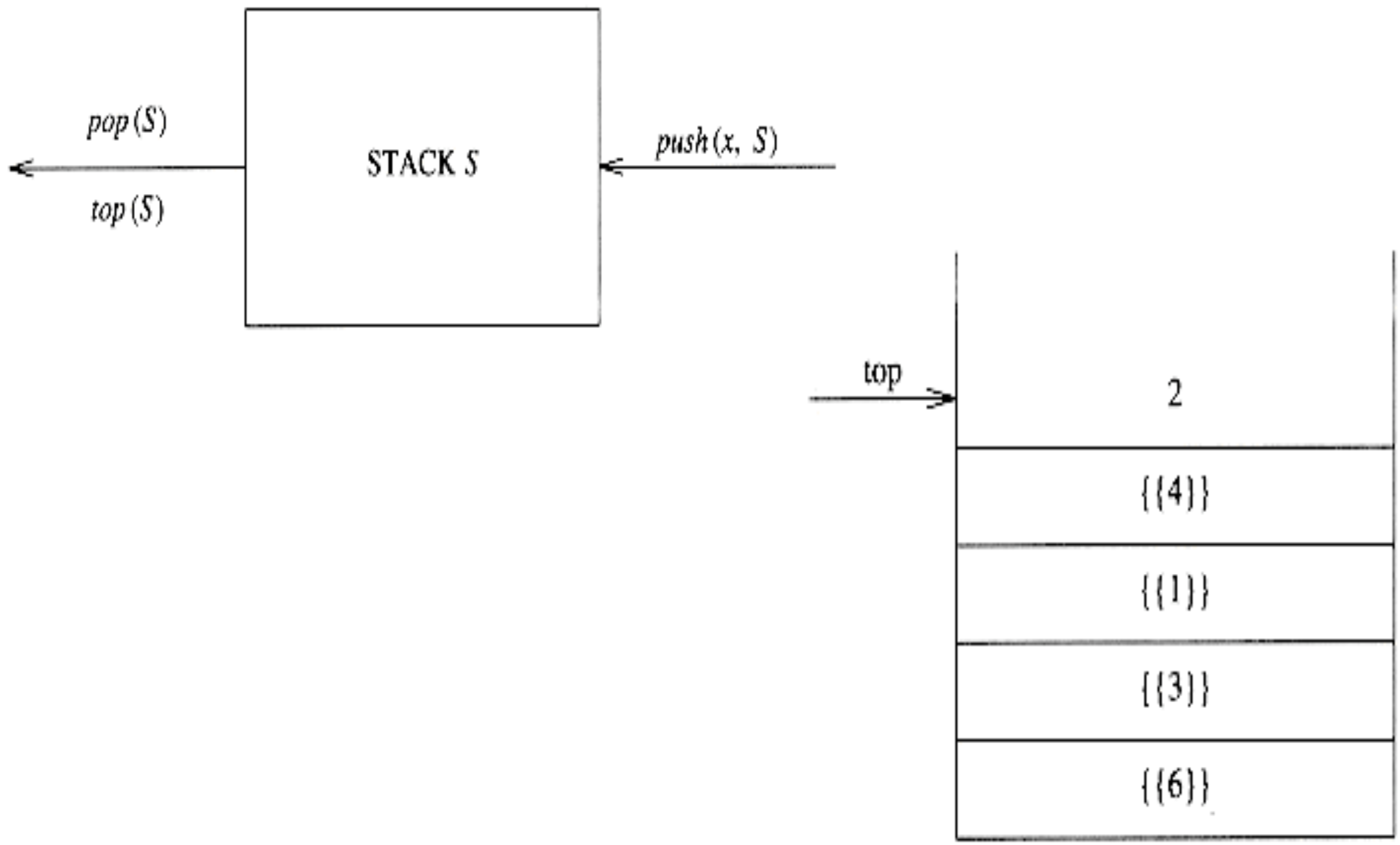
Stacks

- A stack is an ordered list in which all insertions and deletions are made at one end, called the top.
  - The fundamental operations on a stack
    - Push-an insert
    - Pop-a deletion of the most recently inserted element
- LIFO - Last In First Out.

A Stack

- create(S)
- add(i,S)
- delete(S)
- top(S)
- isemts(S)

Example



Implementation

- Linked list implementation
  - Push inserts an element at the front of the list
  - Pop deletes the element at the front of the list
  - Top examines the element at the front of the list
- What is the time complexity of these operations?

Implementation

- Array
    - Size declaration ahead of time
    - Use TopOfStack as a counter variable for the size and pointer to the top of the stack
    - Error testing degrades efficiency

Applications

- Balancing Symbols
  - Compilers check your programs for syntax errors
  - How to check whether there everything is balanced, e.g., [,], (,).
  - [( )] is legal but not [( ])

Balancing Symbols

- Make an empty stack
- Push an opening symbol onto the stack
- Pop a closing symbol
  - If the stack is empty, report an error
  - Otherwise, pop the stack
  - If the symbol popped is not the corresponding opening symbol, report an error.
- If stack is not empty at end, report an error

Application: Reverse Polish Calculator

- Prefix form - when the operators are written before their operands.
- Postfix form (reverse Polish form or suffix form) - when the operators are written after their operands
- Infix form - the usual custom of writing binary operators between their operands

Example

- The infix expression a x b becomes x a b in the prefix form and a b x in the postfix form.
- The infix expression a + b x c becomes + a x b c in the prefix form and a b c x + in the postfix form.
- Note that prefix and postfix forms are not related by taking mirror images or other such simple transformation.

Example

- The major advantage of both Polish forms is that no parentheses are needed to prevent ambiguities in the expression.

- Change the following: $x = (-b + b^2 - 4 \times a \times c)^{0.5}) / (2 \times a)$

Example

- Evaluate (((a + b) x c) + d) / e
- RPN: a b + c x d + e /
- Evaluate a + (b x c) + (d / e)
- RPN: a b c x + d e / +

Problem

- You are a railroad operator and you are asked to see whether you can re-arrange the carts in any order by using an auxiliary track (similar to a stack).
- You are also asked by the boss which sequence of permutation will give rise to the most number of operations performed.

CSC2100B Data Structures Queue

Irwin King

king@cse.cuhk.edu.hk

http://www.cse.cuhk.edu.hk/~king

Department of Computer Science & Engineering

The Chinese University of Hong Kong

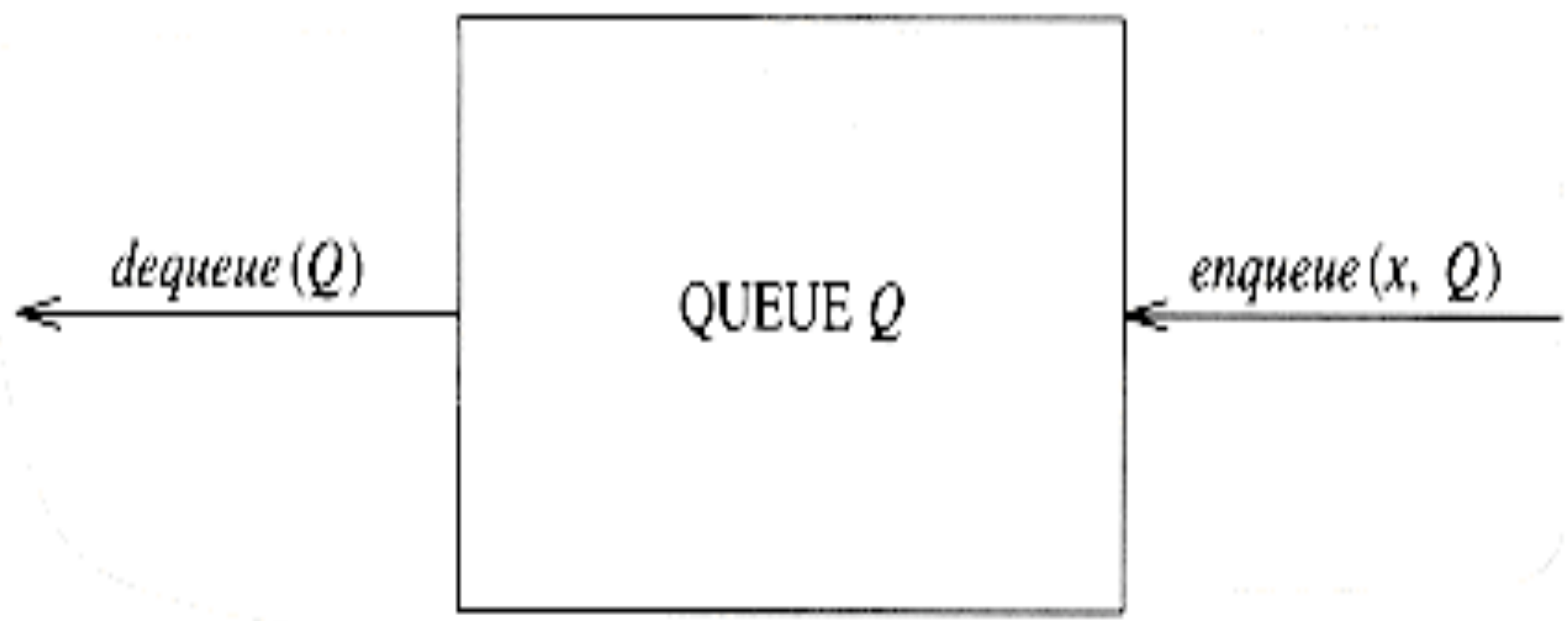
Queue

- A queue is an ordered list in which all insertions take place at one end, the rear, while all deletions take place at the other end, front.
    - FIFO - First In First Out.
    - Basic operations
        - Enqueue-inserts an element at the end of the list
        - Dequeue-deletes the element at the start of the list

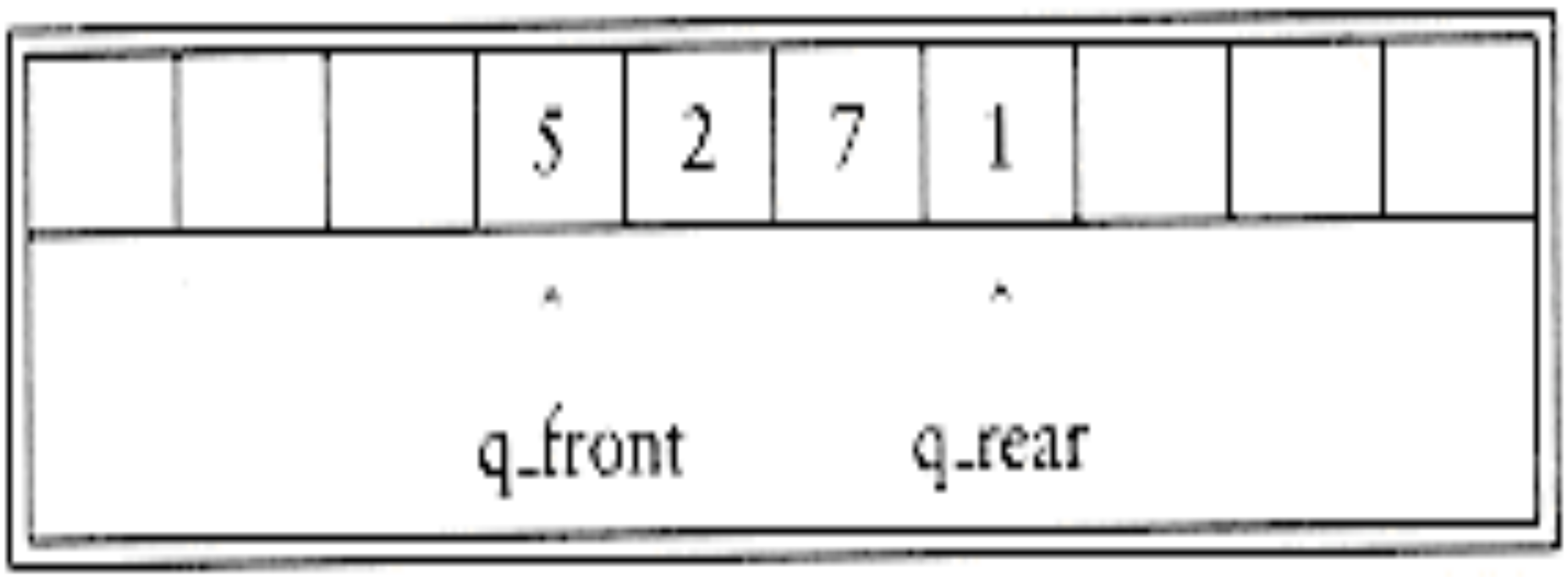
A Queue

- createq(Q)
- add(i,Q)
- delete(Q)
- front(Q)
- isemqs(Q)

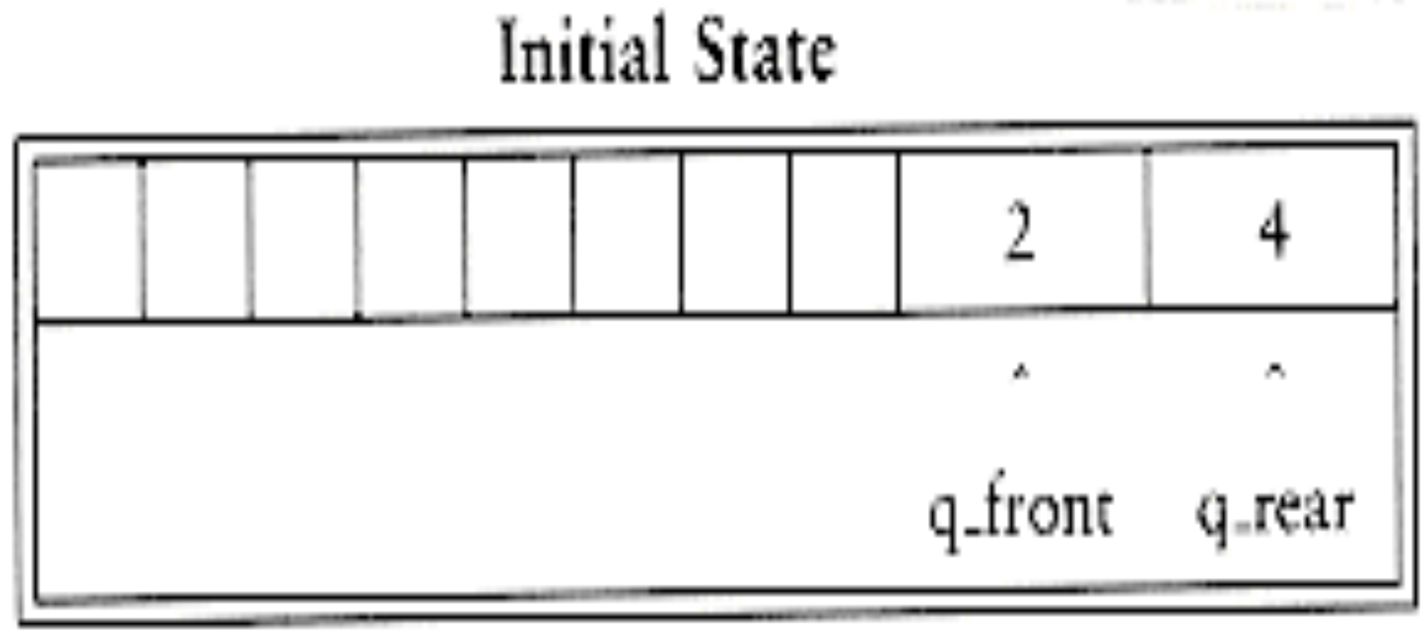
Example



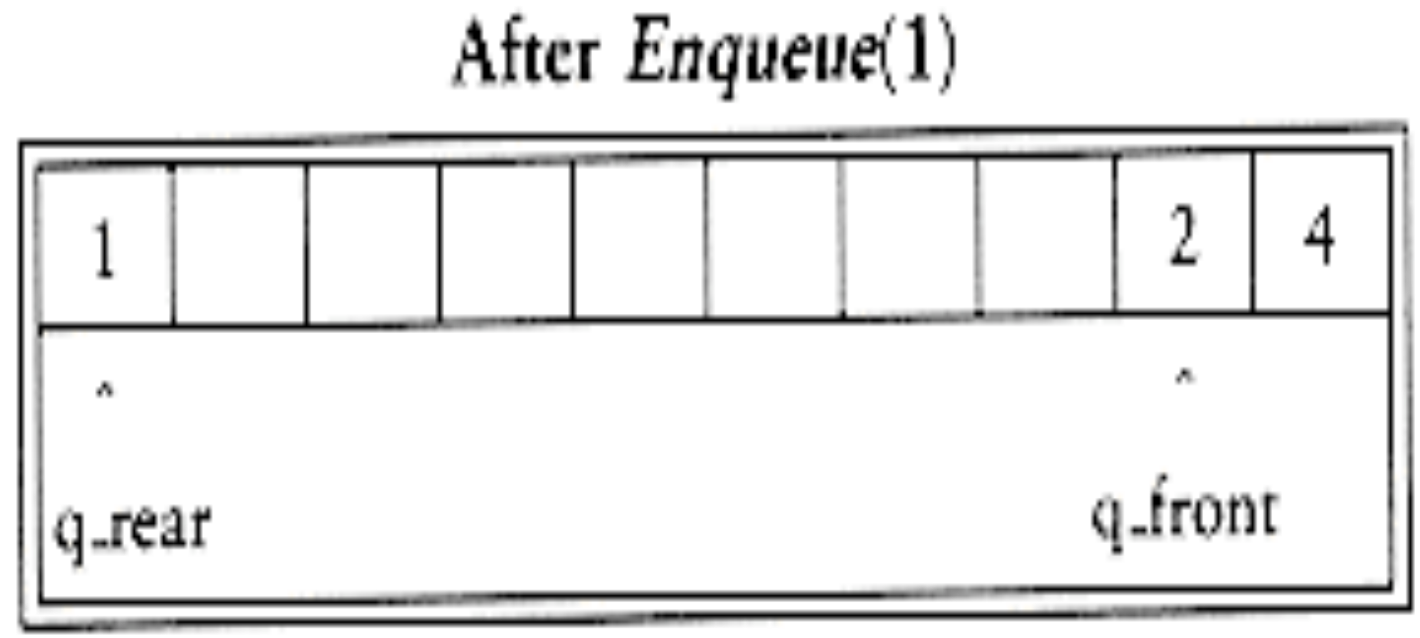
Example



Example



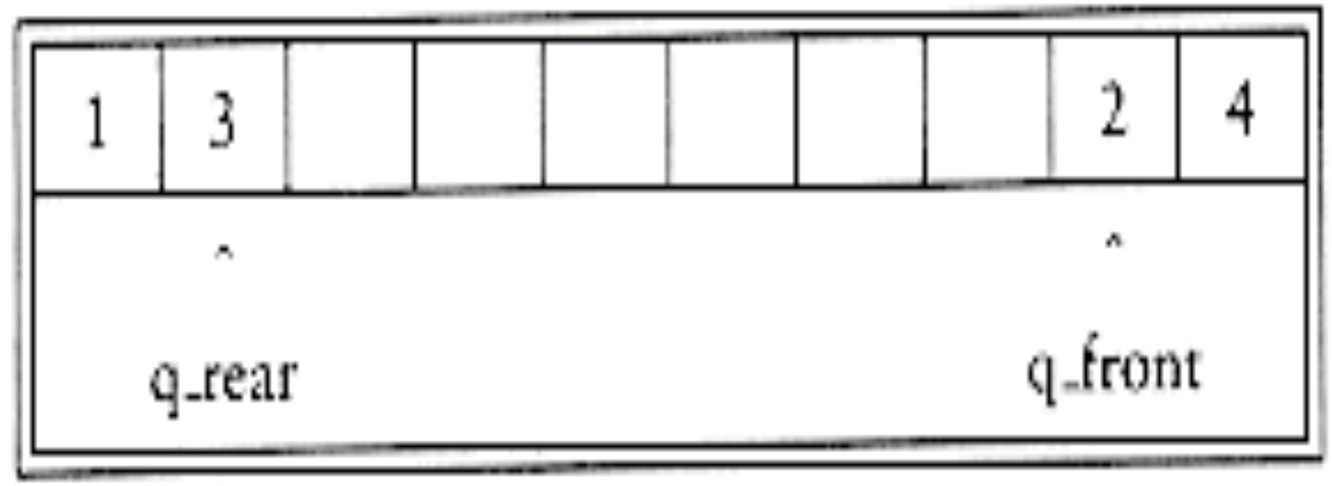
Initial State
2
4
q_front
q_rear



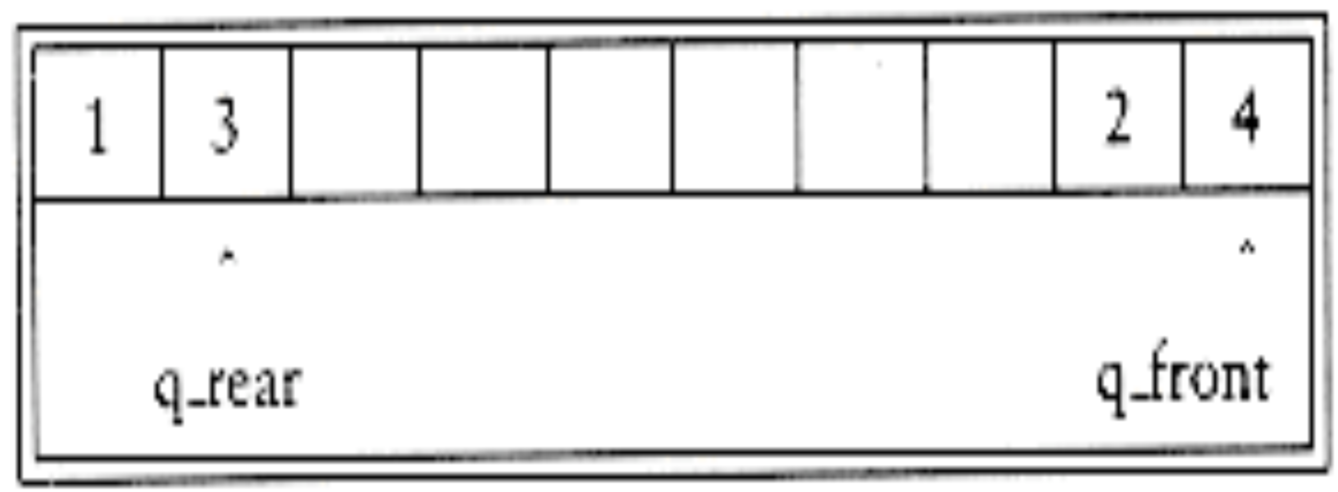
After Enqueue(1)
1
2
4
q_rear
q_front

Example

After *Enqueue* (3)

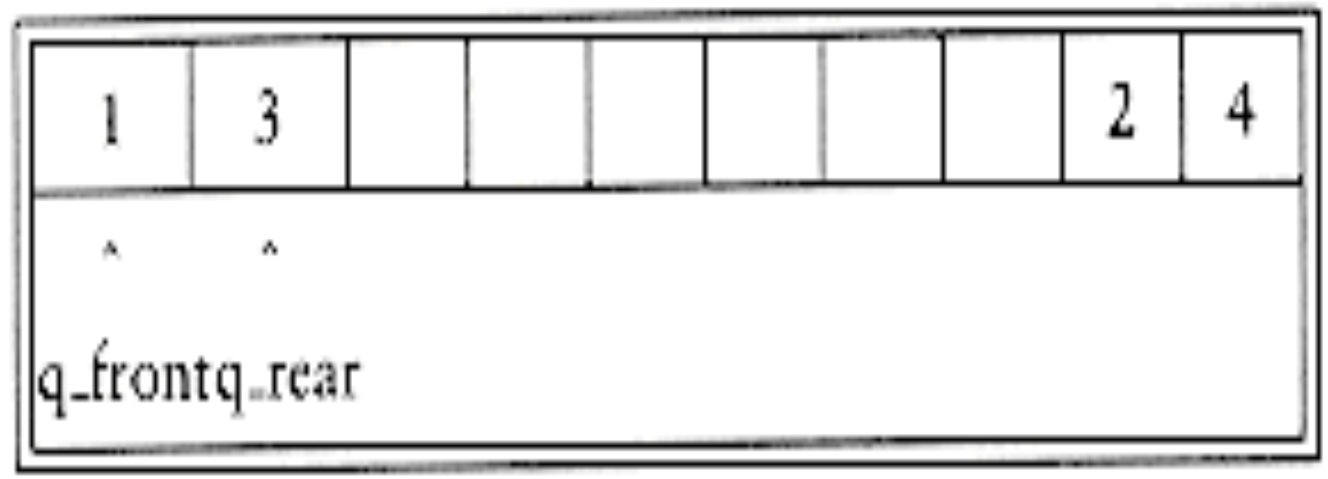


After *Dequeue*, Which Returns 2

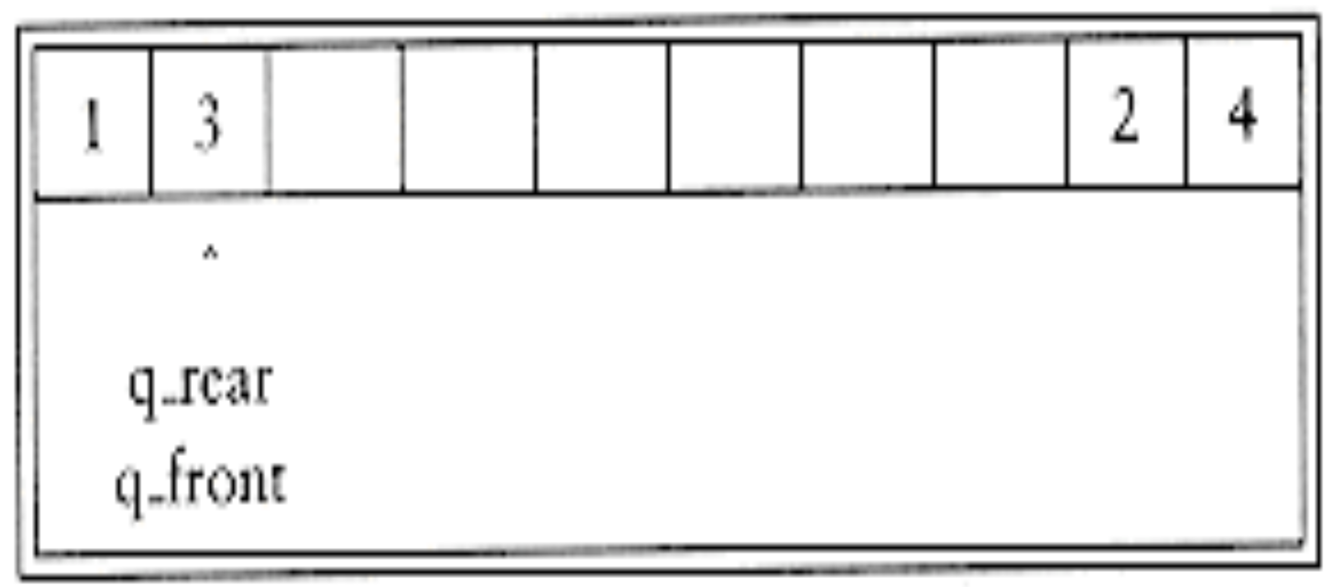


Example

After *Dequeue*, Which Returns 4

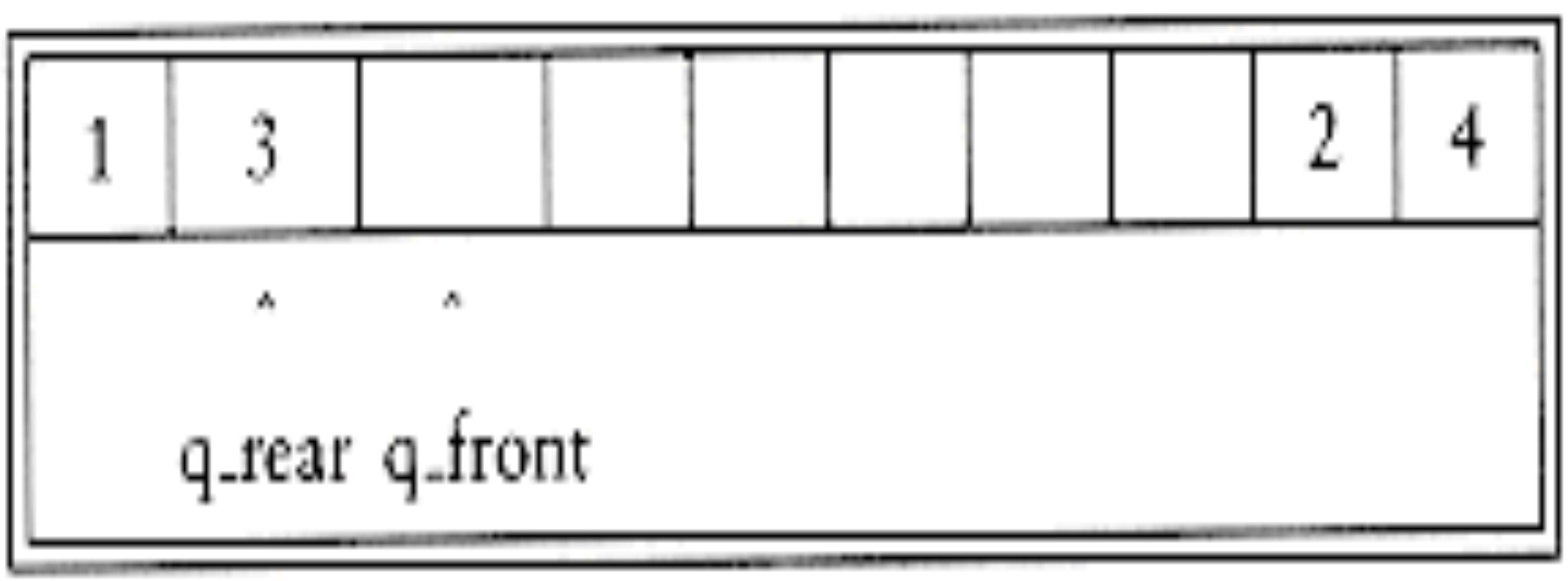


After *Dequeue*, Which Returns 1



Example

After *Dequeue*, Which Returns 3
and Makes the Queue Empty



Applications

- Printer queue (First-come-first-served)
- Airline controller queue
- Bank queue

Summary of Implementations for Queues

- The physical model: a linear array with the front always in the first position and all entries moved up the array whenever the front is deleted. This is generally a poor method for use in computers.
- A linear array with two indices always increasing. This is a good method if the queue can be emptied all at once.
- A circular array with front and rear indices and one position left vacant.

Summary of Implementations for Queues

- A circular array with front and rear indices and a Boolean variable to indicate fullness (or emptiness).
- A circular array with front and rear indices and an integer variable counting entries.
- A circular array with front and rear indices taking special values to indicate emptiness

Review Questions for Queues

- Define the term queue. What operations can be done on a queue?
- List at least four different implementations of queues?
- Is there one implementation of a queue that is almost always better than any other in a computer? If so, which?

Review Questions for Queues

- Is there one implementation of a queue that is almost always worse than any other in a computer? If so, which?
- How is a circular array implemented in a linear array?
- What problem occurs for the extreme cases in a circular array?