

Ordered Array

	Advantages	Disadvantages
Unordered array list	Insertion - $O(1)$	Search - $O(n)$ Deletion - $O(n)$ Fixed array size
Ordered array list	search - $O(\log_2 n)$	Insertion - $O(n)$ Deletion - $O(n)$ Fixed array size

Better performed when searches are more than insert

Disadvantage of using arrays to store data

▣ Disadvantages

- arrays are static structures and therefore cannot be easily extended or reduced to fit the data set.
- arrays are also expensive to maintain new insertions and deletions.

- ▣ In this presentation we consider another data structure called Linked Lists that addresses some of the limitations of arrays.

Linked Lists

- ▣ A linked list is a linear data structure where each element is a separate object.
- ▣ Each element (we will call it a node) of a list is comprising of two items
 - the data and a reference to the next node

LINKED REPRESENTATION

- list elements are stored, in memory, in an arbitrary order
- explicit information (called a link) is used to go from one element to the next

MEMORY LAYOUT

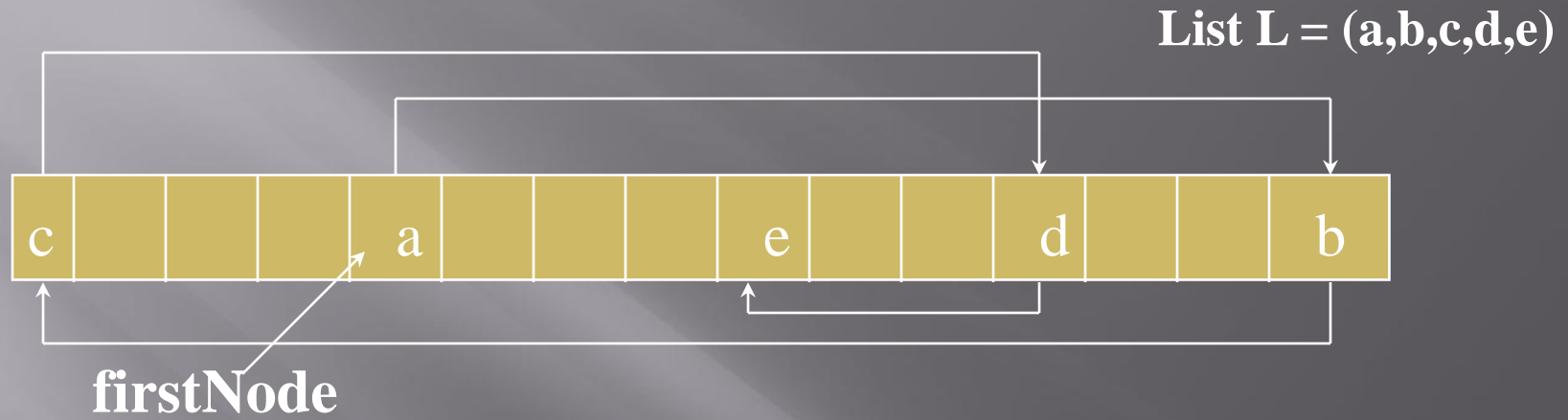
Layout of $L = (a,b,c,d,e)$ using an array representation.



A linked representation uses an arbitrary layout.



LINKED REPRESENTATION

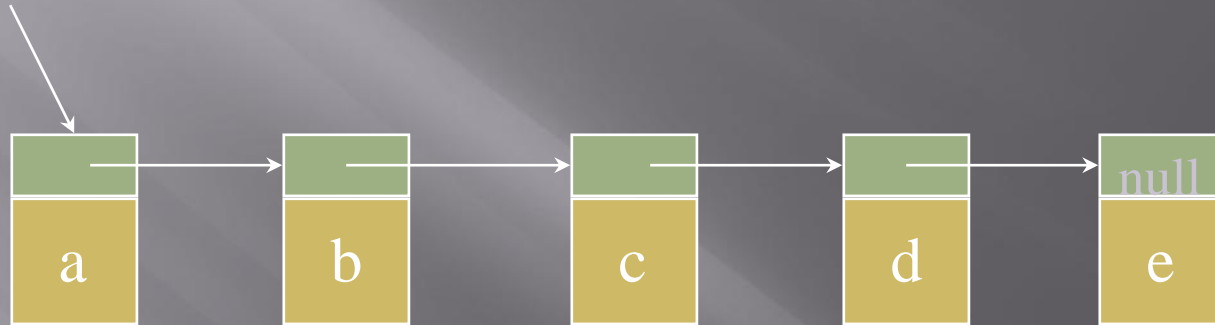


pointer (or link) in e is null

use a variable firstNode to get to the first element a

NORMAL WAY TO DRAW A LINKED LIST

firstNode

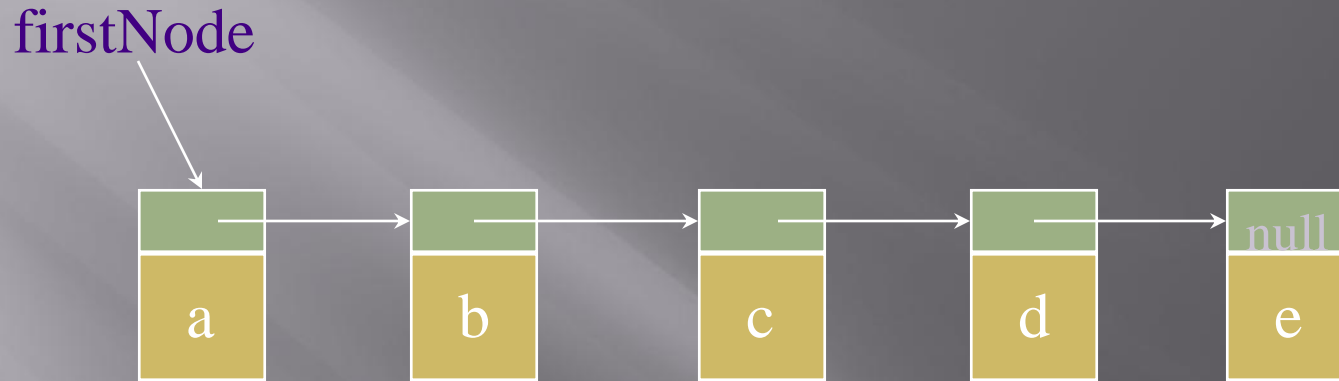


link or pointer field of node



data field of node

LINKED LIST



- In a linked list each node represents one element.
- There is a link or pointer from one element to the next.
- The last node has a null pointer.

NODE REPRESENTATION

```
class Node
{
    // data members
    Object data; // data field
    Node next;  // link field

    // constructors come here
}
```

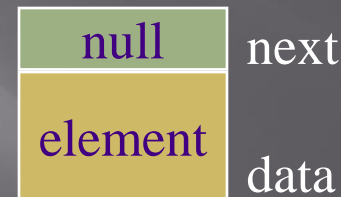


CONSTRUCTORS OF LINKEDLIST NODE

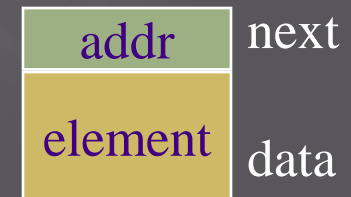
```
Node() {}
```



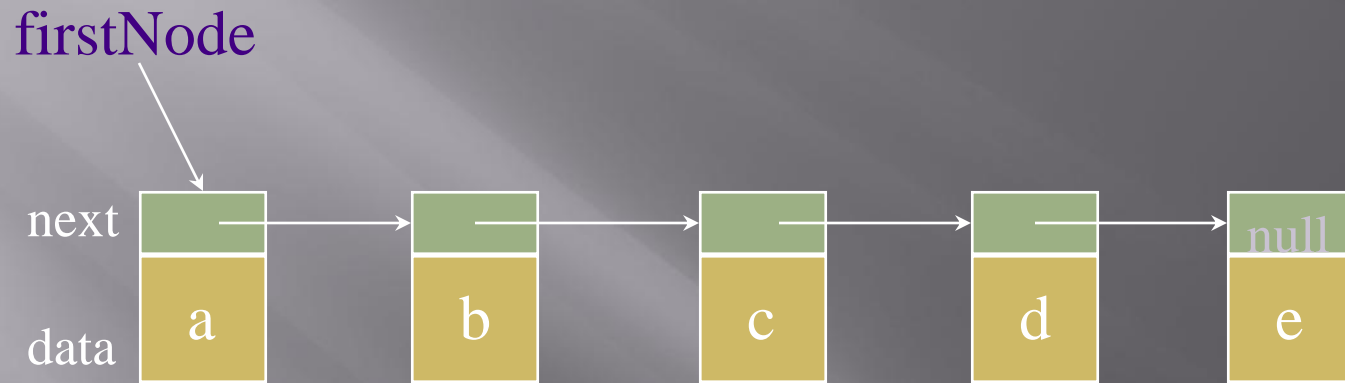
```
Node(Object element)  
{ this.data= element; }
```



```
Node(Object element, Node addr)  
{ this.data = element;  
  this.next = addr; }
```



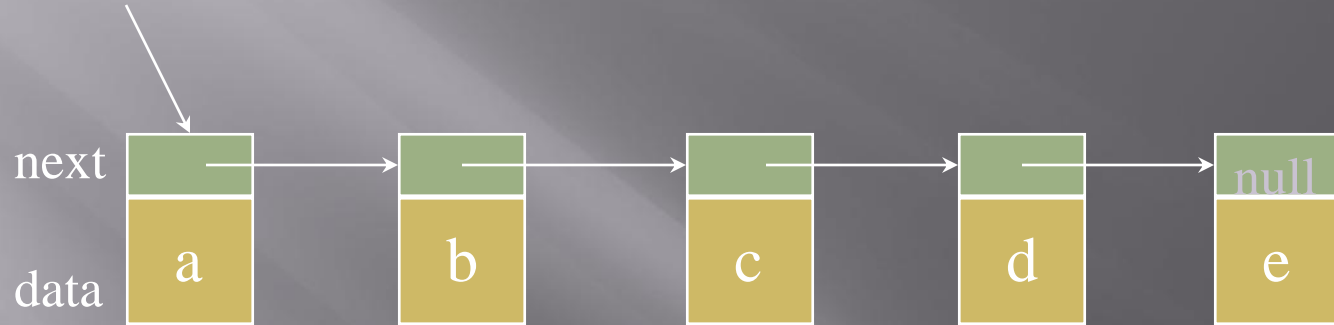
GET FIRST ELEMENT: GET(0)



```
checkIndex(0);  
return firstNode.data; // gets you to first node  
OR  
checkIndex(0);  
desiredNode = firstNode;  
return desiredNode.data;
```

GET SECOND ELEMENT: GET(1)

firstNode



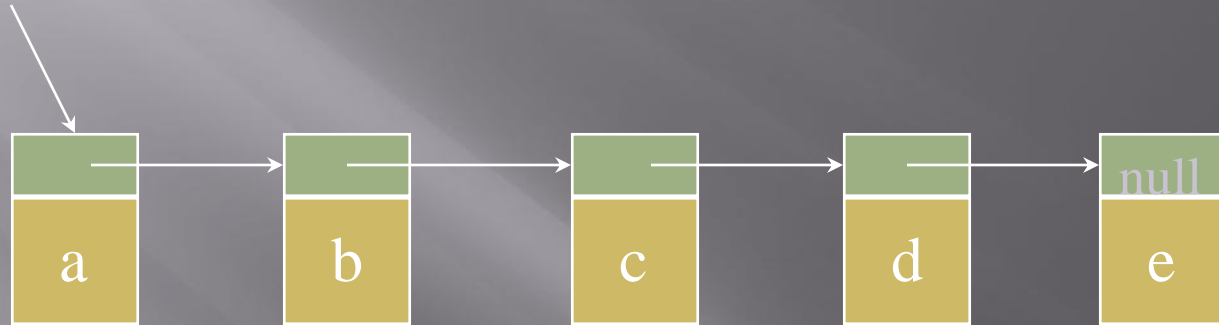
```
checkIndex(1);
```

```
desiredNode = firstNode.next; // gets you to second node
```

```
return desiredNode.data;
```

GET THIRD ELEMENT: GET(2)

firstNode



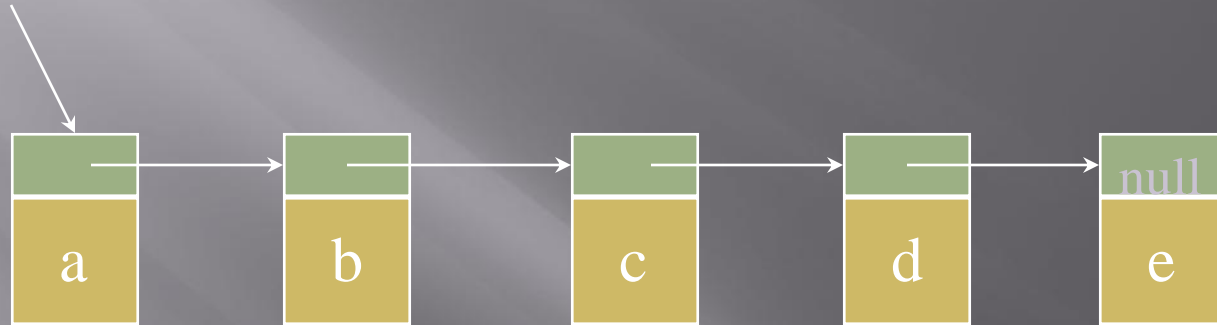
```
checkIndex(2);
```

```
desiredNode = firstNode.next.next; // gets you to third  
node
```

```
return desiredNode.data;
```

GET(5)

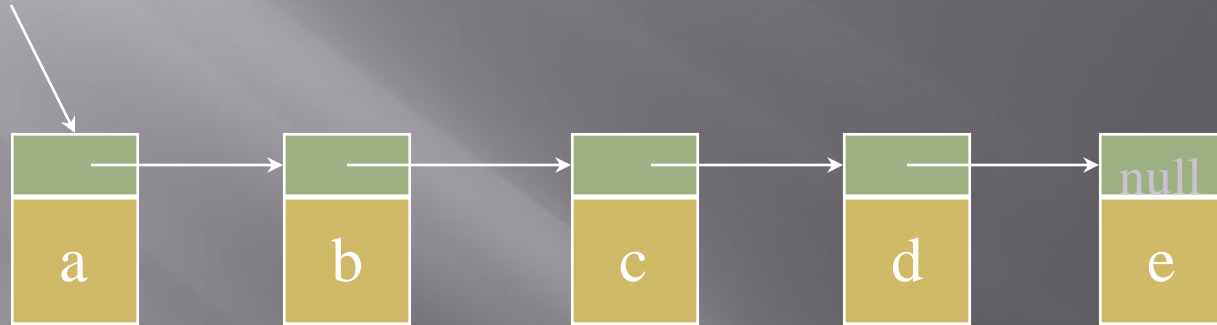
firstNode



```
checkIndex(5);           // throws exception
desiredNode = firstNode.next.next.next.next.next;
                        // desiredNode = null
return desiredNode.data; // null.data
```

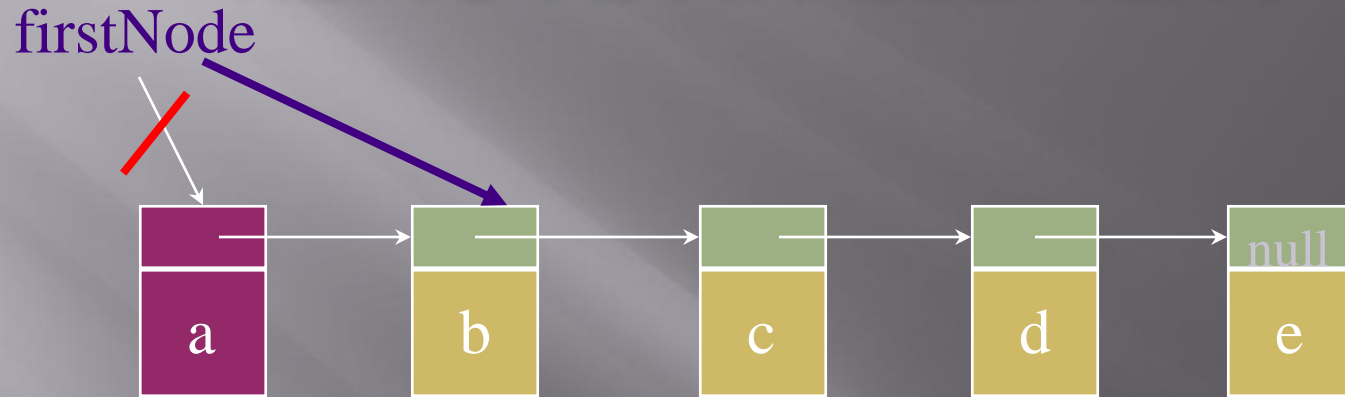
NULL POINTER EXCEPTION

firstNode



```
desiredNode =  
    firstNode.next.next.next.next.next.next;  
    // gets the computer mad  
    // you get a NullPointerException
```

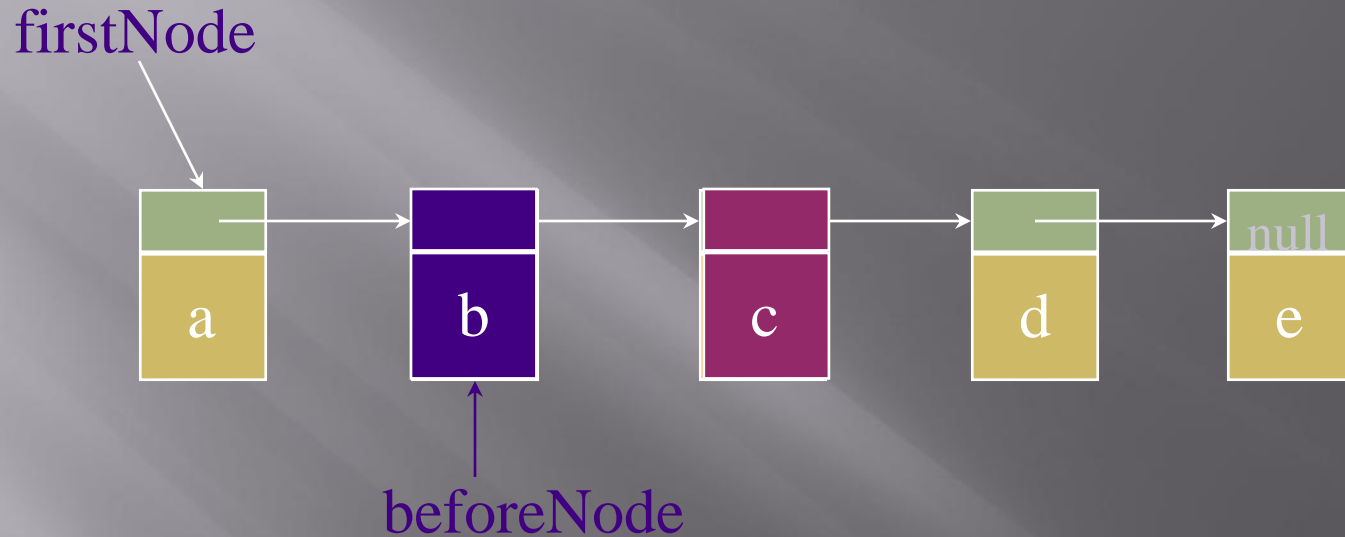
REMOVE AN ELEMENT



`remove(0)`

`firstNode = firstNode.next;`

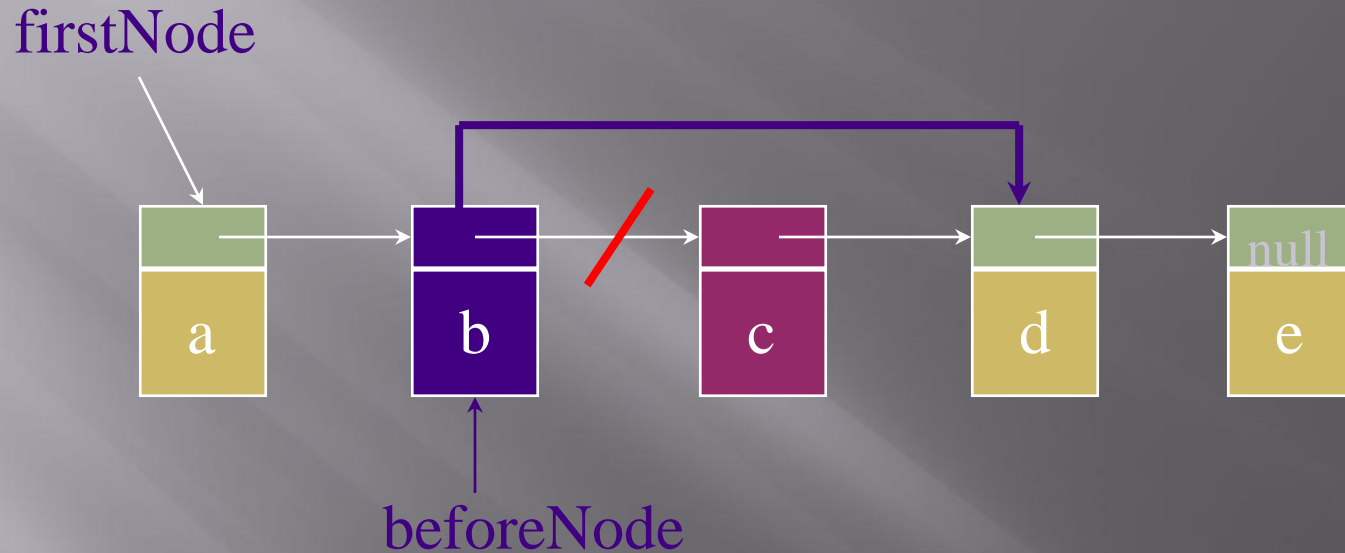
REMOVE(2)



first get to node just before node to be removed

`beforeNode = firstNode.next;`

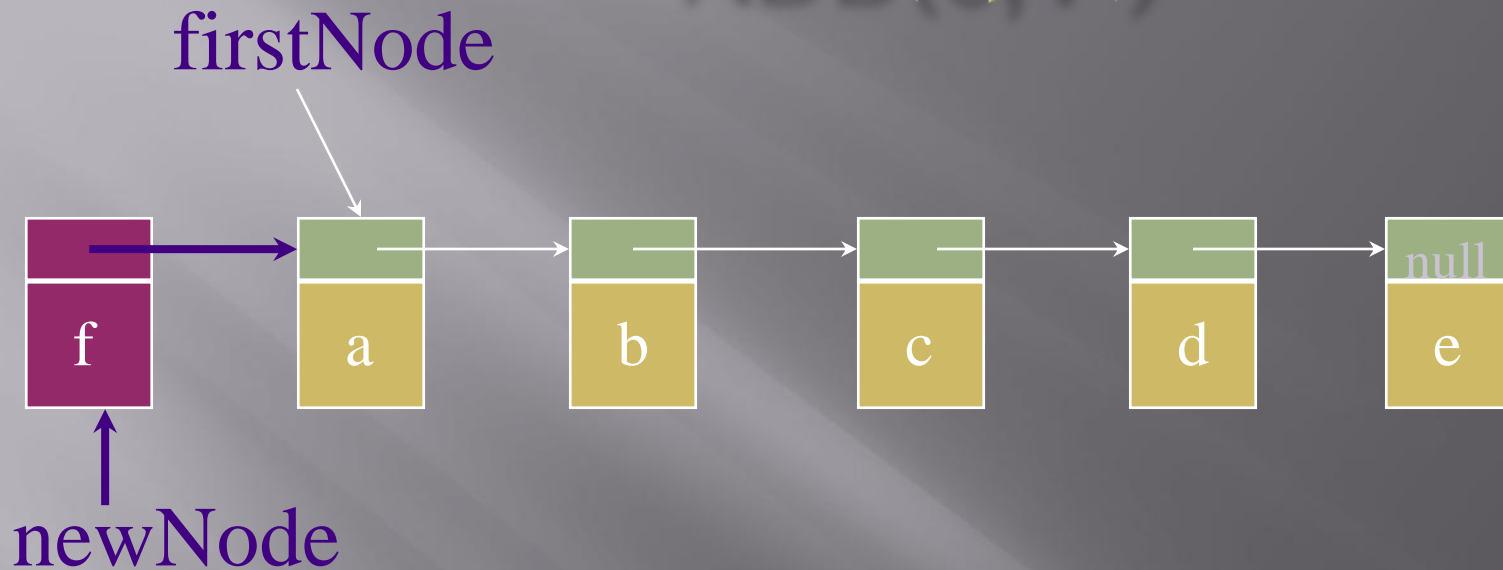
REMOVE(2)



now change pointer in beforeNode

```
beforeNode.next = beforeNode.next.next;
```

ADD(0, 'F')

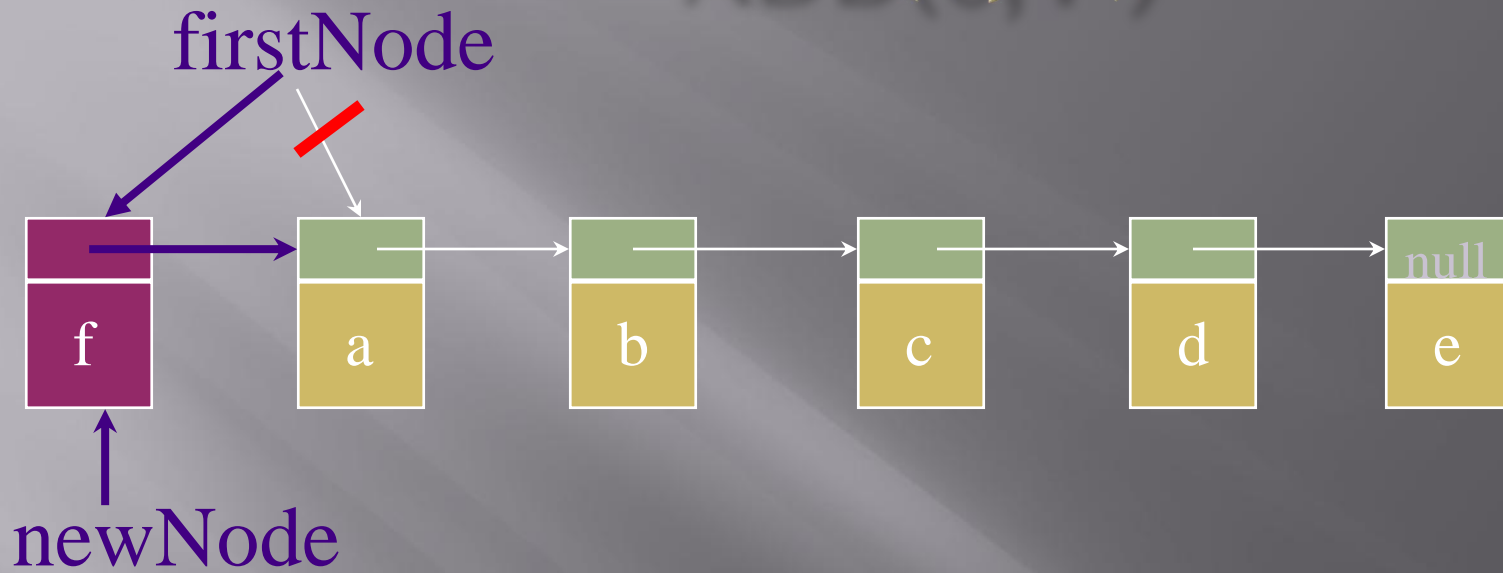


Step 1: get a new node, set its data and link fields

Node newNode =

```
new Node(new Character('f'), firstNode);
```

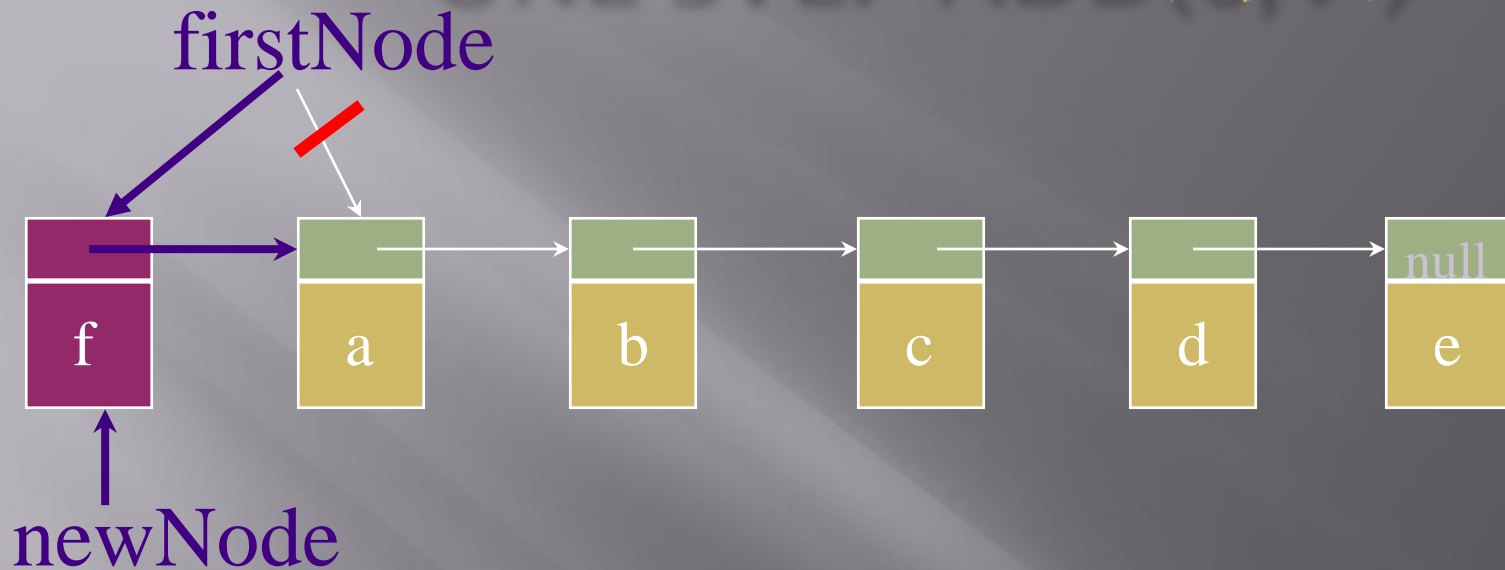
ADD(0,'F')



Step 2: update firstNode

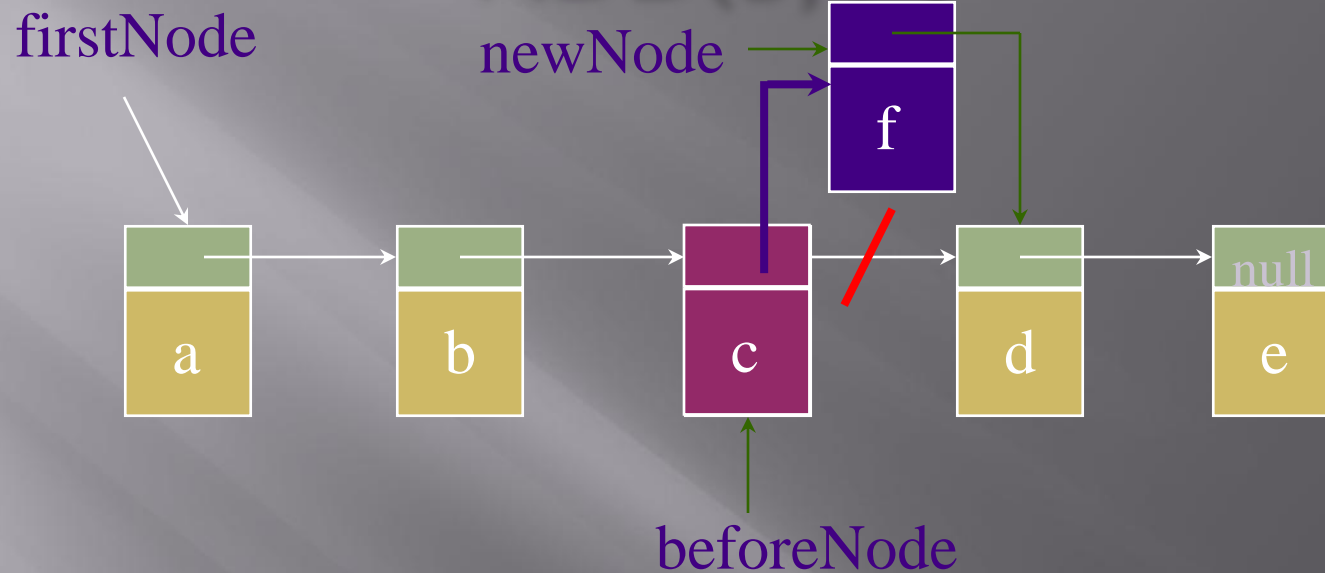
`firstNode = newNode;`

ONE-STEP ADD(0,'F')



```
firstNode = new Node(new Character('f'),  
                      firstNode);
```

ADD(3, 'F')



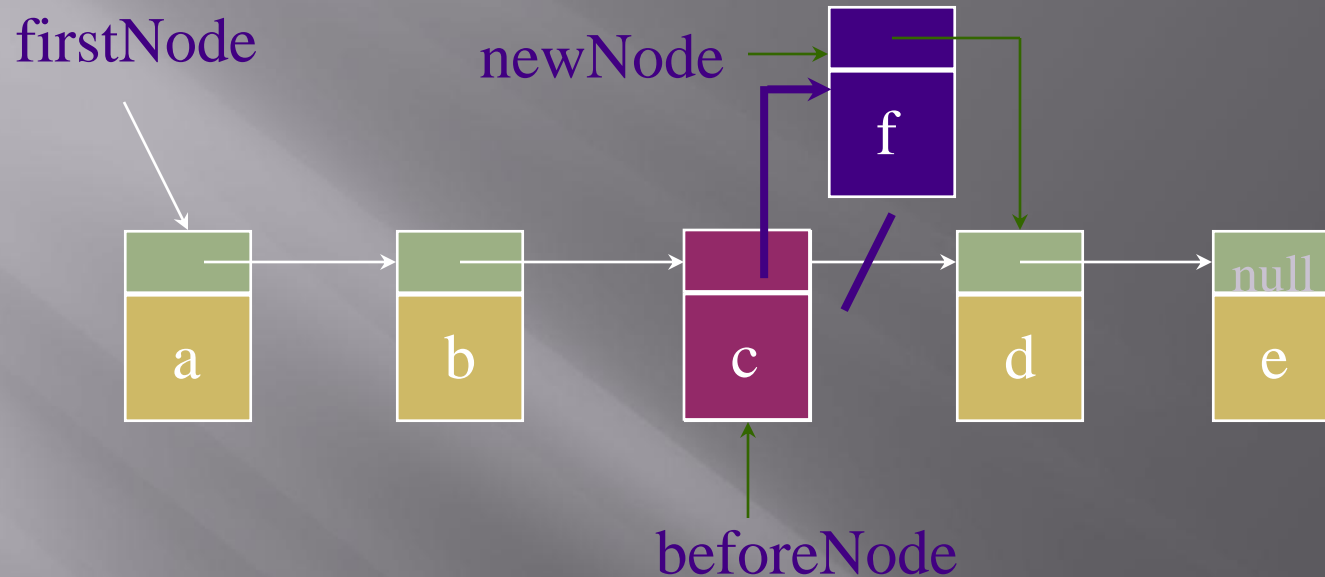
- first find node whose index is 2
- next create a node and set its data and link fields

```
Node newNode = new Node(new Character('f'), beforeNode.next);
```

- finally link beforeNode to newNode

```
beforeNode.next = newNode;
```

TWO-STEP ADD(3,'F')



```
beforeNode = firstNode.next.next;
```

```
beforeNode.next = new Node(new Character('f'),  
                             beforeNode.next);
```

Remember

- ▣ A linked list is a dynamic data structure. The number of nodes in a list is not fixed and can grow and shrink on demand.
- ▣ Any application which has to deal with an unknown number of objects will need to use a linked list.
- ▣ One disadvantage against an array is that
 - it does not allow direct access to the individual elements. If you want to access a particular item then you have to start at the head and follow the references until you get to that item.