CIS112

Linked Lists

BBBF

Yeditepe University

v2024-03-29

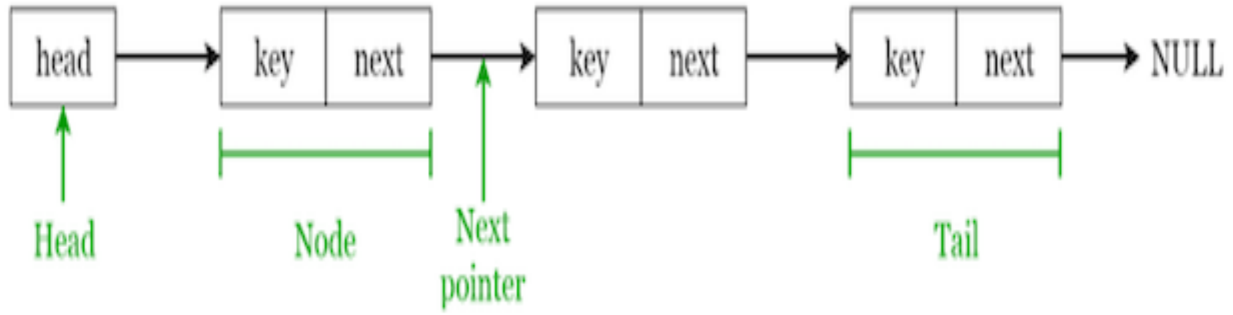
Content

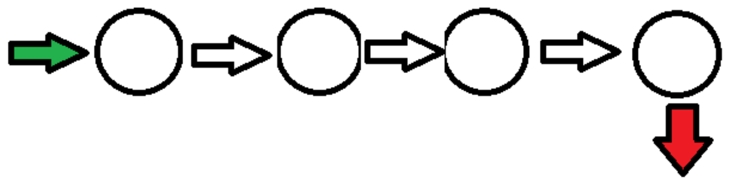
Data Structures with Nodes and Edges

Data Structures with Nodes and Edges

Linked Lists

A linked list is a linear data structure where items are arranged in linear order and linked (connected) to each other. That's why you cannot access random data; you need to access data only in order (sequentially).

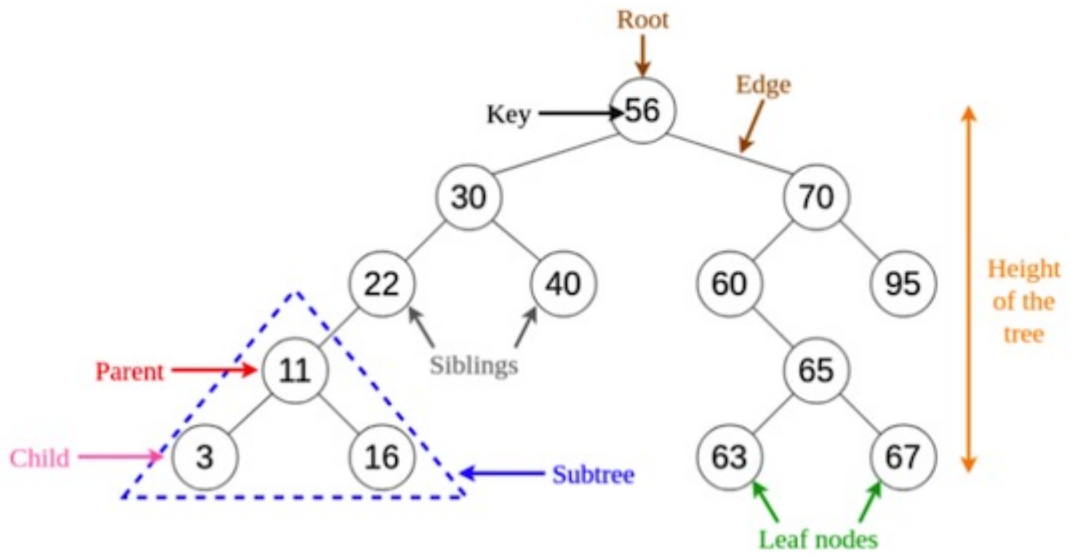




Data Structures with Nodes and Edges

Trees

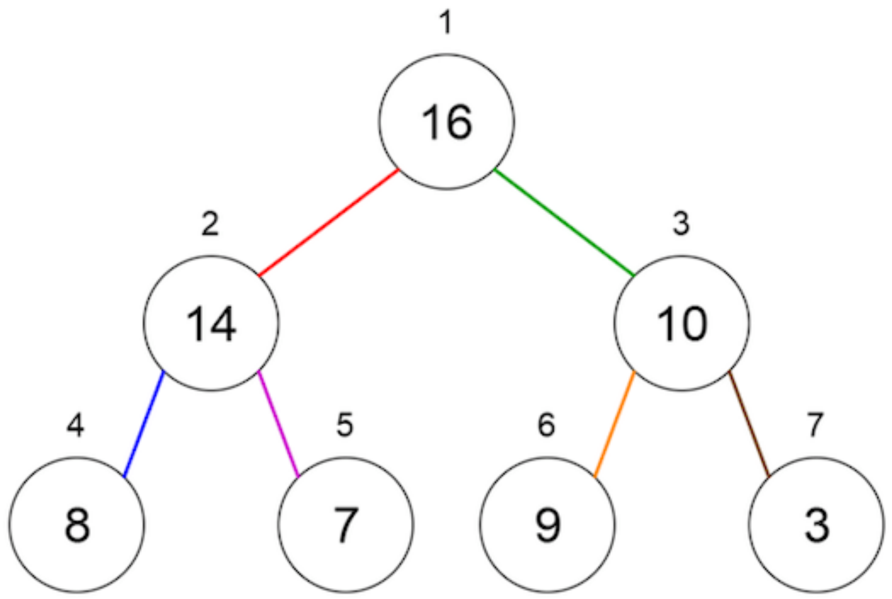
Another basic data structure is a Tree. In the tree structure, data is linked together as in the linked list but organized hierarchically, just like the visual representation of a person's family tree.



Data Structures with Nodes and Edges

Heaps

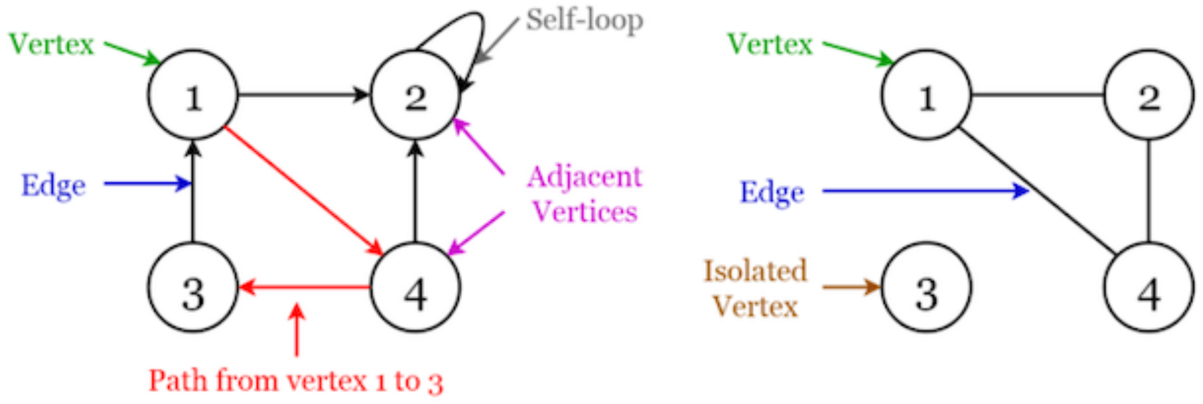
A heap is a specific type of binary tree where the parent nodes are compared to their child nodes, and values are arranged in the nodes accordingly.



Data Structures with Nodes and Edges

Graphs

A graph is a non-linear and abstract data structure that consists of a fixed (finite) set of nodes or vertices and is connected by a set of edges. Edges are the arcs or lines that simply connect nodes in the graph.



Directed Graph

G = {1, 2, 3, 4}
E = {(1, 2), (1, 4), (2, 2), (3, 1), (4, 3), (4, 2)}

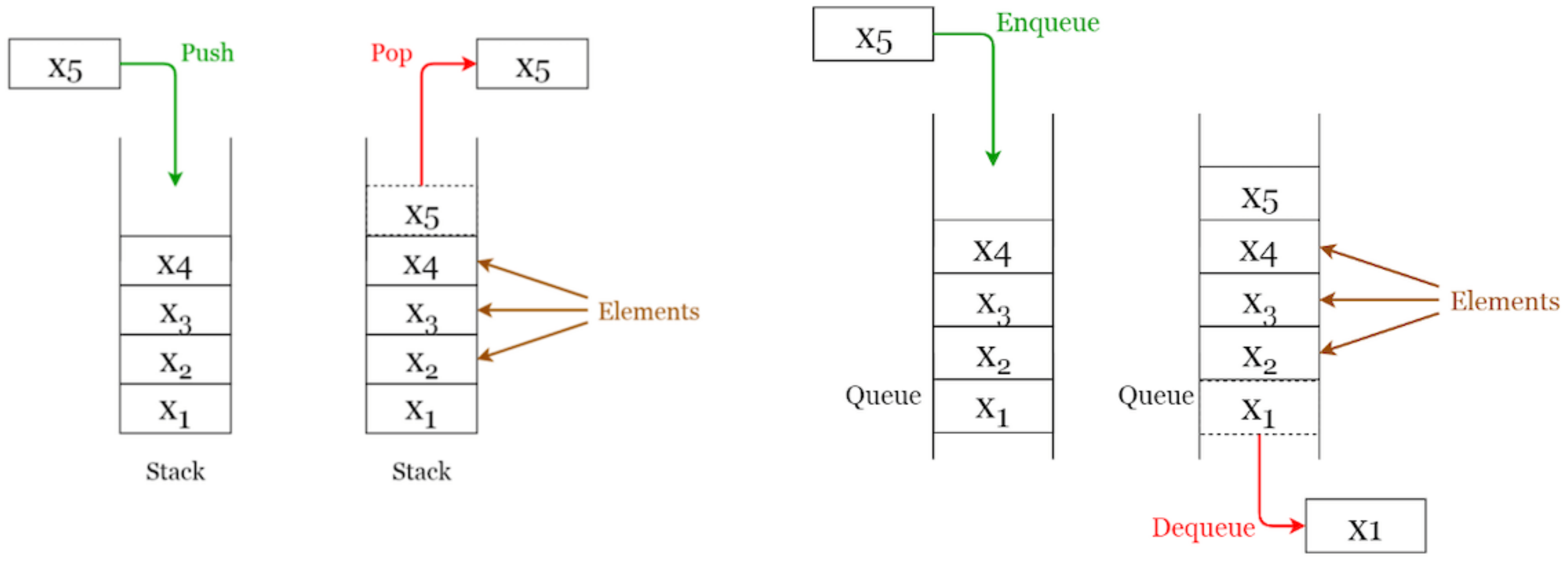
Undirected Graph

G = {1, 2, 3, 4}
E = {(1, 2), (1, 4), (2, 4)}

Data Structures with Nodes and Edges

What about Stacks and Queues?

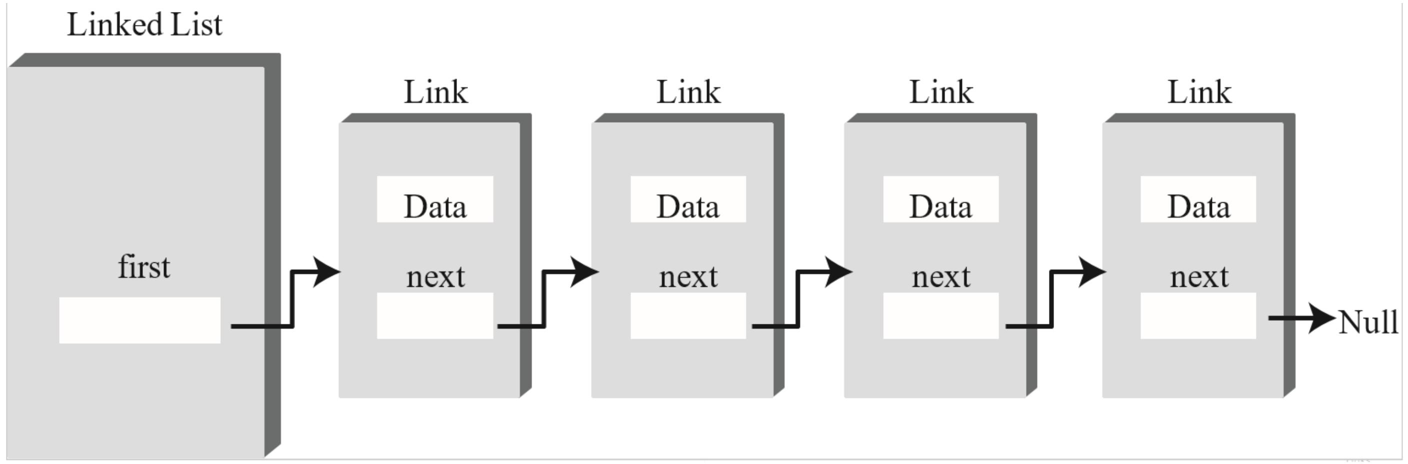
Can they be represented with nodes and edges?



Implementation

Implementation

- **Links: The LinkList Class**



```
class Link
{
public int iData; // data
public double dData; // data
public Link next; // reference to next link
... methods...
}
```

```
class LinkList
{
private Link first;
public void LinkList() // constructor
{
first = null; // no items on list yet
}
....methods ...
}
```

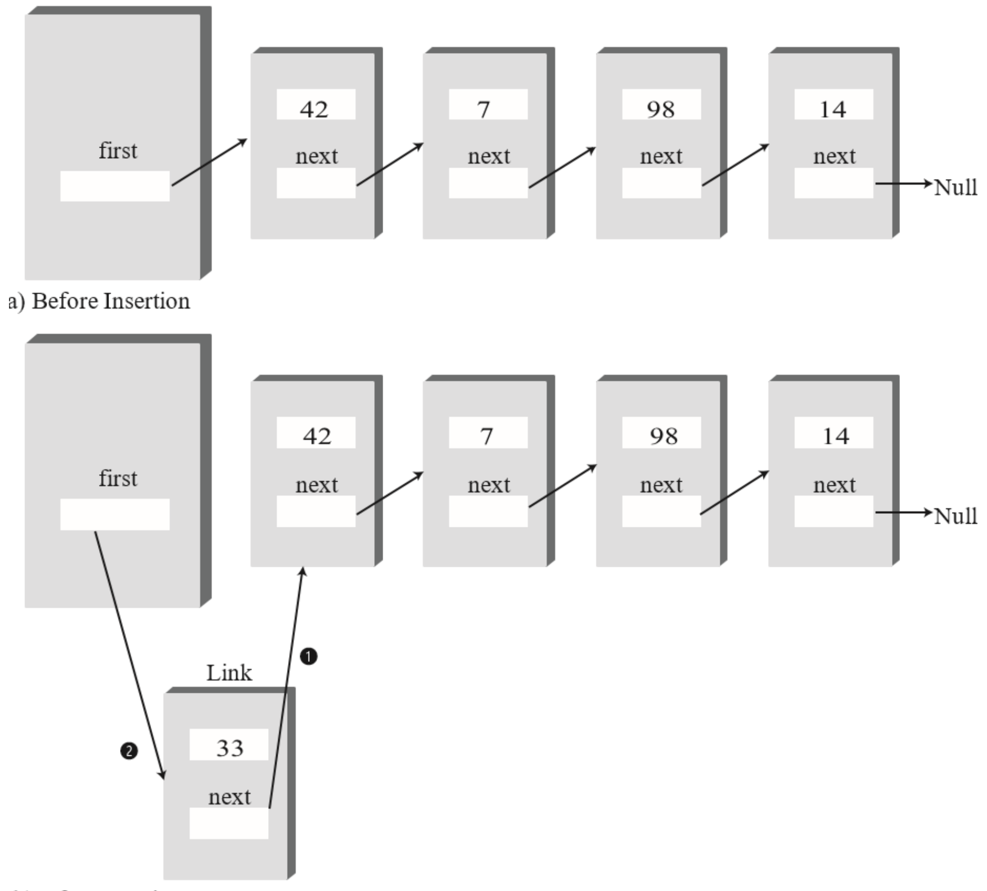
Implementation

- **Links: The Link Class**

```
class Link
{
    public int iData; // data item (key)
    public double dData; // data item
    public Link next; // next link in list

    public Link(int id, double dd) // constructor
    {
        iData = id; // initialize data
        dData = dd; // (‘next’ is automatically set to null)
    }

    public void displayLink() // display yourself
    {
        System.out.print(“{“ + iData + “, “ + dData + “} “);
    }
} // end class Link
```

Implementation

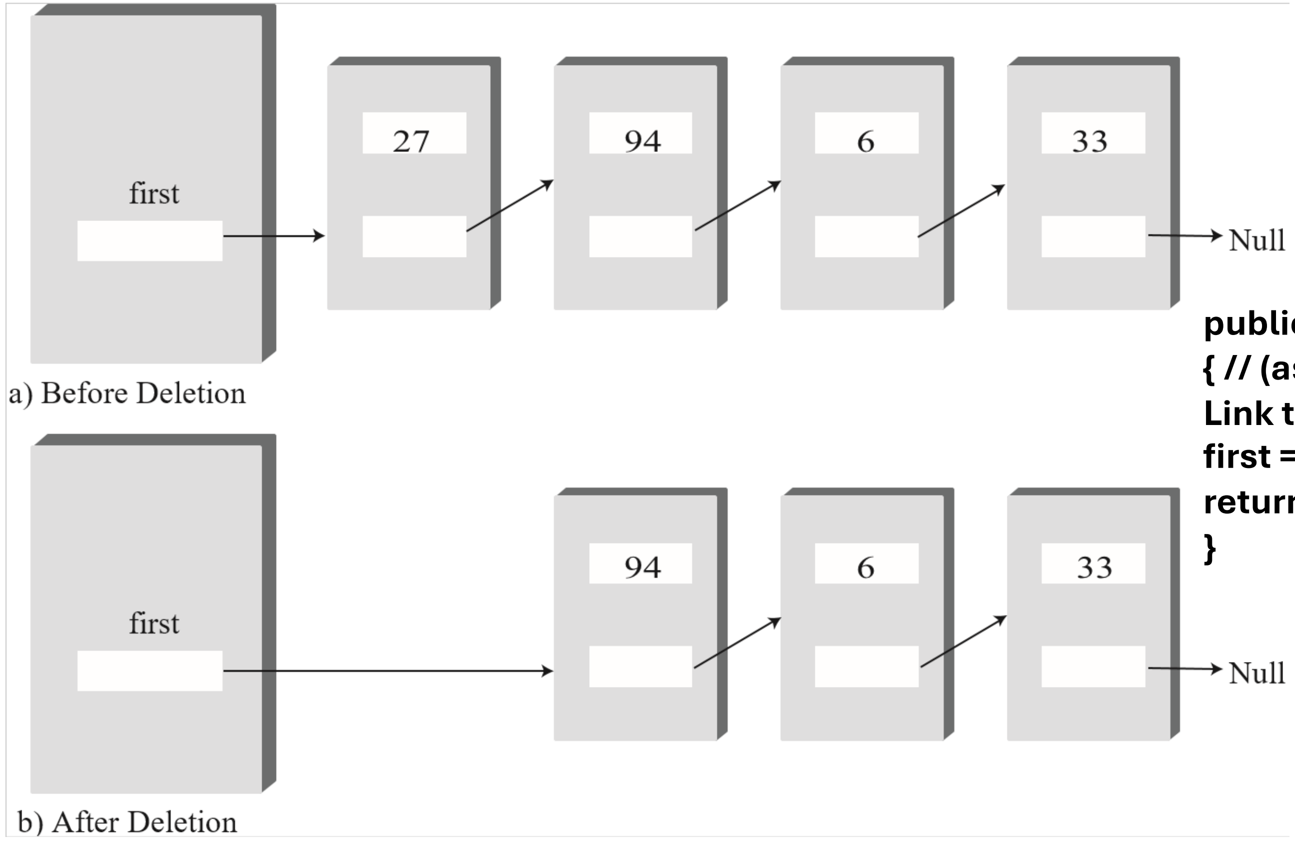
- **Links: The insertFirst() Method**



```
// insert at start of list
public void insertFirst(int id, double dd)
{ // make new link
Link newLink = new Link(id, dd);
newLink.next = first; // newLink --> old first
first = newLink; // first --> newLink
}
```

Implementation

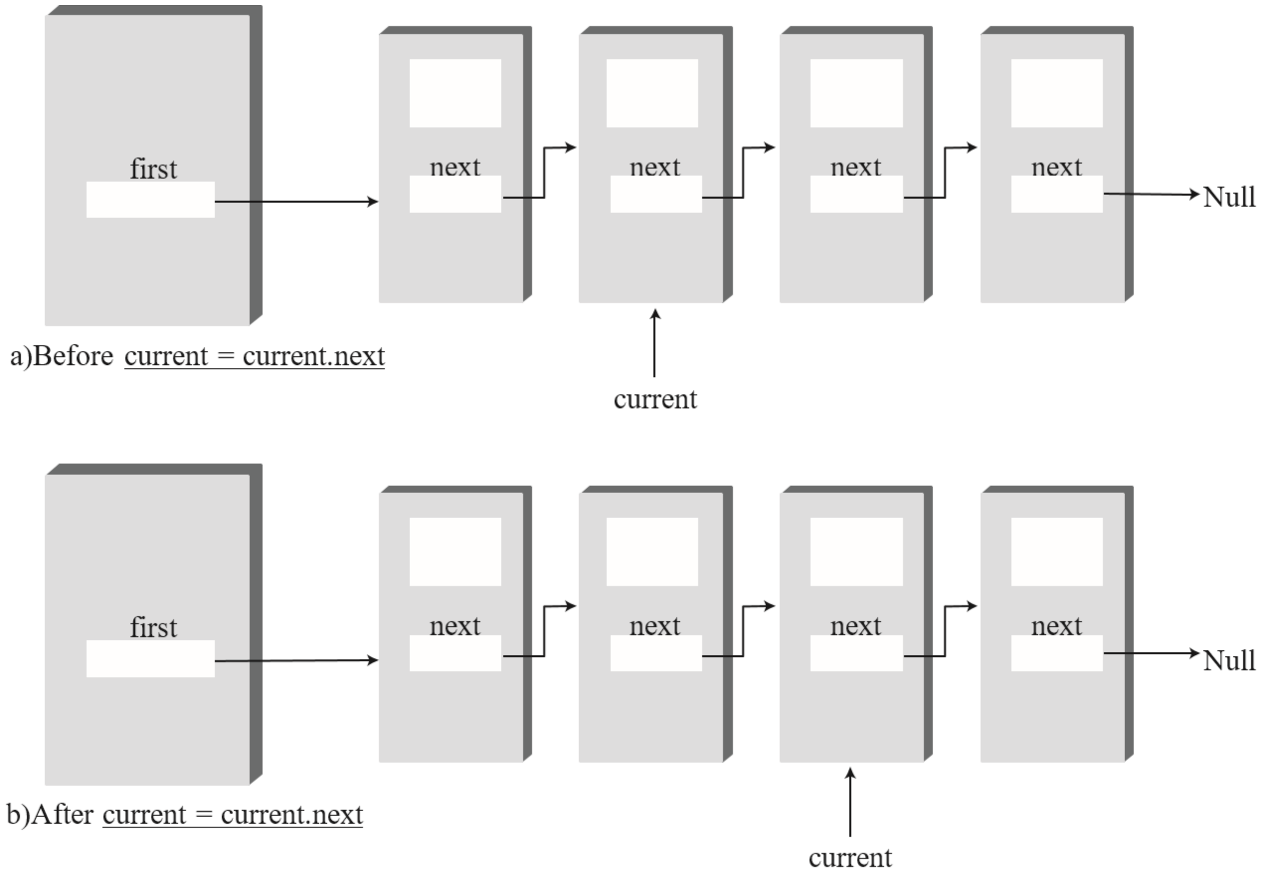
- **Links: The deleteFirst() Method**



```
public Link deleteFirst() // delete first item
{ // (assumes list not empty)
Link temp = first; // save reference to link
first = first.next; // delete it: first-->old next
return temp; // return deleted link
}
```

Implementation

- **Links: The displayList() Method**



a)Before current = current.next

b)After current = current.next

```
public void displayList()
{
System.out.print("List (first-->last): ");
Link current = first; // start at beginning of list
while(current != null) // until end of list,
{
current.displayLink(); // print data
current = current.next; // move to next link
}
System.out.println("");
}
```

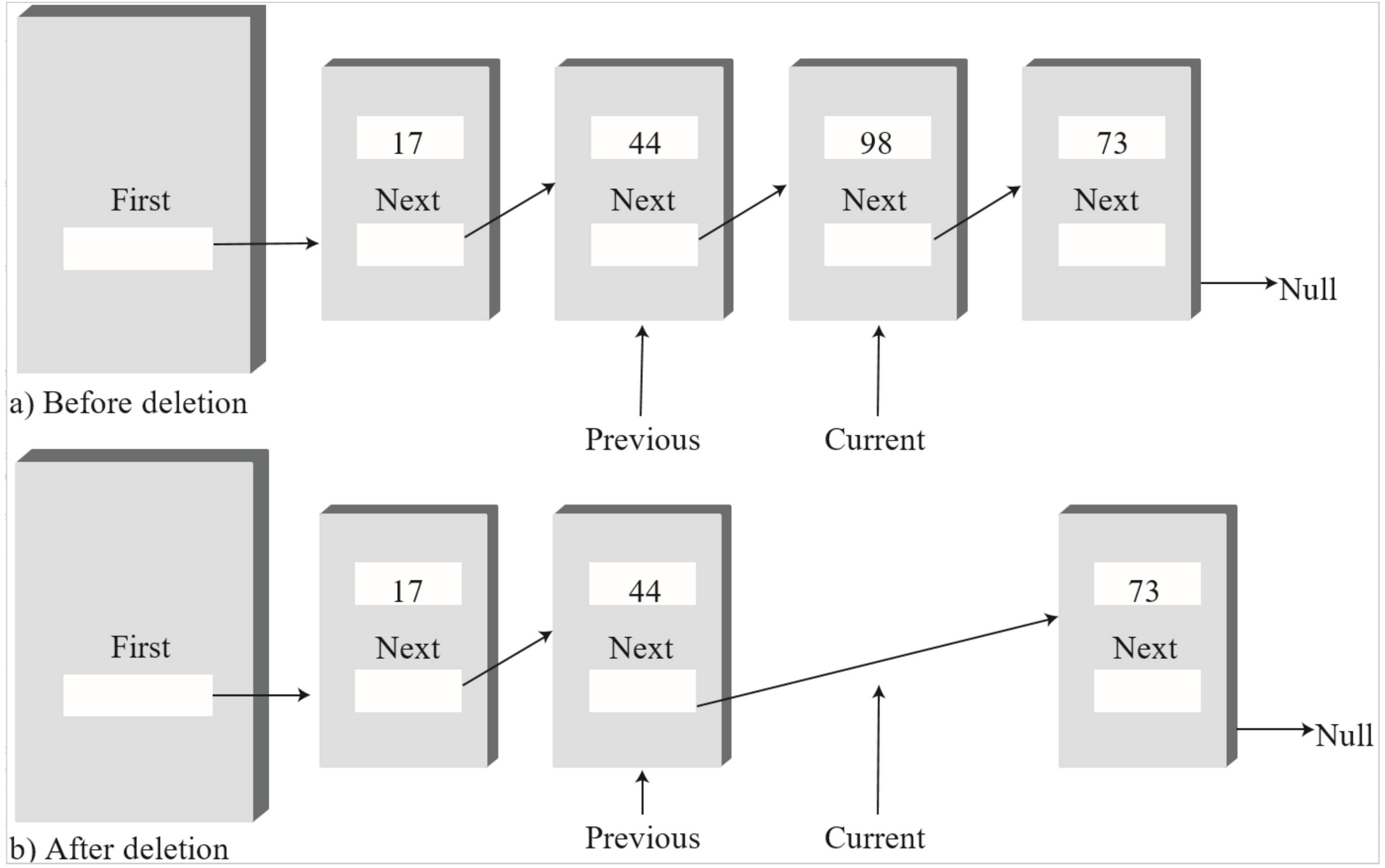
Implementation

- **Links: The find() Method**

```
public Link find(int key) // find link with given key
{ // (assumes non-empty list)
Link current = first; // start at ‘first’
while(current.iData != key) // while no match,
{
if(current.next == null) // if end of list,
return null; // didn’t find it
else // not end of list,
current = current.next; // go to next link
}
return current; // found it
}
```

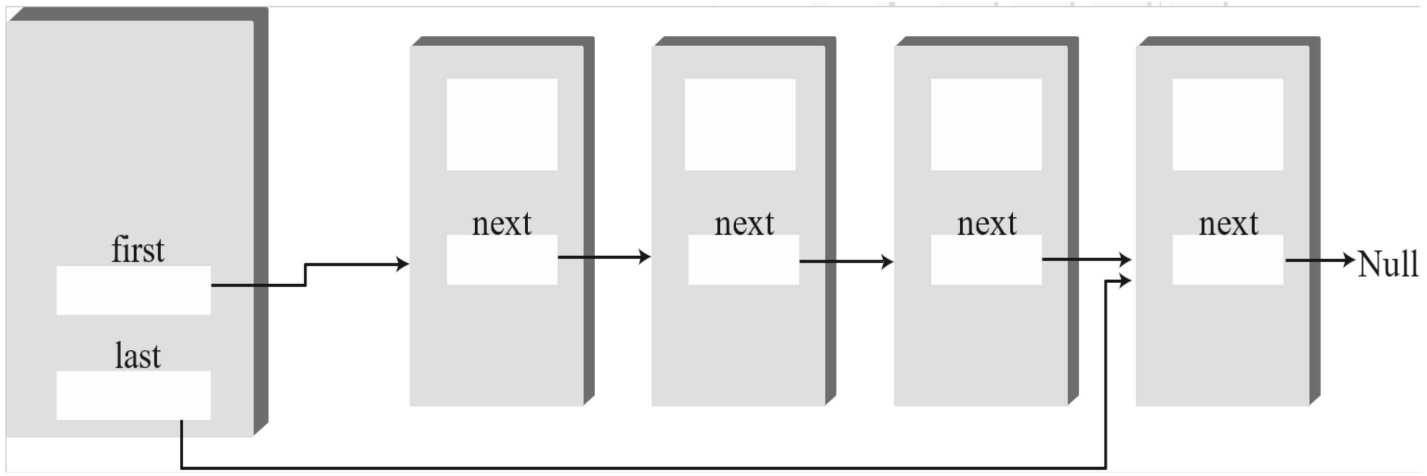
Implementation

- **Links: The delete() Method**



Implementation

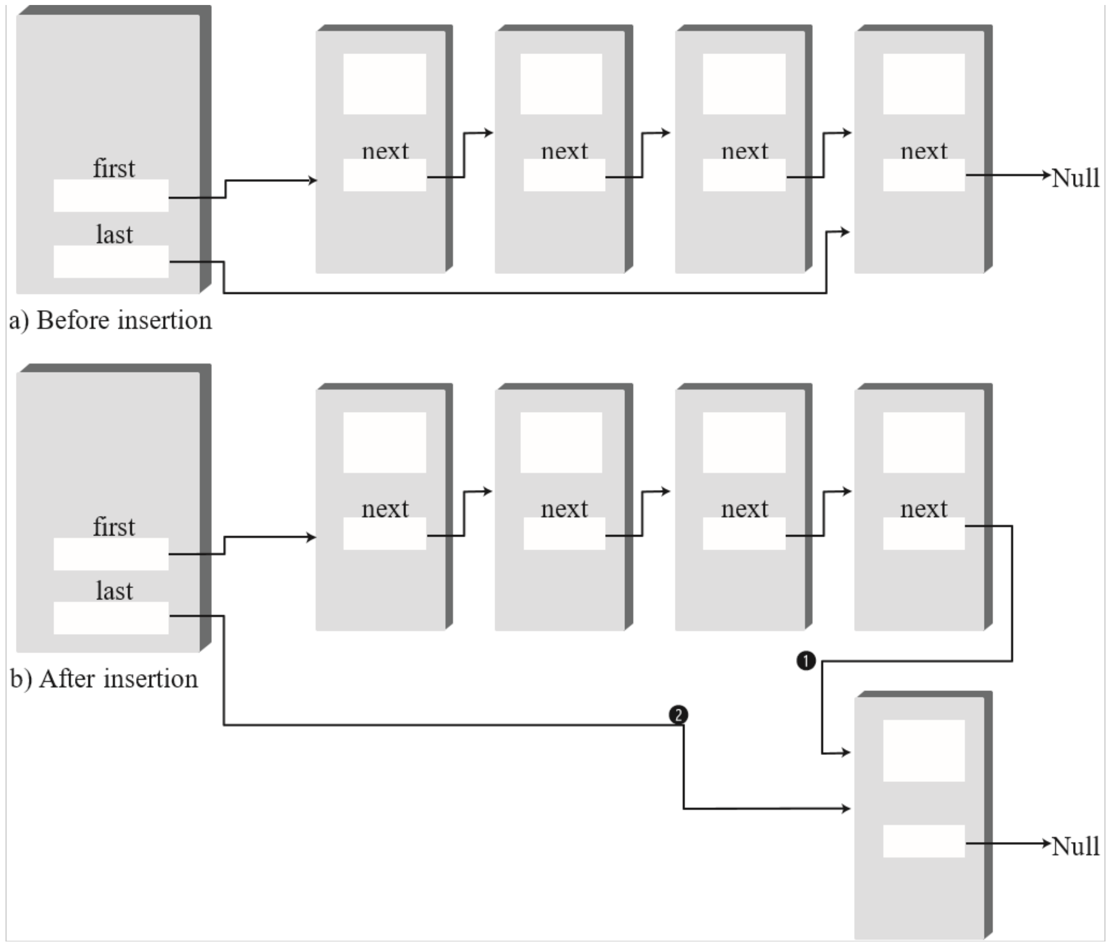
- **Double-Ended Lists:**



```
class FirstLastList
{
private Link first;
// ref to first link
private Link last;
// ref to last link

public FirstLastList() // constructor
{
first = null; // no links on list yet
last = null;
}
 ... methods ...
}
```

Implementation

- **Double-Ended Lists: Insertion at the end of a list**



```
public void insertLast(int id, double dd)
{
Link newLink = new Link(id, dd);
// make new link
if( isEmpty() ) // if empty list,
first = newLink; // first --> newLink
else
last.next = newLink; // old last --> newLink
last = newLink; // newLink <-- last
}
```

Stack and Queue

Stack and Queue

- **Stack** **Queue**

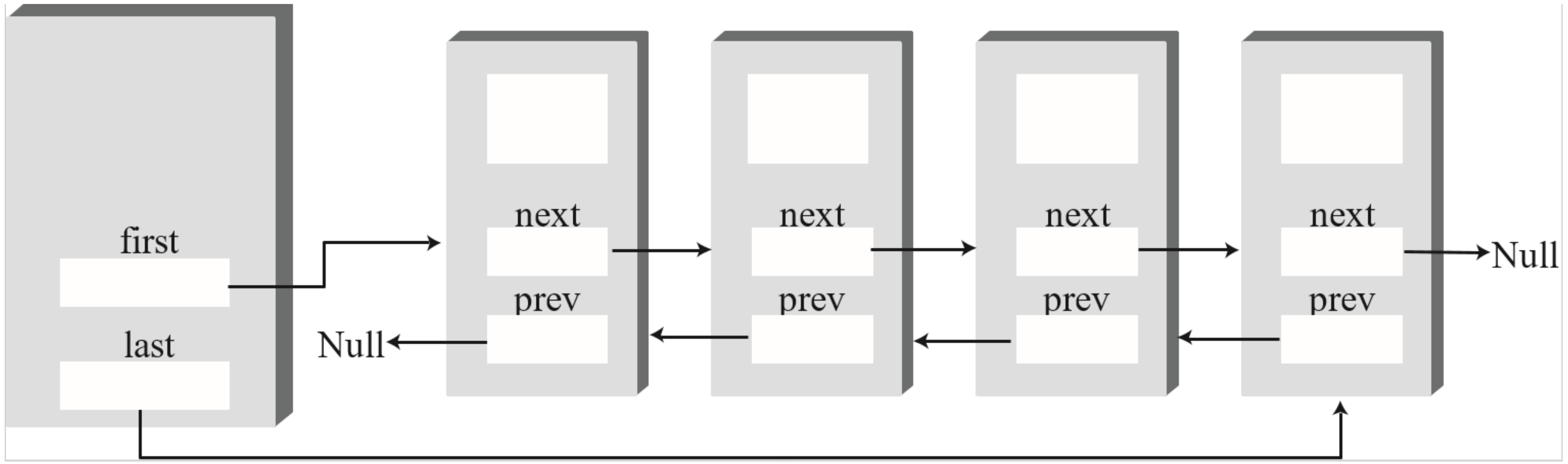
```
class LinkStack
{
private LinkList theList;

public LinkStack() // constructor
{
theList = new LinkList();
}

public void push(int id, double dd)
{
theList.insertFirst(id, dd);
}

public long pop()
{
return theList.deleteFirst();
}
 ... methods ...
}
```

```
class LinkQueue
{
private FirstLastList theList;

public LinkQueue() // constructor
{
theList = new FirstLastList();
}

public void insert(int id, double dd)
// insert, rear of queue
{
theList.insertLast(id, dd);
}

public long remove()
// remove, front of queue
{
return theList.deleteFirst();
 }
 ... methods ...
}
```

Doubly Linked Lists

Doubly Linked Lists

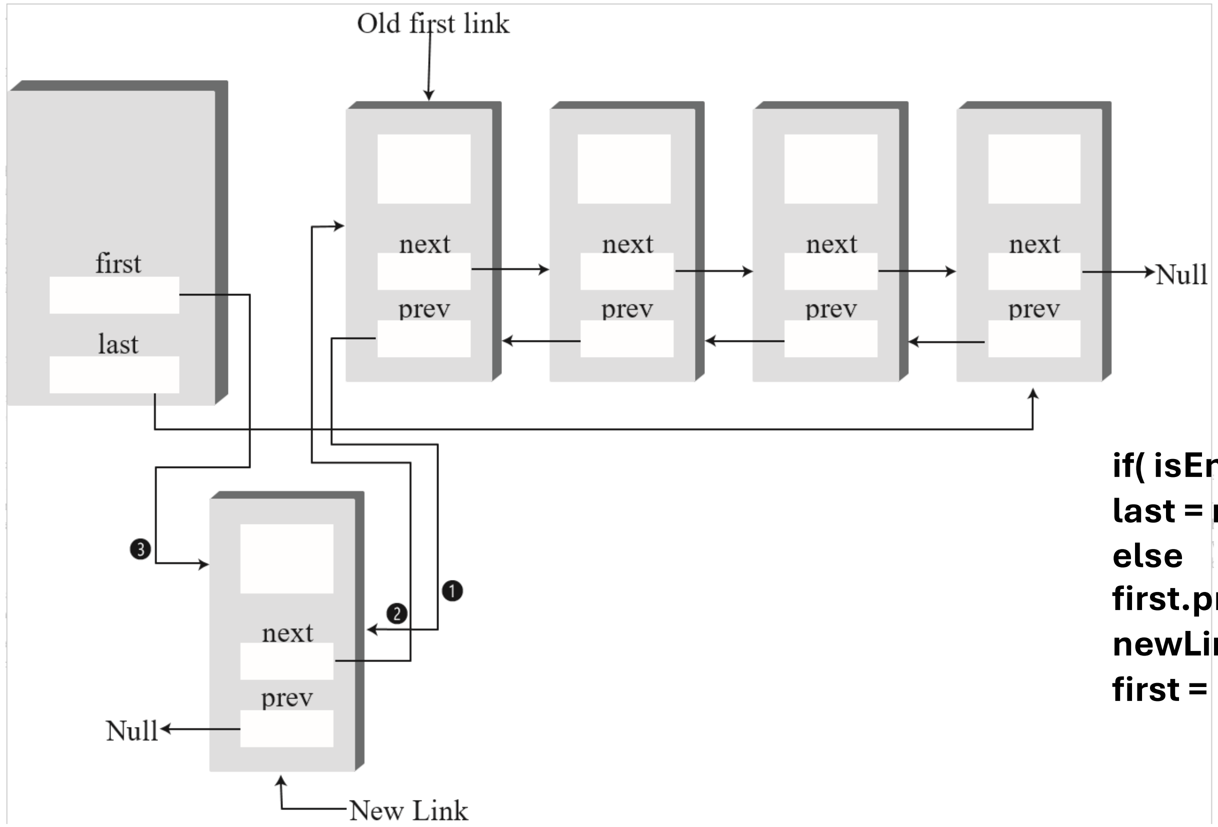
- **Doubly Linked Lists**



```
class Link
{
public long dData; // data item
public Link next; // next link in list
public link previous; // previous link in list
...
}
```

Doubly Linked Lists

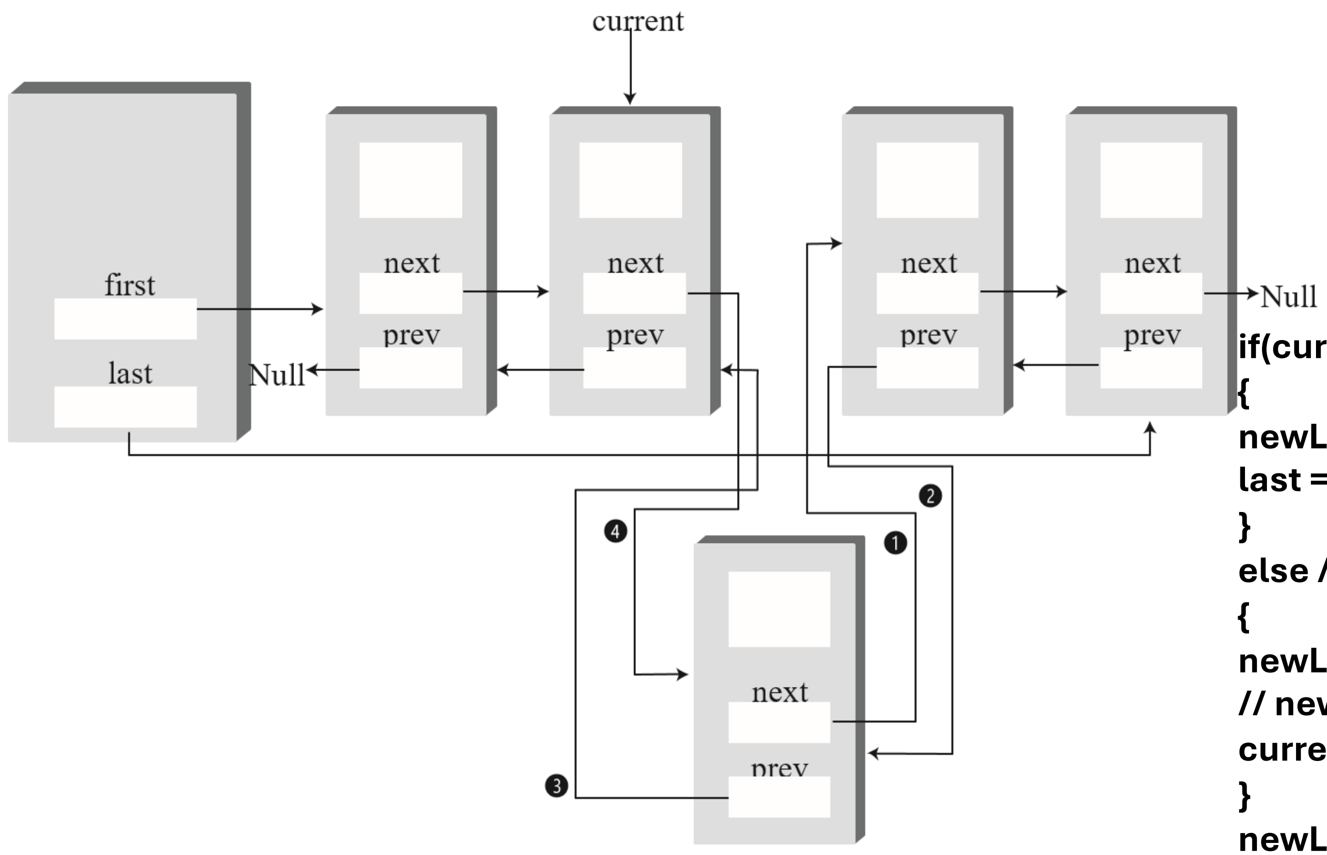
- **Doubly Linked Lists: Insertion at the beginning**



```
if( isEmpty() ) // if empty list,
last = newLink; // newLink <-- last
else
first.previous = newLink; // newLink <-- old first
newLink.next = first; // newLink --> old first
first = newLink; // first --> newLink
```

Doubly Linked Lists

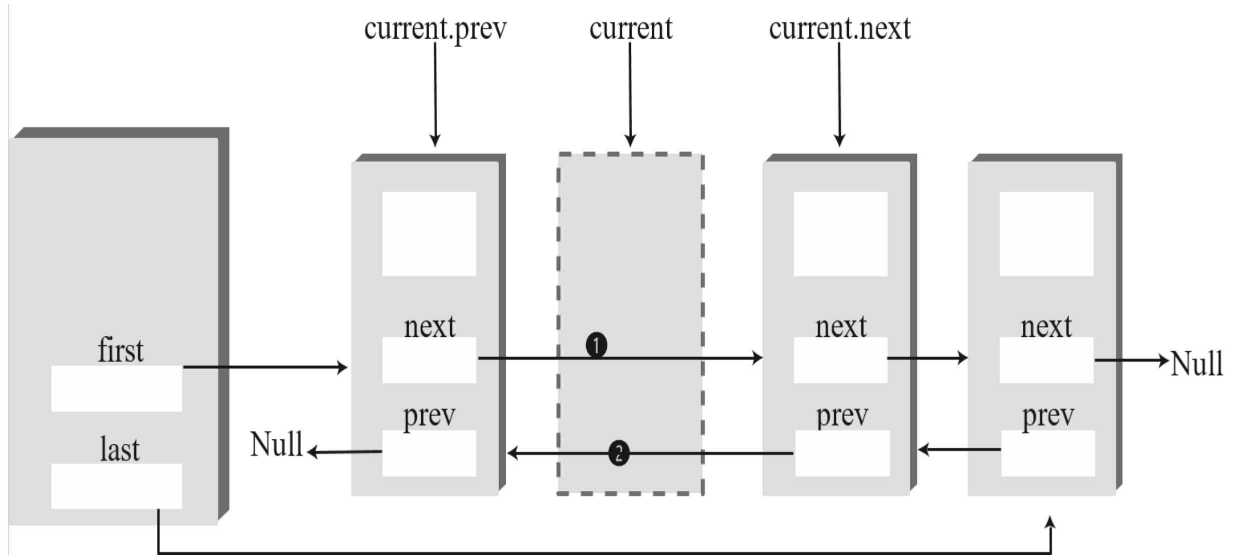
- **Doubly Linked Lists: Insertion at an arbitrary location**



```
if(current==last) // if last link,
{
newLink.next = null; // newLink --> null
last = newLink; // newLink <-- last
}
else // not last link,
{
newLink.next = current.next; // newLink --> old next
// newLink <-- old next
current.next.previous = newLink;
}
newLink.previous = current; // old current <-- newLink
current.next = newLink; // old current --> newLink
```

Doubly Linked Lists

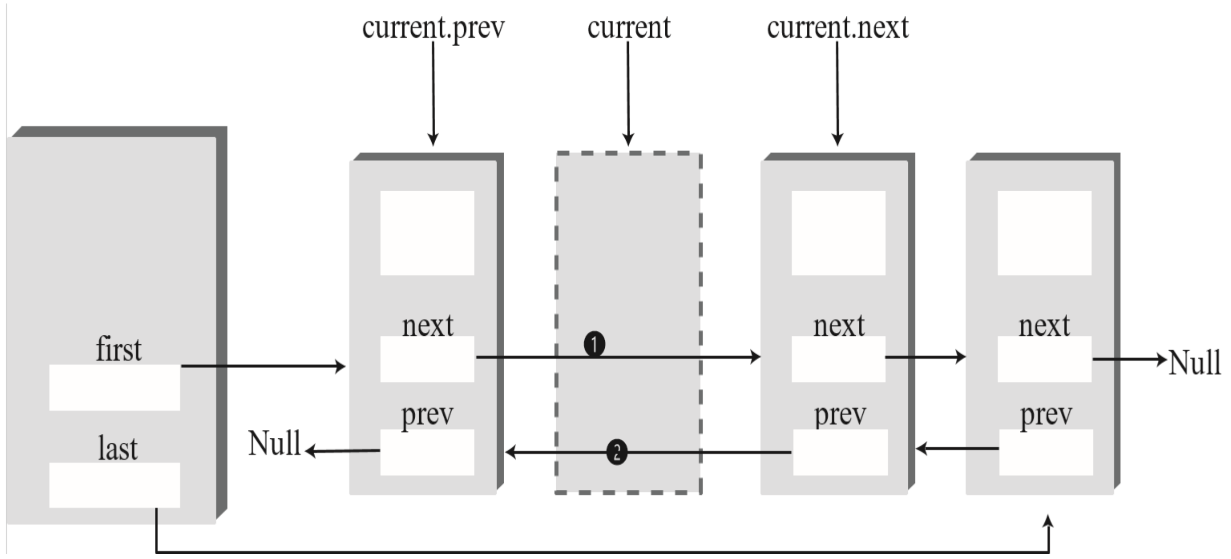
- **Doubly Linked Lists: Deletion an arbitrary link**



```
if(current==first) // first item?
first = current.next; // first --> old next
else // not first
// old previous --> old next
current.previous.next = current.next;
if(current==last) // last item?
last = current.previous; // old previous <-- last
else // not last
// old previous <-- old next
current.next.previous = current.previous;
```

Doubly Linked Lists

- **Doubly Linked Lists: Deletion an arbitrary link**



```
if(current==first) // first item?
first = current.next; // first --> old next
else // not first
// old previous --> old next
current.previous.next = current.next;
if(current==last) // last item?
last = current.previous; // old previous <-- last
else // not last
// old previous <-- old next
current.next.previous = current.previous;
```

References

- [1] https://www.godaddy.com/resources/in/web-pro-in/8-basic-data-structures-every-programmer-should-know

- [2] Robert Lafore, Data Structures & Algorithms in Java, MIT Press, 2022.