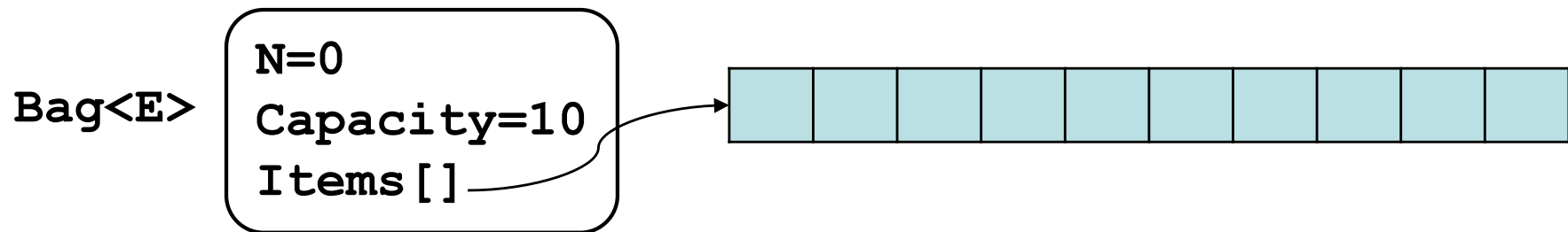


CMSC 132: Object-Oriented Programming II

Linked Lists

Array based Bag

- We implemented the Bag using arrays



- Resizing the bag:
 - Create an array with different size
 - Copy everything from old array to the new array
 - Items references the new array.
 - Old array is garbage.

Arrays

▶ Advantages:

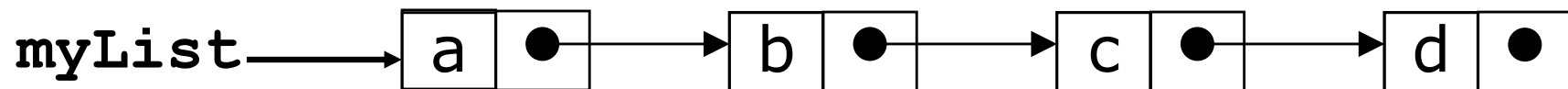
- Good cache performance
- Random access using index

▶ Disadvantages

- Static data structure. Size is fixed.
 - if we allocate more memory than requirement then the memory space will be wasted.
 - if we allocate less memory than requirement, then it will create problem
- Insert/Delete is costly

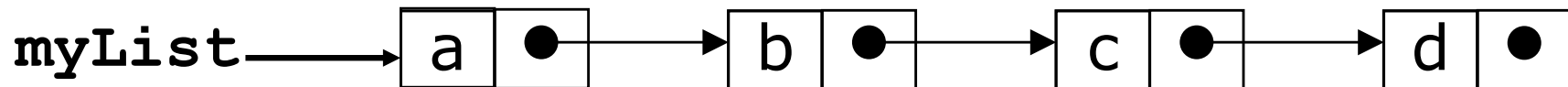
Linked list

- A linked list consists of:
 - A sequence of **nodes**



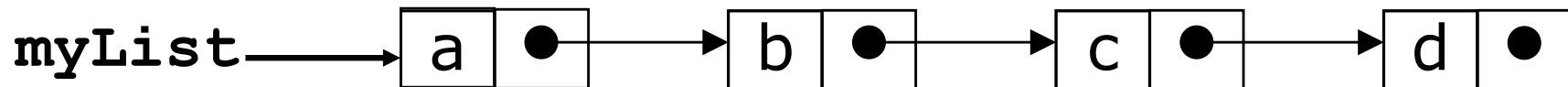
- Each node contains a value and a link (pointer or reference) to some other node
- The last node contains a null link

Linked List Node



```
class Node<E> {  
    E data;  
    Node<E> next;  
  
    Node(E item) {  
        data = item;  
    }  
  
    Node(E item, Node r) {  
        data = item;  
        next = r;  
    }  
}
```

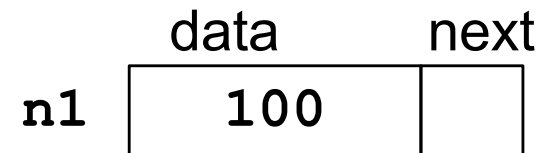
Linked List Node



```
class Node<E> {  
    E data;  
    Node<E> next;
```

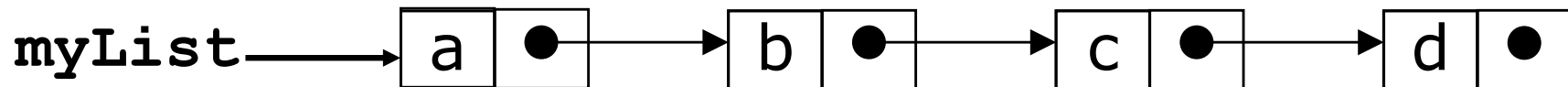
```
    Node(E item) {  
        data = item;  
    }  
}
```

```
    Node(E item, Node r) {  
        data = item;  
        next = r;  
    }  
}
```



```
Node<Integer> n1 = new Node(100);
```

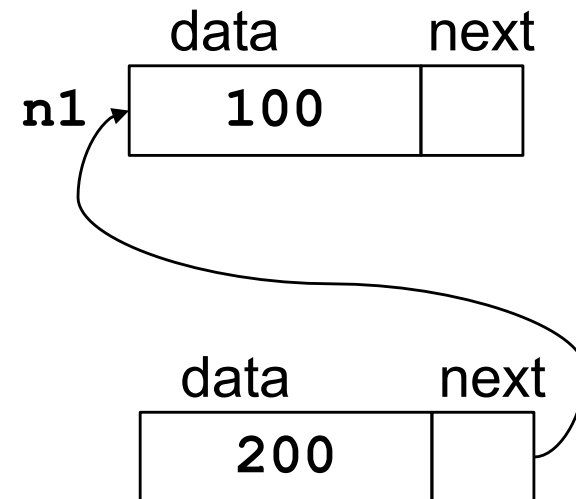
Linked List Node



```
class Node<E> {
    E data;
    Node<E> next;

    Node(E item) {
        data = item;
    }

    Node(E item, Node r) {
        data = item;
        next = r;
    }
}
```

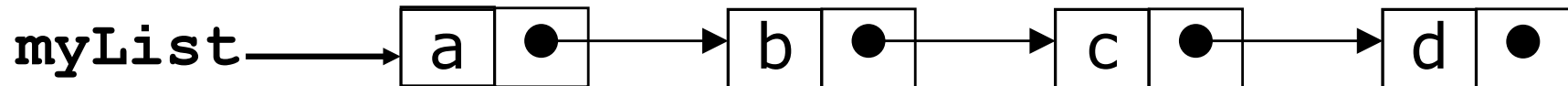


```
Node<Integer> n2 = new Node(200,n1);
```

More terminology

- ▶ A node's successor is the next node in the sequence
 - The last node has no successor
- ▶ A node's predecessor is the previous node in the sequence
 - The first node has no predecessor
- ▶ A list's length is the number of elements in it
 - A list may be empty (contain no elements)

Creating a Linked List



```
Node<String> mylist = new Node<>("a");  
Node<String> n2 = new Node<>("b");  
mylist.next = n2;
```

- **If successor exists:**

```
Node<String> n3 = new Node("c", new Node("d"));
```

```
N2.next = n3
```

Implement the Bag using Linked List

```
public class LinkedBag<E>{
    private int N; //number of items in the bag
    private Node<E> first; //beginning of bag

    private class Node<E> {
        private E data;
        private Node<E> next;
        Node(E item) {
            data = item;
        }
    }
}
```

LinkedBag<E>

N=0
First = null

Insert

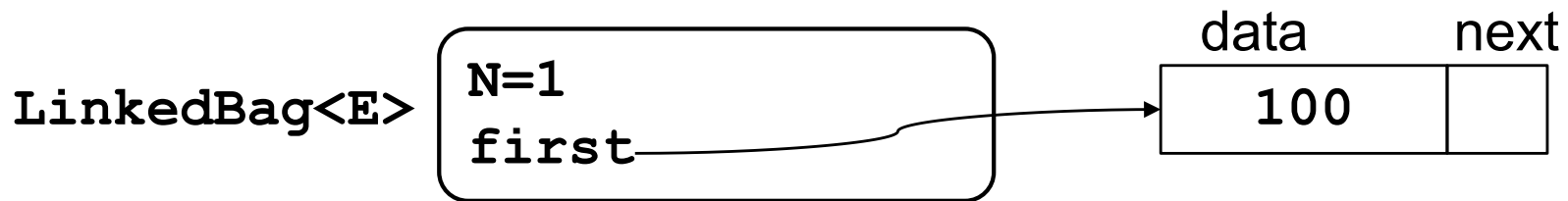
Insert the item as the first node:

```
public void insert(E item) {  
    first = new Node<>(item, first);  
    N++;  
}
```

- No capacity limit
- No need to resize

Insert

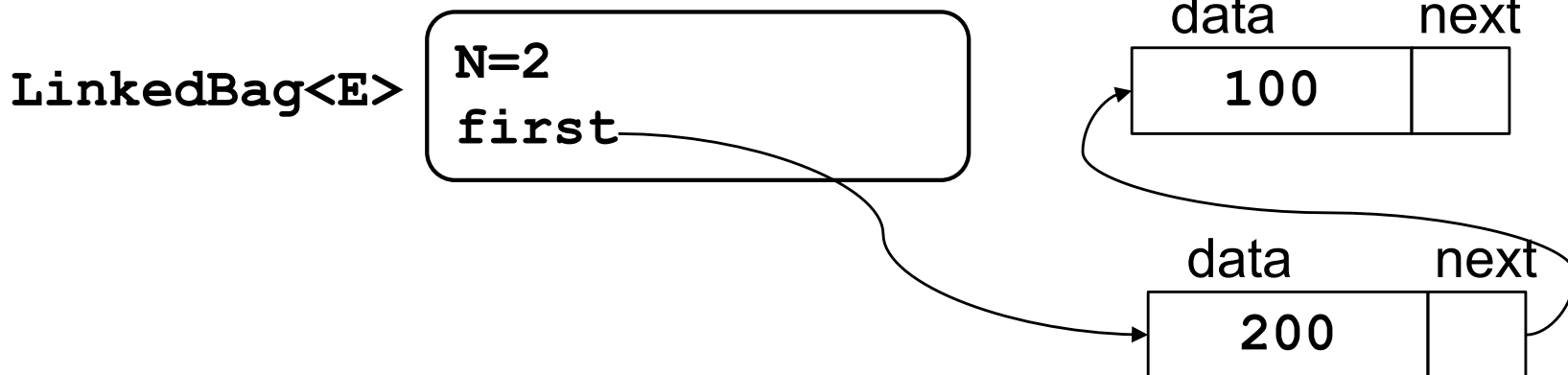
```
public void insert(E item) {  
    first = new Node<E>(item, first);  
    N++;  
}
```



```
LinkedBag<Integer> bag = new LinkedBag();  
bag.insert(100);
```

Insert

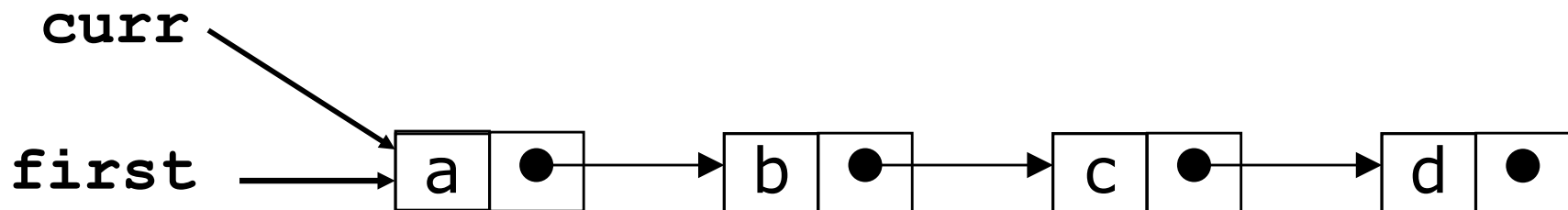
```
public void insert(E item) {  
    first = new Node<E>(item, first);  
    N++;  
}
```



```
LinkedBag<Integer> bag = new LinkedBag();  
bag.insert(100);  
bag.insert(200);
```

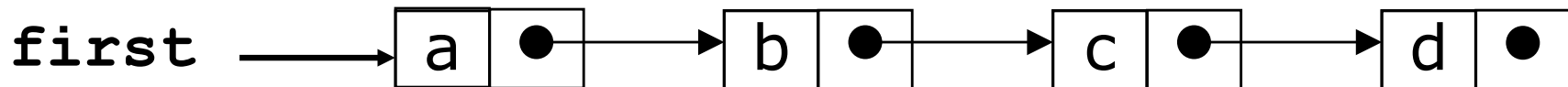
Traversing a Linked List

```
public void print() {  
    Node<E> curr = first;  
    While(curr != null){  
        System.out.print(curr.data + ",");  
        curr = curr.succ;  
    }  
}
```



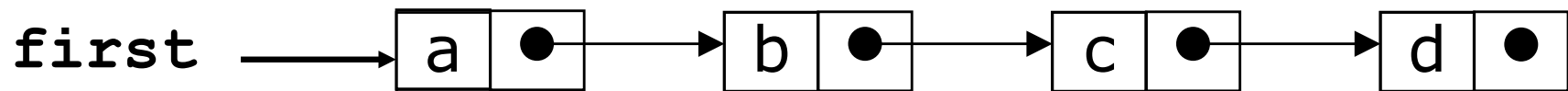
Traversing a Linked List Recursively

```
private void print(Node r) {  
    if(r == null) return;  
    System.out.print(r.data + ",");  
    print(r.next); //recursive call  
}
```

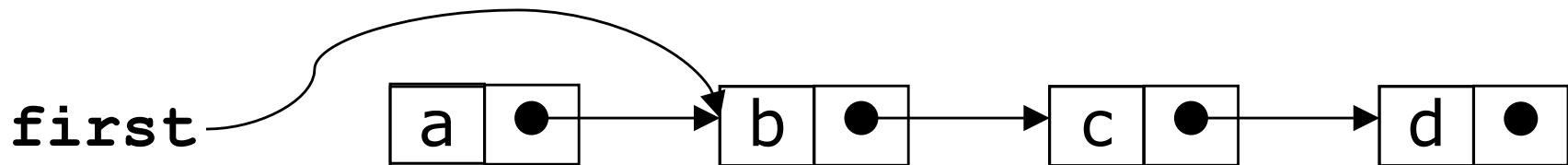


Deleting a node

Delete the first node

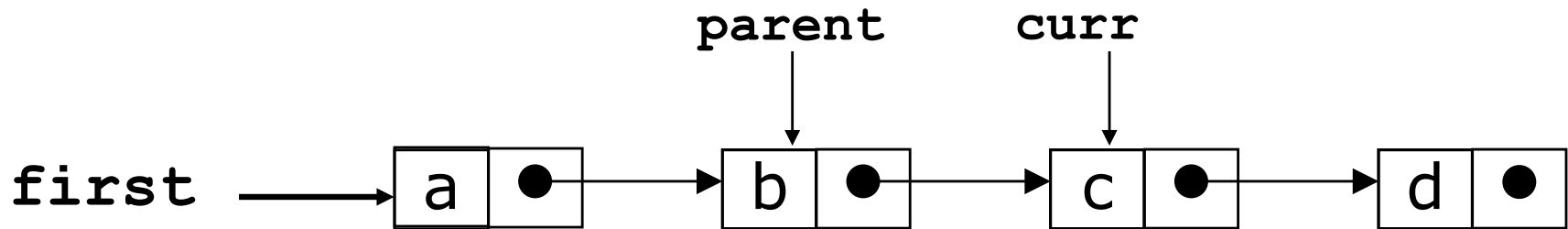


`first = first.next`

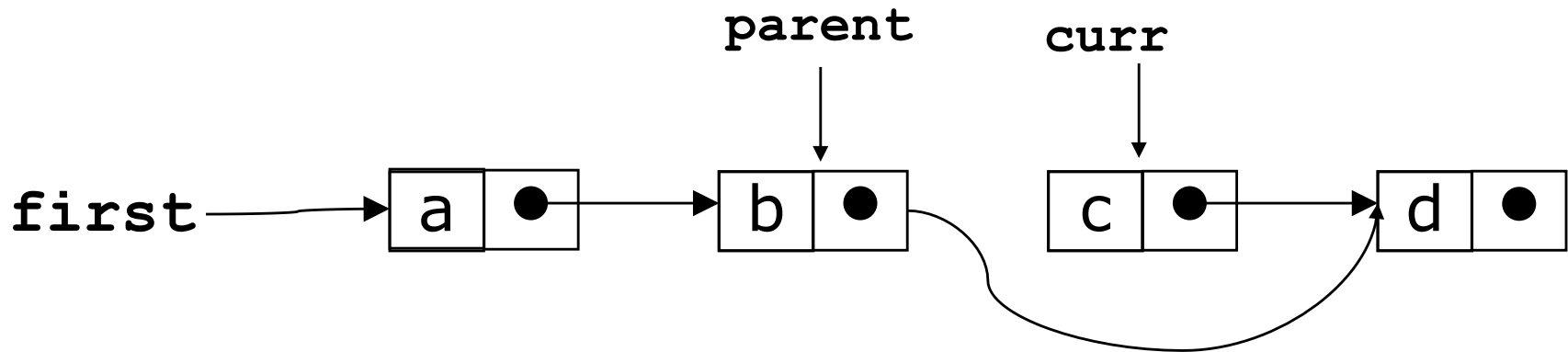


Deleting a node

Delete the node "c"



`parent.next = curr.next;`



Deleting a node

```
public boolean remove(E item) {
    if(isEmpty()) throw new NoSuchElementException();
    if(item == null) return false;
    if(first.data.equals(item)) {
        first = first.next;    return true;
    }
    Node<E> parent = first;
    Node<E> current = first.next;
    while(current != null) {
        if(current.data.equals(item)) {
            parent.next = current.next;
            return true;
        }
        parent = current;
        current = current.next;
    }
    return false;
}
```

Deleting a Node Recursively

```
public void remove_rec(E item) {
    first = remove_rec(first, item);
}

private Node<E> remove_rec(Node<E> r, E item) {
    if( r == null) return null;
    if(r.data.equals(item)) {
        return r.next;
    }
    r.next = remove_rec(r.next, item);
    return r;
}
```

Quiz 1:

What is a Node used for in a linked list?

- A. To store the information and the link to the next item
- B. To check for the end of the list
- C. Not used in a linked list
- D. To check for the beginning of the list

Quiz 1:

What is a Node used for in a linked list?

- A. To store the information and the link to the next item**
- B. To check for the end of the list
- C. Not used in a linked list
- D. To check for the beginning of the list

Quiz 2:

What does the following function do for a given Linked List with first node as head?

```
void foo(Node head) {  
    if(head == null) return;  
    foo(head.next);  
    print(head.data);  
}
```

- A. Prints all nodes of linked lists
- B. Prints all nodes of linked list in reverse order
- C. Prints alternate nodes of Linked List
- D. Prints alternate nodes in reverse order

Quiz 2:

What does the following function do for a given Linked List with first node as head?

```
void foo(Node head) {  
    if(head == null) return;  
    foo(head.next);  
    print(head.data);  
}
```

- A. Prints all nodes of linked lists
- B. Prints all nodes of linked list in reverse order**
- C. Prints alternate nodes of Linked List
- D. Prints alternate nodes in reverse order

Quiz 3:

Which of the following points is/are true about Linked List data structure when it is compared with array

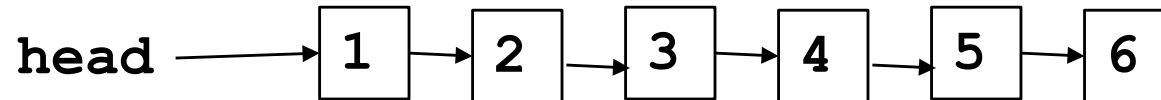
- A. Arrays have better cache locality.
- B. Easy to insert and delete elements in Linked List
- C. Random access is not allowed in Linked Lists
- D. All of the above

Quiz 3:

Which of the following points is/are true about Linked List data structure when it is compared with array

- A. Arrays have better cache locality.
- B. Easy to insert and delete elements in Linked List
- C. Random access is not allowed in Linked Lists
- D. All of the above**

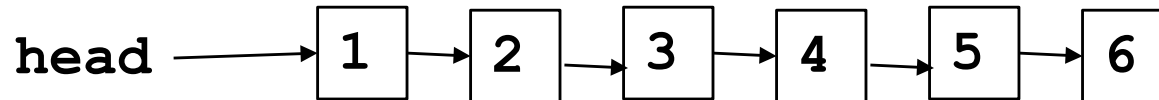
Quiz 4: What is the output of foo?



```
void foo(Node head) {  
    if(head == null) return;  
    print(head.data);  
    if(head.next != null)  
        foo(head.next.next);  
    print(head.data);  
}
```

- A. 1 4 6 6 4 1
- B. 1 3 5 1 3 5
- C. 1 2 3 5
- D. 1 3 5 5 3 1

Quiz 4: What is the output of foo?



```
void foo(Node head) {  
    if(head == null) return;  
    print(head.data);  
    if(head.next != null)  
        foo(head.next.next);  
    print(head.data);  
}
```

- A. 1 4 6 6 4 1
- B. 1 3 5 1 3 5
- C. 1 2 3 5
- D. 1 3 5 5 3 1