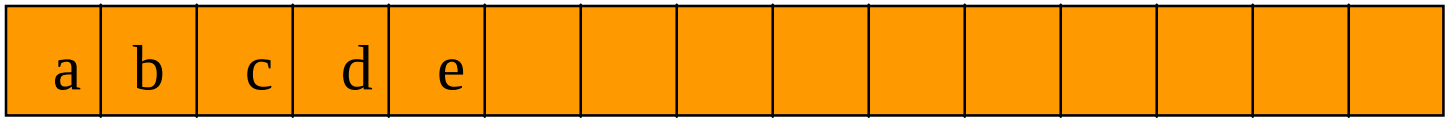
Linked Lists

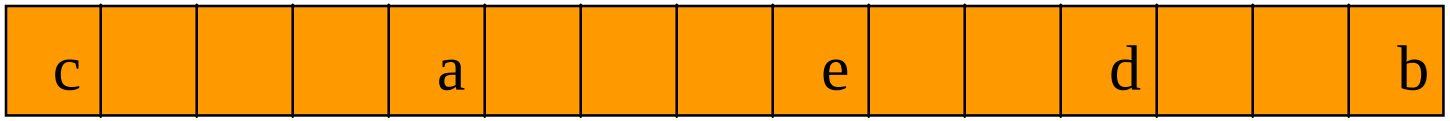
- list elements are stored, in memory, in an arbitrary order
- explicit information (called a link) is used to go from one element to the next

Memory Layout

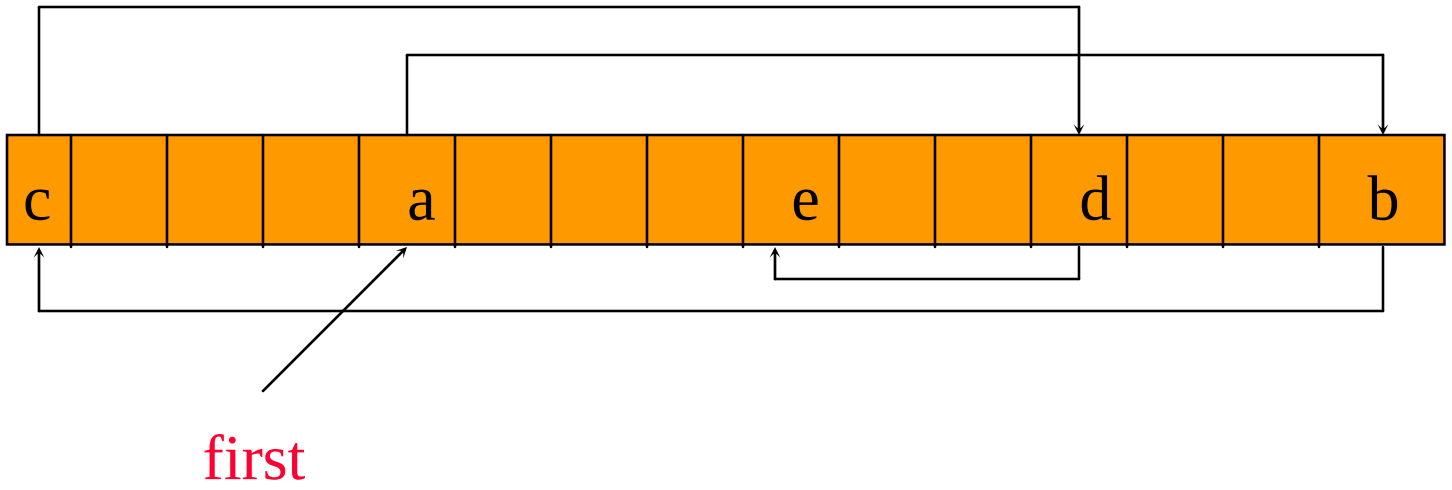
Layout of L = (a,b,c,d,e) using an array representation.



A linked representation uses an arbitrary layout.



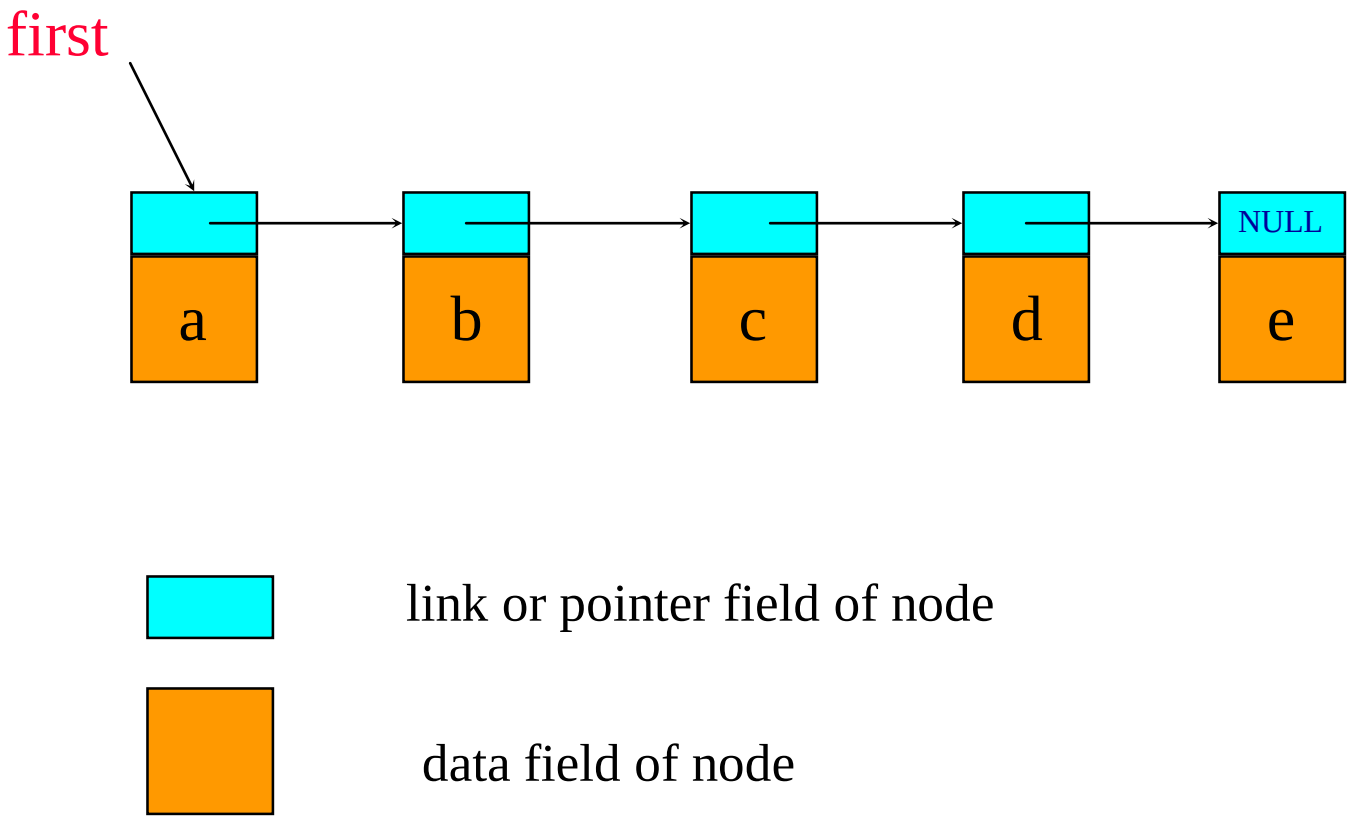
Linked Representation



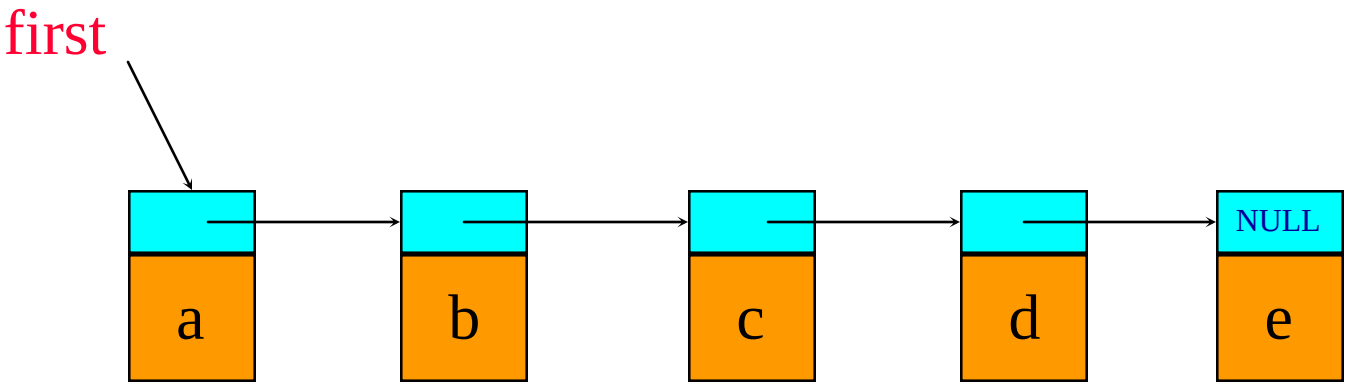
pointer (or link) in e is NULL

use a variable first to get to the first element a

Normal Way To Draw A Linked List



Chain

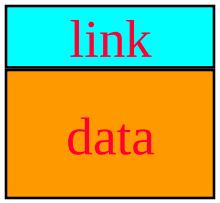


- A chain is a linked list in which each node represents one element.
- There is a link or pointer from one element to the next.
- The last node has a NULL (or 0) pointer.

Node Representation

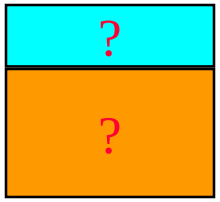
```
template <class T>
class ChainNode
{
  private:
    T data;
    ChainNode<T> *link;

    // constructors come here
};
```

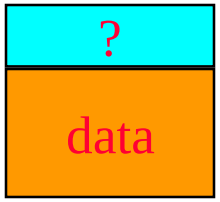


Constructors Of ChainNode

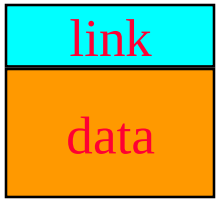
ChainNode() {}



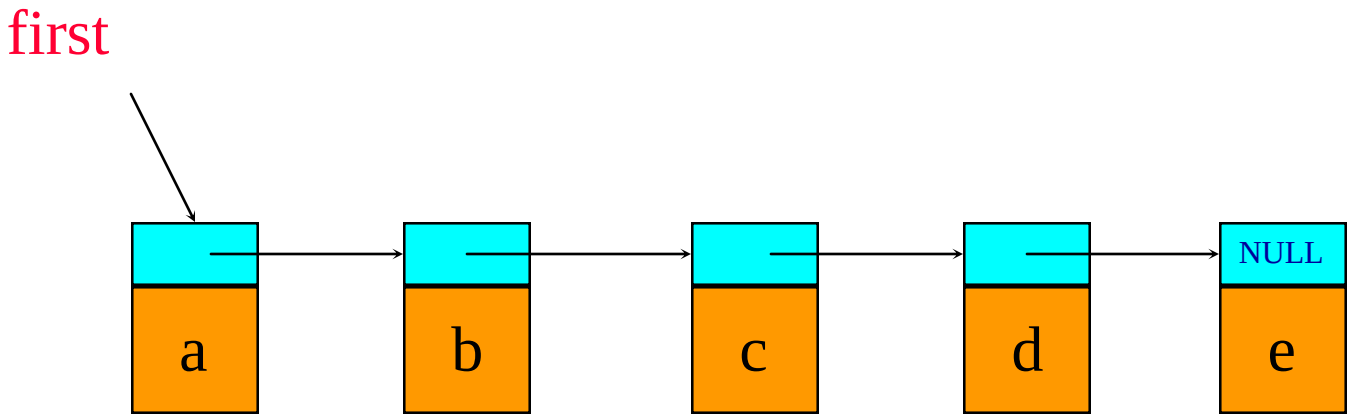
ChainNode(const T& data)
{this->data = data;}



ChainNode(const T& data, chainNode<T>* link)
{this->data = data;
this->link = link;}



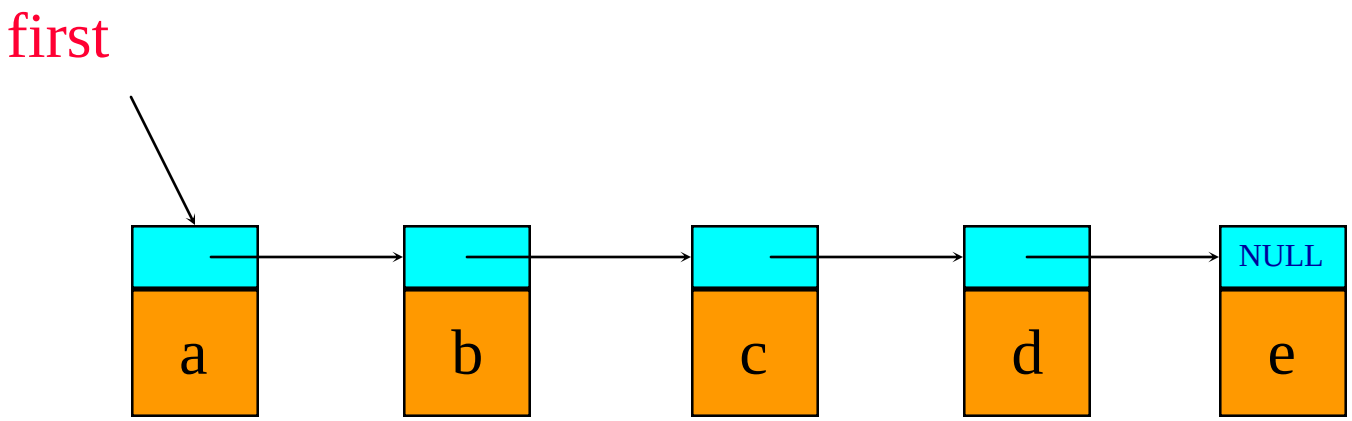
Get(0)



desiredNode = first; // gets you to first node

return desiredNode−•data;

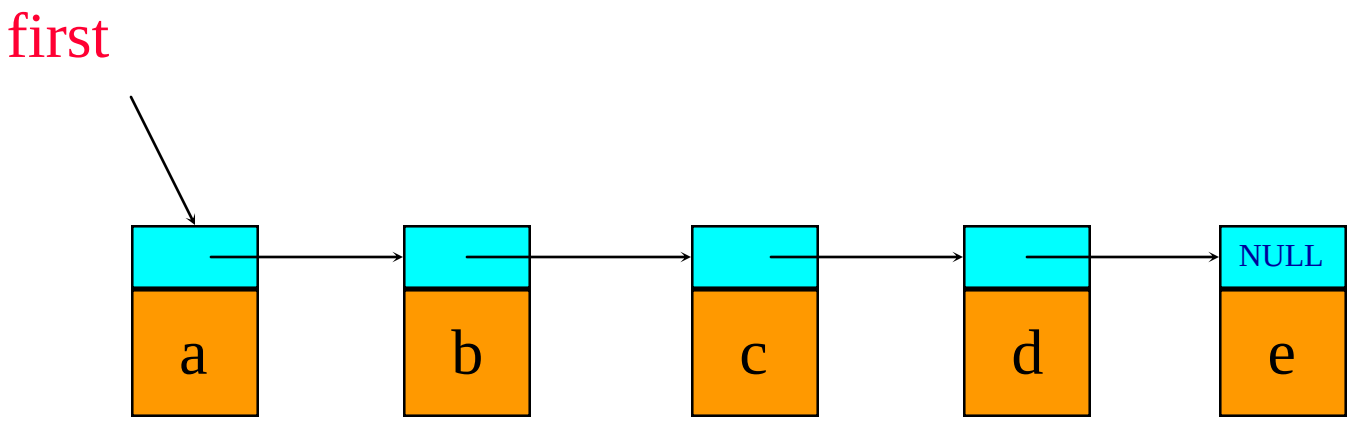
Get(1)



desiredNode = first−•link; // gets you to second node

return desiredNode−•data;

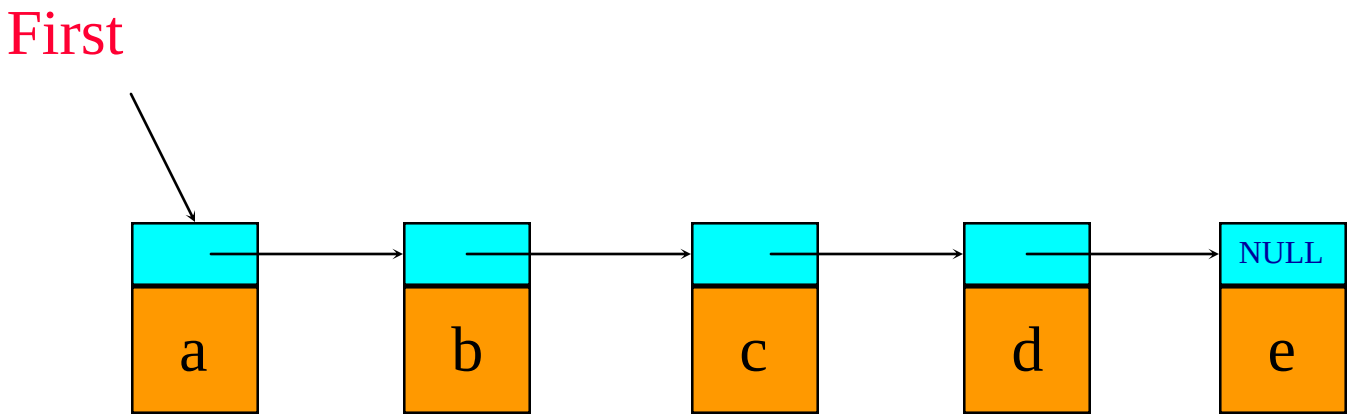
Get(2)



desiredNode = first−•link−•link; // gets you to third node

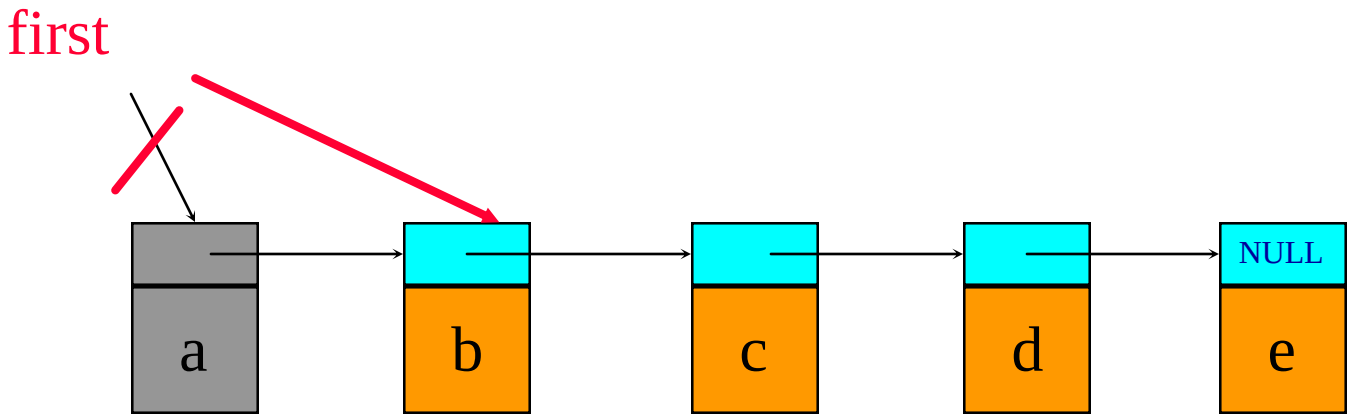
return desiredNode−•data;

Get(5)




```
desiredNode = first−•link−•link−•link−•link−•link;
                                    // desiredNode = NULL
return desiredNode−•data;   // NULL.element
```

Delete An Element



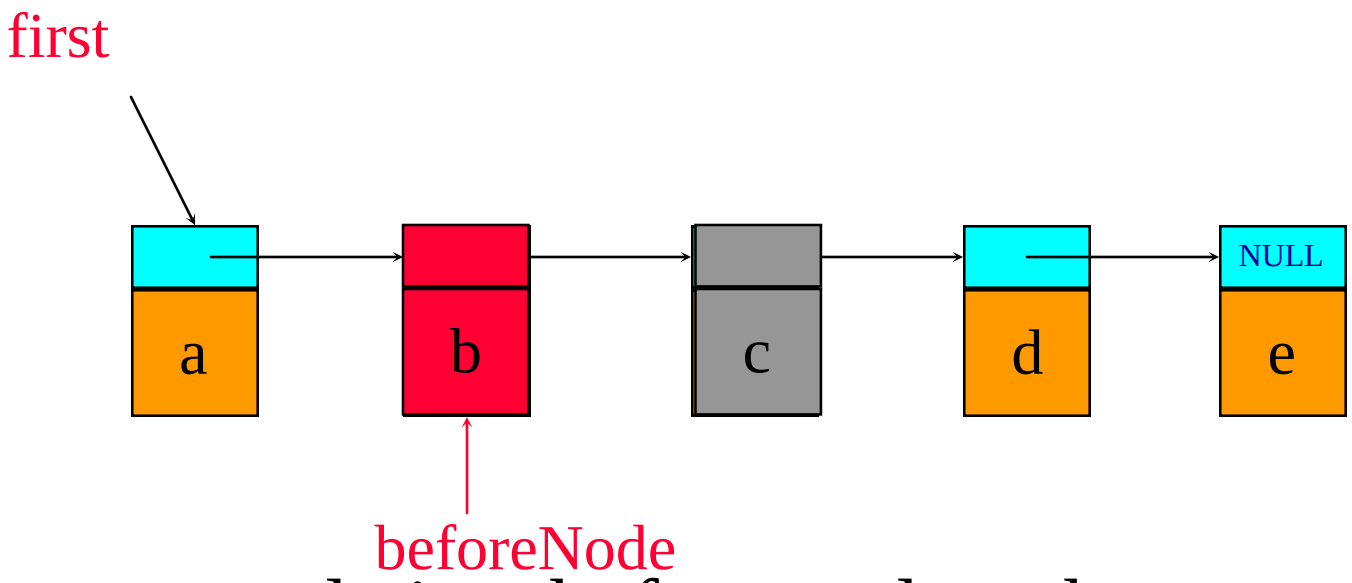
Delete(0)

deleteNode = first;

first = first−•link;

delete deleteNode;

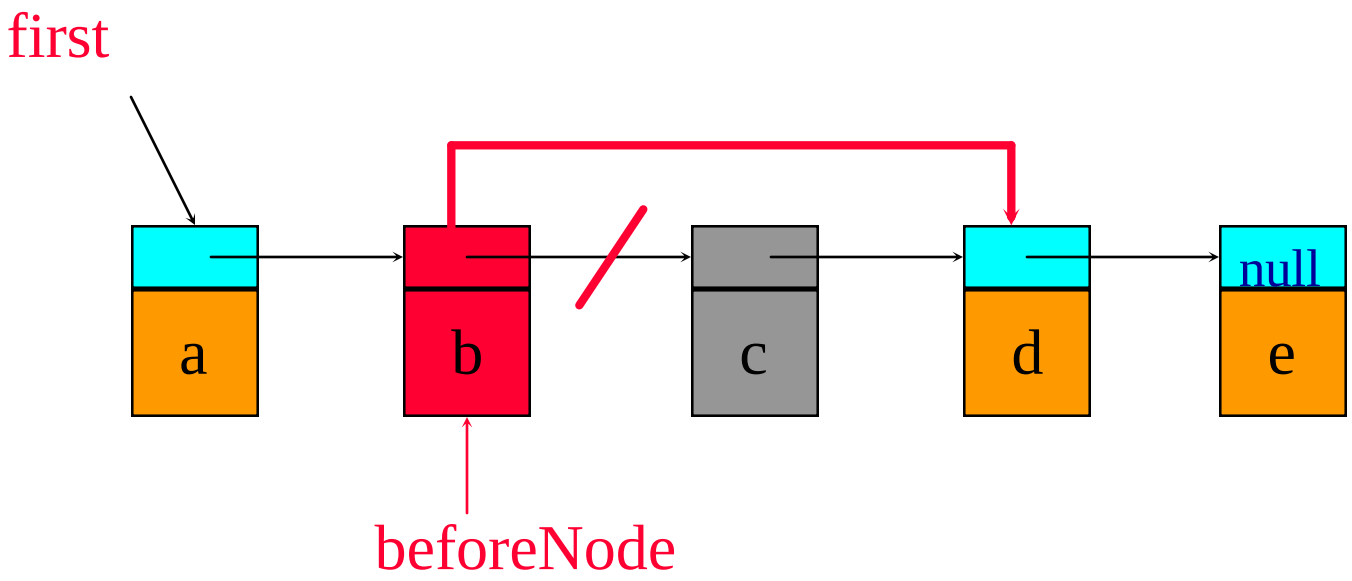
Delete(2)



first get to node just before node to be removed

beforeNode = first−•link;

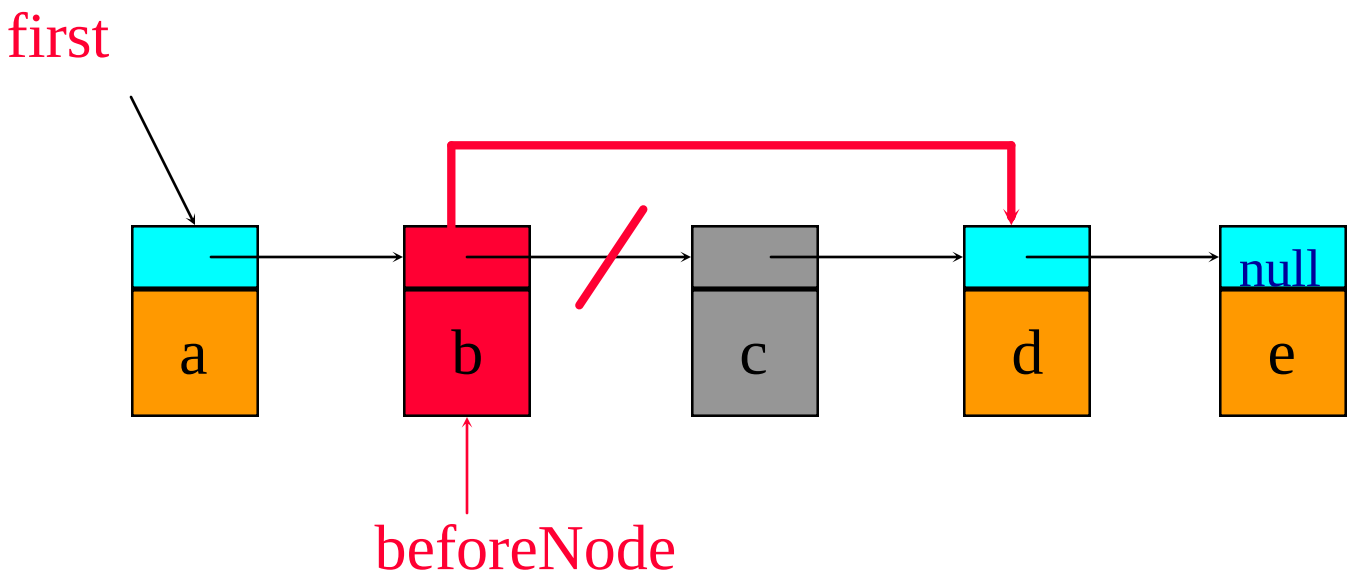
Delete(2)



save pointer to node that will be deleted

deleteNode = beforeNode−•link;

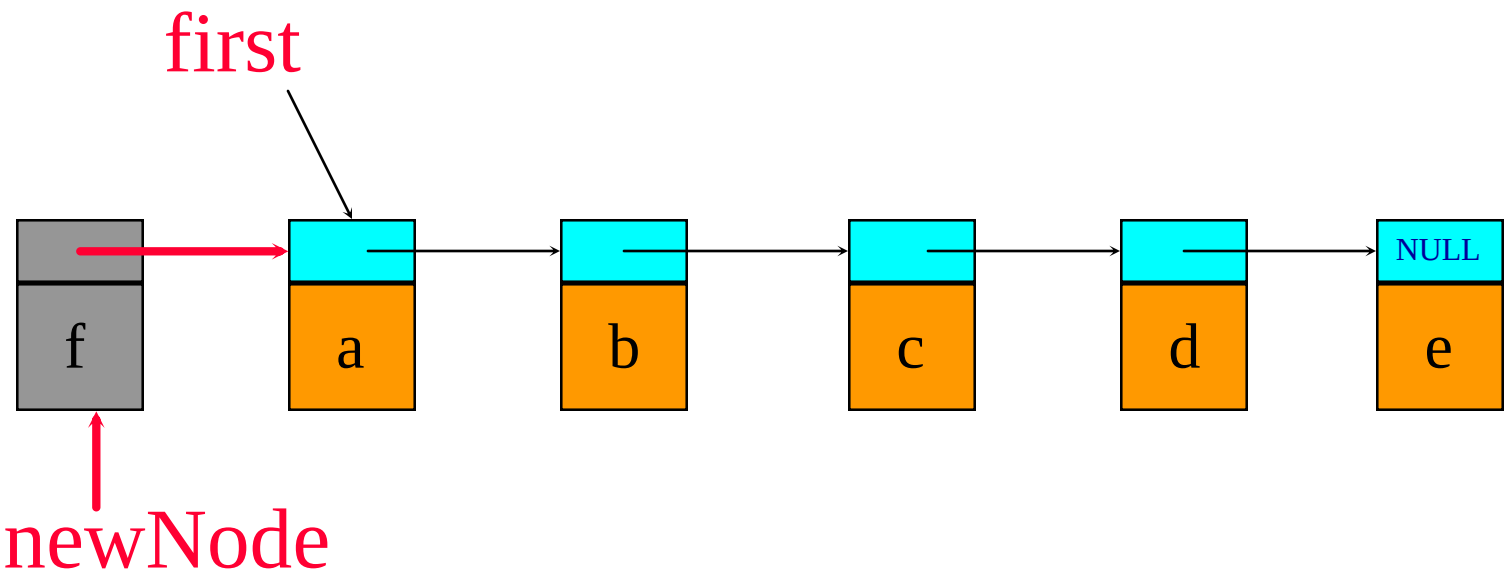
Delete(2)



now change pointer in beforeNode

```
beforeNode->link = beforeNode->link->link;
delete deleteNode;
```

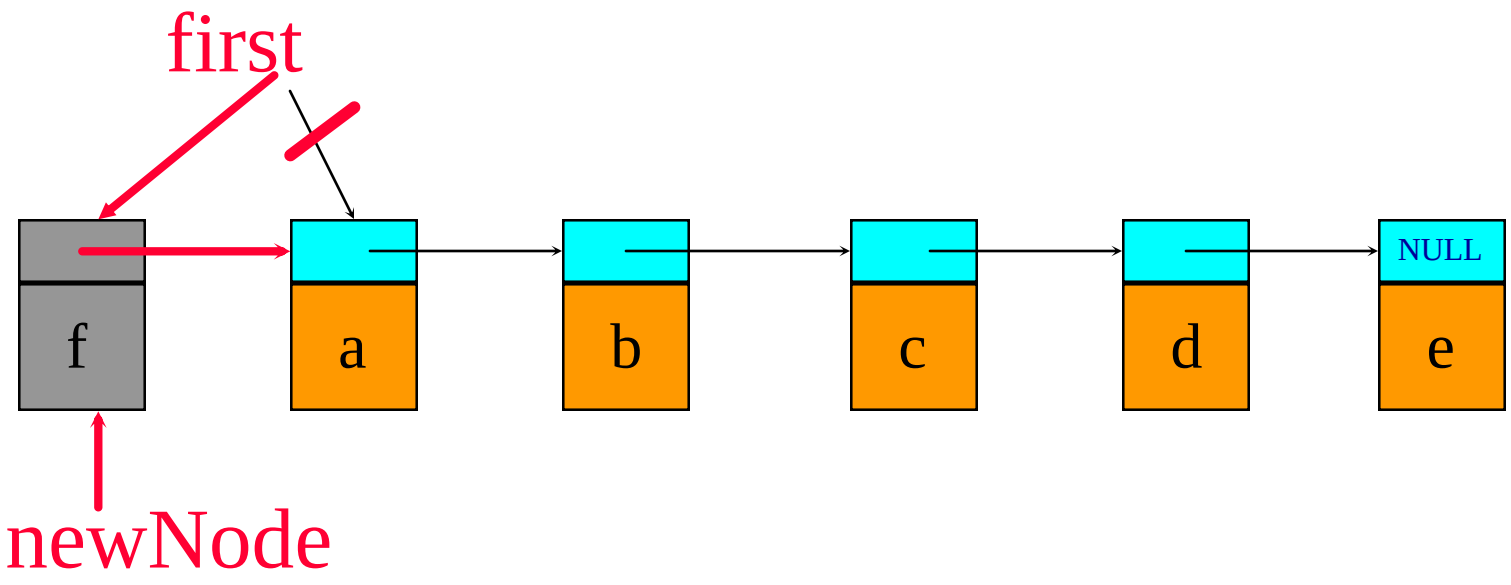
Insert(0,'f')



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

```
newNode = new ChainNode<char>(theElement,
                              first);
```

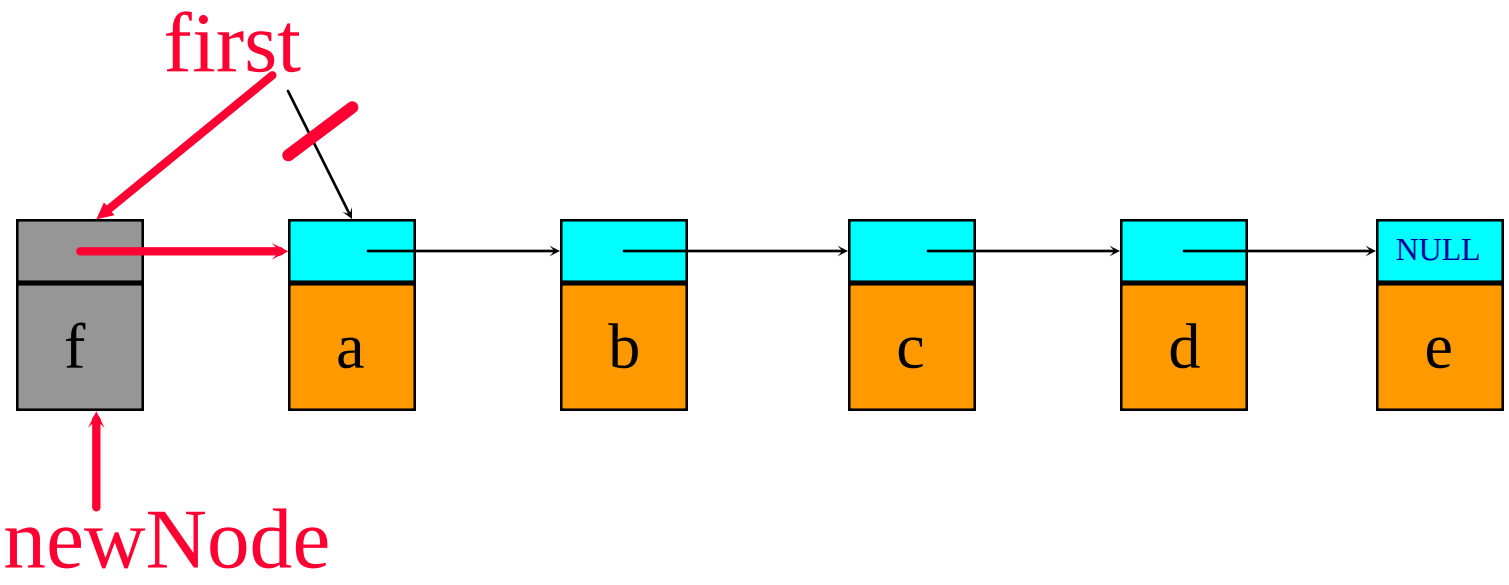
Insert(0,'f')



Step 2: update first

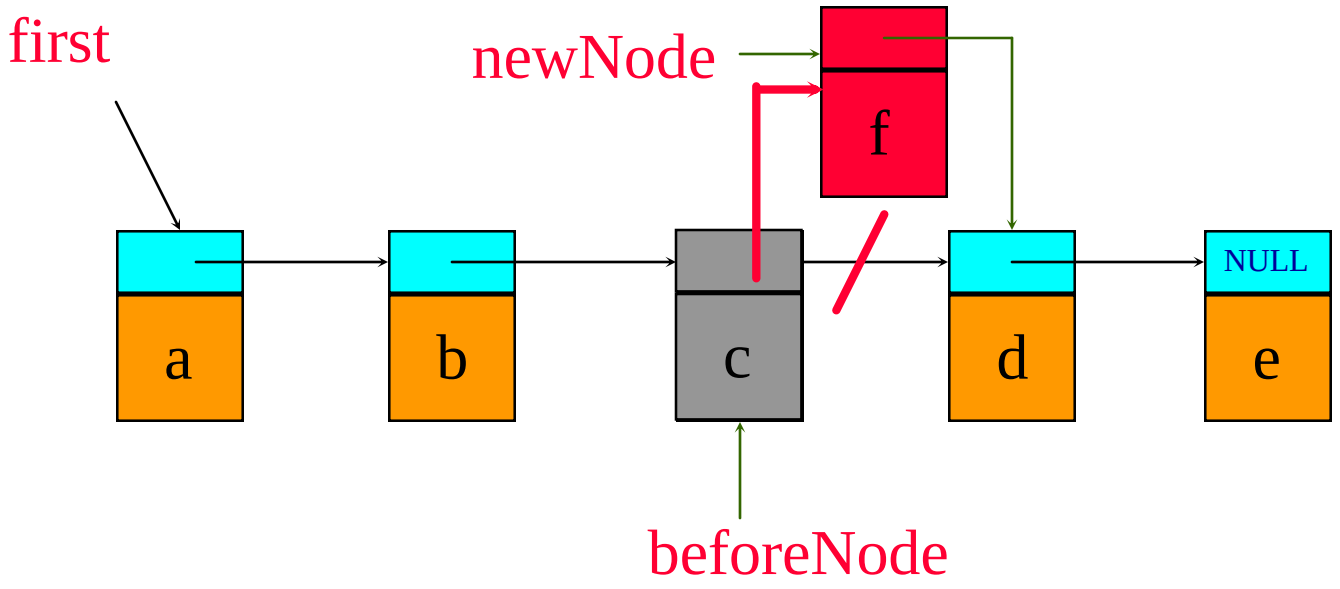
first = newNode;

One-Step Insert(0,'f')



first = new chainNode<char>('f', first);

Insert(3,'f')



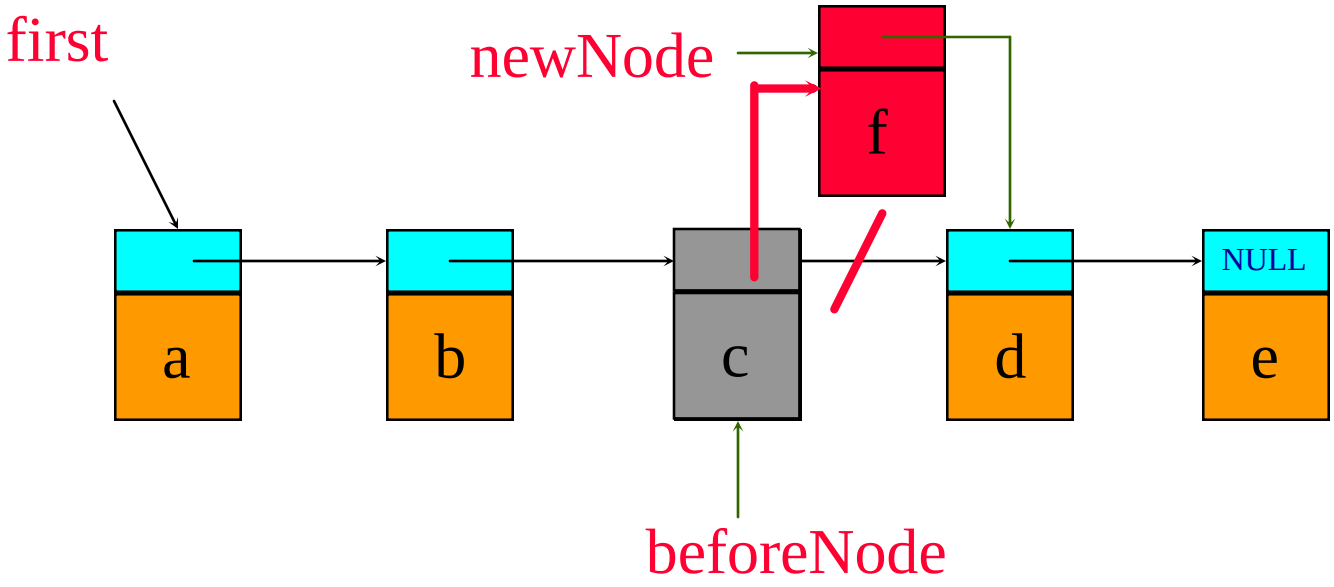
- first find node whose index is 2
- next create a node and set its data and link fields

```
ChainNode<char>* newNode = new ChainNode<char>( 'f',
                                        beforeNode->link);
```

- finally link beforeNode to newNode

```
beforeNode->link = newNode;
```

Two-Step Insert(3,'f')



beforeNode = first−•link−•link;

beforeNode−•link = new ChainNode<char>

('f', beforeNode−•link);