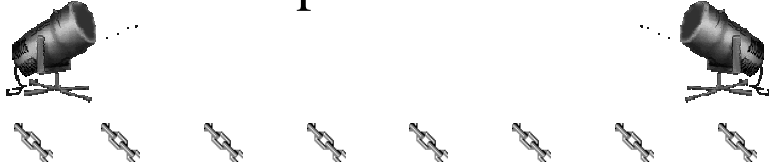




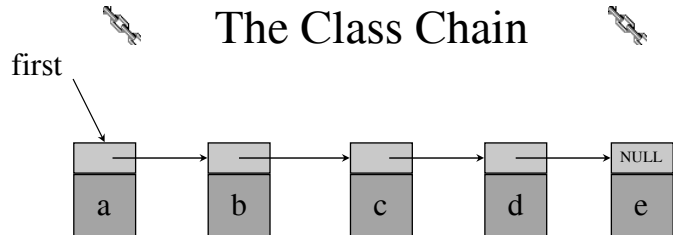
## The Template Class Chain



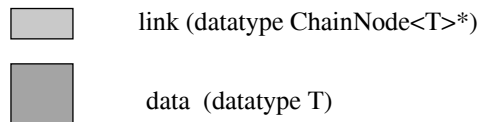
## Chain

- Linear list.
- Each element is stored in a node.
- Nodes are linked together using pointers.

## The Class Chain



### Use ChainNode



## The Template Class Chain

```

template<class T>
class Chain
{
public:
    Chain() {first = 0;}
        // constructor, empty chain
    ~Chain(); // destructor
    bool IsEmpty() const {return first == 0;}
        // other methods defined here
private:
    ChainNode<T>* first;
};
  
```

## The Destructor

```
template<class T>
chain<T>::~~chain()
{
  // Chain destructor. Delete all nodes
  // in chain.
  while (first != NULL)
  {
    // delete first
    ChainNode<T>* next = first->link;
    delete first;
    first = next;
  }
}
```

5

## The Method IndexOf

```
template<class T>
int Chain<T>::IndexOf(const T& theElement) const
{
  // search the chain for theElement
  ChainNode<T>* currentNode = first;
  int index = 0; // index of currentNode
  while (currentNode != NULL &&
         currentNode->data != theElement)
  {
    // move to next node
    currentNode = currentNode->next;
    index++;
  }
}
```

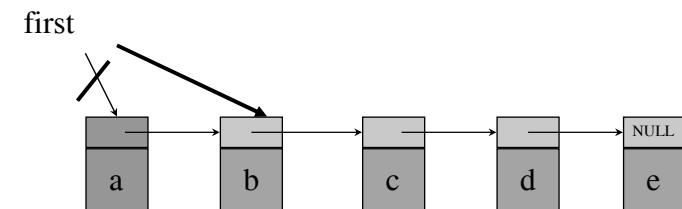
6

## The Method IndexOf

```
// make sure we found matching element
if (currentNode == NULL)
  return -1;
else
  return index;
}
```

7

## Delete An Element



```
delete(0)
```

```
deleteNode = first;
```

```
first = first->link;
```

```
delete deleteNode;
```

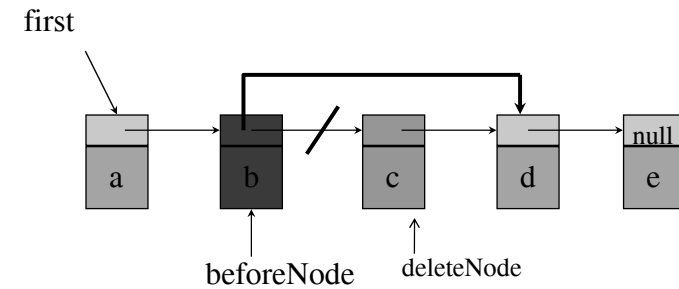
8

## Delete An Element

```
template<class T>
void Chain<T>::Delete(int theIndex)
{
    if (first == 0)
        throw "Cannot delete from empty chain";
    ChainNode<T>* deleteNode;
    if (theIndex == 0)
        { // remove first node from chain
            deleteNode = first;
            first = first->link;
        }
}
```

9

## Delete(2)



Find & change pointer in beforeNode  
`beforeNode->link = beforeNode->link->link;`  
`delete deleteNode;`

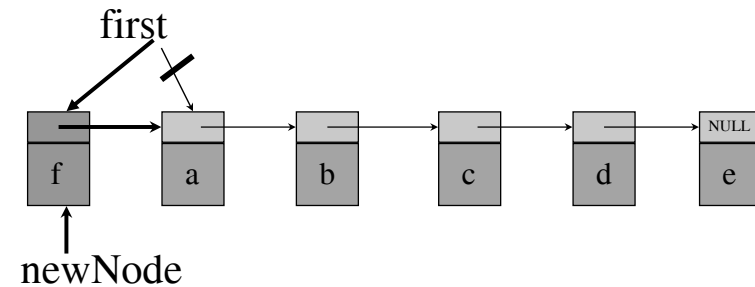
10

## Delete An Element

```
else
{ // use p to get to beforeNode
    ChainNode<T>* p = first;
    for (int i = 0; i < theIndex - 1; i++)
        {if (p == 0)
            throw "Delete element does not exist";
            p = p->next;}
    deleteNode = p->link;
    p->link = p->link->link;
}
delete deleteNode;
}
```

11

## One-Step Insert(0,'f')



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

12

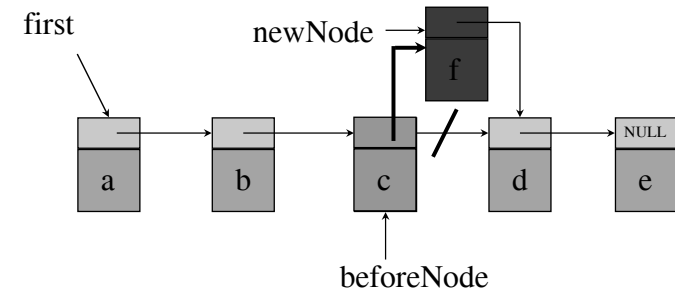
## ✂ Insert An Element ✂

```
template<class T>
void Chain<T>::Insert(int theIndex,
                    const T& theElement)
{
    if (theIndex < 0)
        throw "Bad insert index";

    if (theIndex == 0)
        // insert at front
        first = new chainNode<T>
            (theElement, first);
```

13

## Two-Step Insert(3,'f')



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

14

## ✂ Inserting An Element ✂

```
else
{ // find predecessor of new element
    ChainNode<T>* p = first;
    for (int i = 0; i < theIndex - 1; i++)
    {if (p == 0)
        throw "Bad insert index";
        p = p->next;}
    // insert after p
    p->link = new ChainNode<T>
        (theElement, p->link);
}
}
```

15