LINKED LISTS

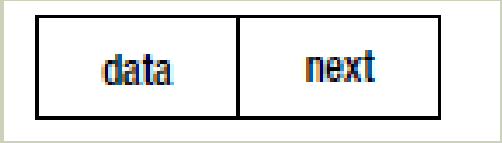
Chapter 1

OBJECTIVES

- Introduction
- 1.1 Basic operations
  - 1.1.1 Counting nodes
  - 1.1.2 Searching
  - 1.1.3 Printing the values

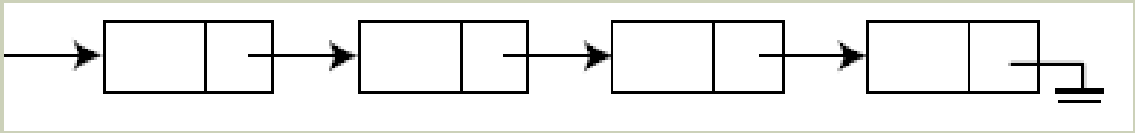
INTRODUCTION

- When values are stored in a 1D array (*x*[0] to *x*[n]), they can be thought of as being organized as a "linear list." A linear list means that the nodes are arranged in a linear order.
- In many situations, we use an array for representing a linear list. But we can also represent such a list by using an organization in which each node in the list points *explicitly* to the next node. This new organization is referred to as a *linked list*.
- In a linked list, each node contains a pointer that points to the next node in the list. We can think of each node as a cell with two components, like this:



INTRODUCTION

- The data item can actually be one or more fields, and next is a reference to ("points to") the next node of the list. We can say that the next reference is a link between two nodes.
- Since the next field of the *last* node does not refer (point) to anything, we must set it to null.
- In addition to the nodes of the list, we need an object reference variable (head) that "points to" the first node in the list known as the head node.
- If the list is empty, the value of head is null.



1.1 BASIC OPERATIONS

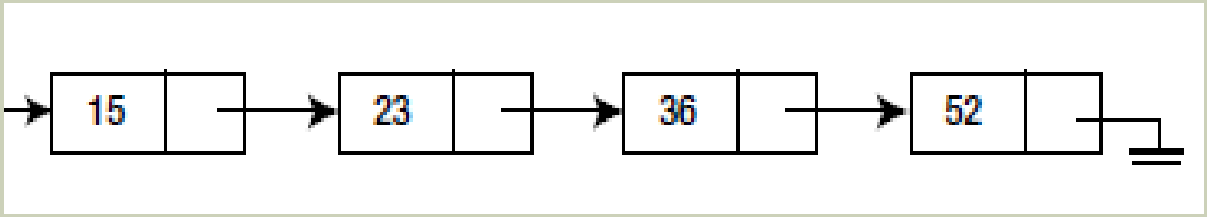
1.1.1 Counting nodes

1.1.2 Searching

1.1.3 Printing the values

1.1.1 COUNTING NODES

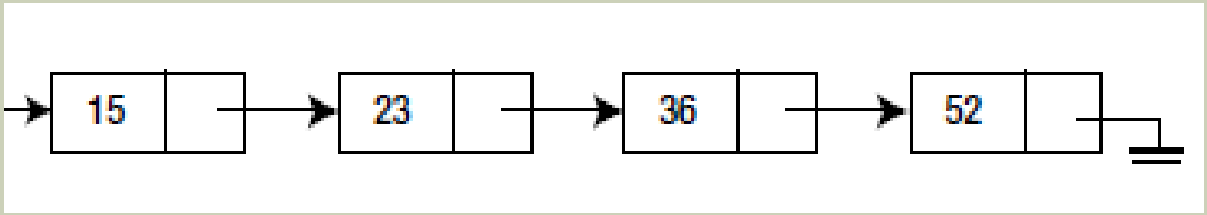
Suppose we want to count the nodes in the below linked list. We initialize a counter to zero and start from the head node and then we keep moving until the end of the list (until null is reached) while incrementing the counter at each node.




```
public int countNodes() {
    int count = 0; // Details of the class Node will be explained later
    Node curr = head; // start from the first node
    while(curr != null){ // as long as we did not reach the end
        count++; // count this node
        curr = curr.next; // move to the next node
    }
    return count;
}
```

1.1.2 SEARCHING

Suppose we want to search for a value in the below linked list. We start from the head node and then we keep moving until the end of the list (until null is reached). Once we find the value we stop the search.

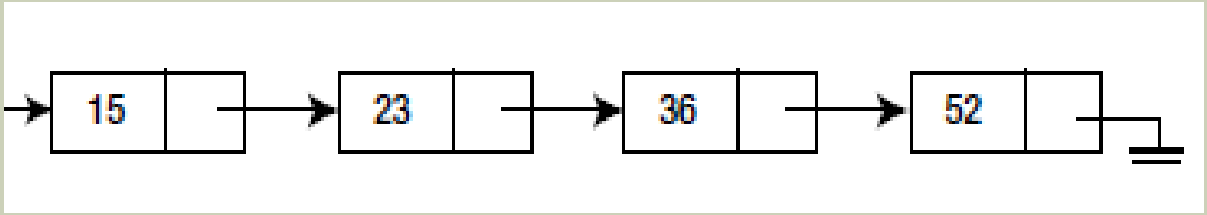



```
public boolean search(NodeData item) {
    Node curr = head; // start from the first node
    while(curr != null) {// as long as we did not reach the end
        if(curr.data.compareTo(item) == 0)
            return true; // item found => stop the search
        curr = curr.next; // move to the next node
    }
    return false; // item not found
}
```

// The details of the class NodeData will be explained later

1.1.3 PRINTING THE VALUES

Suppose we want to print the values in the below linked list. We start from the head node and then we keep moving until the end of the list (until null is reached) printing the values in each node.




```
public void print() {
    Node curr = head; // start from the first node
    while(curr != null){ // as long as we did not reach the end
        System.out.print(curr.data.value + " ");
        curr = curr.next; // move to the next node
    }
}
```