CSI2114
Data Structures
Spring 2006

CSI 2114 (Spring 2006)
Data Structures

Prof: Amiya Nayak

Office: SITE 5001

Email: nayak@uottawa.ca

Office Hours: Thurs (13:00-15:00), Fri (13:30-14:30)

Course Web site

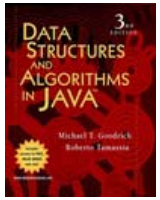
http://www.site.uottawa.ca/~anayak/CSI2114S06/

For slides, assignments, information ...

Textbook

Michael Goodrich, Roberto Tamassia
Data Structures and Algorithms in Java
(3rd or 2nd ed.), Wiley, 2004 (2000)

available at **AGORA**
(104.11 $ +tax)



Available free on the website:
Hints for the Exercises
Animations

http://ww3.java3.datastructures.net

Labs

What are they ?
Why are they important ?

Lab Schedule:

Wednesday 12:30-14:30 STE 2060

Evaluation

Assignments	30 %
Midterm exam (closed book, 2 hours)	20 %
Final exam (closed book, 3 hours)	50 %

To pass the course you must get at least 50 % in the average of the two exams

Assignments

- 4 assignments

Each assignment will be composed by two parts:
- theory questions
- programming exercises

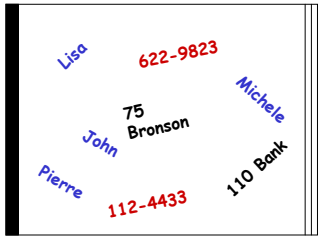
NOTE: Plagiarism will not be tolerated

Data Structures ?

Example:
Electronic Phone Book

Contains different **DATA**:

- names
- phone numbers
- addresses



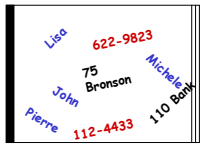
Need to perform certain **OPERATIONS**:

- add
- delete
- look for a phone number
- look for an address

How to organize the data so to optimize the efficiency of the operations

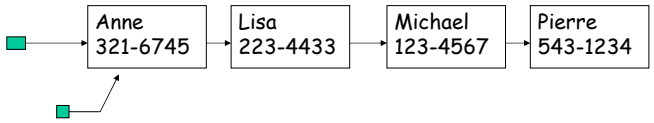
Data Structures ?

Example:



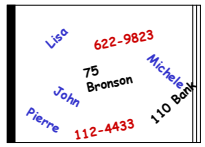
Lisa 223-4433	Pierre 543-1234	Michael 123-4567	Anne 321-6745

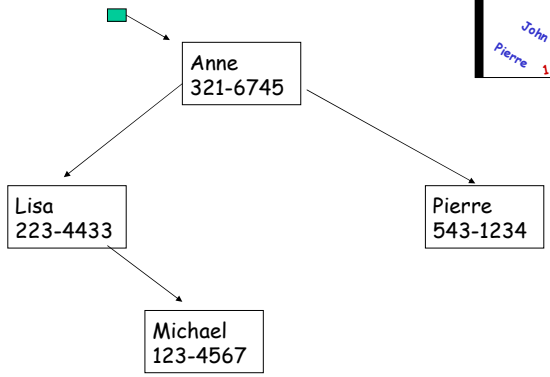
Anne 321-6745	Lisa 223-4433	Michael 123-4567	Pierre 543-1234



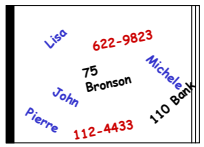
Data Structures ?

Example:



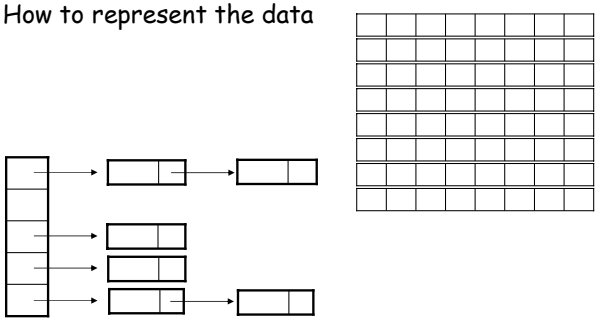


Data Structures ?



Data Structures ?

How to represent the data



so to perform the operations efficiently

Data Structures ?

Keep in mind the operations you need to perform

Choose the **best** structure for your data

Study different data structures

How to understand if a data structure is good

Objectives of the course

Present in a systematic fashion the most commonly used data structures, emphasizing their abstract properties.

Discuss typical algorithms that operate on each kind of data structure, and analyze their performance.

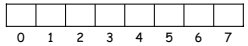
Compare different Data Structures for solving the same problem, and choose the best.

Overview of the course

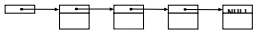
- Review ….
- Stacks, queues, deques (review)
- Algorithm analysis techniques
- Vectors, Lists, Sequences
- Trees
- Heaps
- Dictionaries
- Search trees
- Tries
- Graphs
- Sorting

Review

- Arrays

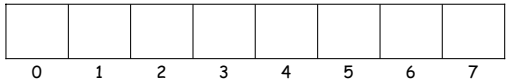


- Linked Structures



Array

A:

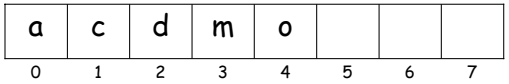


Numbered collection of variables of the same type. Fixed length.

- **Static** structure
- Direct access

Insertion ?
Deletion ?

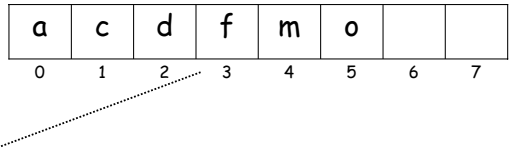
Array



f

example of insertion in a sorted array

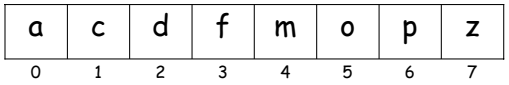
Array



move "m" and "o"
to make room for "f"

example of insertion in a sorted array

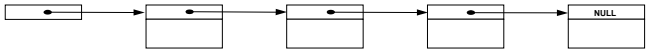
Array



1) For insertions and deletions
elements MUST BE MOVED

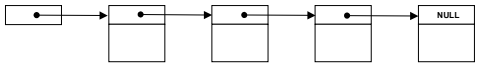
2) What happens when the array is FULL ?

Linked Structures



- **Dynamic** structure (never full)
- Sequential access (no direct access)
- Insertion and deletion occur without moving elements

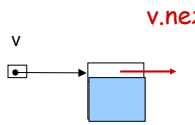
Single Linked Lists



Node v

Object element

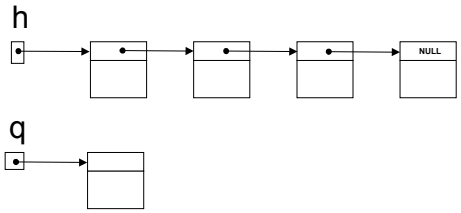
Node next



v.element

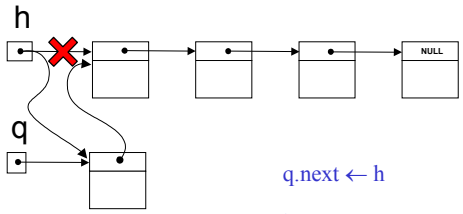
Insertion

Original configuration:



Goal: to insert the element q into list h.

Insertion at the beginning



q.next ← h

h ← q

(easy)

… we are using pseudocode …

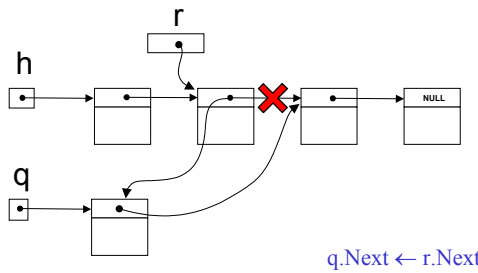
pseudocode

q.next ← h

variable **q.next** gets the value of variable **h**

(q.next:= h)

Insertion after r

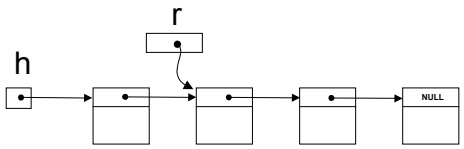


(easy)

q.Next ← r.Next

r.Next ← q

Insertion before r



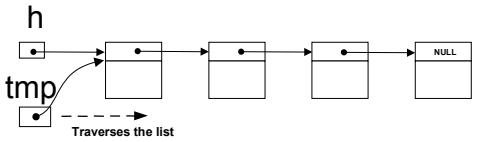
(more difficult)

- Must maintain a pointer to the preceding element

or

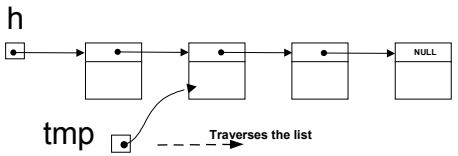
- Exchange the contents pointed to by r and q, and insert q after r.

Search



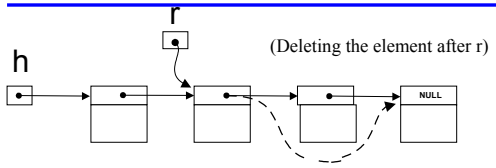

```
Node tmp;
tmp ← h;
while (tmp != null ) {
    if tmp .element  is  what-I-m-looking-for {
       return tmp ; }
   else
```

Search




```
Node tmp;
tmp ← h;
while (tmp != null ) {
    if tmp .element  is  what-I-m-looking-for {
       return tmp ; }
    else {tmp ← tmp .next; }
  return tmp ;
  }
```

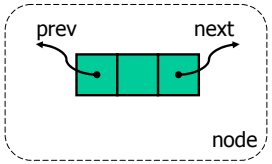

Deletion

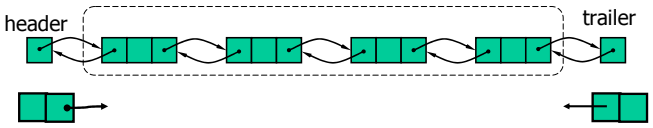


First element (easy)	h ← h.Next
Element after r (easy)	r.Next ← r.Next.Next
Element at r (difficult)	• Use a pointer to the preceding element, or • Exchange the content of the element at r with the contents of the element following r, and delete the element after r. (this is impossible if r points to the last element)

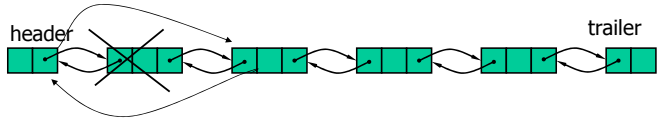
Doubly Linked List

- Nodes store:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes





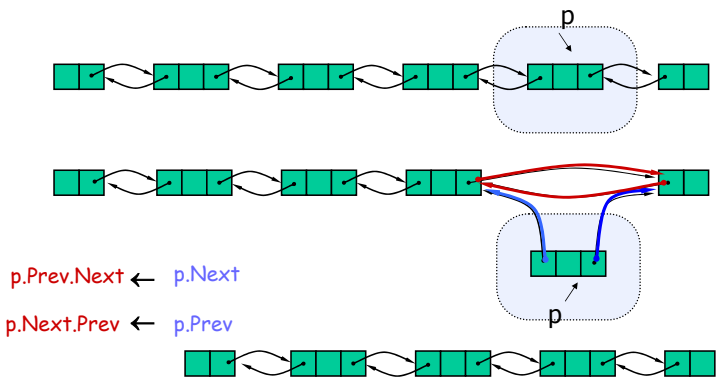
Deletion (first element)



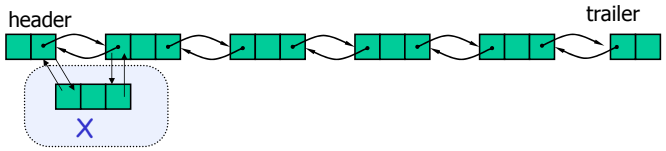
header.next ← header.next.next

header.next.prev ← header

Deletion (element p)



Insertion (beginning)



X.next ← header.next

header.next ← X

X.prev ← header

X.next.prev ← X

Insertion (before p)