

## Topic 6

# Linked Data Structures

4-1

## Objectives

- Describe linked structures
- Compare linked structures to array-based structures
- Explore the techniques for managing a linked list
- Discuss the need for a separate node class to form linked structures

4-2

## Array Limitations

- What are the limitations of an array, as a data structure?
  - Fixed size
  - Physically stored in consecutive memory locations
  - To insert or delete items, may need to shift data

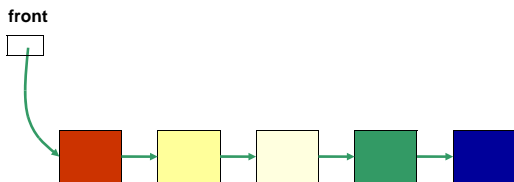
4-3

## Linked Data Structures

- A **linked** data structure consists of items that are linked to other items
  - How? each item **points to** another item
- **Singly linked list**: each item points to the next item
- **Doubly linked list**: each item points to the next item *and* to the previous item

4-4

## Conceptual Diagram of a Singly-Linked List



4-5

## Advantages of Linked Lists

- The items do **not** have to be stored in consecutive memory locations: the successor can be anywhere physically
  - So, can insert and delete items without shifting data
  - Can increase the size of the data structure easily
- Linked lists can grow **dynamically** (i.e. at run time) – the amount of memory space allocated can grow and shrink as needed

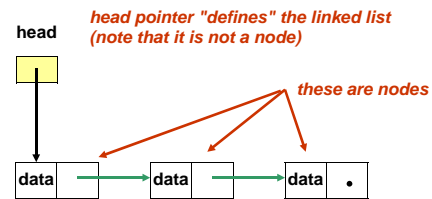
4-6

## Nodes

- A linked list is an ordered sequence of items called **nodes**
  - A **node** is the basic unit of representation in a linked list
- A **node** in a **singly linked list** consists of two fields:
  - A **data** portion
  - A **link (pointer)** to the **next** node in the structure
- The first item (node) in the linked list is accessed via a **front** or **head** pointer
  - The linked list is defined by its head (this is its starting point)

4-7

## Singly Linked List



4-8

## Linked List

**Note:** we will hereafter refer to a singly linked list just as a "**linked list**"

- **Traversing the linked list**
  - How is the first item accessed?
  - The second?
  - The last?
- What does the last item point to?
  - We call this the **null link**

4-9

## Discussion

- How do we get to an item's successor?
- How do we get to an item's predecessor?
- How do we access, say, the 3rd item in the linked list?
- How does this differ from an array?

4-10

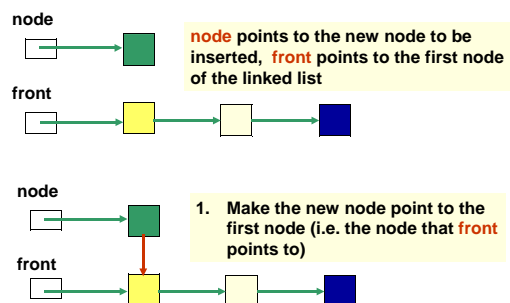
## Linked List Operations

We will now examine linked list operations:

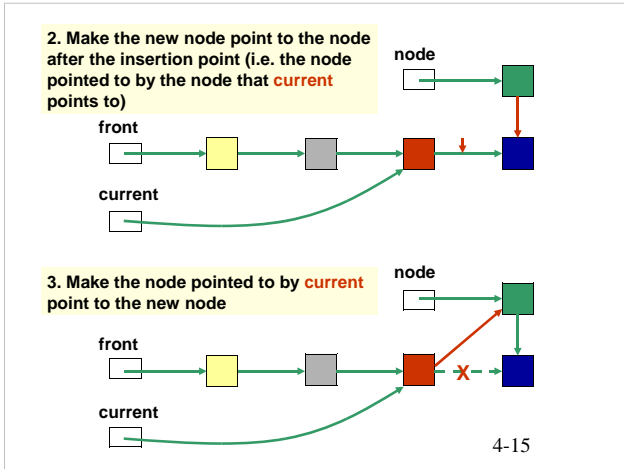
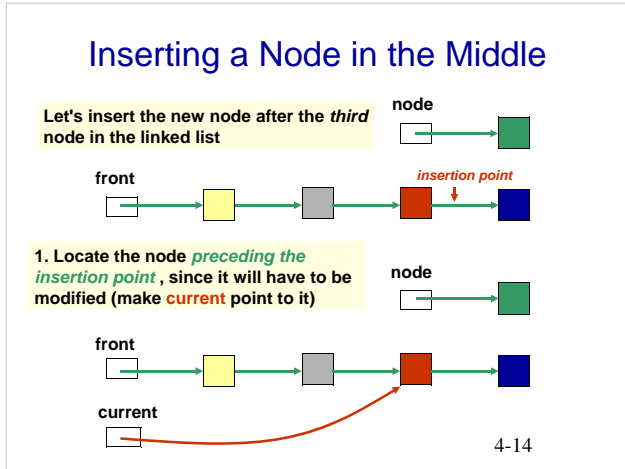
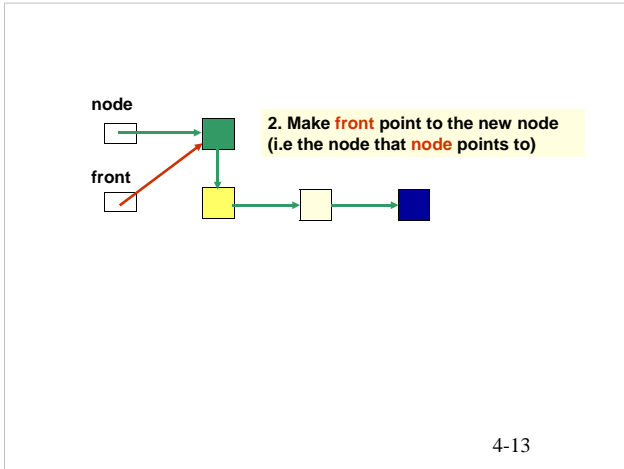
- **Add** an item to the linked list
  - We have 3 situations to consider:
    - insert a node **at the front**
    - insert a node **in the middle**
    - insert a node **at the end**
- **Delete** an item from the linked list
  - We have 3 situations to consider:
    - delete the node **at the front**
    - delete an **interior** node
    - delete the **last** node

4-11

## Inserting a Node at the Front



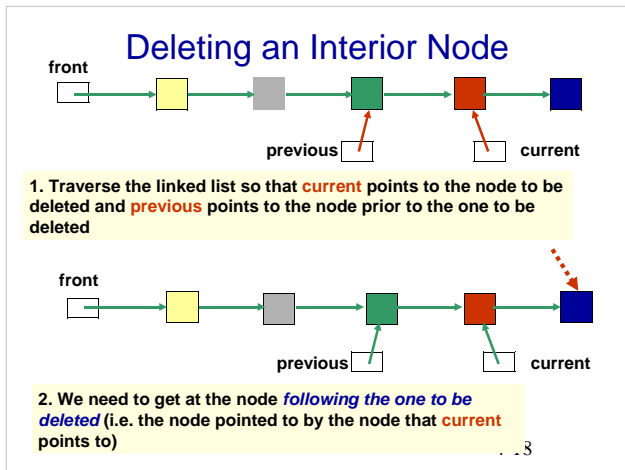
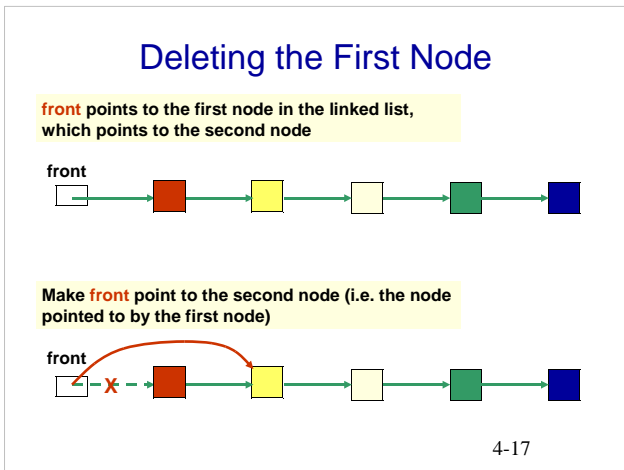
4-12

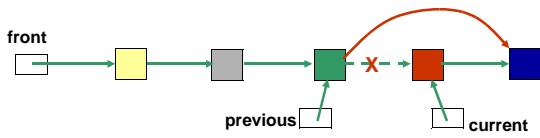


### Discussion

- Inserting a node at the front is a special case; why?
- Is inserting a node at the end a special case?

4-16





3. Make the node that **previous** points to, point to the node following the one to be deleted

4-19

## Discussion

- Deleting the node at the front is a special case; why?
- Is deleting the last node a special case?

4-20

## References As Links

- Recall that in Java, a reference variable contains a reference or **pointer** to an object
  - We can show a reference variable **obj** as **pointing to** an object:



- A linked structure uses **references** to link one object to another

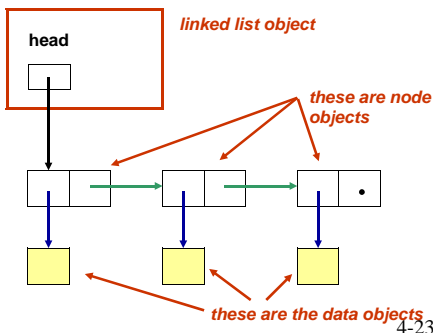
4-21

## Implementation of Linked List

- In Java, a linked list is a list of **node objects**, each of which consists of two references:
  - A reference to the **data object**
  - A reference to the **next node object**
- The **head pointer** is the reference to the linked list, *i.e.* to the first node object in the linked list
- The last node has the **null** value as its reference to the “next” node object

4-22

## Linked List of Node Objects



4-23

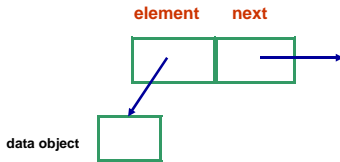
## Node Objects

- For our linked list implementations, we will define a class called **LinearNode** to represent a node
  - It will be defined for the **generic type T**
- Why is it a good idea to have separate node class?
- Note that it is called “**LinearNode**” to avoid confusion with a different class that will define nodes for non-linear structures later

4-24

## The **LinearNode** Class

- Attributes (instance variables):
  - element**: a reference to the data object
  - next**: a reference to the next node
    - so it will be of type LinearNode



4-25

## The LinearNode Class

- Methods: we only need
  - Getters
  - Setters

4-26

```
public class LinearNode<T>
{
    private LinearNode<T> next;
    private T element;

    public LinearNode() {
        next = null;
        element = null;
    }

    public LinearNode (T elem){
        next = null;
        element = elem;
    }
    // cont'd..
}
```

**LinearNode.java**

4-27

```
public LinearNode<T> getNext() {
    return next;
}

public void setNext (LinearNode<T> node){
    next = node;
}

public T getElement() {
    return element;
}

public void setElement (T elem) {
    element = elem;
}
}
```

**LinearNode.java  
(cont'd)**

4-28

## Example: Create a LinearNode Object

- Example: create a node that contains the integer 7

```
Integer intObj = new Integer(7);
LinearNode<Integer> inode =
    new LinearNode<Integer> (intObj);
```

or

```
LinearNode<Integer> inode =
    new LinearNode<Integer> (new Integer(7));
```

4-29

## Exercise: Build a Linked List

- Exercise: create a linked list that contains the integers 1, 2, 3, ..., 10

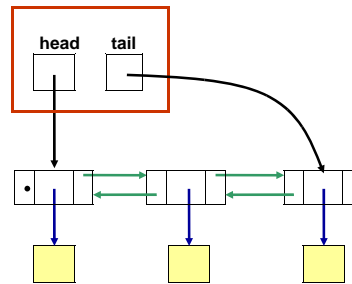
4-30

## Doubly Linked Lists

- In a **doubly linked list**, each node has two links:
  - A reference to the **next node** in the list
  - A reference to the **previous node** in the list
    - What is the “previous” reference of the first node in the list?
- What is the advantage of a doubly linked list?
- What is a disadvantage?

4-31

## Doubly Linked List



4-32