Data Abstraction & Problem Solving with
JAVA
3rd Edition Walls & Mirrors
JANET J. PRICHARD | FRANK M. CARRANO

Chapter 5

Linked Lists

Preliminaries

- Options for implementing an ADT
  - Array
    - Has a fixed size
    - Data must be shifted during insertions and deletions
  - Linked list
    - Is able to grow in size as needed
    - Does not require the shifting of items during insertions and deletions

Preliminaries

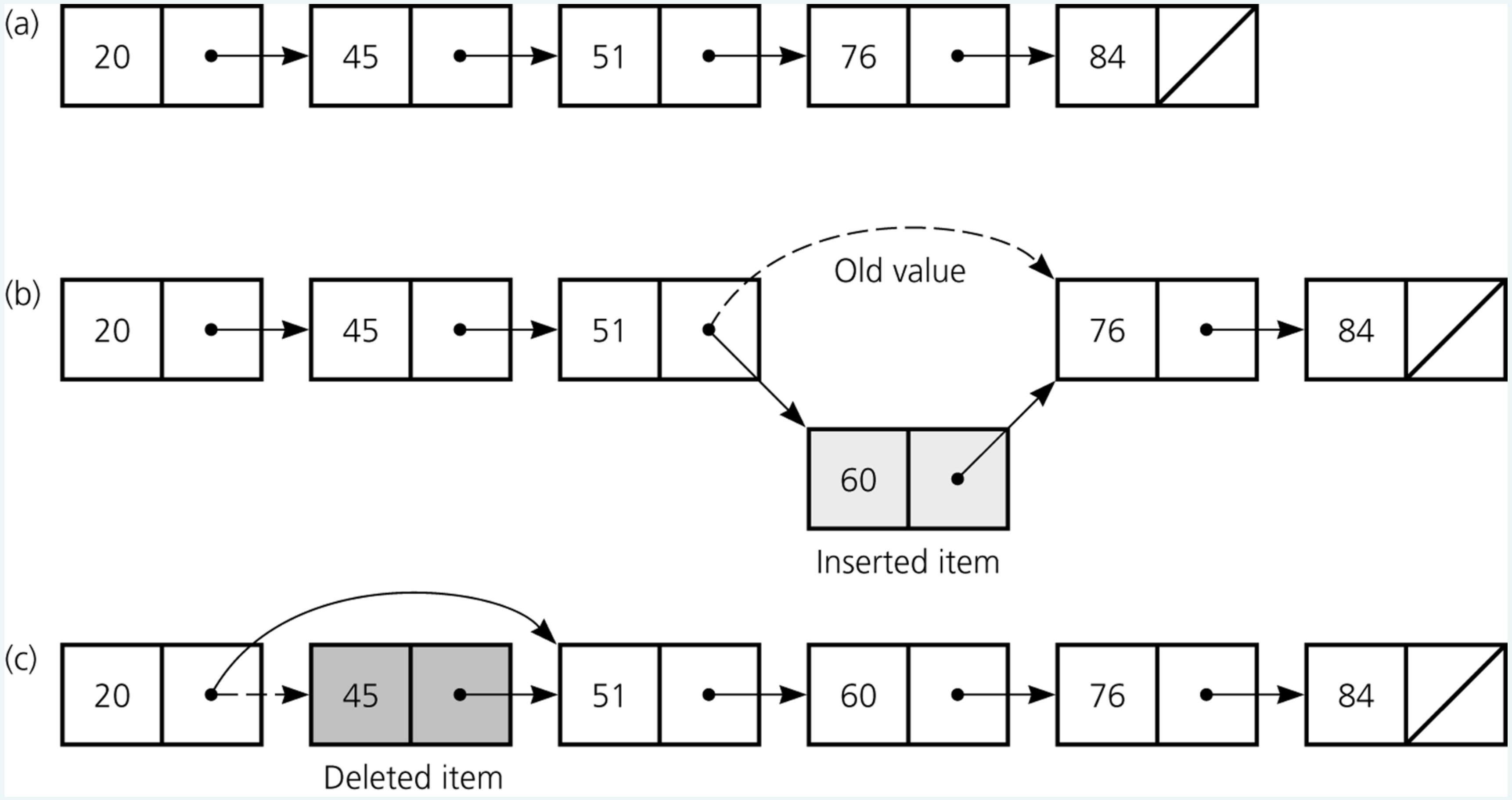


Figure 5-1

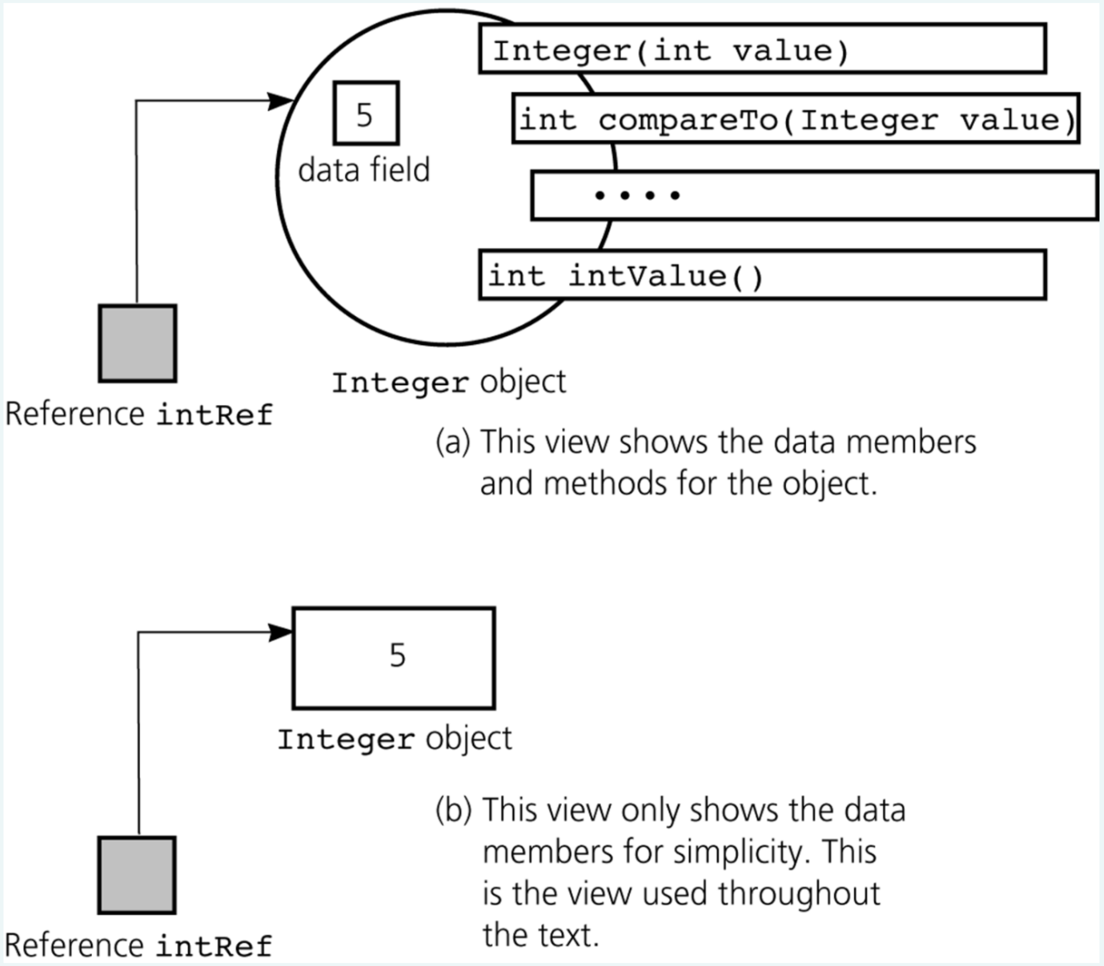
a) A linked list of integers; b) insertion; c) deletion

Object References

- A reference variable
  - Contains the location of an object
  - Example

    ```
    Integer intRef;
    intRef = new Integer(5);
    ```
  - As a data field of a class
    - Has the default value `null`
  - A local reference variable to a method
    - Does not have a default value

Object References



Integer(int value)
int compareTo(Integer value)
• • • •
int intValue()
5
data field
Reference intRef
Integer object
(a) This view shows the data members and methods for the object.
5
Integer object
Reference intRef
(b) This view only shows the data members for simplicity. This is the view used throughout the text.

Figure 5-2

A reference to an ***Integer*** object

Object References

- When one reference variable is assigned to another reference variable, both references then refer to the same object

```
Integer p, q;
p = new Integer(6);
q = p;
```

- A reference variable that no longer references any object is marked for garbage collection

Object References

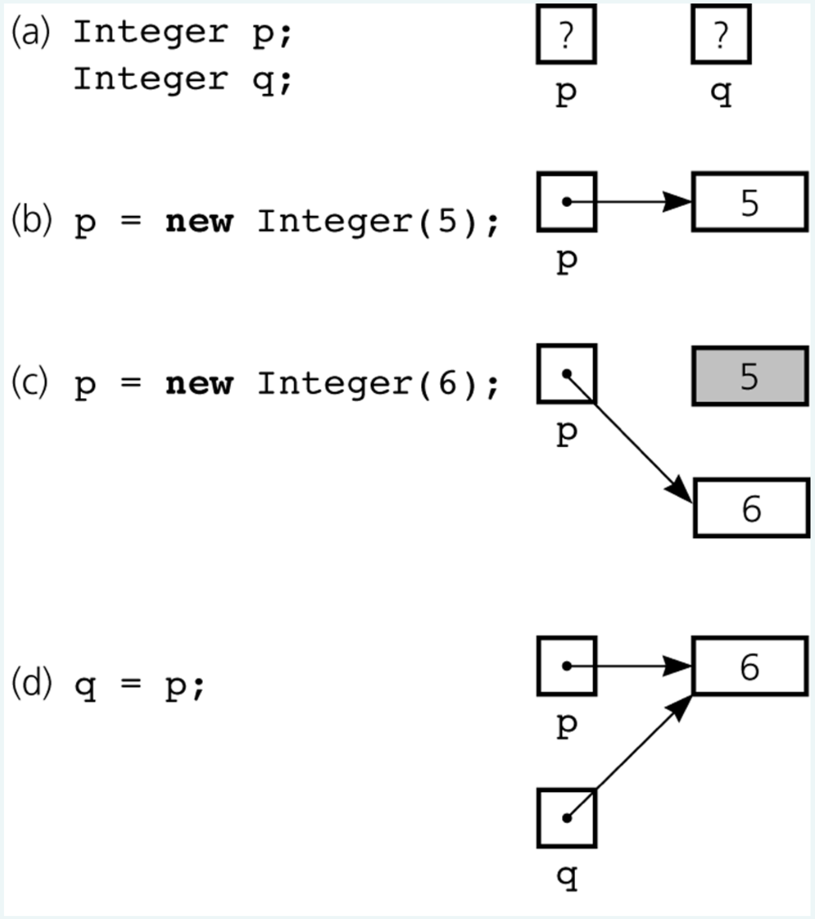


Figure 5-3a-d

a) Declaring reference variables; b) allocating an object; c) allocating another object, with the dereferenced object marked for garbage collection

Object References

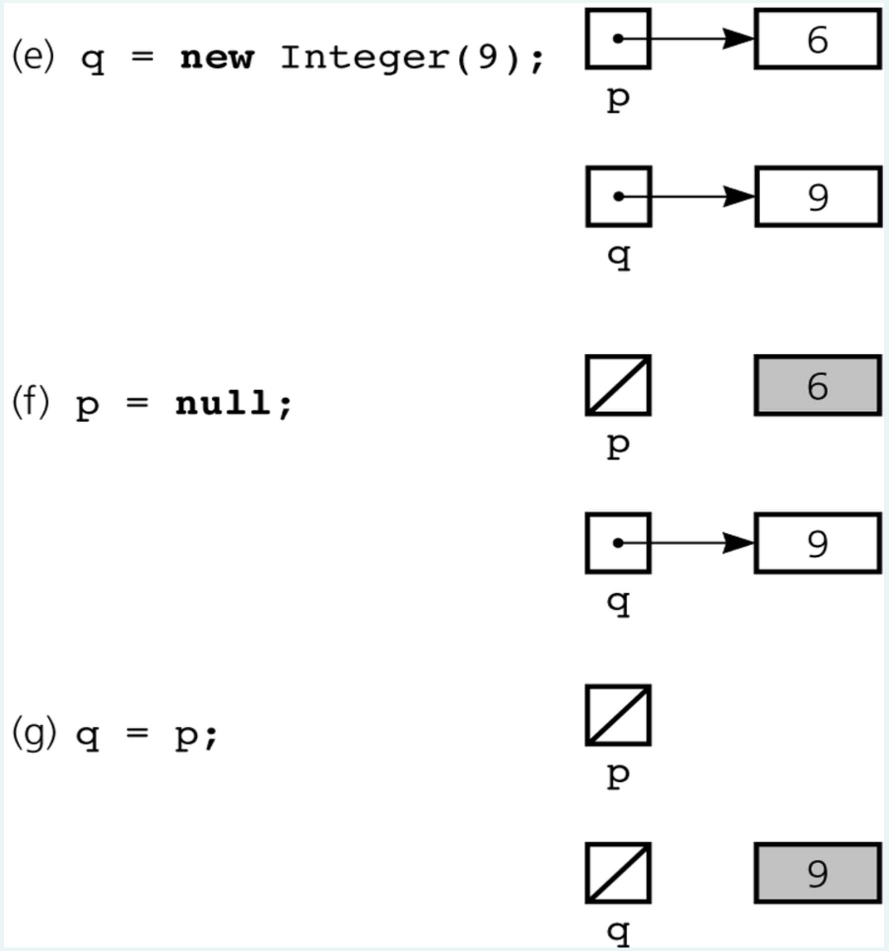


Figure 5-3e-g

e) allocating an object; f) assigning ***null*** to a reference variable; g) assigning a reference with a ***null*** value

Object References

- An array of objects
  - Is actually an array of references to the objects
  - Example

```
Integer[] scores = new Integer[30];
```

  - Instantiating Integer objects for each array reference

```
scores[0] = new Integer(7);
scores[1] = new Integer(9); // and so on …
```

Object References

- Equality operators (== and !=)
  - Compare the values of the reference variables, not the objects that they reference
- `equals` method
  - Compares objects field by field
- When an object is passed to a method as an argument, the reference to the object is copied to the method's formal parameter
- Reference-based ADT implementations and data structures use Java references

Resizable Arrays

- The number of references in a Java array is of fixed size
- Resizable array
  - An array that grows and shrinks as the program executes
  - An illusion that is created by using an allocate and copy strategy with fixed-size arrays
- `java.util.Vector` class
  - Uses a similar technique to implement a growable array of objects

Reference-Based Linked Lists

- Linked list
  - Contains nodes that are linked to one another
  - A node contains both data and a link to the next item
  - Access is package-private

```
package List;
class Node {
    Object item;
    Node next;
    // constructors, accessors,
    // and mutators …
} // end class Node
```

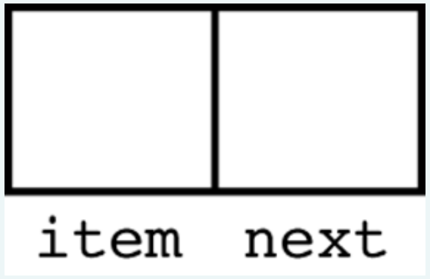


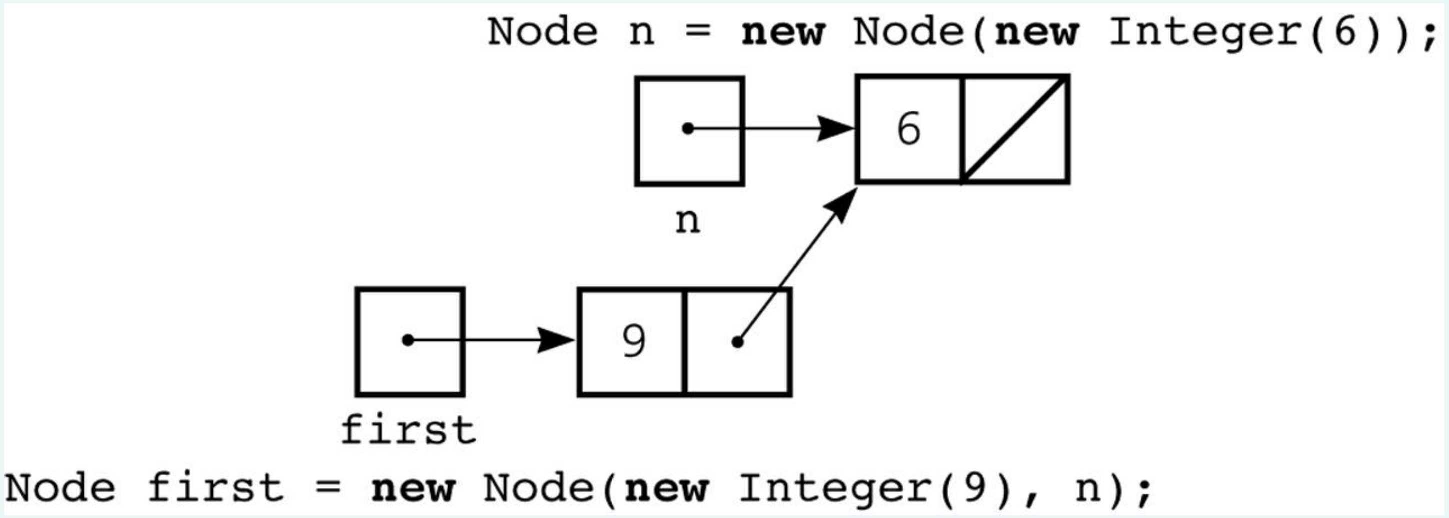
Figure 5-5

A node

Reference-Based Linked Lists

- Using the Node class

```
Node n = new Node (new Integer(6));
Node first = new Node (new Integer(9), n);
```



Node n = new Node(new Integer(6));
6
n
9
first
Node first = new Node(new Integer(9), n);

Figure 5-7

Using the ***Node*** constructor to initialize a data field and a link value

Reference-Based Linked Lists

- Data field `next` in the last node is set to `null`
- `head` reference variable
  - References the list's first node
  - Always exists even when the list is empty

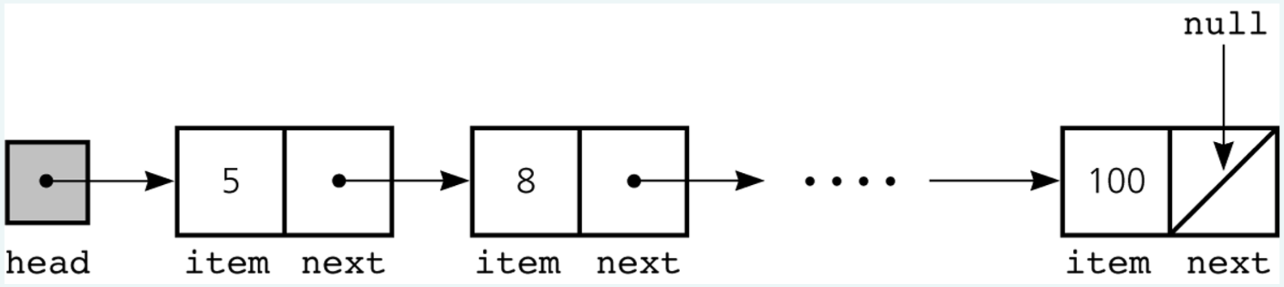


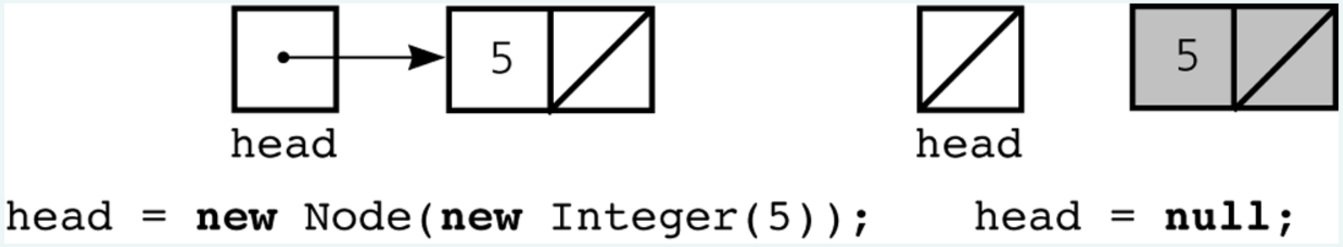
Figure 5-8

A ***head*** reference to a linked list

Reference-Based Linked Lists

- `head` reference variable can be assigned `null` without first using `new`
  - Following sequence results in a lost node

```
head = new Node(); // Don't really need to use new here
head = null; // since we lose the new Node object here
```



```
head = new Node(new Integer(5));     head = null;
```

Figure 5-9

A lost node

Programming with Linked Lists: Displaying the Contents of a Linked List

- `curr` reference variable
  - References the current node
  - Initially references the first node
- To display the data portion of the current node

```
System.out.println(curr.item);
```

- To advance the current position to the next node

```
curr = curr.next;
```

Displaying the Contents of a Linked List

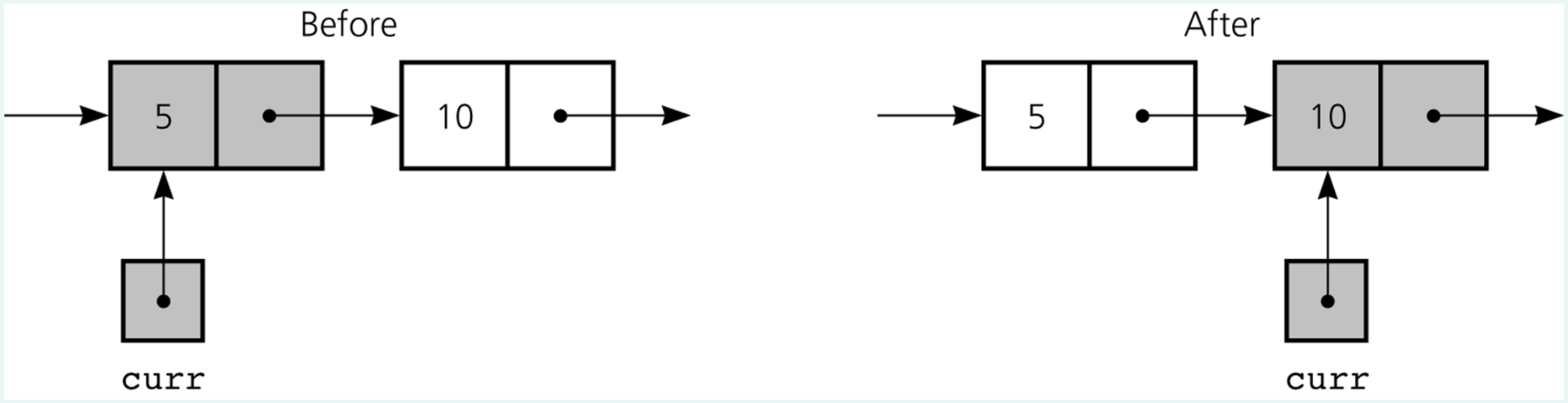


Figure 5-10

The effect of the assignment ***`curr = curr.next`***

Displaying the Contents of a Linked List

- To display all the data items in a linked list

```
for (Node curr = head; curr != null; curr =
    curr.next) {
 System.out.println(curr.item);
} // end for
```

Deleting a Specified Node from a Linked List

- To delete node N which `curr` references
  - Set `next` in the node that precedes N to reference the node that follows N

    ```
    prev.next = curr.next;
    ```

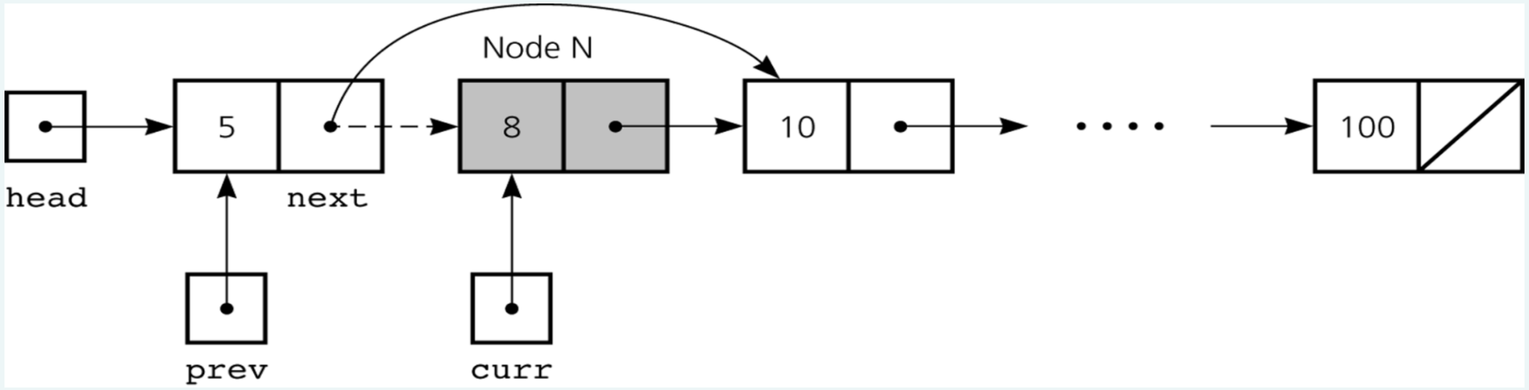


Figure 5-11

Deleting a node from a linked list

Deleting a Specified Node from a Linked List

- Deleting the first node is a special case

```
    head = head.next;
```

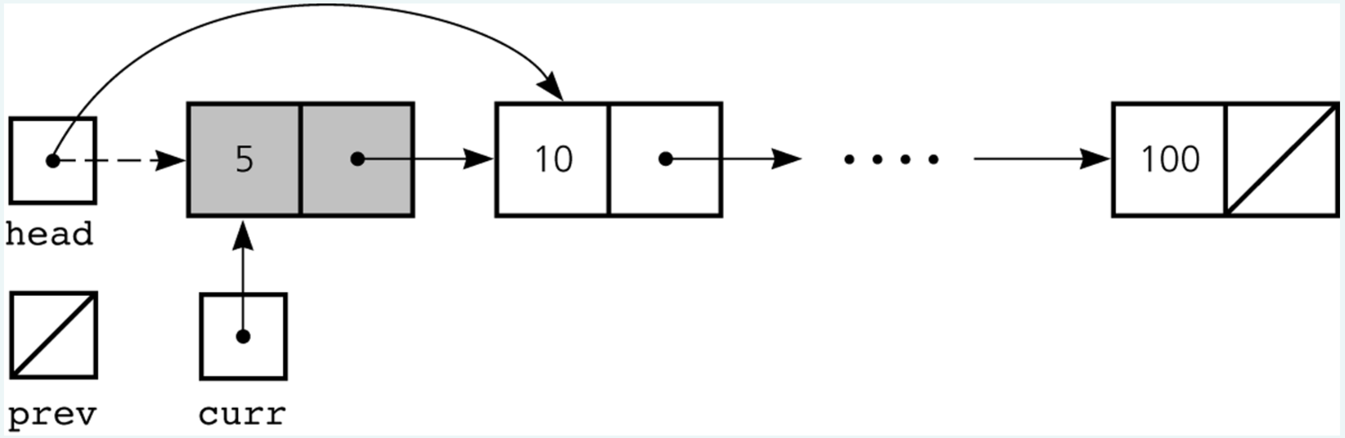


Figure 5-12

Deleting the first node

Deleting a Specified Node from a Linked List

- To return a node that is no longer needed to the system

  ```
  curr.next = null;
  curr = null;
  ```

- Three steps to delete a node from a linked list
  - Locate the node that you want to delete
  - Disconnect this node from the linked list by changing references
  - Return the node to the system

Inserting a Node into a Specified Position of a Linked List

- To create a node for the new item

```
newNode = new Node(item);
```

- To insert a node between two nodes

```
newNode.next = curr;
prev.next = newNode;
```

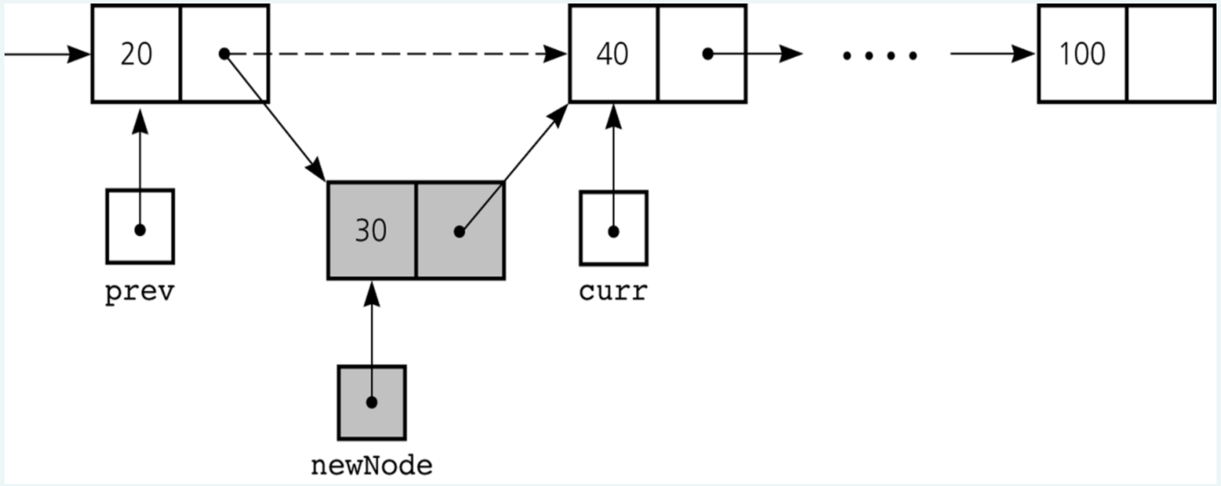


Figure 5-13

Inserting a new node into a linked list

Inserting a Node into a Specified Position of a Linked List

- To insert a node at the beginning of a linked list

```
newNode.next = head;
head = newNode;
```

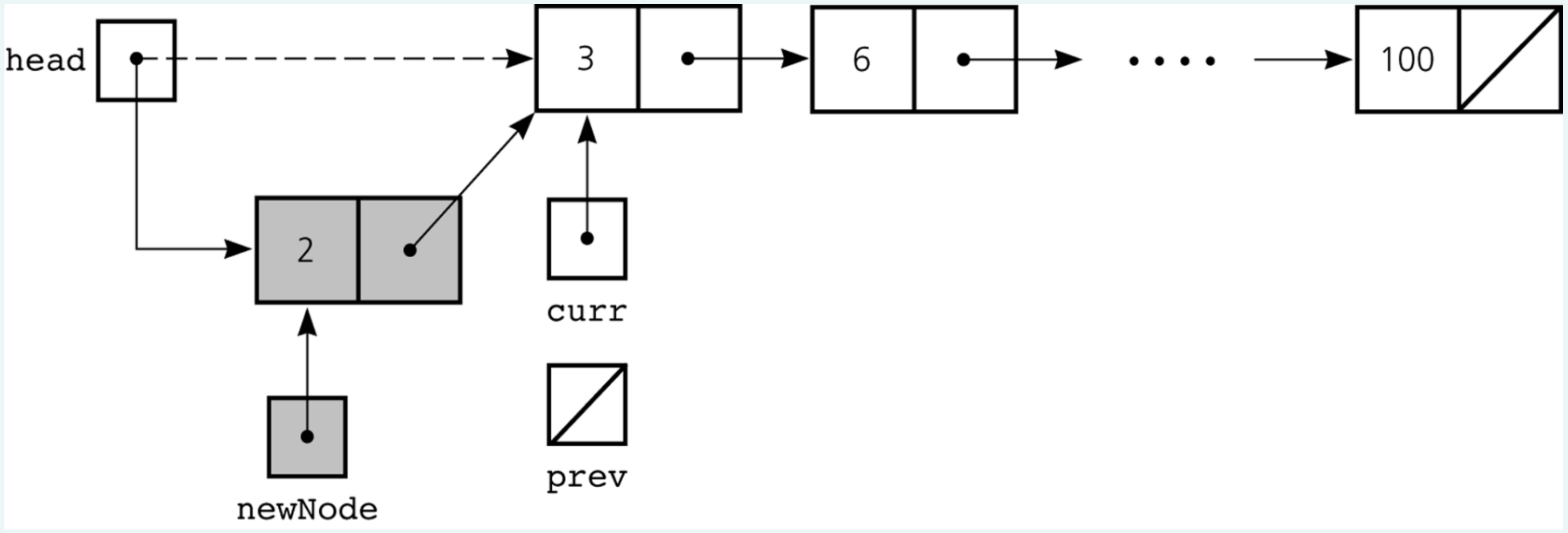


Figure 5-14

Inserting at the beginning of a linked list

Inserting a Node into a Specified Position of a Linked List

- Inserting at the end of a linked list is not a special case if `curr` is `null`

```
newNode.next = curr;
prev.next = newNode;
```

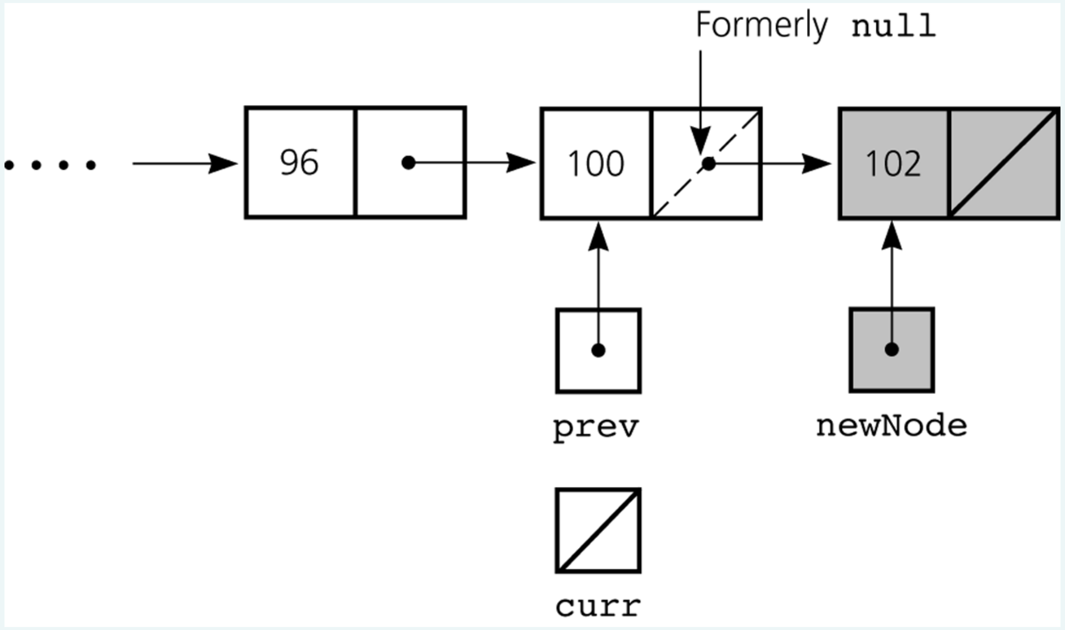


Figure 5-15

Inserting at the end of a linked list

Inserting a Node into a Specified Position of a Linked List

- Three steps to insert a new node into a linked list
  - Determine the point of insertion
  - Create a new node and store the new data in it
  - Connect the new node to the linked list by changing references

Determining `curr` and `prev`

- Determining the point of insertion or deletion for a sorted linked list of objects

```
for ( prev = null, curr = head;
        (curr != null) &&
        (newValue.compareTo(curr.item) > 0);
       prev = curr, curr = curr.next ) {
} // end for
```

Please open file *carrano_ppt05_B.ppt* to continue viewing chapter 5.