ENSF 593/594
Data Structures – Linked Lists

Mohammad Moshirpour

Outline

- Linked List
  - Insertion
  - Deletion
- Doubly Linked List
  - Insertion
  - Deletion
- Circular List
  - Insertion
  - Deletion

Goal

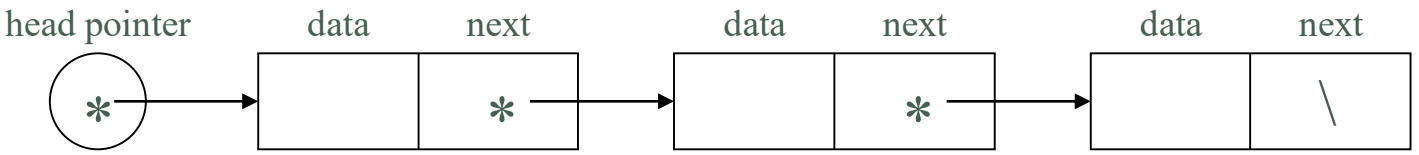
- In this lecture we will learn about another data structure, linked list, to implement a list. We will learn how to create a linked list and how to insert or delete elements from a linked list.
- To study usefulness of doubly-linked lists
- To study circular lists

Review

- Common data structures to implement a list:
  - Arrays
  - Link List

Linked Lists

- Are also called *logically ordered lists*
- Definition: A linear data structure that consists of zero or more *nodes* (elements), where each node contains data and a pointer to the next node



Linked Lists (cont'd)

- A head pointer is used to point to the first node of the list (the *head*)
- In the last node (the *tail*), the *next* pointer is set to *null* to signify the end of the list
- A linked list can grow and shrink without limit
  - A node is allocated dynamically at run time whenever a new item is added to the list
  - A node is freed (garbage collected in Java) whenever an item is deleted from the list

Linked Lists (cont'd)

- The data field for a node may be
  - A primitive data type, or
  - A compound data type, or
  - A reference to an object
- In Java, each node is an object of a class such as the following:

Linked Lists (cont'd)

```
public class Node {
   public double data;
   public Node next;
public Node(double d, Node n) {  //  Constructor
       data = d;
       next = n;
   }
public void setNext(Node n){     //Accessor methods
     next = n;
}
public Node getNext(){     //Accessor methods
     return next;
}
…
}
```

Linked Lists (cont'd)

- A linked list is initially empty, so set the head pointer to *null*
  - E.g. `Node head = null;`

Linked Lists (cont'd)

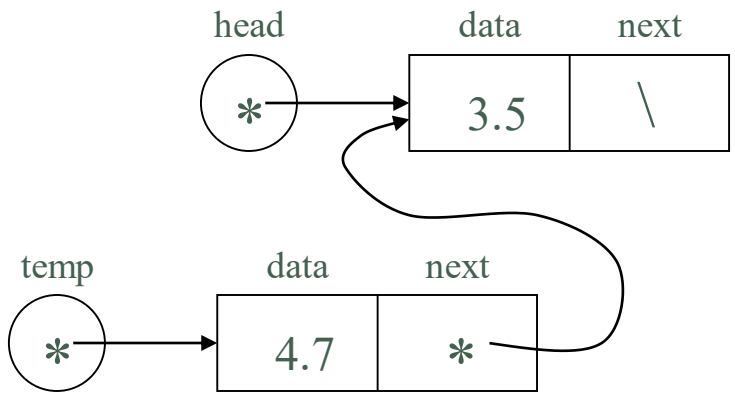
- When you add an item to the list, you create a new node object, and link it in
  - E.g. `head = new Node(3.5, null);`

Linked Lists (cont'd)

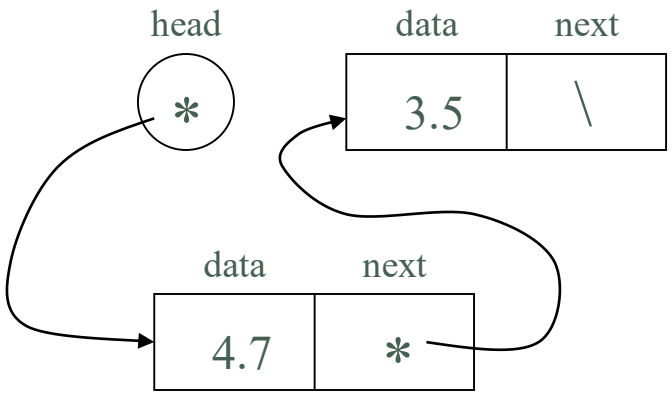
- To insert into the beginning of a list:
  - Allocate a new node
    - Use a temporary variable to point to it
    - Set the *data* field to the desired value
    - Set the *next* pointer to the value in head pointer
  - Set the head pointer to the temporary variable
  - E.g.
```
Node temp = new Node(4.7, head);
head = temp;
```

Linked Lists (cont'd)

Linked Lists (cont'd)



Linked Lists (cont'd)



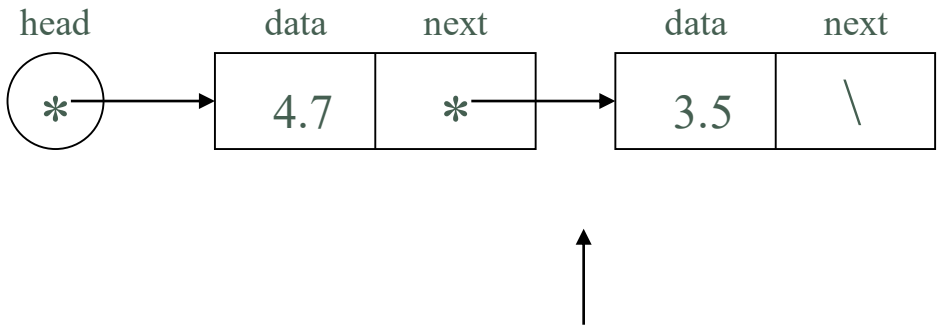
Linked Lists (cont'd)

- To insert into the middle or end of a list:
  - Find the predecessor node
    - May require a search
    - Use a variable to point to it
  - Allocate a new node
    - Use a temporary variable to point to it
    - Set the *data* field to the desired value
    - Set the *next* pointer to the value in the predecessor's *next* field
  - Set the predecessor's *next* pointer to the temporary variable

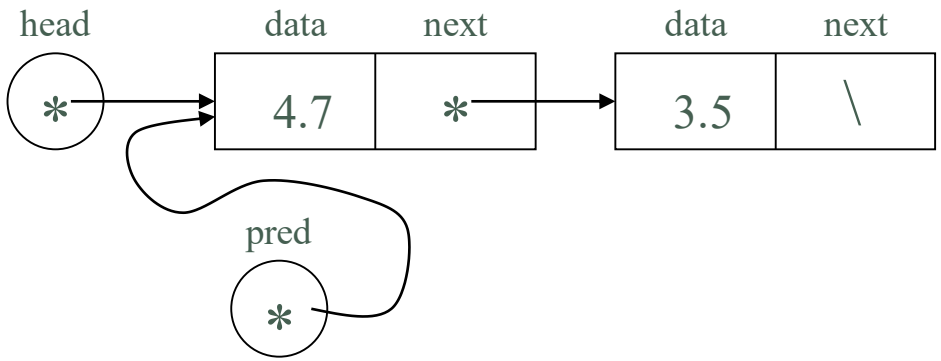
Linked Lists (cont'd)

- E.g.

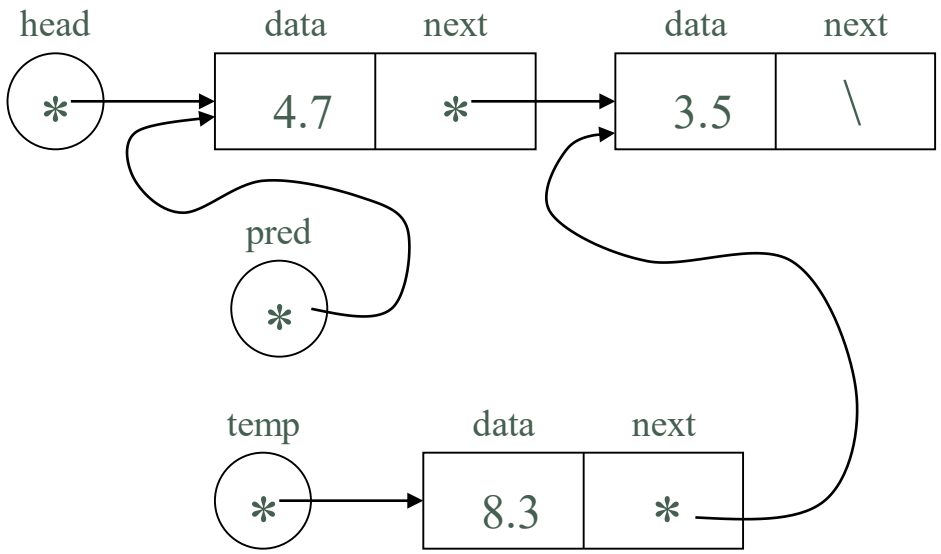
```
Node pred = ???;
Node temp = new Node(8.3, pred.getNext());
       pred.setNext(temp);
```



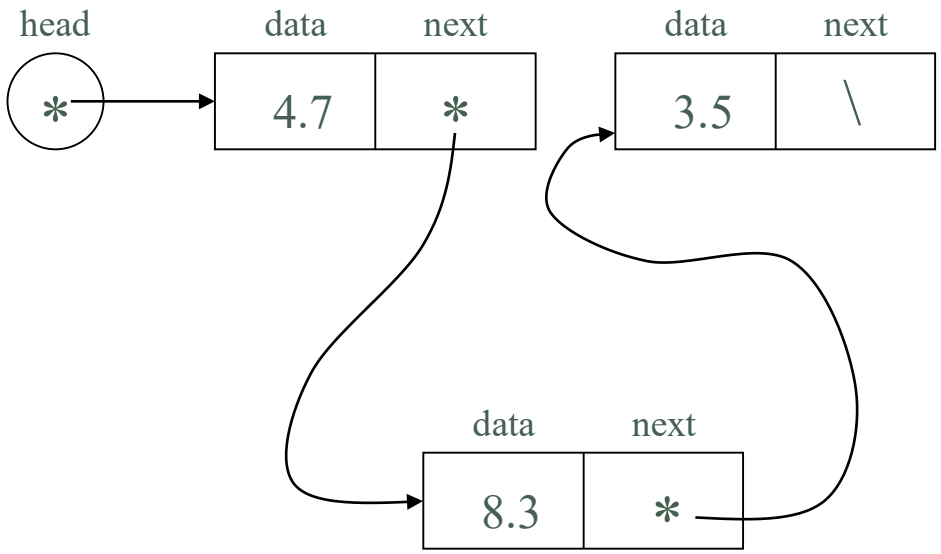
Linked Lists (cont'd)



Linked Lists (cont'd)



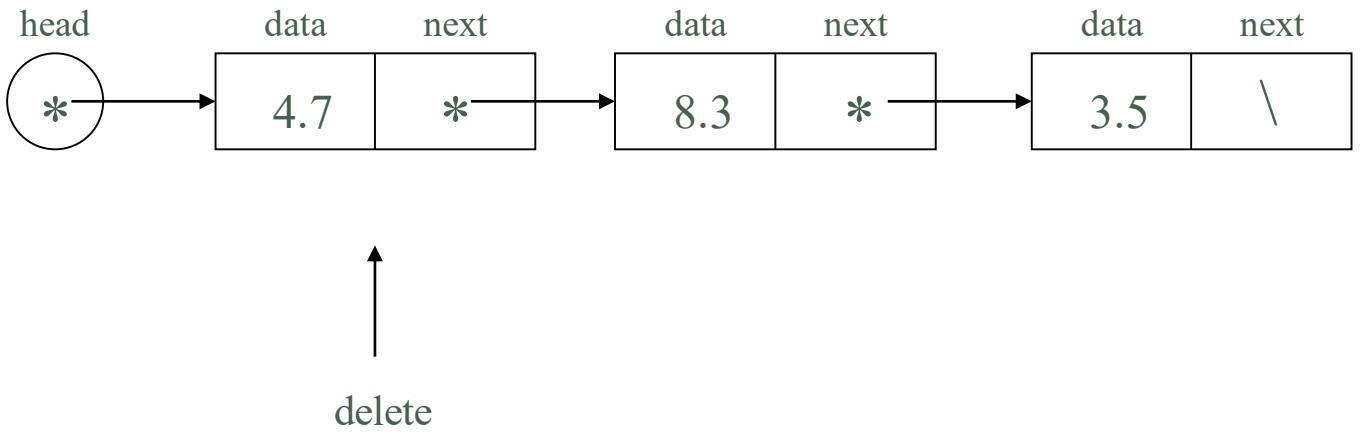
Linked Lists (cont'd)



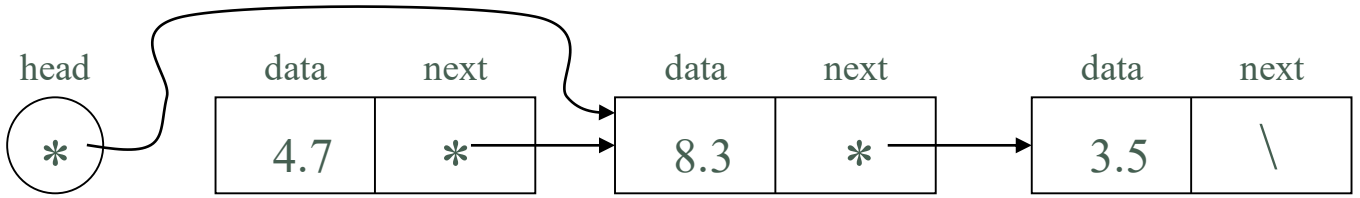
Linked Lists (cont'd)

- To delete from the beginning of a list:
  - Set the *head* pointer to point to the successor node
  - Free the deleted node
    - In Java, will be garbage collected if no longer referred to
    - Other languages require an explicit free operation
      - Need a temporary pointer to the node
  - E.g. `head = head.getNext();`

Linked Lists (cont'd)



Linked Lists (cont'd)



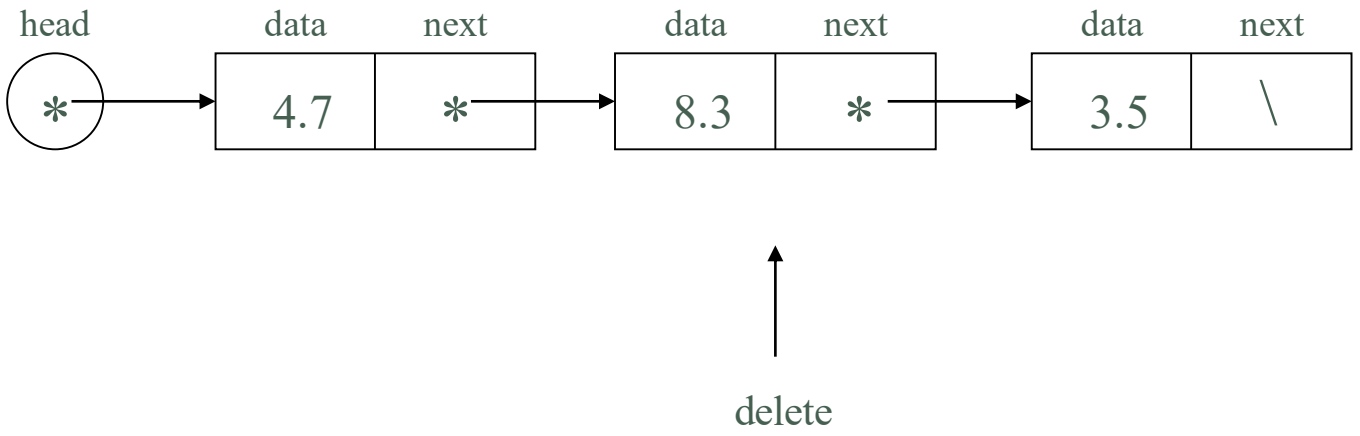
Linked Lists (cont'd)

Linked Lists (cont'd)

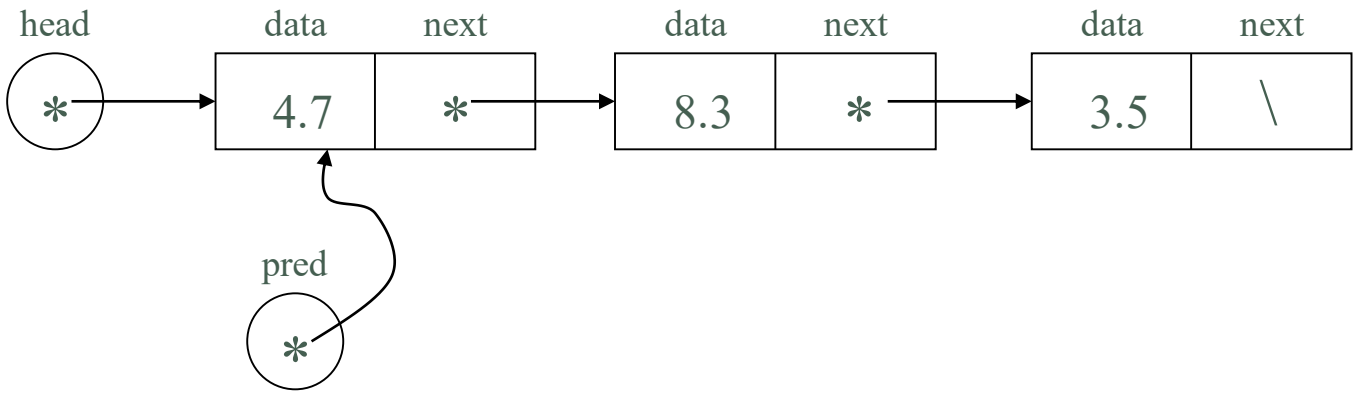
- To delete from the middle or end of a list:
  - Find the predecessor node
    - May require a search
    - Use a variable to point to it
  - Set the predecessor's *next* pointer to the delete node's *next* value
    - i.e the successor node, or *null*
  - Free the deleted node
  - E.g.

```
Node pred = …
pred.setNext((pred.getNext()).getNext());
```

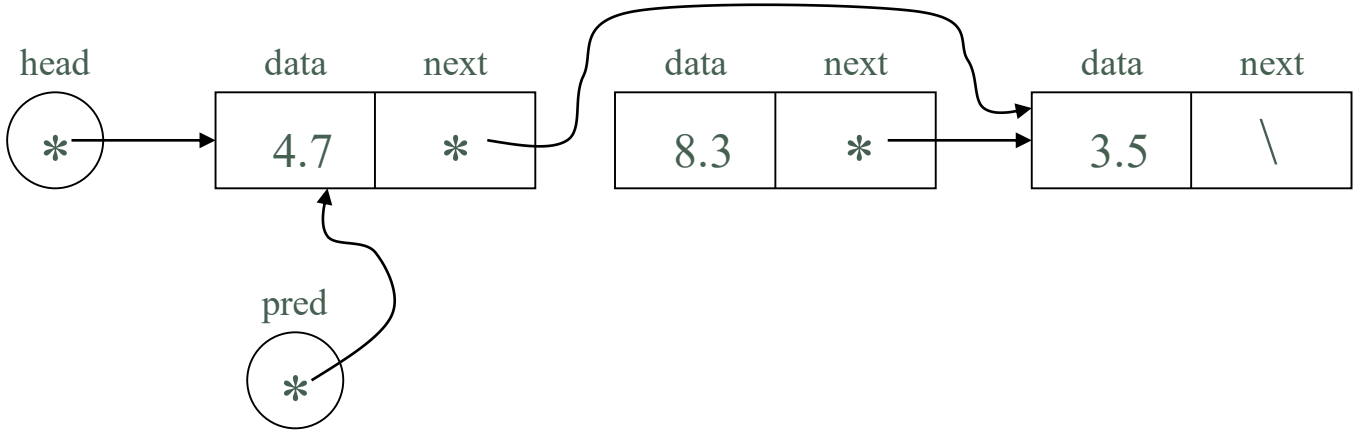
Linked Lists (cont'd)



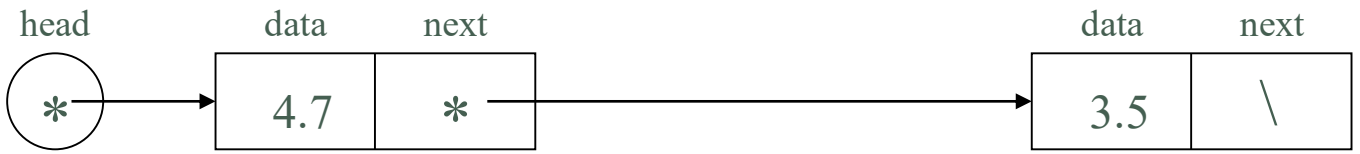
Linked Lists (cont'd)



Linked Lists (cont'd)

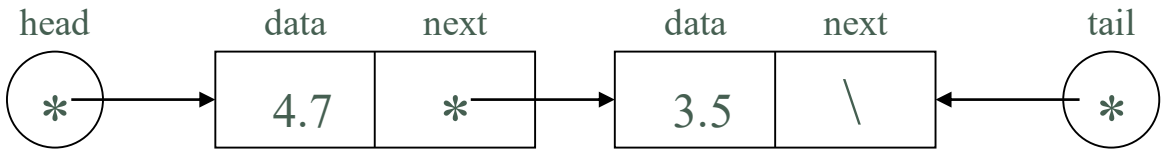


Linked Lists (cont'd)



Linked Lists (cont'd)

- Be sure never to delete from an empty list
  - i.e. Check for null *head* pointer
- A *tail pointer* points to the last node in the list
  - Makes inserting/deleting to/from the end of the list easier
  - Note: requires slight changes to preceding algorithms



Linked Lists (cont'd)

- To *traverse* the linked list, follow the pointers until *null* reached
  - Use a temporary pointer
  - E.g.

```
Node tmp;
for (tmp = head; tmp != null; tmp = tmp.getNext())
   System.out.println(tmp.getData());
```

Linked Lists (cont'd)

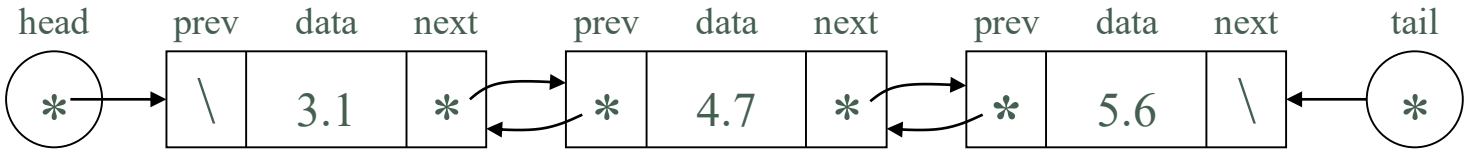
- Insertion and deletion are *O*(1), once the position is known
  - However, if finding the predecessor node requires a search, this is *O*(*n*) in the average and worst cases
  - Insertion/deletion using the head (or tail) pointer is *O*(1)

Linked Lists (cont'd)

- Getting an entry at a position other than the head (or tail, if tail pointer used) requires sequential access
  - Is $O(n)$ in the average and worst cases

Doubly Linked Lists

- Enhance *singly linked lists* by adding pointers to predecessor nodes
  - Allow list traversal from tail to head



Doubly Linked Lists (cont'd)

- Requires modification to node structure:

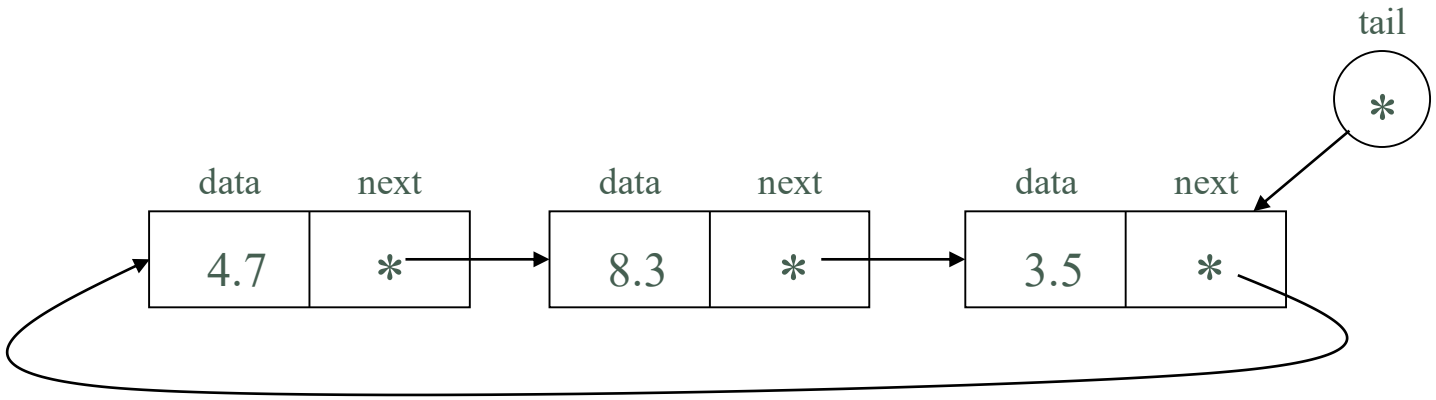
```
public class Node {
    public double data;
    public Node prev, next;

    //  Constructor
    public Node(double d, Node p, Node n) {
        data = d;
        prev = p;
        next = n;
    }
   // Accessor methods
   …
}
```

Doubly Linked Lists (cont'd)

- Insertion and deletion are trickier to implement
  - Especially at the start or end of the list, or when the list is, or about to become, empty
- Java provides a generic implementation with the class `java.util.LinkedList`

Circular Lists

- Are linked lists where the last node points to the first node

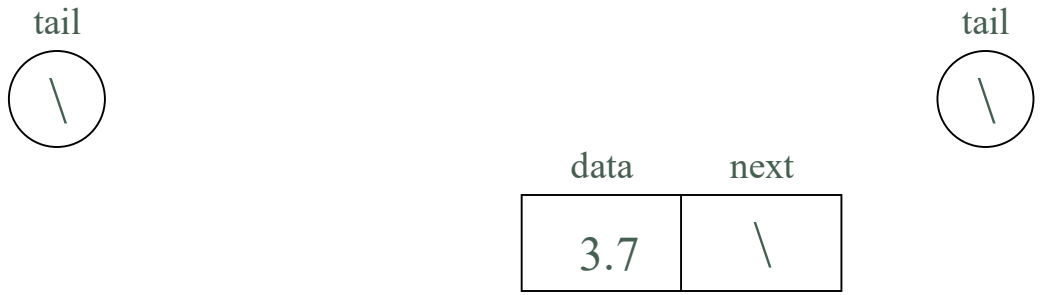


Circular Lists (cont'd)

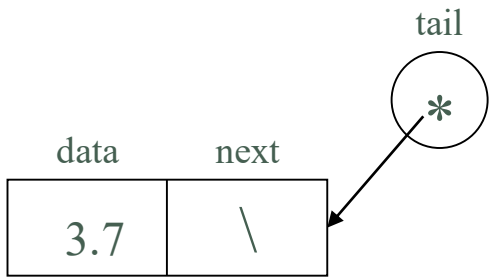
- Only a tail pointer is needed, since the head is pointed to by `tail.getNext()`
- To insert into an empty list:
  - Allocate a new node
    - Set the *data* field to the desired value
  - Assign the node's address to the tail pointer
  - Set the node's *next* field to the same value (a self reference)

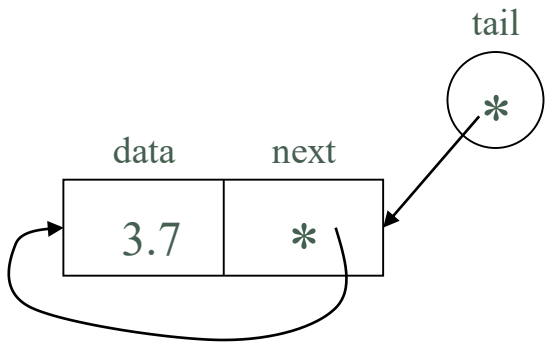
Circular Lists (cont'd)

- E.g.
```
tail = new Node(3.7, null);
tail.setNext(tail);
```



Circular Lists (cont'd)



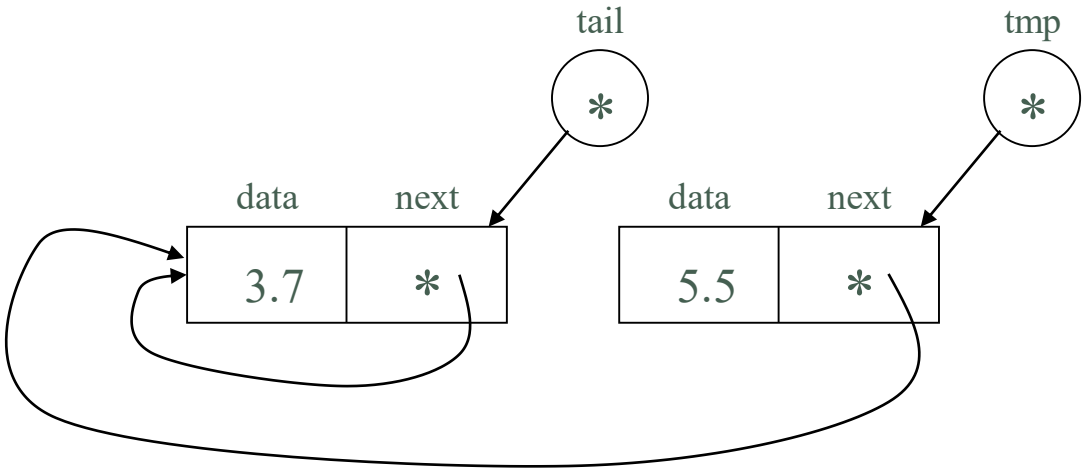


Circular Lists (cont'd)

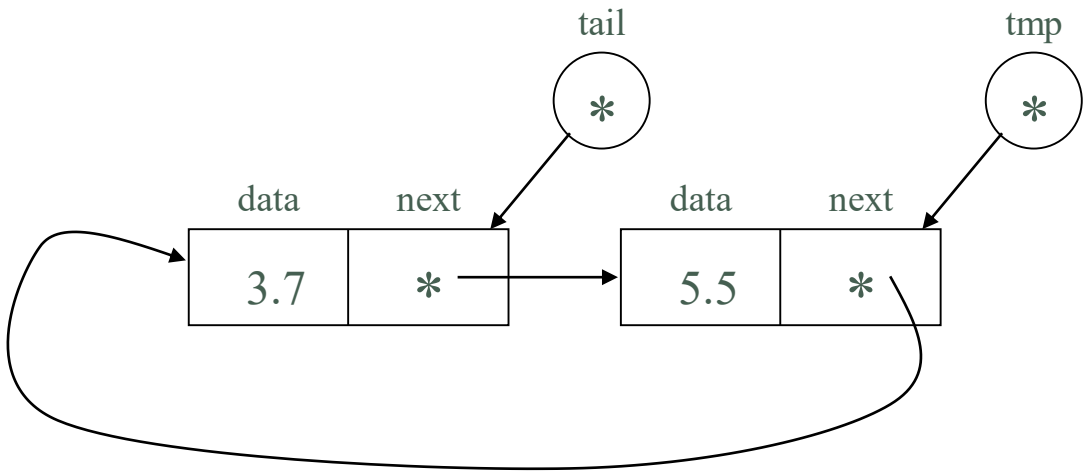
- To insert into the end of the list:
  - Allocate a new node
    - Use a temporary pointer to point to it
    - Set the *data* field to the desired value
    - Set the *next* field to tail.next
  - Set tail.next to the temporary variable
  - Set tail to the temporary variable

Circular Lists (cont'd)

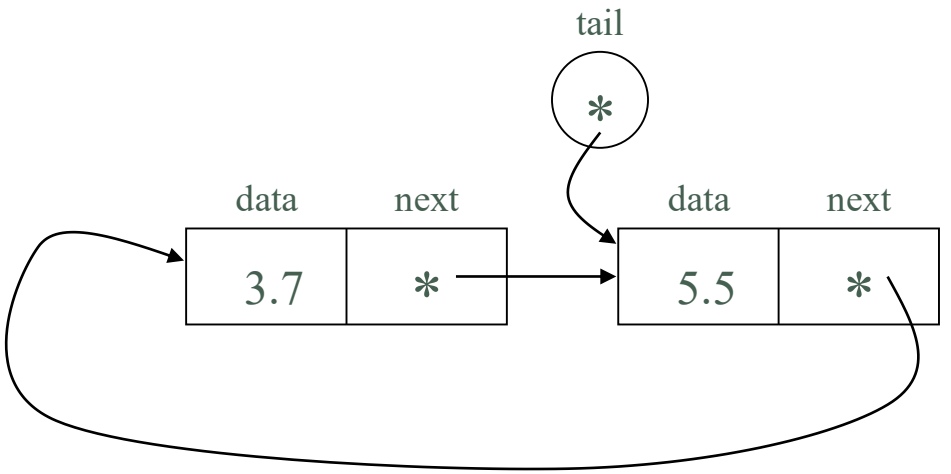
- E.g.
```
tmp = new Node(5.5, tail.getNext());
tail.setNext(tmp);
tail = tmp;
```



Circular Lists (cont'd)



Circular Lists (cont'd)



Applications

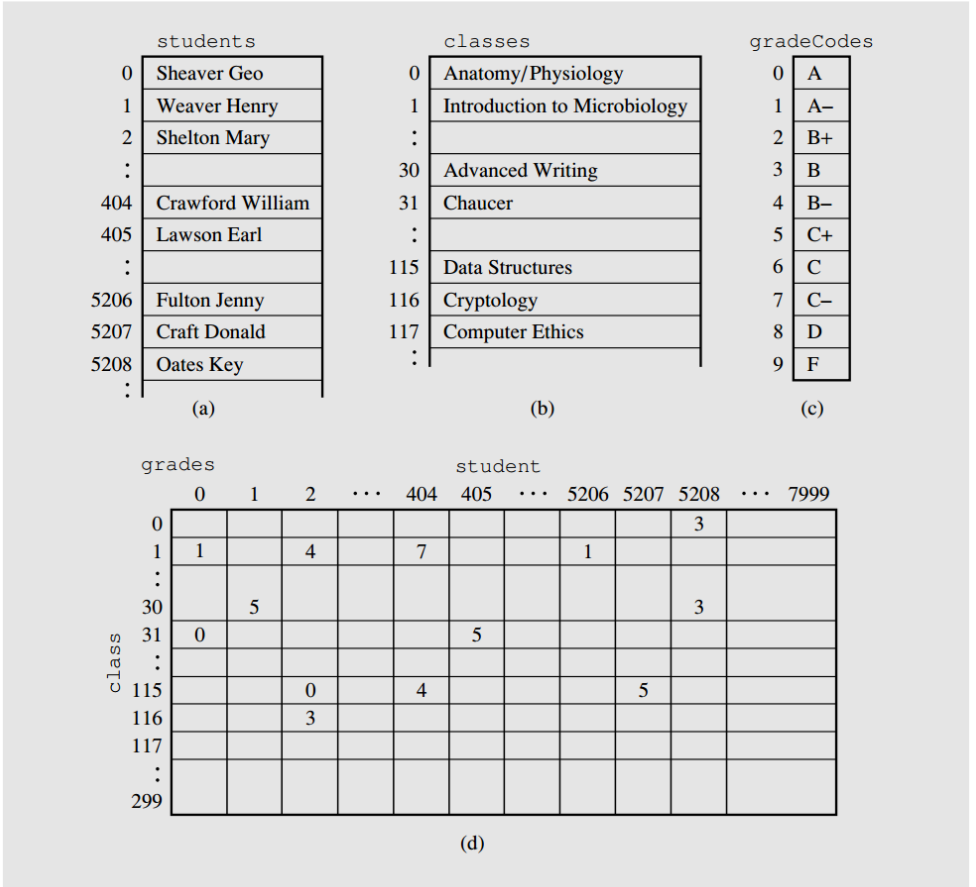
- Linked lists can be sued to save storage for sparse tables

Sparse Tables

- A table is normally implemented using a two-dimensional array
- A *sparse table* has mostly empty cells
  - Thus much space is wasted

Sparse Tables (cont'd)

Arrays and sparse table used for storing student grades.



students

0	Sheaver Geo
1	Weaver Henry
2	Shelton Mary
:	
404	Crawford William
405	Lawson Earl
:	
5206	Fulton Jenny
5207	Craft Donald
5208	Oates Key
:	

(a)

classes

0	Anatomy/Physiology
1	Introduction to Microbiology
:	
30	Advanced Writing
31	Chaucer
:	
115	Data Structures
116	Cryptology
117	Computer Ethics
:	

(b)

gradeCodes

0	A
1	A–
2	B+
3	B
4	B–
5	C+
6	C
7	C–
8	D
9	F

(c)

grades

class \ student	0	1	2	···	404	405	···	5206	5207	5208	···	7999
0										3		
1	1		4		7			1				
:												
30		5								3		
31	0					5						
:												
115			0		4				5			
116			3									
117												
:												
299												

(d)

Sparse Tables (cont'd)

- Space can be saved by using:
  - One-dimensional arrays of references for:
    - Columns
    - Rows
  - Linked lists of nodes, where each node represents a filled cell. Each has:
    - Data
    - A pointer to the next filled cell in the column
    - A pointer to the next filled cell in the row

Sparse Tables (cont'd)

Two-dimensional arrays for storing student grades.

classesTaken1

	0	1	2	...	404	405	...	5206	5207	5208	... 7999
0	1	30	1		1	31		1	115	0	
1	31		115		115	64		33	121	30	
2	124		116		218	120		86	146	208	
3	136				221			121	156	211	
4					285			203		234	
5					292						
6											
7											

classesTaken2

	0	1	2	...	404	405	...	5206	5207	5208	... 7999
0	1	5	4		7	5		1	5	3	
1	0		0		4	5		1	5	3	
2	0		3		6	4		0	3	2	
3	2				5			2	0	3	
4					3			2		1	
5					3						
6											
7											

(a)

studentsInClass1

	0	1	...	30	31	...	115	116	... 299
0	5208	0		1	0		2	2	
1		2		5208	405		404		
2		404					5207		
3		5206							
249									

studentsInClass2

	0	1	...	30	31	...	115	116	... 299
0	3	0		5	0		0	3	
1		0		3	5		4		
2		7					5		
3		1							
249									

(b)

Summary

- Linked list is a linear data structure that consists of zero or more nodes.
- Linked list is like a chain of nodes that are connected together by pointers.
- Each node contains data and a pointer to the next node.
- Linked list has a head (and sometimes tail) pointer to get access to the first (and last) element in the list.
- You can delete or insert a node to a linked list in O(1)
- Finding a specific node in a linked list requires O(n).

Summary (Cont'd)

- In doubly linked list each node has a pointer to predecessor, allow list traversal from tail to head.
- In Circular Lists the last node's next pointer points to the first node.
- Deletion and insertion to both doubly and circular lists need careful attention.
- Sparse tables has mostly empty cells

Review Questions

- What is a linked list?
- How can linked list grow and shrink without limit?
- How can we insert a node to the beginning, middle and end of a linked list?
- How can we delete a node to the beginning, middle and end of a linked list?
- What is the complexity of getting an entry at a position other than the head (or tail, if tail pointer used)?

Review Questions (Cont'd)

- What is the difference between singly and doubly linked lists?
- How can we insert a node to an empty circular list?
- How can we insert a node to the end of a circular list?
- What is a sparse table?
- How can we implement a sparse table?

Any questions?