Linked List

Dr. Rajni Bala

Disadvantages of Array

- Arrays are nice and simple for storing things in a certain order, but they have drawbacks.
- They are not very adaptable. For instance, we have to fix the size *n* of an array in advance, which makes resizing an array difficult.
- Insertions and deletions are difficult because elements need to be shifted around to make space for insertion or to fill empty positions after deletion.

Linked list

- A ***linked list***, in its simplest form, is a collection of ***nodes*** that together form a linear ordering.
- As in the children's game "Follow the Leader," each node stores a pointer, called *next*, to the next node of the list.

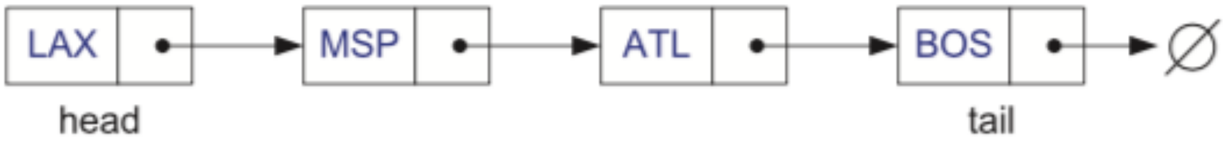
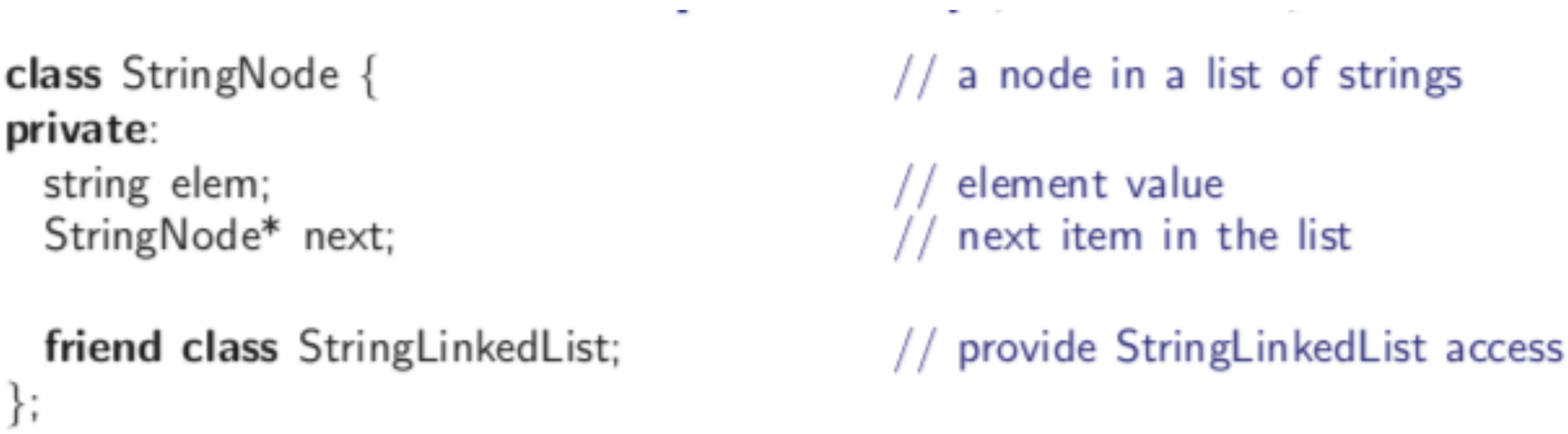


Figure 3.9: Example of a singly linked list of airport codes. The *next* pointers are shown as arrows. The null pointer is denoted by ∅.

Singly linked list

- The *next* pointer inside a node is a ***link*** or ***pointer*** to the next node of the list. Moving from one node to another by following a *next* reference is known as ***link hopping*** or ***pointer hopping***.
- The first and last nodes of a linked list are called the ***head*** and ***tail*** of the list, respectively. Thus, we can link-hop through the list, starting at the head and ending at the tail.
- The tail is the node having a null *next* reference.
- The structure is called a ***singly linked list*** because each node stores a single link.
- Like an array, a singly linked list maintains its elements in a certain order, as determined by the chain of *next* links.
- Unlike an array, a singly linked list does not have a predetermined fixed size. It can be resized by adding or removing nodes

Class to represent a node



```
class StringNode {                       // a node in a list of strings
private:
  string elem;                           // element value
  StringNode* next;                      // next item in the list

  friend class StringLinkedList;         // provide StringLinkedList access
};
```

Code Fragment 3.13: A node in a singly linked list of strings.

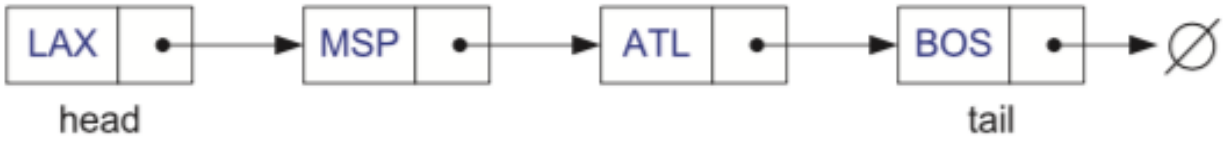
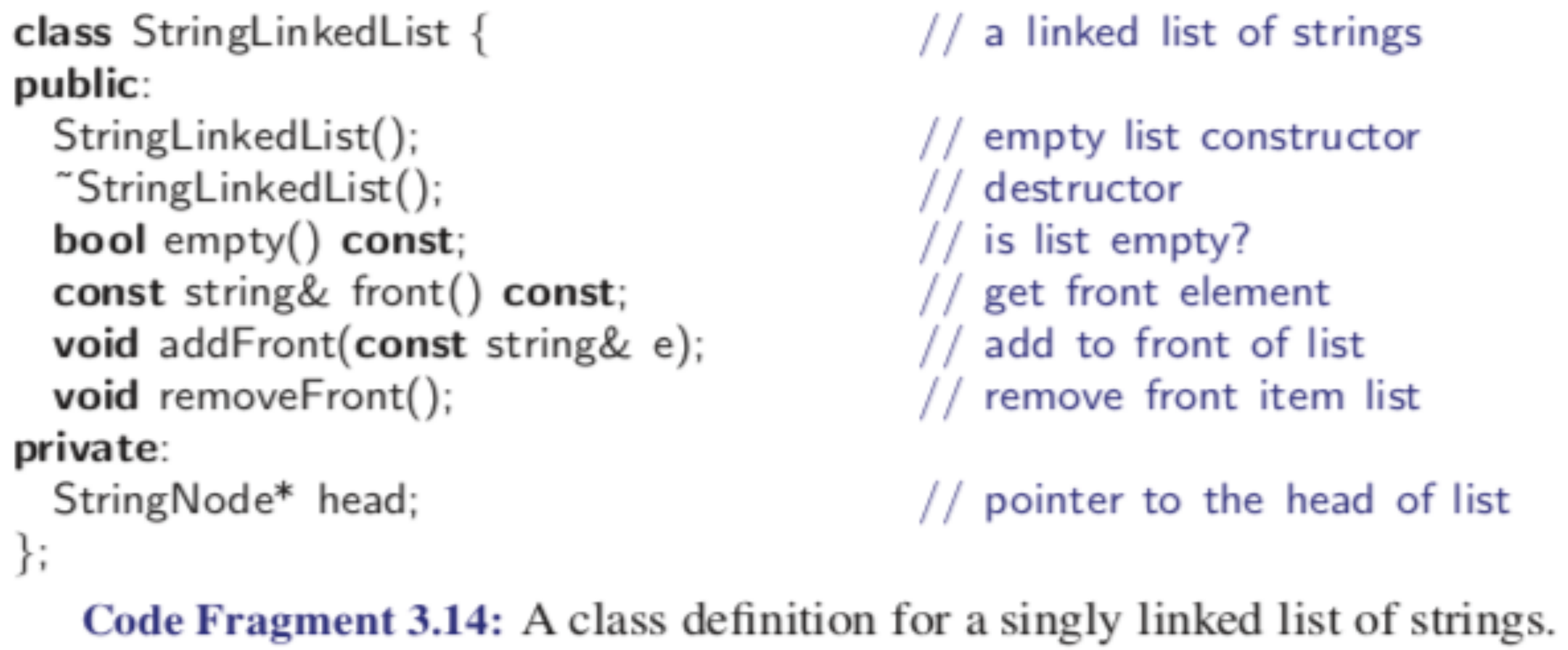


Figure 3.9: Example of a singly linked list of airport codes. The *next* pointers are shown as arrows. The null pointer is denoted by ∅.

Singly Linked List Class



```
class StringLinkedList {                    // a linked list of strings
public:
  StringLinkedList();                       // empty list constructor
  ~StringLinkedList();                      // destructor
  bool empty() const;                       // is list empty?
  const string& front() const;              // get front element
  void addFront(const string& e);           // add to front of list
  void removeFront();                       // remove front item list
private:
  StringNode* head;                         // pointer to the head of list
};
```

Code Fragment 3.14: A class definition for a singly linked list of strings.

StringLinkedList l1;
L1.addFront(“Abc”);
L1.removeFront();

StringNode n;

To access the data members one can use

n.elem   n.next

StringNode *head=NULL

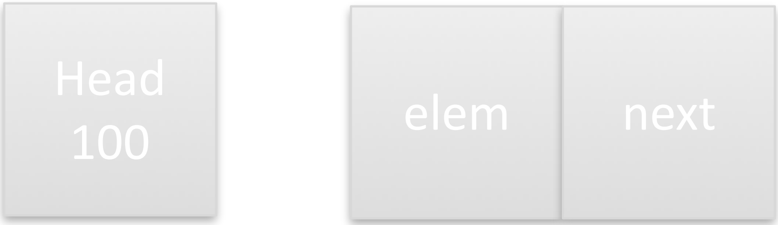
head =0

head = new StringNode()

To access the elements using pointer to node (head)

head->elem

head->next

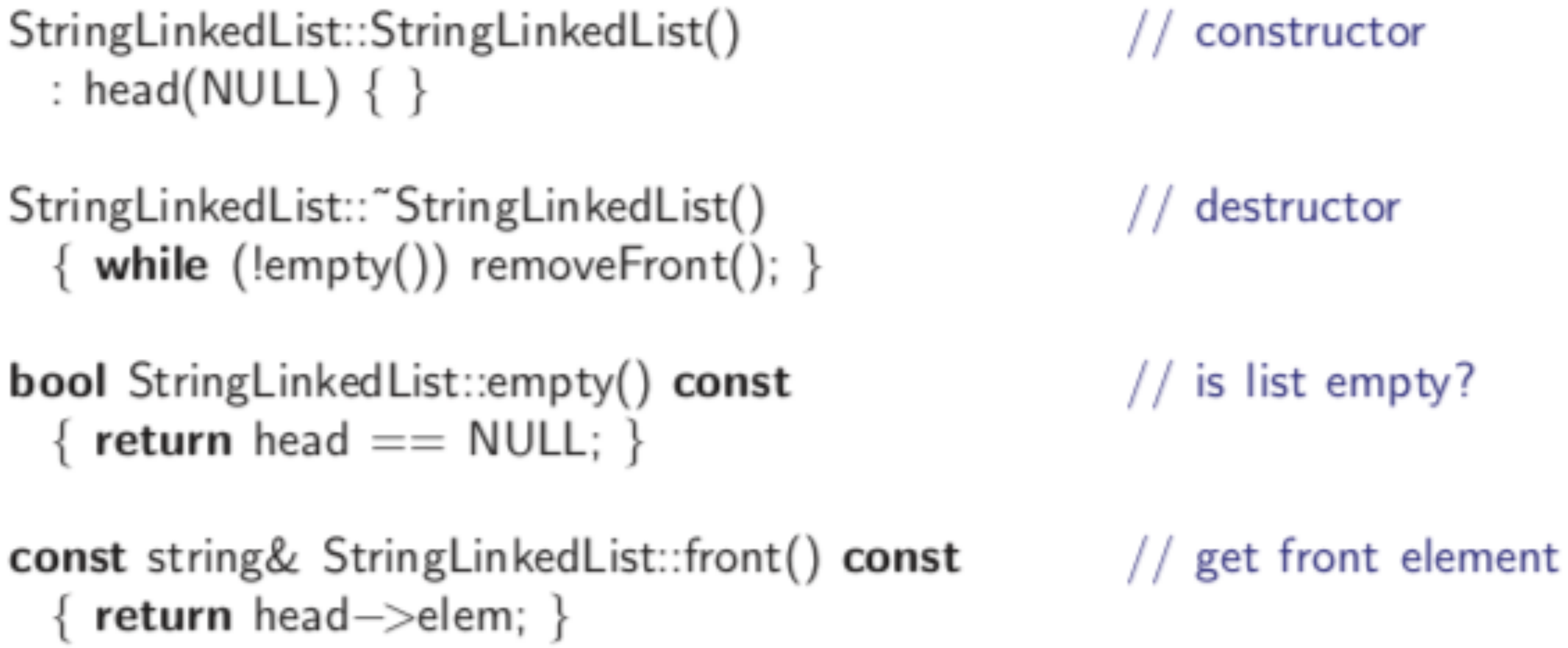


Head
100

elem

next

Singly linked list



```
StringLinkedList::StringLinkedList()                      // constructor
  : head(NULL) { }

StringLinkedList::~StringLinkedList()                     // destructor
  { while (!empty()) removeFront(); }

bool StringLinkedList::empty() const                      // is list empty?
  { return head == NULL; }

const string& StringLinkedList::front() const             // get front element
  { return head->elem; }
```

Destructor of linked list

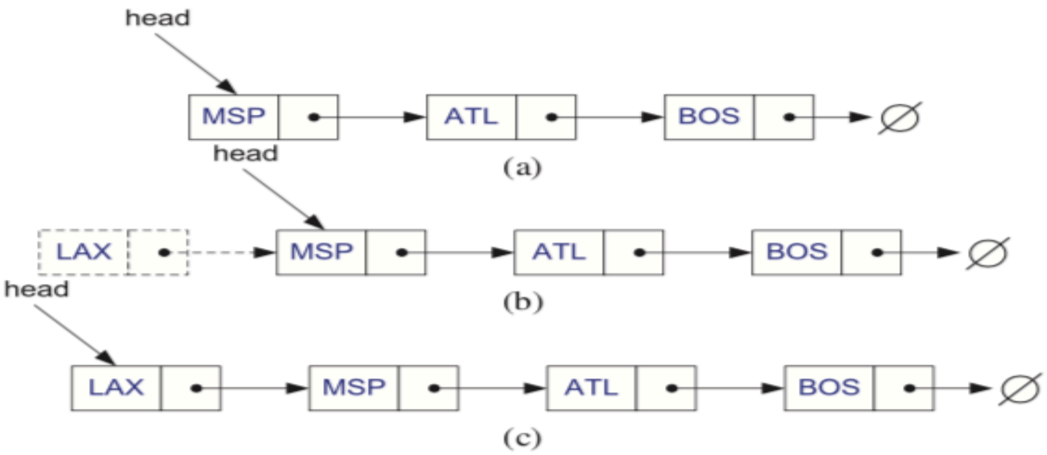
```
Node *p=NULL;
while(head!=NULL)
{
      p=head;
      head=head->next;
    delete p;
}
```

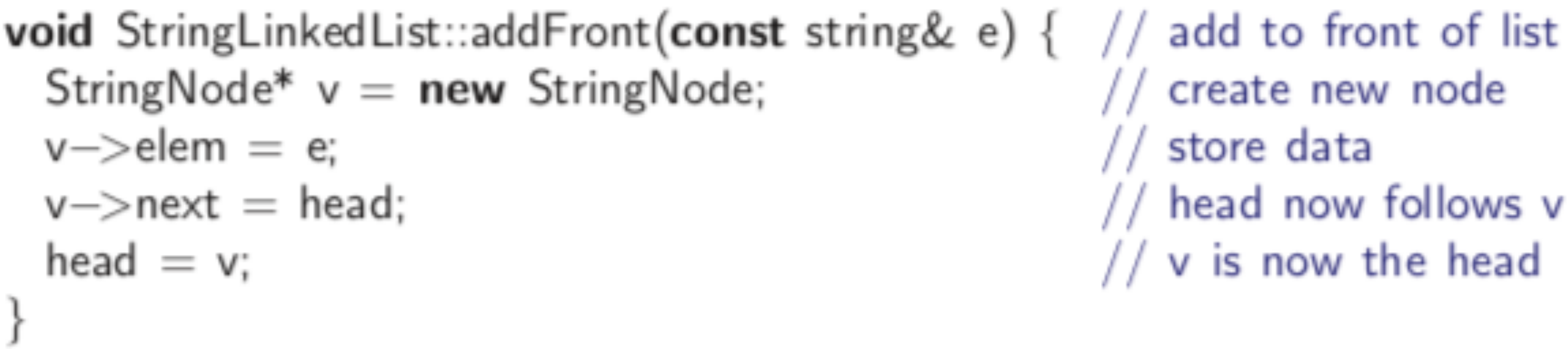
Copy Constructor

- Head   1->2->3->4
- l2.head   1->2->3->4

```
LinkedList l2(l1);      l2.head=l1.head;
LinkedList(const LinkedList &l1)
{
p=l1.head;
head=null;
while(p!=null)
{   newnode = new IntNode;
     newnode->elem=p->elem;
     newnode->next=null;
     if (head==NULL) { head=newnode;p1=head;}
     else
      {p1->next=newnode;
       p1=p1->next;
      }
    p=p->next;
}
```

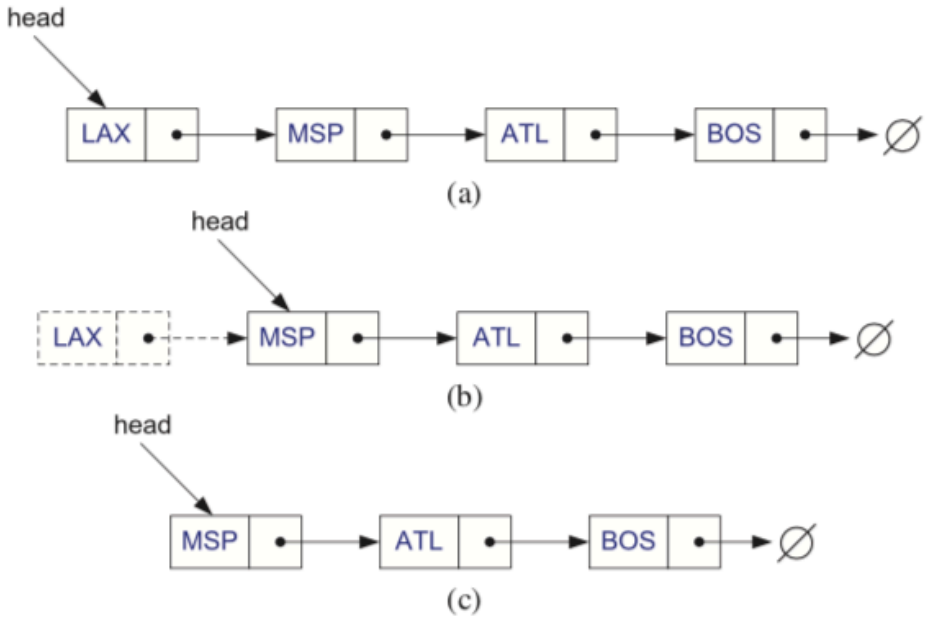
Adding the node at the front

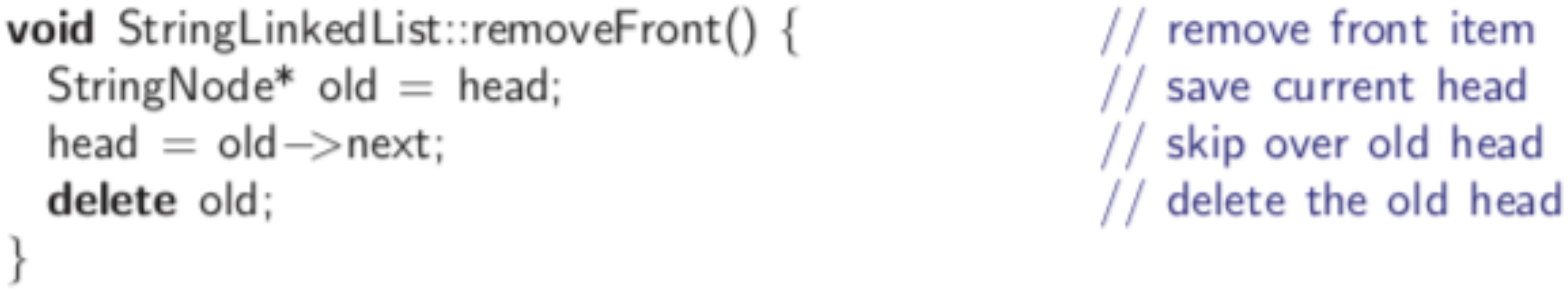




```
void StringLinkedList::addFront(const string& e) {  // add to front of list
  StringNode* v = new StringNode;                    // create new node
  v->elem = e;                                       // store data
  v->next = head;                                    // head now follows v
  head = v;                                          // v is now the head
}
```

Removing node from the front





```
void StringLinkedList::removeFront() {            // remove front item
  StringNode* old = head;                         // save current head
  head = old->next;                               // skip over old head
  delete old;                                     // delete the old head
}
```

Display the linked list

```
void IntLinkedList::display()
{
    if(head==NULL)  cout<< "Empty list ";
    else {
              IntNode* pt=head;
              while(pt!=NULL)
             {
                // Printing the current
                cout<<pt->elem;
               // moving to next element
                pt=pt->next;
              }
}
```

1->2->3->10

2->4->6->20

```
int IntLinkedList::count()
{
      int count=0;
      IntNode* ptr=head;
      while(ptr!=NULL)
       {
            count++;
            ptr=ptr->next;
      }
       return count;
}
```

Add at last (integer list)

```
void LinkedList::addLast(int x)
{     IntNode *p = new IntNode;
       p->elem=x;
       p->next=NULL;

      IntNode *temp=head;
      if (empty())  {head=p; return;}
      while(temp->next!=NULL)     temp=temp->next;
       temp->next= p;
}
```

Insert at a position

1-> 2-> 5-> 10

3 N

1-> 2-> 3-> 5-> 10

1-> 2-> 5-> 10

Insert at a position

- 1-> 2-> 5-> 10
- L1.insert(pos=3,value=6)
- 1->2->6->5->10

```
insert(int pos, int value)
{
       Intnode *p = new Intnode;
       p->elem=value;
       ptr = head;
       if (pos==1)
      {
             p->next=head;head=p;return;}
       count=1;
       while(ptr!=NULL && count<pos-1)
       {    ptr=ptr->next;
             count++;
       }
       if (ptr==NULL) throw “Invalid Position”;
       p->next= ptr->next;
        ptr->next = p;

}
```

Delete an element a given pos

- 1->2->6->5->10
- 1->2-> 5->10
- delete(pos=3)

```
delete(int pos)
{
     ptr = head;
     if (head==0)   throw “Empty List. Deletion not possible”;
     if (pos==1)
    {
         old=head;
        head= old->next; delete old;;return;}
     count=1;
     while(ptr!=NULL && count<pos-1)
     {    ptr=ptr->next;
          count++;
     }
     if (ptr==NULL) throw “Invalid Position”;
     intnode *old=ptr->next;
     p-tr>next= old->next;
     delete old;

}
```

Delete a given value

- 1->2->6->5->10
- delete(5)

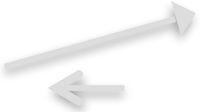
```
delete(int value)
{
     if (head==0)   throw “Empty List. Deletion not possible”;
      IntNode* ptr = head;
      IntNode* prev =null;
      while(ptr!=NULL&& ptr->elem!=value)
      {
         prev=ptr;
         ptr=ptr->next;
    }
      if (ptr==NULL) throw “Invalid value”;
      if (prev==NULL)  head=head->next;
      else  prev->next= ptr->next;
      delete ptr;

}
```

Class to represent linkedlist

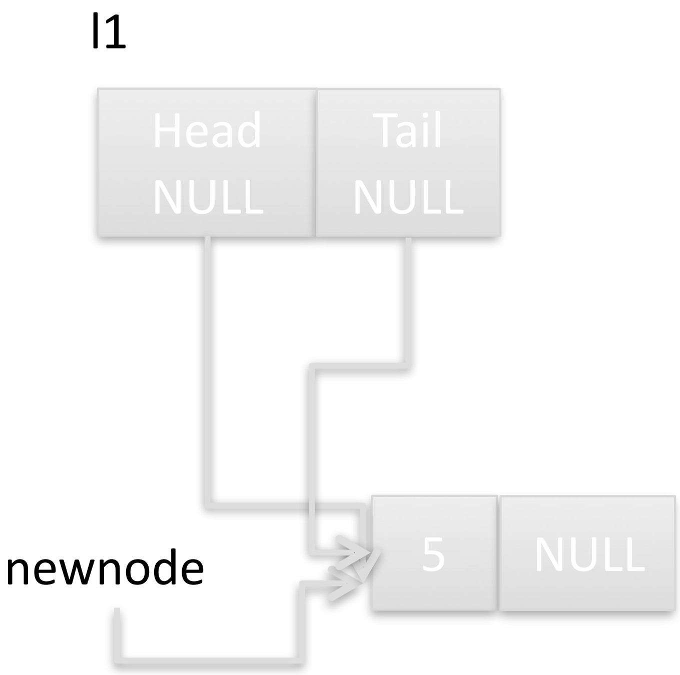
```
Class LinkedList
{       IntNode *head,*tail;
public:
        LinkedList(){ head=tail=0;}
        void addFront(int x);
        void addTail(int x);
        void removeTail();
        void removeHead();
        void display();
        int count();
        void delete(int pos)
        void delete(int value)
        void insert(pos, value);
}
```

- LinkedList l1;
- L1.addFront(2) head -> 2
- tail



- L1.addFront(3) 3->2

addFront



l1

Head
NULL

Tail
NULL

newnode

5

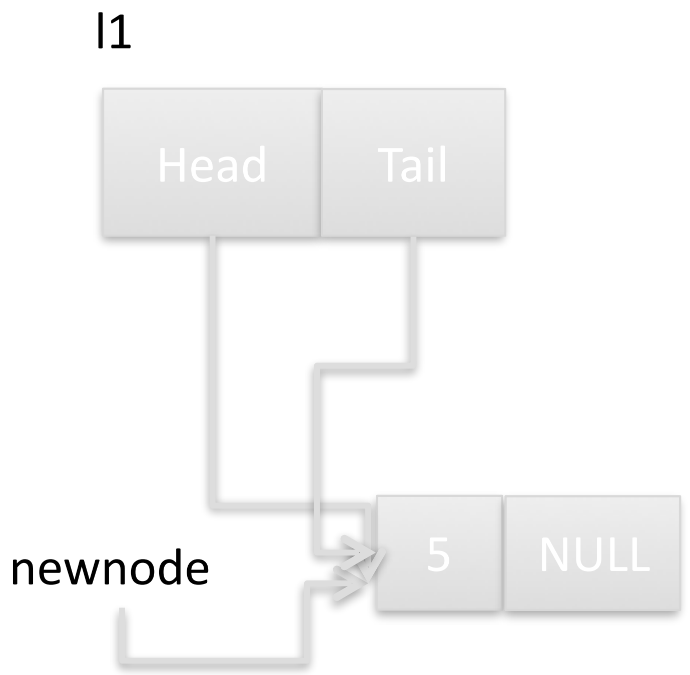
NULL

```
LinkedList l1;

 l1.addFront(5)

 IntNode *newnode = new IntNode;
 Newnode->elem = x;
 Newnode->next = NULL
 If (head==NULL)
       head = tail = newnode;
```

addFront cont.



l1

Head

Tail

5

NULL

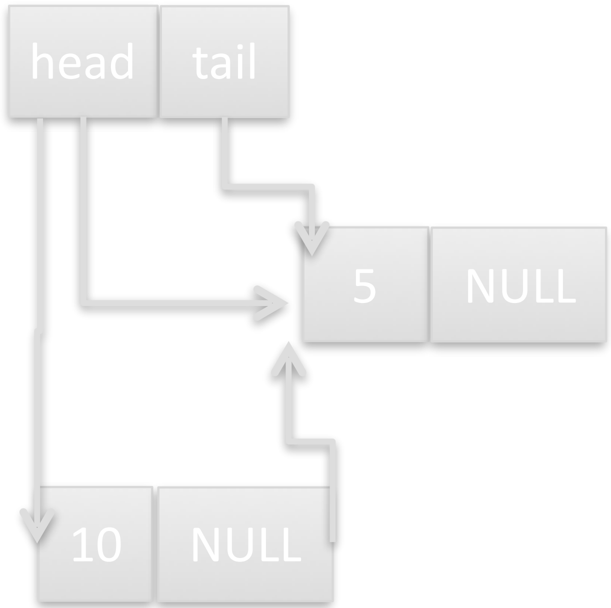
newnode

```
LinkedList l1;
```

```
l1.addFront(5)
```

```
IntNode *newnode = new IntNode;
Newnode->elem = x;
Newnode->next = NULL
If (head==NULL)
      head = tail = newnode;
```

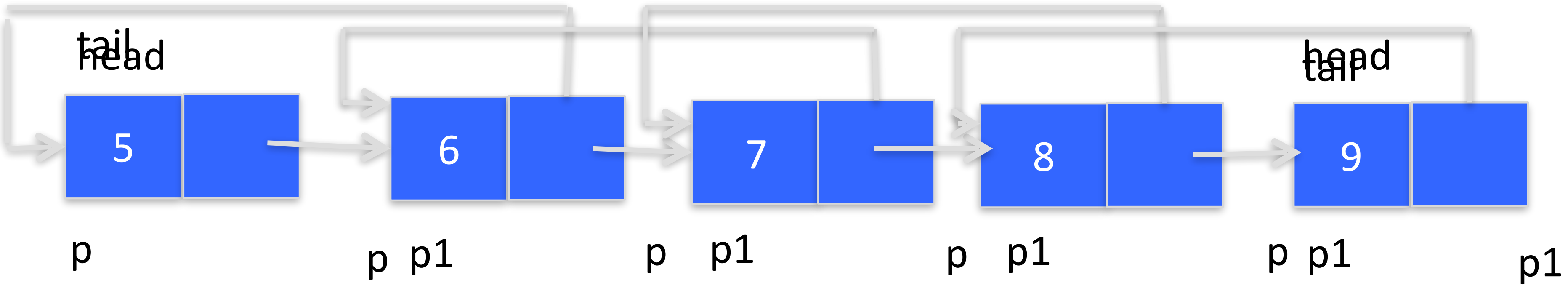
addFront



l1.addFront(5)

```
IntNode *newnode = new IntNode;
Newnode->elem = x;
Newnode->next = NULL
If (head==NULL)
       head = tail = newnode;
Else
    newnode->next=head;
    head=newnode;
```

Reversing a SLL



```
if (empty()) return;
p=head;
p1 = head->next;
if (p1==NULL) return;
while(p1!=NULL){
       p2=p1->next;
        p1->next =p;
        p = p1;
        p1 = p2;
}
head->next=NULL;
tail = head;
head= p;
return;
```