King Fahd University of Petroleum & Minerals
College of Computer Science & Engineering

Information & Computer Science Department

Unit 3

Linked Lists

Information and Computer Science ICS

Reading Assignment

- "Data Structures and Algorithms in Java", 3rd Edition, Adam Drozdek, Cengage Learning, ISBN 978-9814239233
  - Chapter 3 (Sections 1 – 3)
  - Sections 3.4-3.9 are not included.

Objectives

Discuss the following topics:

- Singly Linked Lists
- Doubly Linked Lists
- Circular Lists

Singly Linked Lists

- A **linked structure** is a collection of nodes storing data and links to other nodes
- A **linked list** is a data structure composed of nodes, each node holding some information and a reference to another node in the list
- A **singly linked list** is a node that has a link only to its successor in this sequence

Singly Linked Lists (continued)

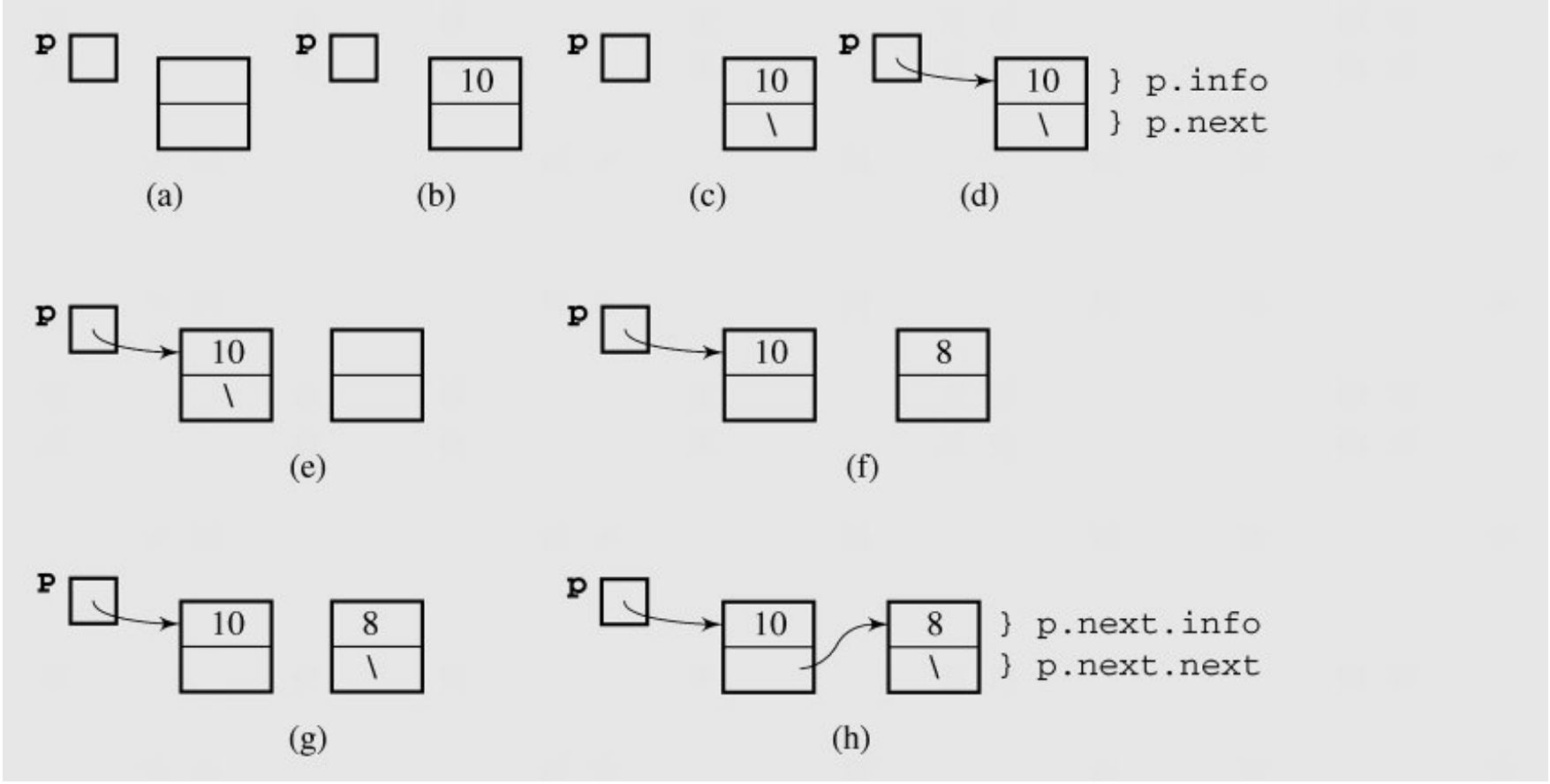

Figure 3-1 A singly linked list

Singly Linked Lists (continued)

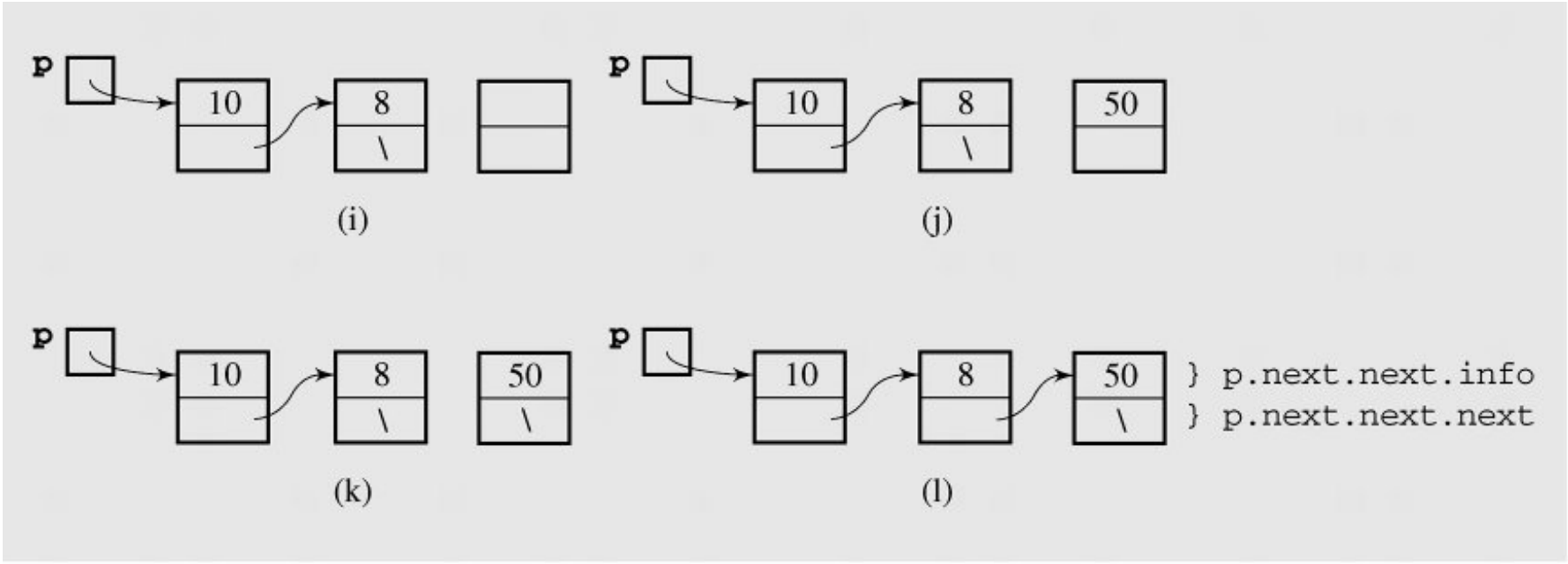

Figure 3-1 A singly linked list (continued)

A Node in a Singly Linked List

```
//***********  SLLNode.java  ******************
// node in a generic singly linked list class

public class SLLNode<T> {
    public T info;
    public SLLNode<T> next;
    public SLLNode() {
        this(null,null);
    }
    public SLLNode(T el) {
        this(el, null);
    }
    public SLLNode(T el, SLLNode<T> ptr) {
        info = el; next = ptr;
    }
}
```

A List in a Singly Linked List

```
//*****************  SLL.java  ********************
//  a generic singly linked list class

public class SLL<T> {
    protected SLLNode<T> head, tail;
    public SLL() {
        head = tail = null;
    }
    public boolean isEmpty() {
        return head == null;
    }
```

Singly Linked List Visualization

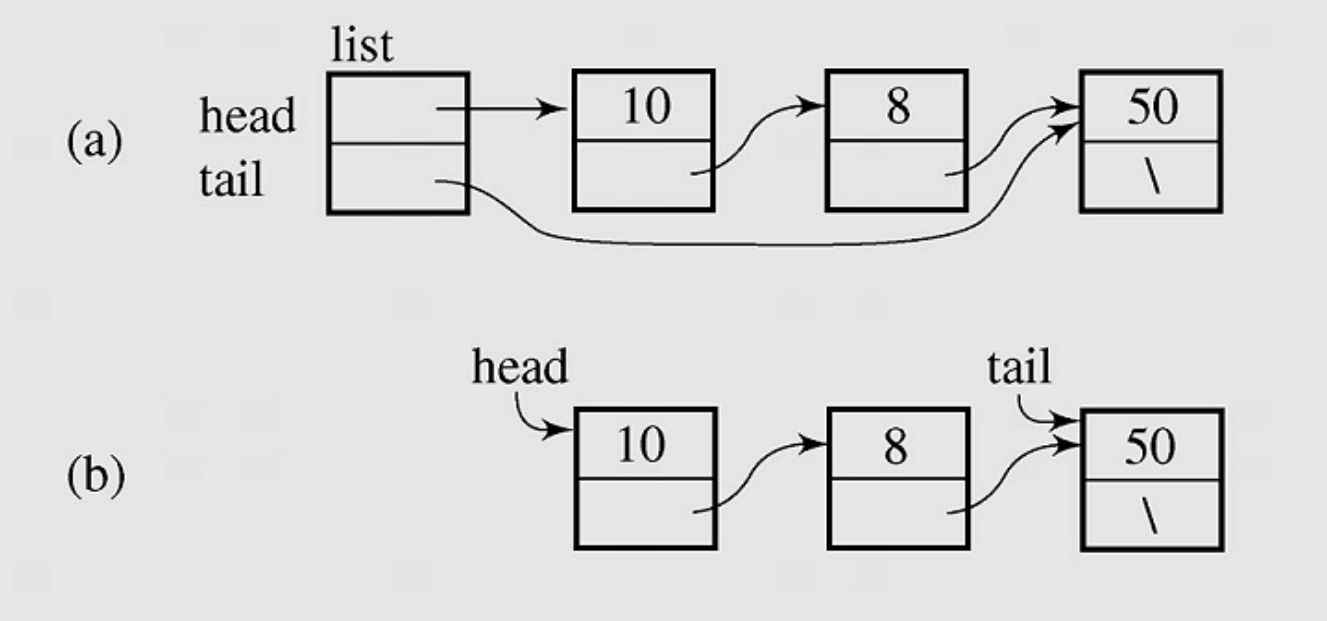

Figure 3-3 A singly linked list of integers

Insertion into Singly Linked Lists

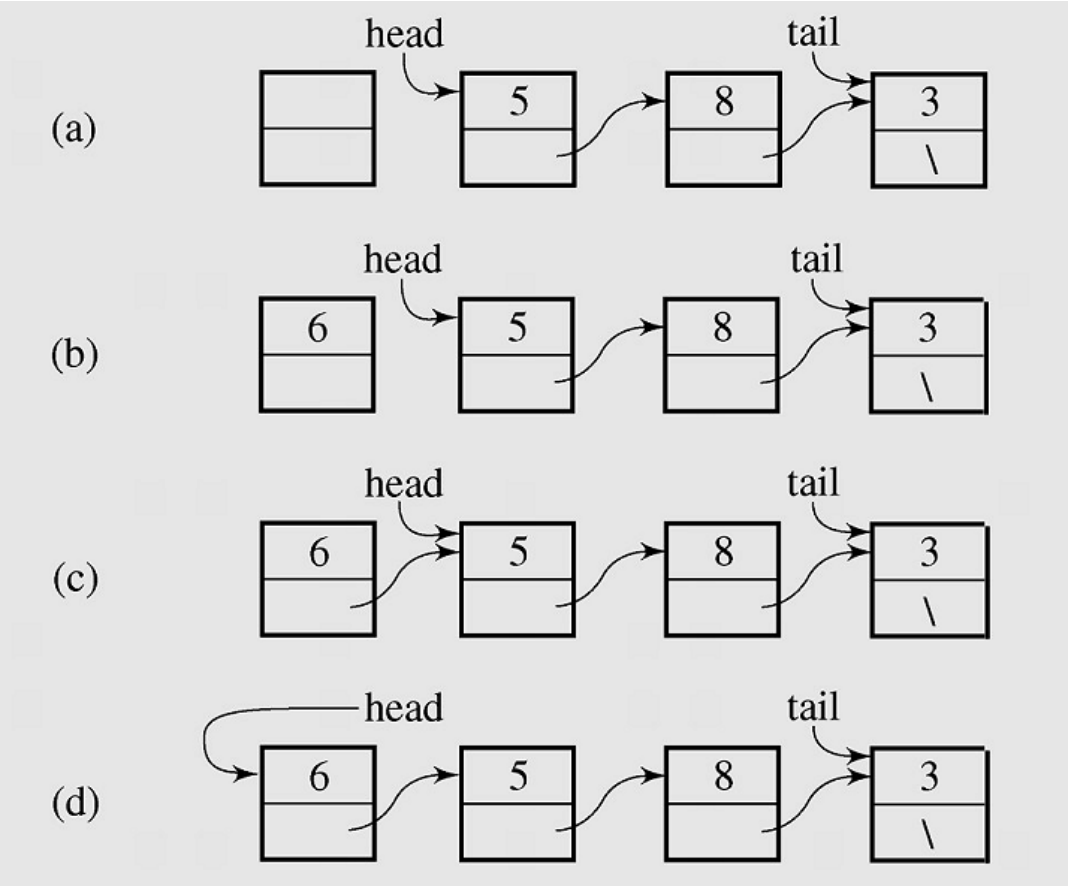

Figure 3-4 Inserting a new node at the beginning of a singly linked list

In the worst case, this operation is in O( ).

Insertion into Singly Linked Lists (cont.)

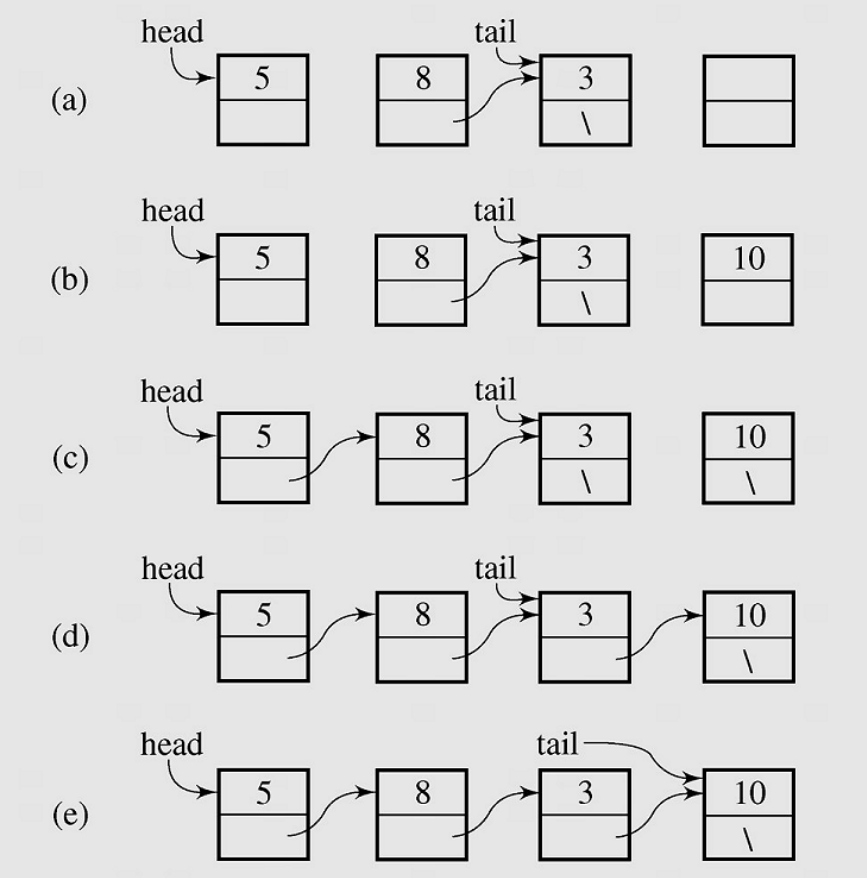

Figure 3-5 Inserting a new node at the end of a singly linked list

In the worst case, this operation is in O( ).

Insertion into Singly Linked Lists (cont.)

```
public void addToHead(T el) {
        head = new SLLNode<T>(el,head);
        if (tail == null)
            tail = head;
    }


public void addToTail(T el) {
        if (!isEmpty()) {
            tail.next = new SLLNode<T>(el);
            tail = tail.next;
        }
        else head = tail = new SLLNode<T>(el);
    }
```

Deletion from Singly Linked Lists

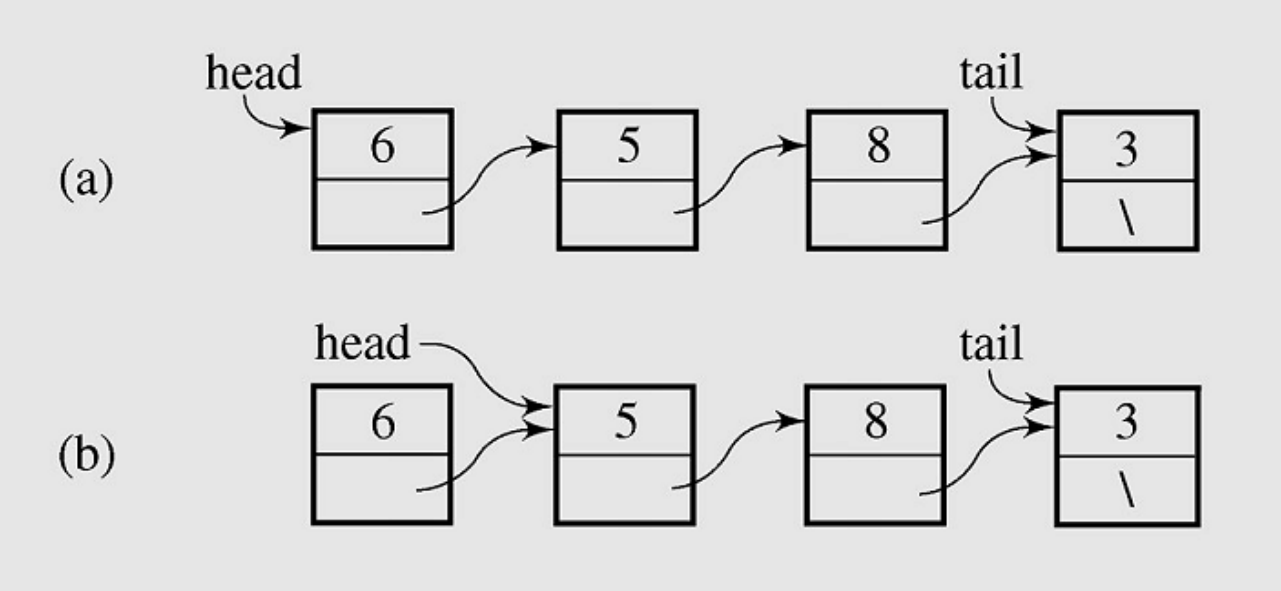

Figure 3-6 Deleting a node from the beginning of a singly linked list

In the worst case, this operation is in O( ).

Deletion from Singly Linked Lists (cont.)

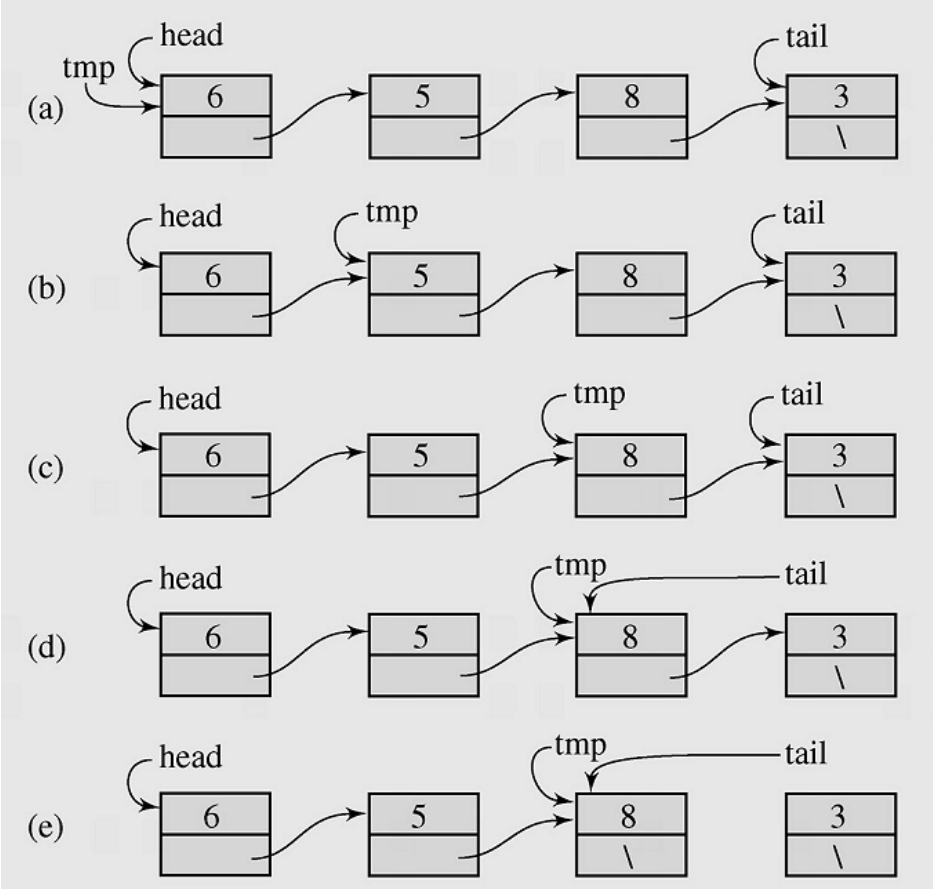

Figure 3-7 Deleting a node from the end of a singly linked list

In the worst case, this operation is in O( ).

Deletion from Singly Linked Lists (cont.)

```
public T deleteFromHead() { // delete head and return its
                                    // info;
        if (isEmpty())
             return null;
        T el = head.info;
        if (head == tail)    // if only one node on the list;
             head = tail = null;
        else head = head.next;
        return el;
    }
```

Deletion from Singly Linked Lists (cont.)

```
public T deleteFromTail() { // delete tail, return its info;
    if (isEmpty())
        return null;
    T el = tail.info;
    if (head == tail)       // if only one node in list;
        head = tail = null;
    else {                  // if more than one node in list,

        SLLNode<T> tmp;     // find predecessor of tail;
        for (tmp = head; tmp.next != tail; tmp = tmp.next);
        tail = tmp;         // predecessor of tail becomes tail;
        tail.next = null;
    }
    return el;
}
```

Deletion from Singly Linked Lists (cont.)

```
public void delete(T el) {  // delete the node with element el;
  if (!isEmpty())
      if (head == tail && el.equals(head.info)) // if only one
           head = tail = null;        // node on the list;
      else if (el.equals(head.info)) // if > one node on list;
           head = head.next;   // and el is in the head node;
      else {                   // if more than one node in list
           SLLNode<T> pred, tmp;// and el is in a nonhead node;
           for (pred = head, tmp = head.next;
                tmp != null && !tmp.info.equals(el);
                pred = pred.next, tmp = tmp.next);
           if (tmp != null) {   // if el was found;
               pred.next = tmp.next;
               if (tmp == tail) // if el is in the last node;
                  tail = pred;
           }
      }
}
```

Printing a Singly Linked List and Checking an Element in a List

```
  public void printAll() {
        for (SLLNode<T> tmp = head; tmp != null; tmp = tmp.next)
            System.out.print(tmp.info + " ");
    }
    public boolean isInList(T el) {
        SLLNode<T> tmp;
        for (tmp = head; tmp != null && !tmp.info.equals(el);
   tmp = tmp.next);
        return tmp != null;
    }
}
```

Doubly Linked Lists

- A **doubly linked list** is when each node in a linked list has two reference fields, one to the successor and one to the predecessor

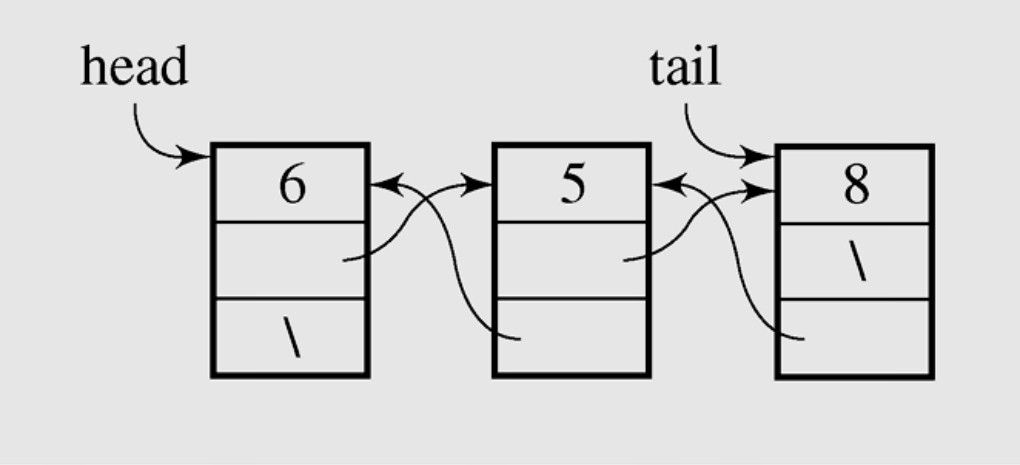

Figure 3-9 A doubly linked list

A Doubly Linked List Node

```
//********************  DLLNode.java  **********************
//        node of generic doubly linked list class

public class DLLNode<T> {
    public T info;
    public DLLNode<T> next, prev;
    public DLLNode() {
        next = null; prev = null;
    }
    public DLLNode(T el) {
        info = el; next = null; prev = null;
    }
    public DLLNode(T el, DLLNode<T> n, DLLNode<T> p) {
        info = el; next = n; prev = p;
    }
}
```

A Doubly Linked List

```
//*******************  DLL.java  **********************
//         generic doubly linked list class

public class DLL<T> {
    private DLLNode<T> head, tail;
    public DLL() {
        head = tail = null;
    }
    public boolean isEmpty() {
        return head == null;
    }
    public void setToNull() {
        head = tail = null;
    }
```

Insertion into Doubly Linked Lists

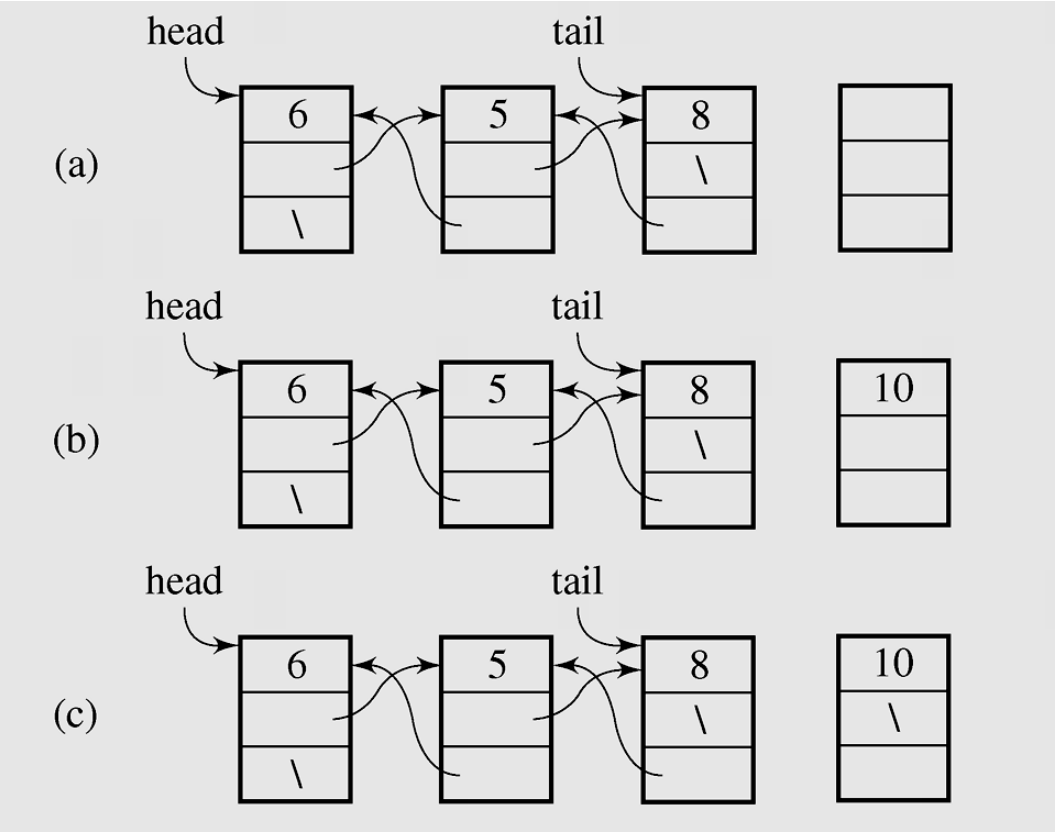

Figure 3-11 Adding a new node at the end of a doubly linked list

Insertion into Doubly Linked Lists (cont.)

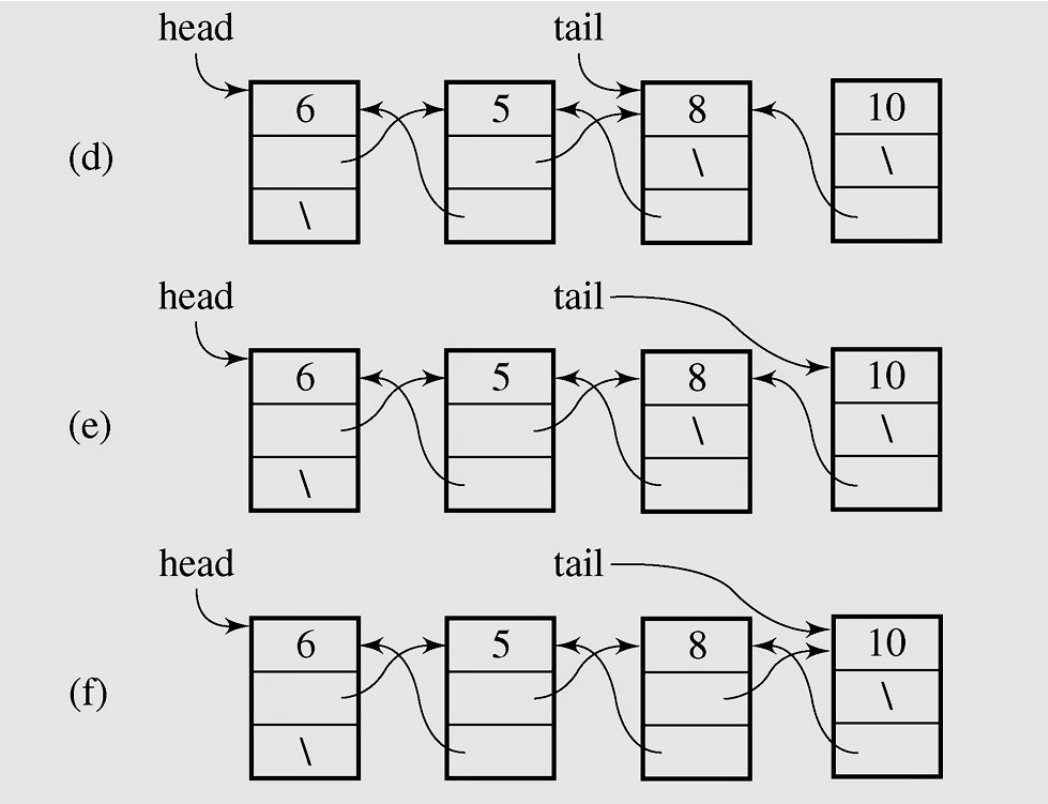

Figure 3-11 Adding a new node at the end of a doubly linked list (continued)

In the worst case, this operation is in O( )

Insertion into Doubly Linked Lists (cont.)

```
public void addToTail(T el) {
        if (tail != null) {
              tail = new DLLNode<T>(el,null,tail);
              tail.prev.next = tail;
        }
        else head = tail = new DLLNode<T>(el);
    }
```

Deletion from Doubly Linked Lists

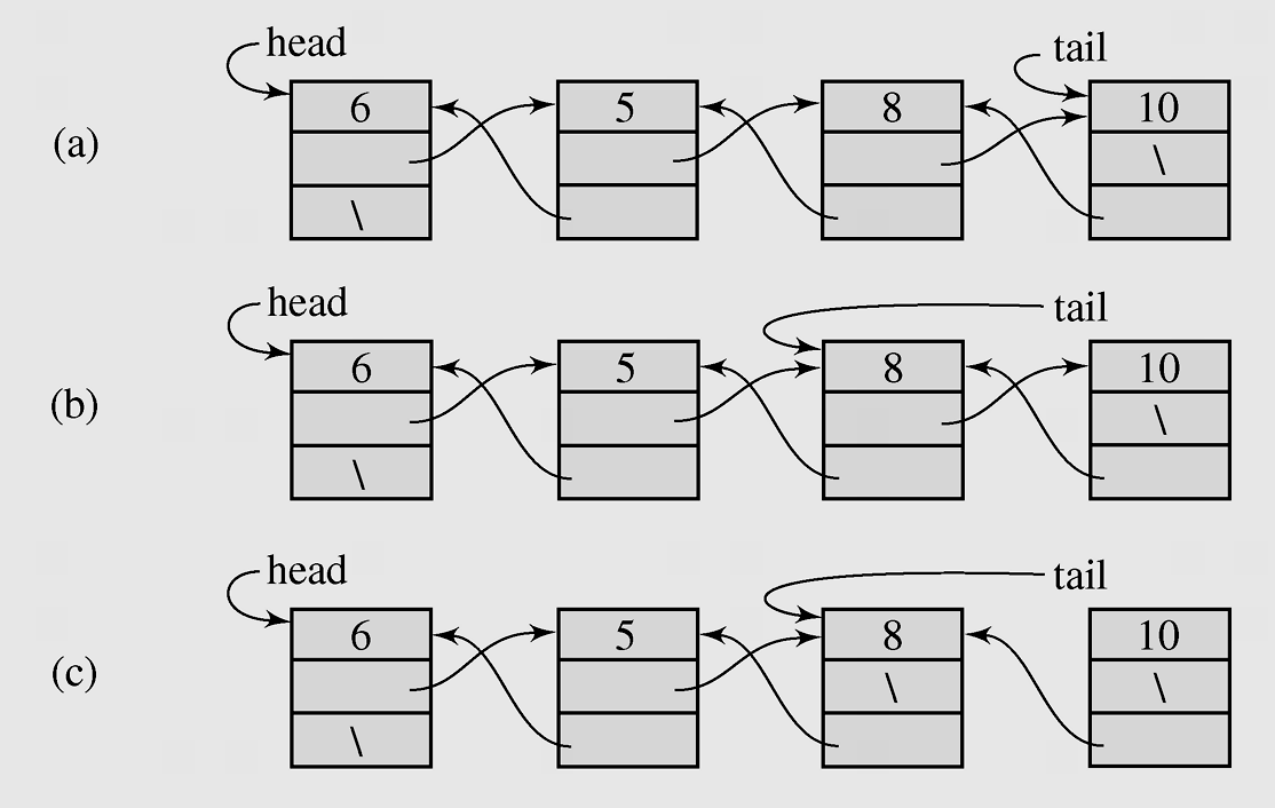

Figure 3-12 Deleting a node from the end of a doubly linked list

Deletion from Doubly Linked Lists (cont.)

```
public T deleteFromTail() {
        if (isEmpty())
             return null;
        T el = tail.info;
        if (head == tail)    // if only one node on the list;
             head = tail = null;
        else {               // if more than one node in list;
             tail = tail.prev;
             tail.next = null;
        }
        return el;
    }
```

Circular Lists

- A **circular list** is when nodes form a ring: The list is finite and each node has a successor

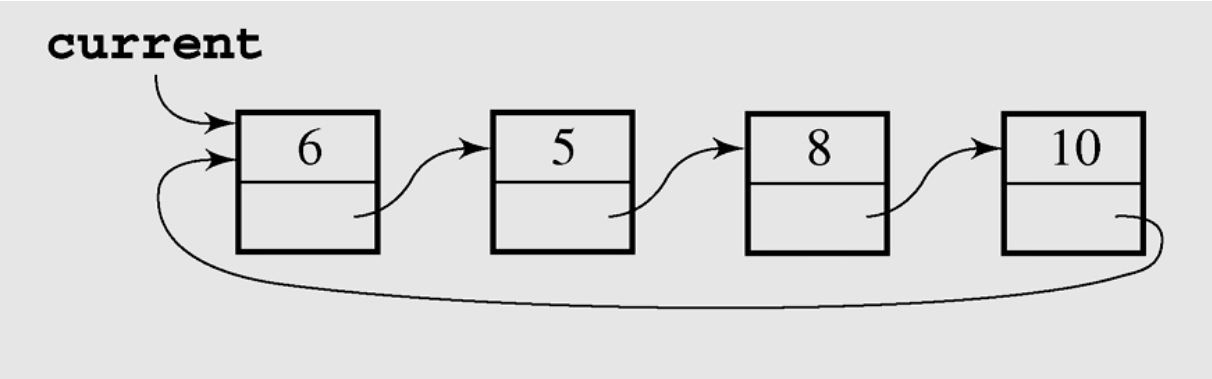

Figure 3-13 A circular singly linked list

Circular Lists (continued)

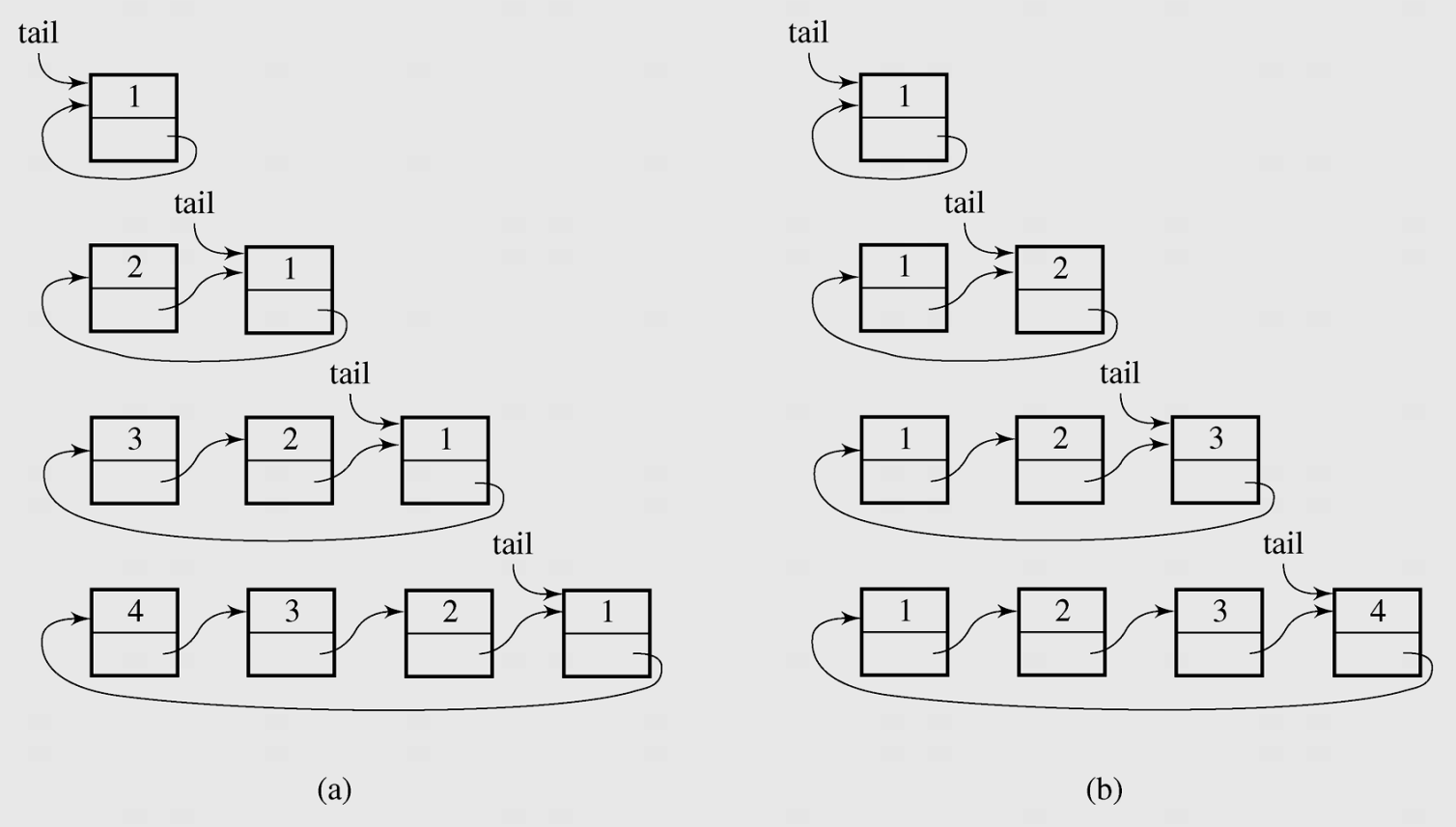

Figure 3-14 Inserting nodes at the front of a circular singly linked list (a) and at its end (b)

Circular Lists (continued)

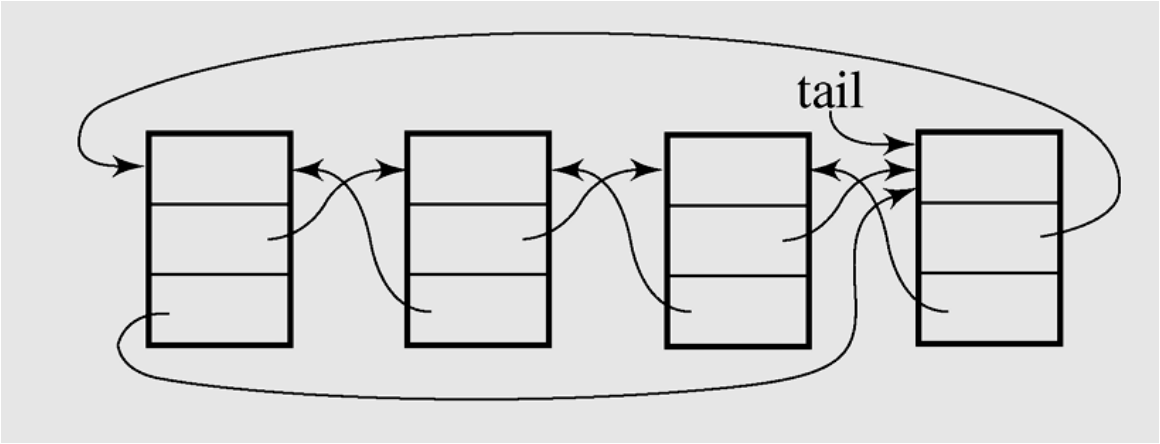

Figure 3-15 A circular doubly linked list

Summary

- A linked structure is a collection of nodes storing data and links to other nodes.
- A linked list is a data structure composed of nodes, each node holding some information and a reference to another node in the list.
- A node in a singly linked list has a link only to its successor in this sequence.
- A node in a doubly linked list has links to its successor and predecessor in this sequence.
- A circular list is when nodes form a ring: The list is finite and each node has a successor.
- The advantage of arrays over linked lists is that they allow random accessing.