Mrudang Mehta
Associate Professor
Department of Computer Engineering,
Faculty of Technology,
Dharminsh Desai University, Nadiad, Gujarat, India
mrudang.mehta@gmail.com

Chapter 4

Linked List

Book Reference:

Fundamentals of Data Structures in C++ by Horowitz, Sahni, Mehta, Galgotia Publisher, 2007 or later version

Other Reference Book

Data Structures Using C and C++ (2nd Edition)

By Langsam, Augenstein, Tenenbaum

Linked List (Chain)

- Linear list.
- Each element is stored in a node.
- Nodes are linked together using pointers.
- Ordered List:
  - Sequential Mapping(array), Insert and Delete operation become expensive
  - BAT,EAT,HAT, MAT elements inserted in sequential mapping (then insert CAT,FAT,JAT,LAT)
  - Which steps are required? (shifting??)

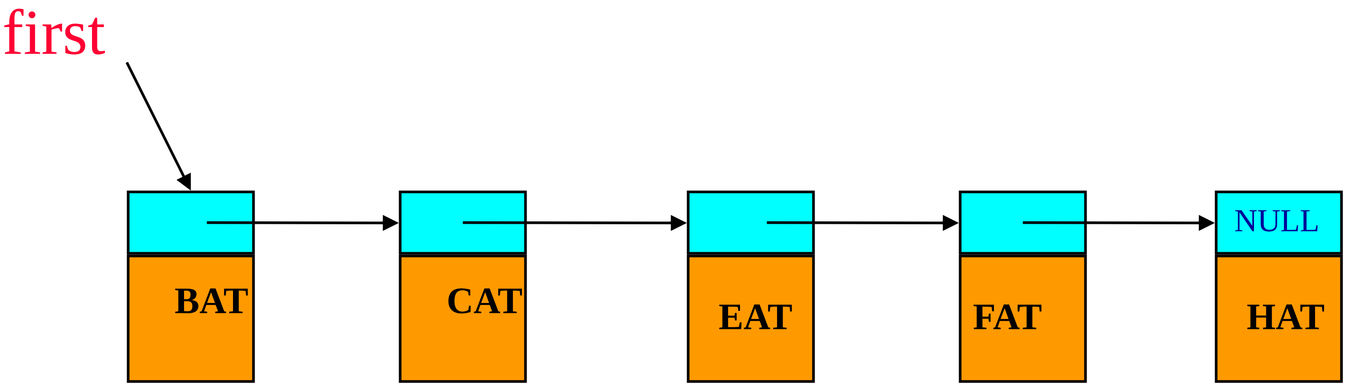
Can you represent Linked List using Array?

Linked Representation

- Items may be placed anywhere in memory
- Each data item is associated with a link field which contains address to next element. (*Array Representation of* **Linked List**)

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
HAT		CAT	EAT		BAT			FAT			JAT		
12		4	9		3			1			-1		

The Class List



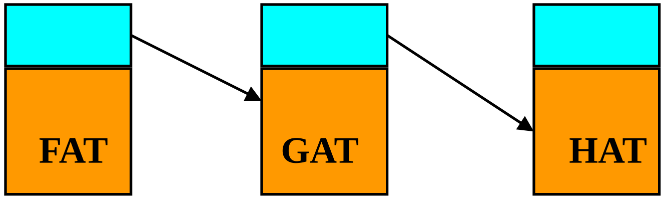
Use ListNode

🟦 link (datatype ChainNode<T>*)

🟧 data (datatype T)

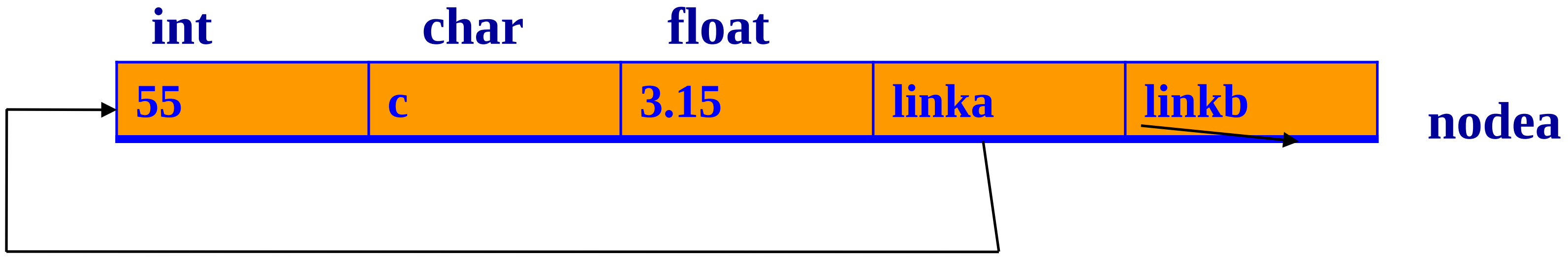
Insert an element

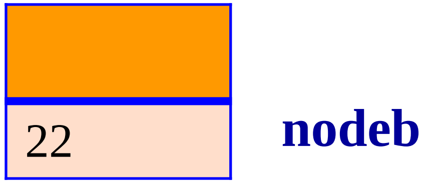
- Get a node that is currently unused. Let its address be x.
- Set data field of node to GAT
- Set link field of node to link pointing *next to FAT*
- *Link of FAT becomes x*.



Linked List using Class

- Basic building block of list →Node



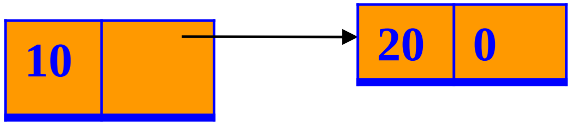


```
class nodea{
  int d1;char d2; float d3;
   nodea *linka;
   nodeb *linkb;
};
```

```
class nodeb{
 int d;
 nodeb *link;
};
```

```
class TLNL;   //three letter node list
class TLN {      //three letter node
      friend class TLNL;
private:
        char data[3]; TLN *link;
};
class TLNL{
  public: //list manipulation functions
  private: TLN *first;
}
```

Creating linked list



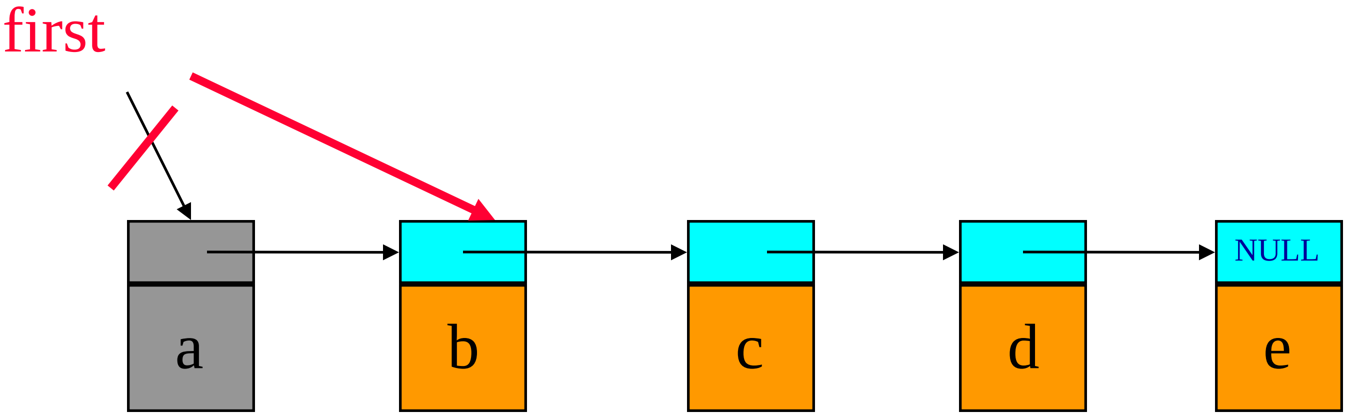
```
class ListNode{
  int data; ListNode *link;
};
void List::Create2() {  first = new ListNode(10);
first->link= new ListNode(20);
}
ListNode::ListNode(int element=0)
{       data= element; link=0;}
```

- **Insert a node after a given node ???**

```
void List:: Insert50(ListNode *x)
{   ListNode  *t = new ListNode(50);
    if(!first){  // empty list
        first=t;
        return;
     }
     t->link = x->link;
     x->link=t;
}
```

- **Delete a node given by x whose previous node is y ???**

Delete An Element



delete(0)

deleteNode = first;

first = first–>link;

delete deleteNode;

Delete Operation

```
void List:: Delete(ListNode *x, ListNode *y)
{
                          if(!y){
                         first = first->link;
                         }
                         else{
                         y->link = x->link;
                         }
                         delete x;
}
```

Write an algorithm for

(a) To check that the current node(element) in list is not null

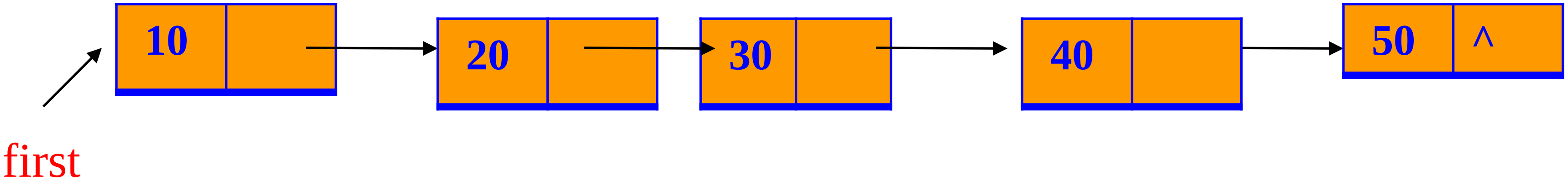
(b) To check that the next node in list is not null

(c) Return a pointer to the first element of list

(d) Return a pointer to the next element of list

(e) To compute the sum of elements

(f) Attaching a node to the end of a list



List has member first pointer;

We can use current=first; current=current->link;

(a) *To check that the current element in list is not null*

```
         boolean NotNull() {
          if(current)
                    return TRUE;
          else
                    return FALSE;
         }
```

(b) *To check that the next element in list is not null*

```
boolean NextNotNull() {
          if(current && current->link)
                    return TRUE;
          else
                    return FALSE;
     }
```

(c) Return a pointer to the first element of list

```
        (type)  FirstElement()  {
           if(first)
                 return &list.first->data;
           else
                 return 0;
          }
```

(d) Return a pointer to the next element of list

```
  (type)  NextNode(){
         if(current) {
                 current=current->link;
                 if(current)
                    return &current->data;
              }
            else
                 return 0;
     }
```

(e) To compute the sum of elements

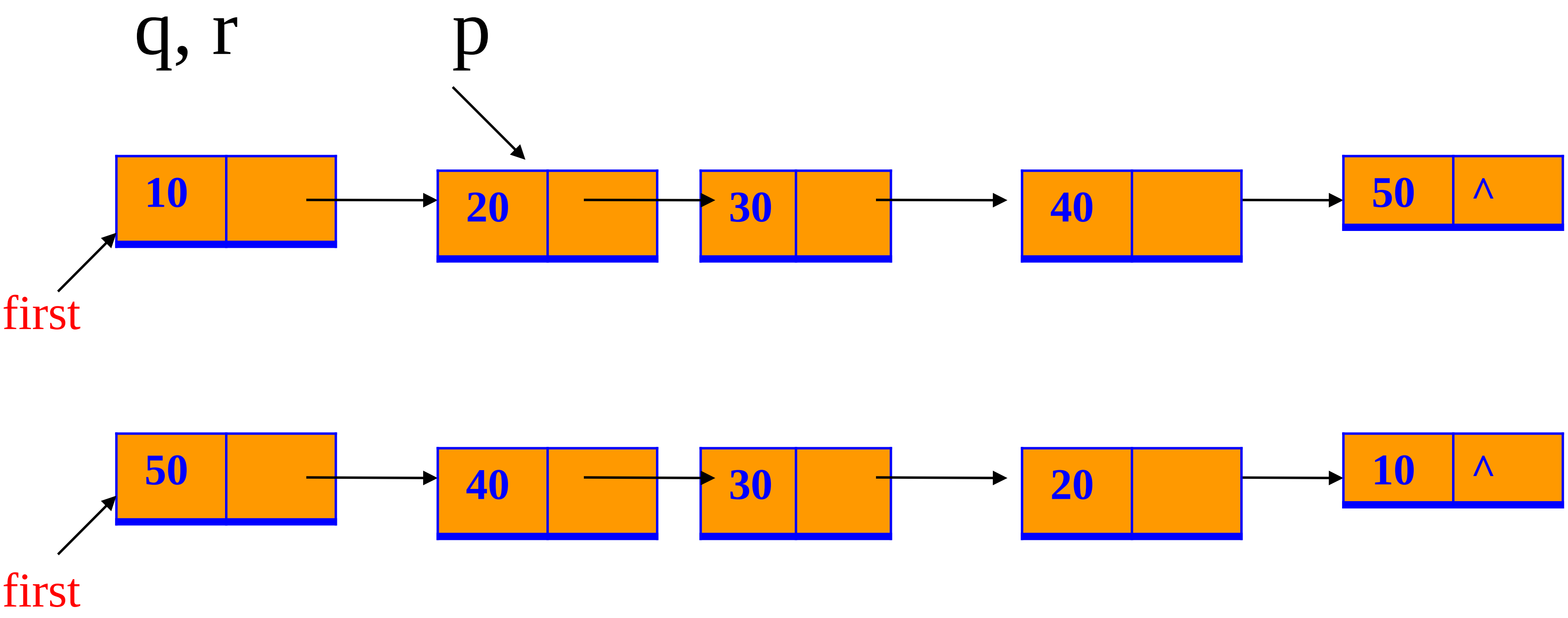
```
int sum(List &list1)
{
        //check empty list
if(!list1.NotNull())
            return 0;

int retValue = *list1.FirstNode();

while(list1.NextNotNull()){
            retValue += *list1.NextNode();
}

return retValue;
}
```

Write an Algorithm to Invert a List



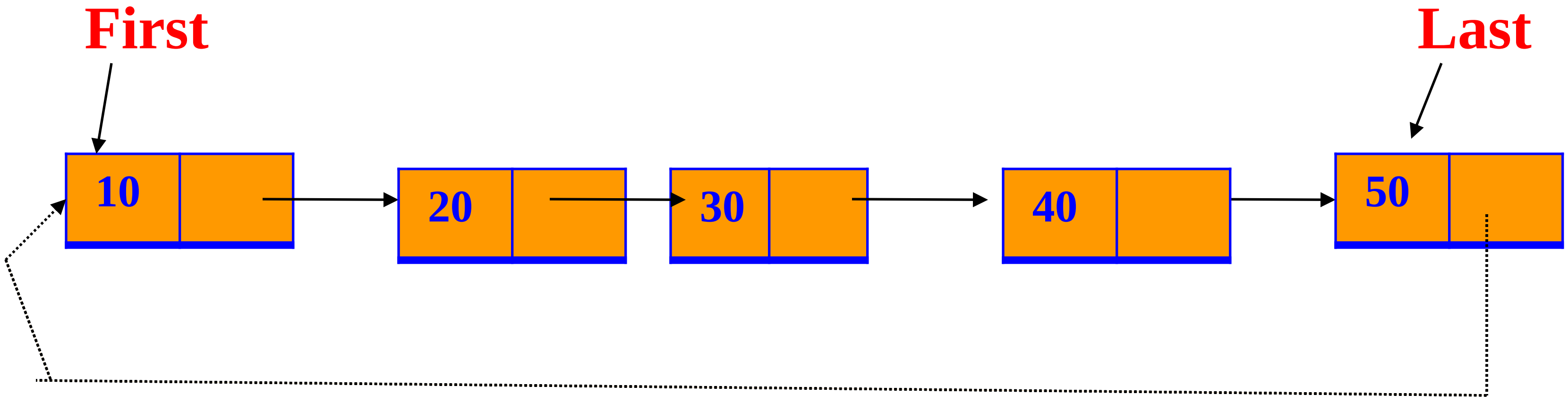
```
void list:: Invert() {
            ListNode *p = first; *q=0; //q trails p
            while(p){
                  ListNode *r=q; q=p; //r trails q
                  p=p->link;
                  q->link=r;
            }
         first=q;
}
```

Write an algorithm to Concatenate Two Lists

```
void List:: Concatenate(List b)
{                   if(!first) {
                              first = b.first;
                              return;
                    }
                    if(b.first) {
                    for(ListNode *p=first; p->link;p=p->link)
                              p->link=b.first;
                    }
}
```

Circular Lists

- Singly linked list : last node link field is null
- In, Circular List, last node's link points to first node

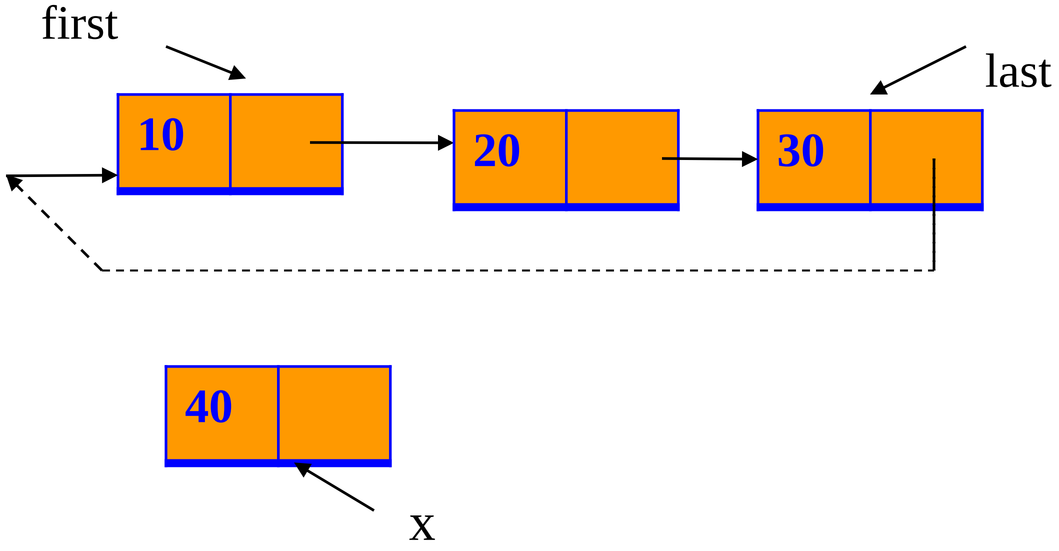


- To check whether current node is last node
  - In singly linked list (?)
  - In Circular linked list (?)
- In singly linked list
  - *current->link==0*
- In circular linked list
  - *current->link==first*

Insert operation in Circular List

- **We assume,**
  - **ListNode class for list node**
  - **CircList class for circular lists**
- **void CircList::InsertFront(ListNode *x)**
- **{**

```
        //insert the node pointed at by x at the front of the circular list, we have last pointer
        if( !last) { //empty list
                last= x;
                x->link =x;
        }
        else{
                x->link=last->link;
                last->link=x;
        }
}
```



Write an algorithm to,

Insert node (element) at the last

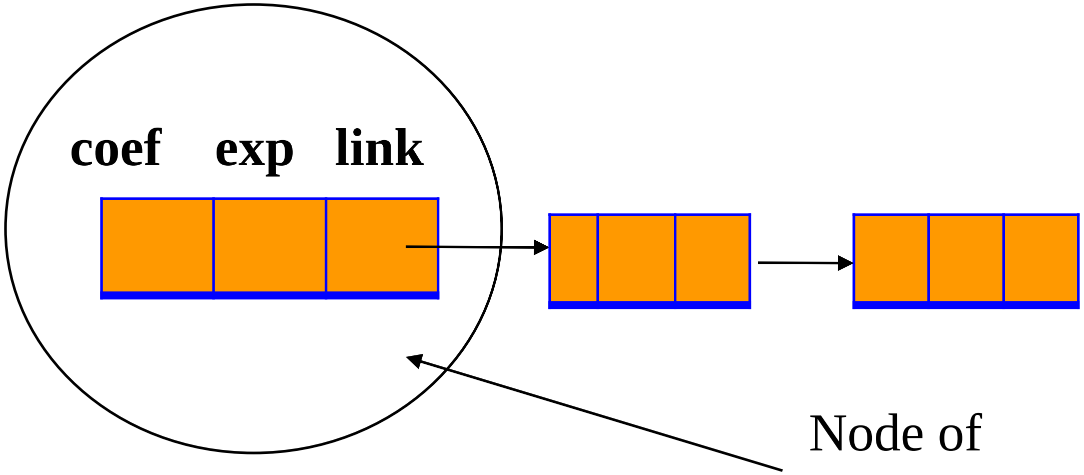
Insert a node after given node

Delete a node from front

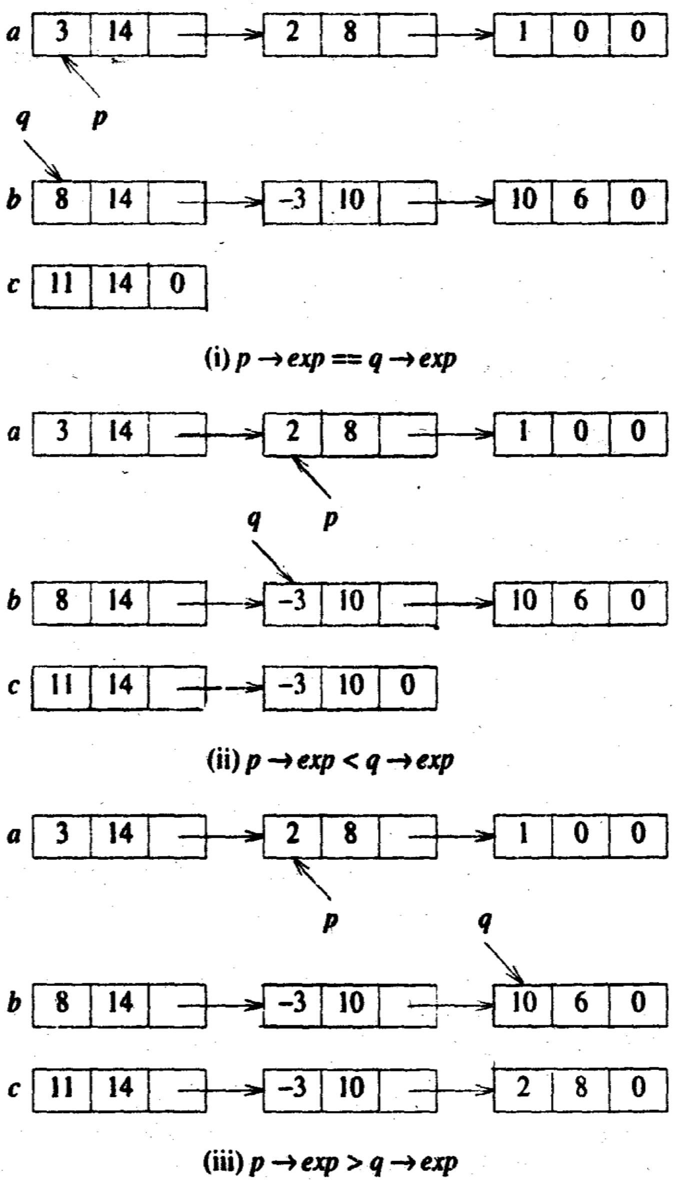
Delete a node at the last

Delete a given node

Polynomial Addition using Linked List

- Polynomial Representation
  - $A(x) = a_m x^{em} + a_{m-1} x^{em-1} + \ldots\ldots\ldots a_0 x^{e0}$
- 

Node of Polynomial List



(i) $p \rightarrow exp == q \rightarrow exp$

(ii) $p \rightarrow exp < q \rightarrow exp$

(iii) $p \rightarrow exp > q \rightarrow exp$

Figure 4.19: Generating the first three terms of $c = a + b$

From Horowitz, Sahni and Mehta's Book

Using ListNode, List and ListIterator classes

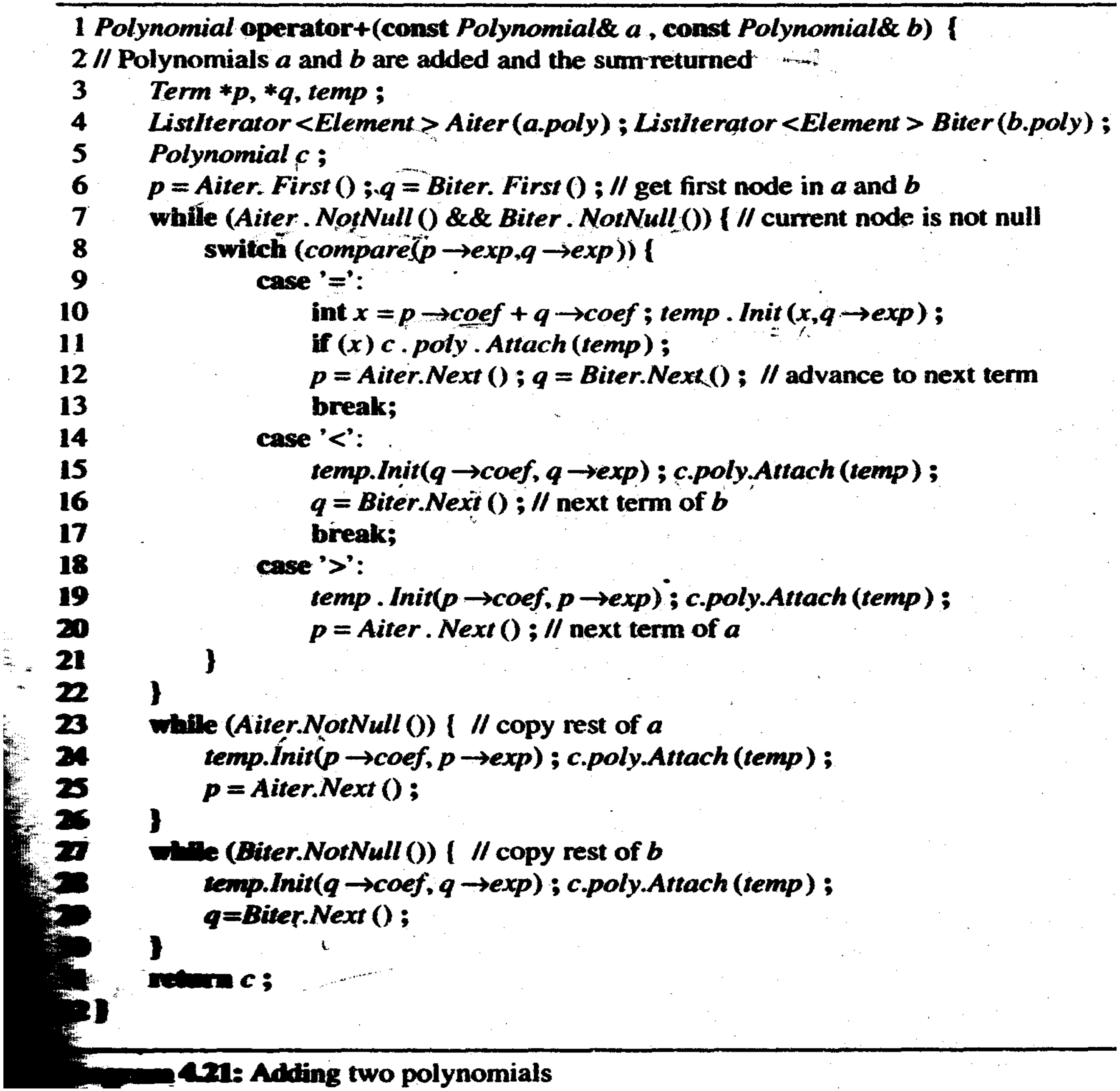
List and ListIterator are friend of ListNode

ListIterator is friend of List

ListIterator’s private members are objects of ListNode and List class

Refer Source Code Poly.cpp

```
struct Term
// all members of Term are public by default
{ int coef; int exp;
    void Init(int c, int e){ coef=c; exp=e;}
 };
class Polynomial {
    friend Polynomial operator+(const Polynomial&, const
    Polynomial&);
private:
    List poly;
};
```



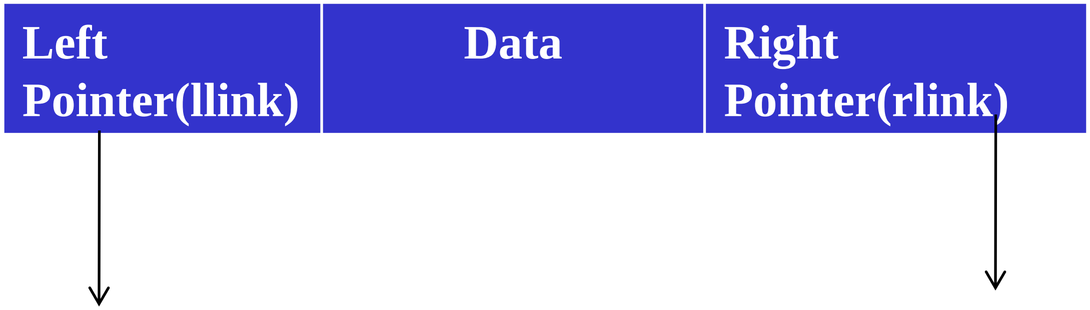
```
Polynomial operator+(const Polynomial& a , const Polynomial& b) {
// Polynomials a and b are added and the sum returned
    Term *p, *q, temp ;
    ListIterator <Element > Aiter (a.poly) ; ListIterator <Element > Biter (b.poly) ;
    Polynomial c ;
    p = Aiter. First () ; q = Biter. First () ; // get first node in a and b
    while (Aiter . NotNull () && Biter . NotNull ()) { // current node is not null
        switch (compare(p →exp,q →exp)) {
            case '=':
                int x = p →coef + q →coef ; temp . Init (x,q →exp) ;
                if (x) c . poly . Attach (temp) ;
                p = Aiter.Next () ; q = Biter.Next () ; // advance to next term
                break;
            case '<':
                temp.Init(q →coef, q →exp) ; c.poly.Attach (temp) ;
                q = Biter.Next () ; // next term of b
                break;
            case '>':
                temp . Init(p →coef, p →exp) ; c.poly.Attach (temp) ;
                p = Aiter . Next () ; // next term of a
        }
    }
    while (Aiter.NotNull ()) { // copy rest of a
        temp.Init(p →coef, p →exp) ; c.poly.Attach (temp) ;
        p = Aiter.Next () ;
    }
    while (Biter.NotNull ()) { // copy rest of b
        temp.Init(q →coef, q →exp) ; c.poly.Attach (temp) ;
        q=Biter.Next () ;
    }
    return c ;
}
```

4.21: Adding two polynomials

From Horowitz, Sahni and Mehta’s Book

Doubly Linked List

- Singly linked list(Compare with array)
- Circular Linked list(compare with singly list-Limitation ??)
- Doubly Linked list:
  - Minimum three fields in a db list node
    - Data, Left pointer and Right pointer

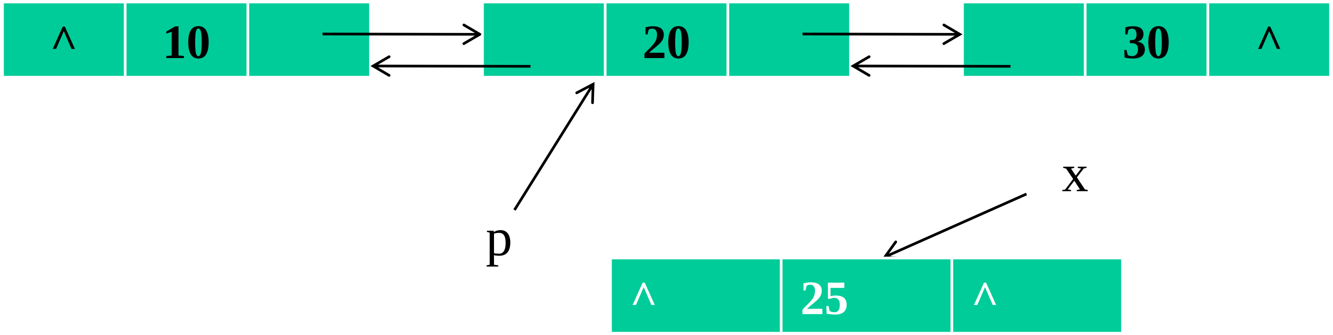


Left Pointer(llink)

Data

Right Pointer(rlink)

Insert Operation on doubly linked list



∧ 10

20

30 ∧

p

x

∧ 25 ∧

Inserting a node

```
void DbList:: InsertNode(DbListNode *p, DbListNode *x)
{  //p is pointer to given node
    //x is pointer to the node we want to insert
    x->llink=p;   // x’s left pointer points to p
    x->rlink=p->rlink;      // x’s right pointer points to where p’s right pointer points
    p->rlink->llink=x; // previous right node of p must now point to x
    p->rlink=x; // p’s right pointer must now point to x


}
```

Deleting a node

```
void DbList:: Delete( DbListNode *x)
{ // delete a node pointed by x
    if(x==first)
          cout<<“delete not allowed\n”;
     else{
          x->llink->rlink = x->rlink;
          x->rlink->llink = x->llink;
          delete x;
     }
}
```

Exercise

- Perform following operations on doubly linked list
  - Insert before
  - Insert in the middle
  - Delete after
  - Delete before
  - Delete a node having specific data value