Doubly Linked List

Doubly-Linked Lists

- It is a way of going *both* directions in a linked list, *forward* and *reverse*.
- Many applications require a quick access to the *predecessor* node of some node in list.

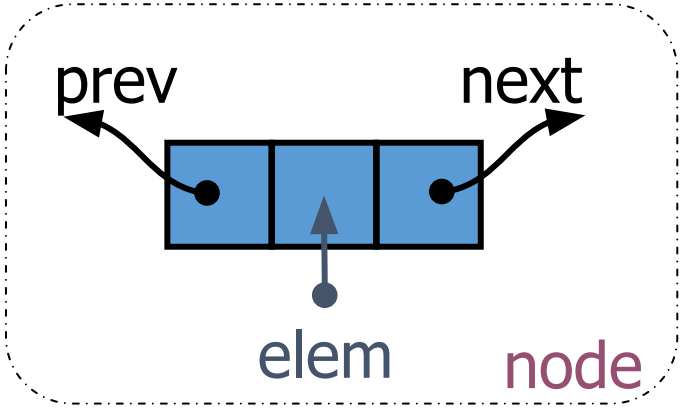
Advantages over Singly-linked Lists

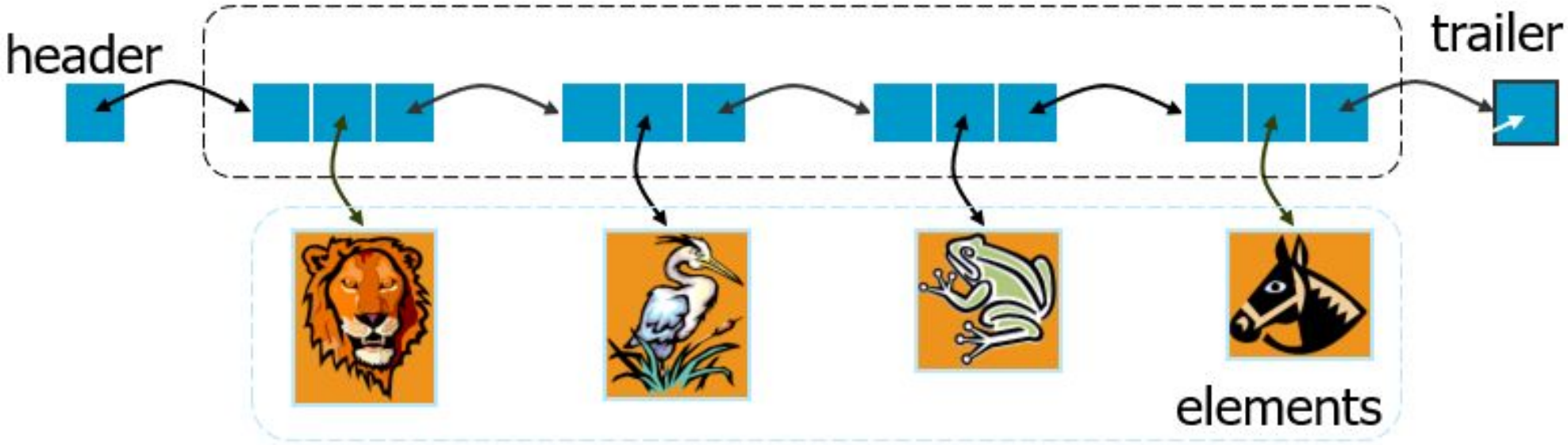
- Quick update operations:
  such as: insertions, deletions at *both* ends (head and tail), and also at the middle of the list.

- A node in a doubly-linked list store two references:

  - A *next* link; that points to the next node in the list, and
  - A *prev* link; that points to the previous node in the list.

- Nodes structure include
- **element**
- **link to the previous node**
- **link to the next node**
- Special trailer and header nodes





Code Snippet:

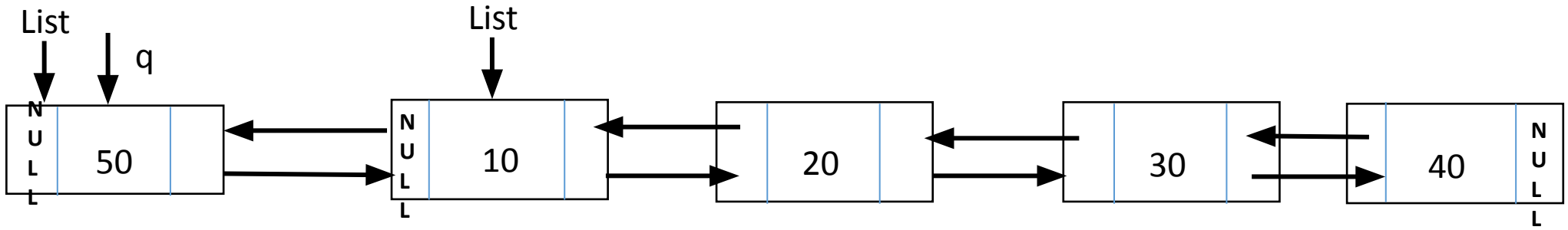
```
struct Node{
    data_type data; //Any data type
    struct Node *prev;      //Pointer to the struct Node
    struct Node *next;
};

int main()
{
    Node *header, *trailer;
}
```

Insertion in doubly linked list

There are four situations for inserting a node in Doubly linked list.

- Insert node in the beginning
- Insert node in the end
- Inserting node towards right of a specific node
- Inserting node towards left of a specific node

Insert node in the beginning



Algorithm:

1.q = getnode();

2.info(q) = x;

3. If(list!=NULL)/*If list is not empty*/

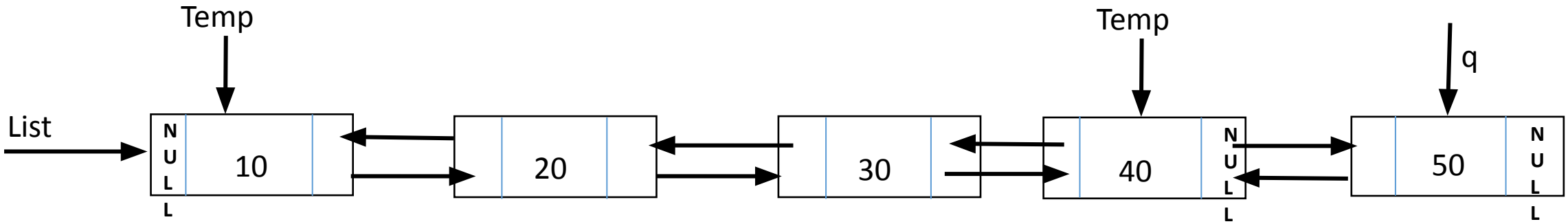
4. list->left = q;

5. q->left = NULL;

6. q->right = list;

7. list = q;

Insert node in the end



Algorithm:

1.q = getnode();

2.info(q) = x;

3.right(q) = NULL;

4. temp = list;

5.if(list==NULL)/*If list is empty*/

i)list = q;

ii)q->left=NULL;

6.else

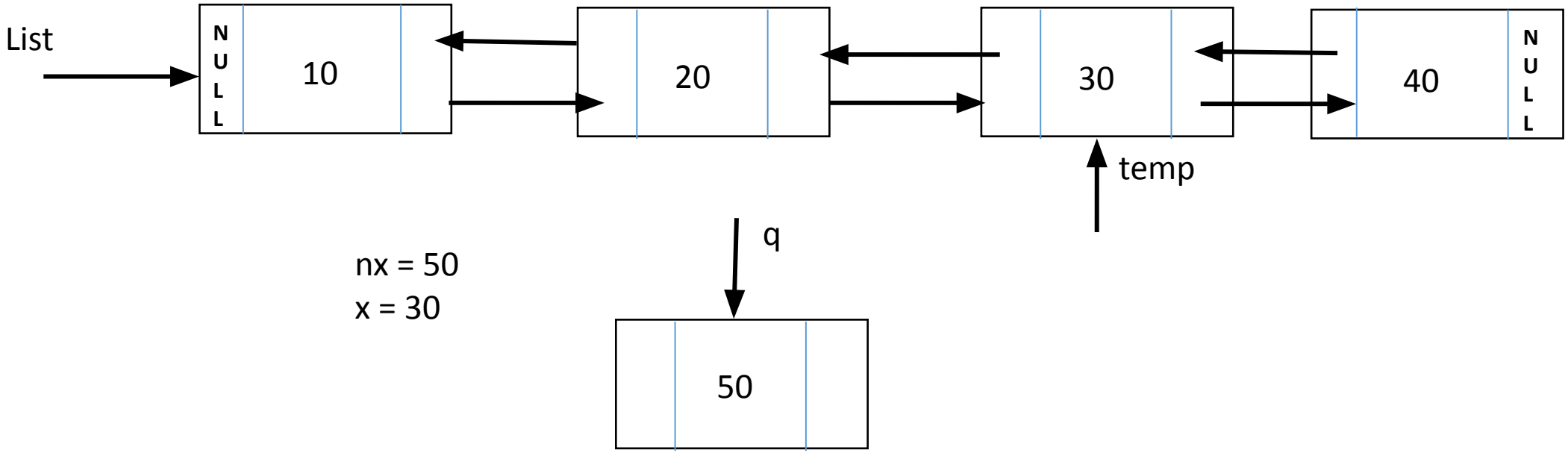
i)while(temp->right!=NULL)repeat following step

temp=temp->right

ii)temp->right=q;

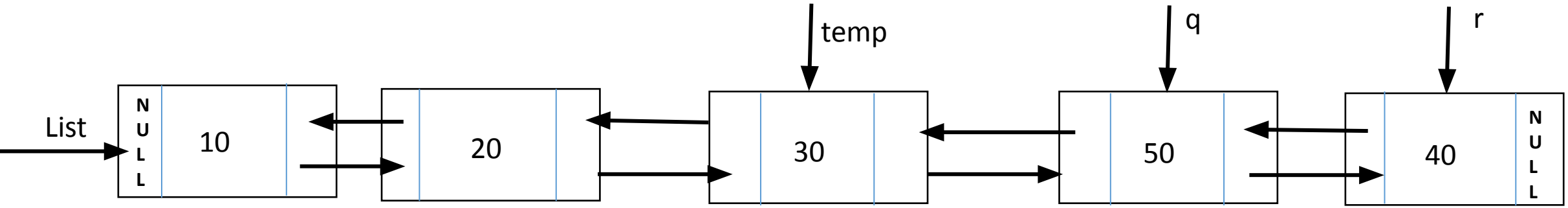
iii)q->left=temp;

Insert node towards right of a specific node



Before inserting new node containing 50 towards right of node containing 30 in its info field

After inserting new node containing 50 towards right of node containing 30 in its info field



Algorithm:

1.temp=list;

2.do

if(info(temp)==x) /*node found*/

break;

else

temp=temp->right;

while(temp!=NULL);

3. if(temp==NULL)

printf(“Element not found”)

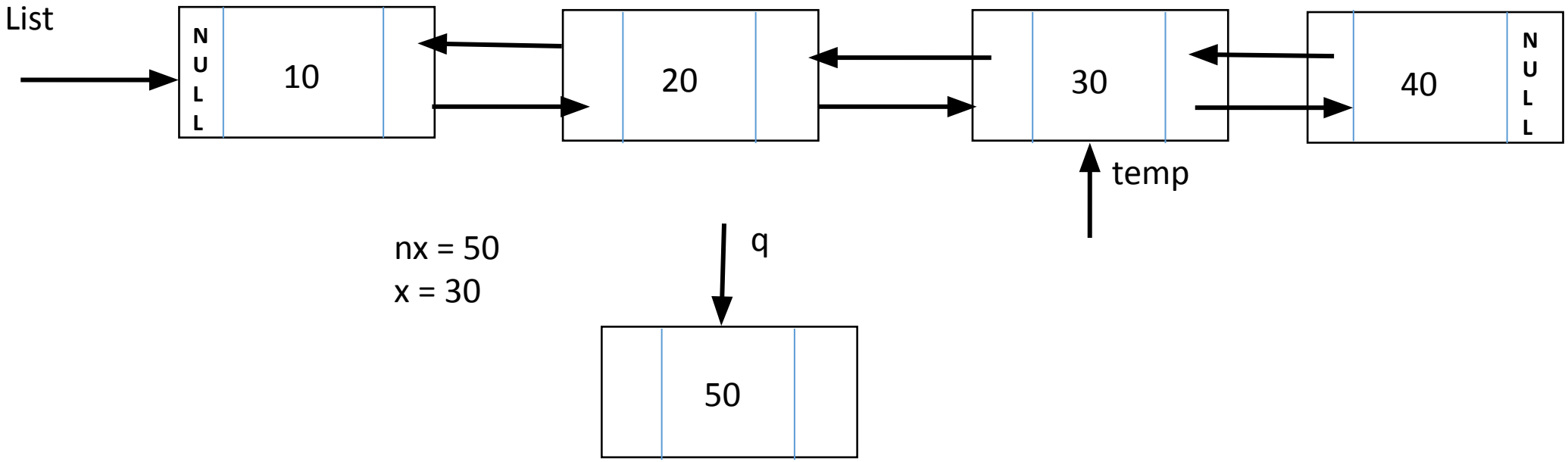
return;

4. q=getnode();
5. info(q)=nx;
6. r=right(temp);
7. If(r!=NULL)

left(r)=q;

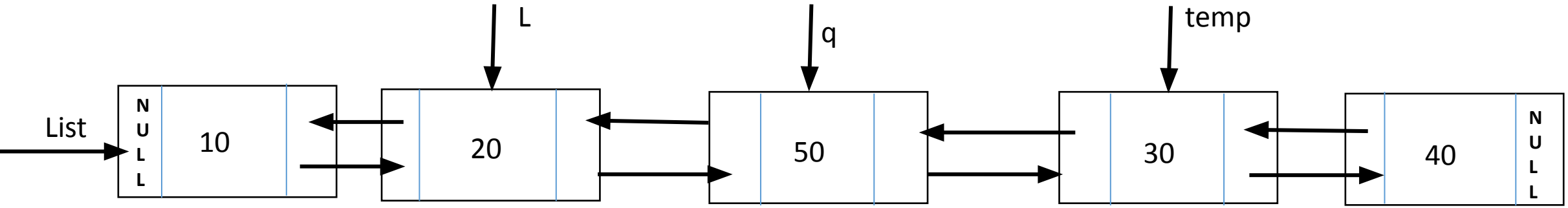
8. right(q)=r;
9. left(q)=temp;
10. right(temp)=q;

Insert node towards left of a specific node



Before inserting new node containing 50 towards left of node containing 30 in its info field

After inserting new node containing 50 towards left of node containing 30 in its info field



Algorithm:

1.temp=list;

2.do

if(info(temp)==x) /*node found*/

break;

else

temp=temp->right;

while(temp!=NULL);

3. if(temp==NULL)

printf(“Element not found”)

return;

4. q=getnode();

5. info(q)=nx;

6. L=left(temp);

7. If(L!=NULL)

right(L)=q;

8. right(q)=temp;

9. left(q)=L;

10. left(temp)=q;

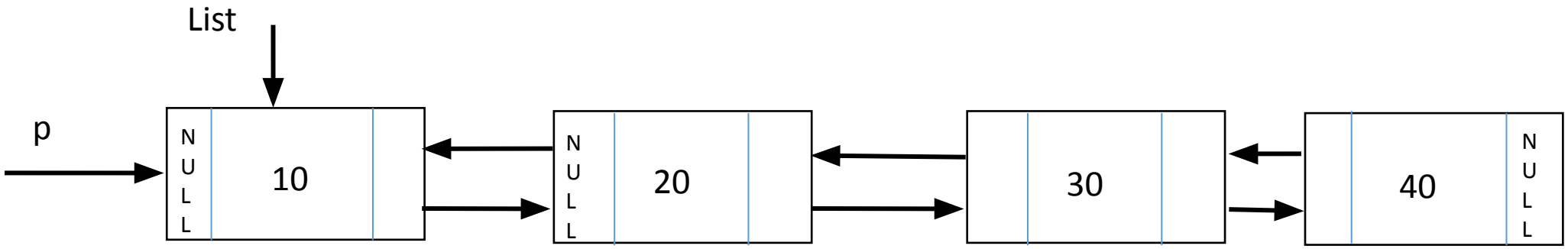
11. if(temp==list)

list=q;

Deletion in doubly linked list

- Delete beginning node
- Delete ending node
- Deleting a middle node

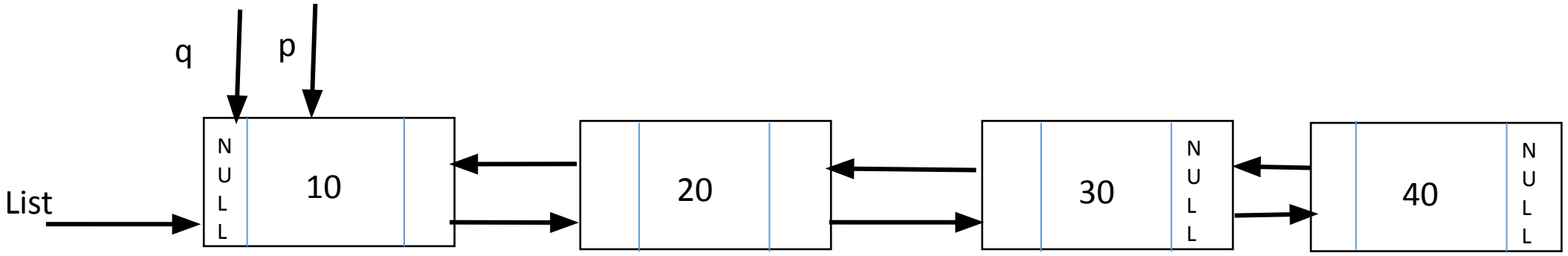
Delete beginning node




```
p= List
if(List==NULL)
    printf("List is empty")
```

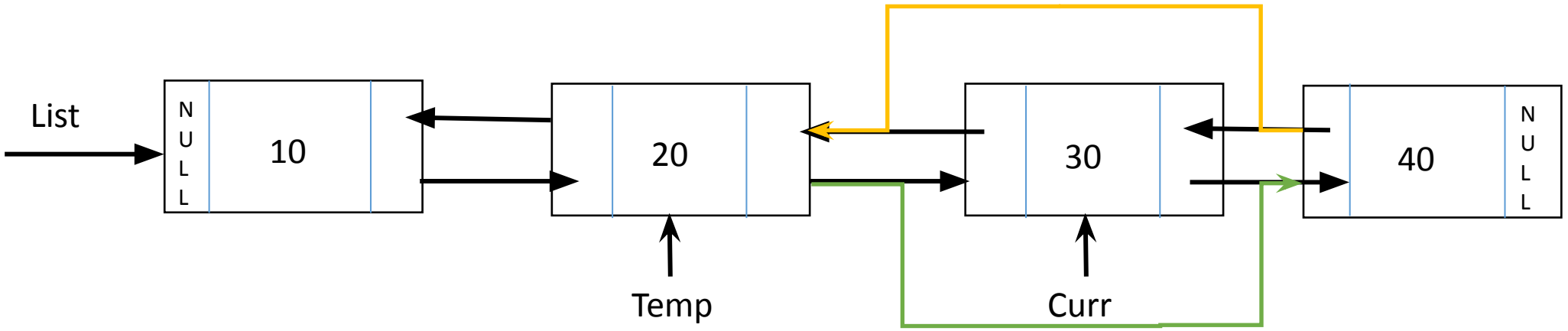
```
x=info(p)
List = List->next
List->prev = null
freenode(p)
```

Delete ending node




```
q=List
if(List==NULL)
      print”List is empty”
while(right(q)!=Null)
      p=q
      q=right(q)
right(p)=Null
Freenode(      q)
```

Delete middle node



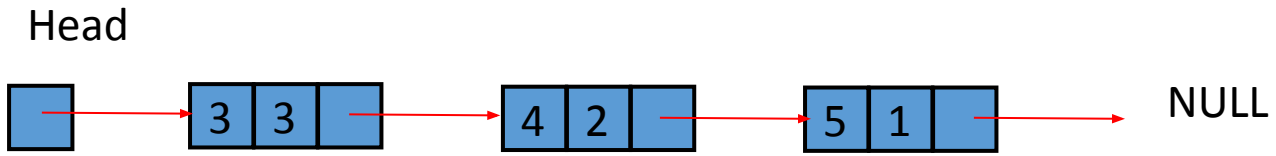
Temp->right = Curr->right

left(right(Curr)=Temp

Applications of Link List:

1)Polynomial Addition and Subtraction

- Consist of coefficient and Exponent.
- One expression can be represented as link list
- **3X^3 + 4x^2 + 5X**



Each Term can be represented inside a node.

```
struct Node
{
    int coff, exp;
    struct Node *next;
};
```

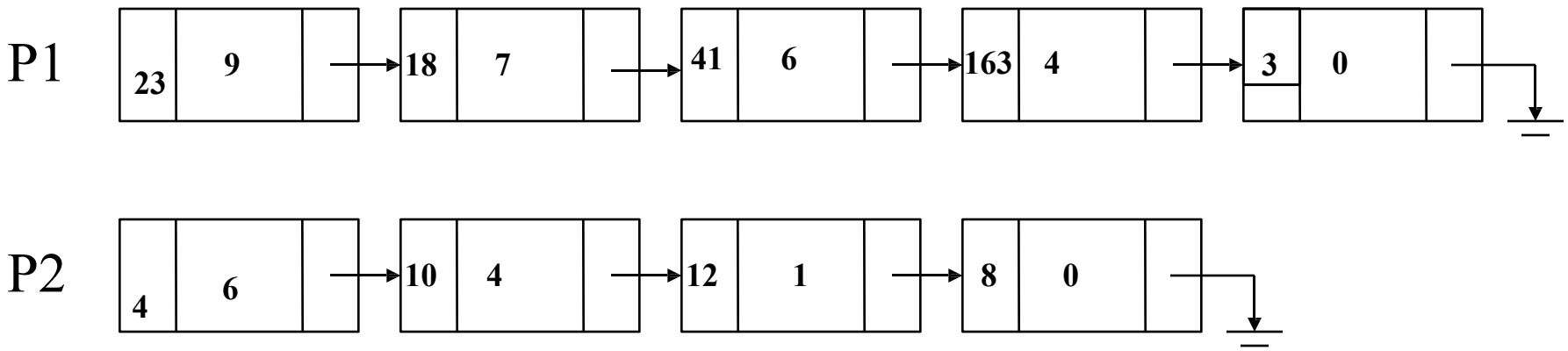
Steps to find Addition/Subtraction:

- The result will be stored in the third link list say L3.
- Start with highest power of any polynomial.
- If no node is having same exponent, simply append in L3.
- When exponents are matched, add the coefficient and store in L3.
- If one list gets exhausted earlier and the other list still contains some lower order terms, then simply append the remaining terms to the new list.

Polynomial Representation

- Linked list Implementation:

- $p1(x) = 23x^9 + 18x^7 + 41x^6 + 163x^4 + 3$
- $p2(x) = 4x^6 + 10x^4 + 12x + 8$

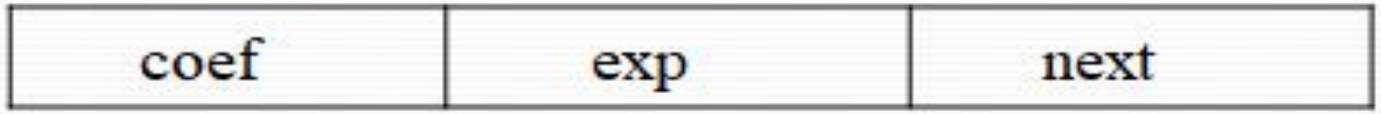


Polynomials

$$A(x) = a_{m-1}x^{e_{m-1}} + a_{m-2}x^{e_{m-2}} + \cdots + a_0 x^{e_0}$$

Representation

```
struct polynode {
    int coef;
    int exp;
    struct polynode * next;
};

typedef struct polynode *polyptr;
```



•Adding polynomials using a Linked list representation:
(storing the result in p3)

To do this, we have to break the process down to cases:

•Case 1: exponent of p1 > exponent of p2

Copy node of p1 to end of p3. [go to next node]

•Case 2: exponent of p1 < exponent of p2

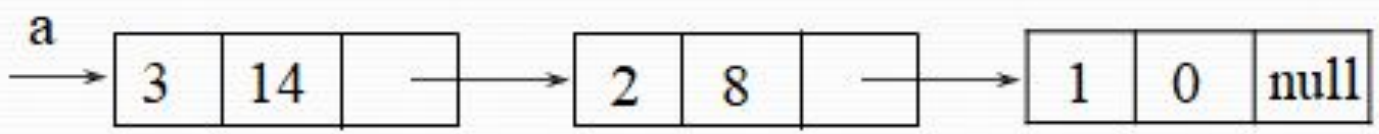
Copy node of p2 to end of p3. [go to next node]

•Case 3: exponent of p1 = exponent of p2

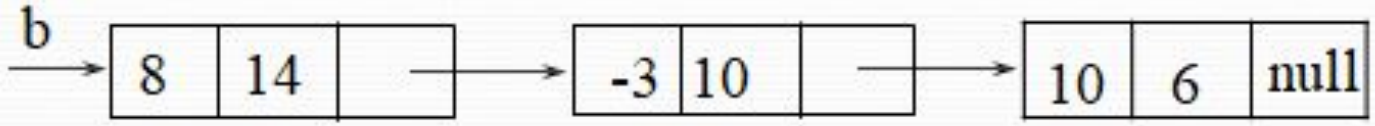
Create a new node in p3 with the same exponent and with the sum of the coefficients of p1 and p2.

Example

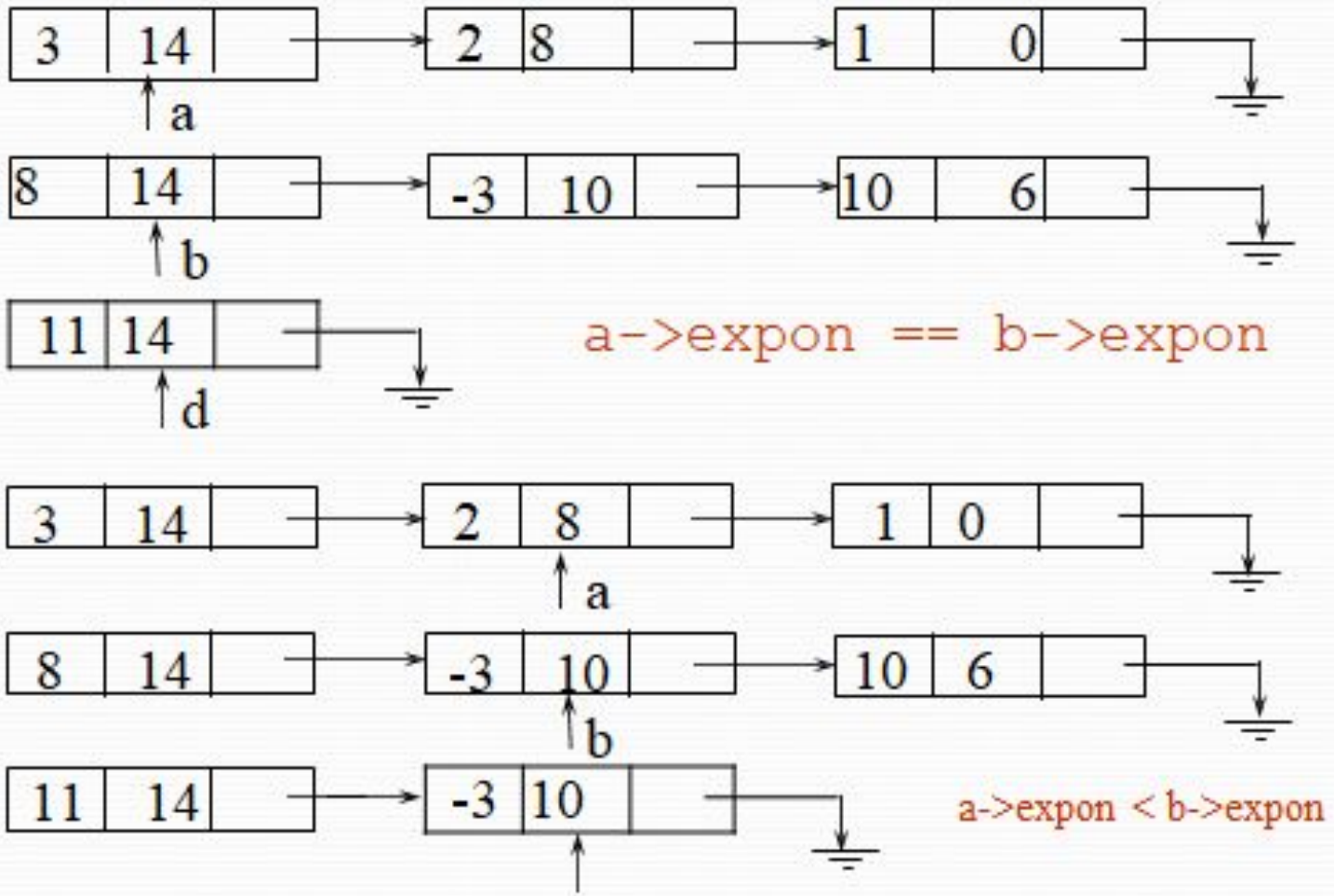
$$a = 3x^{14} + 2x^{8} + 1$$

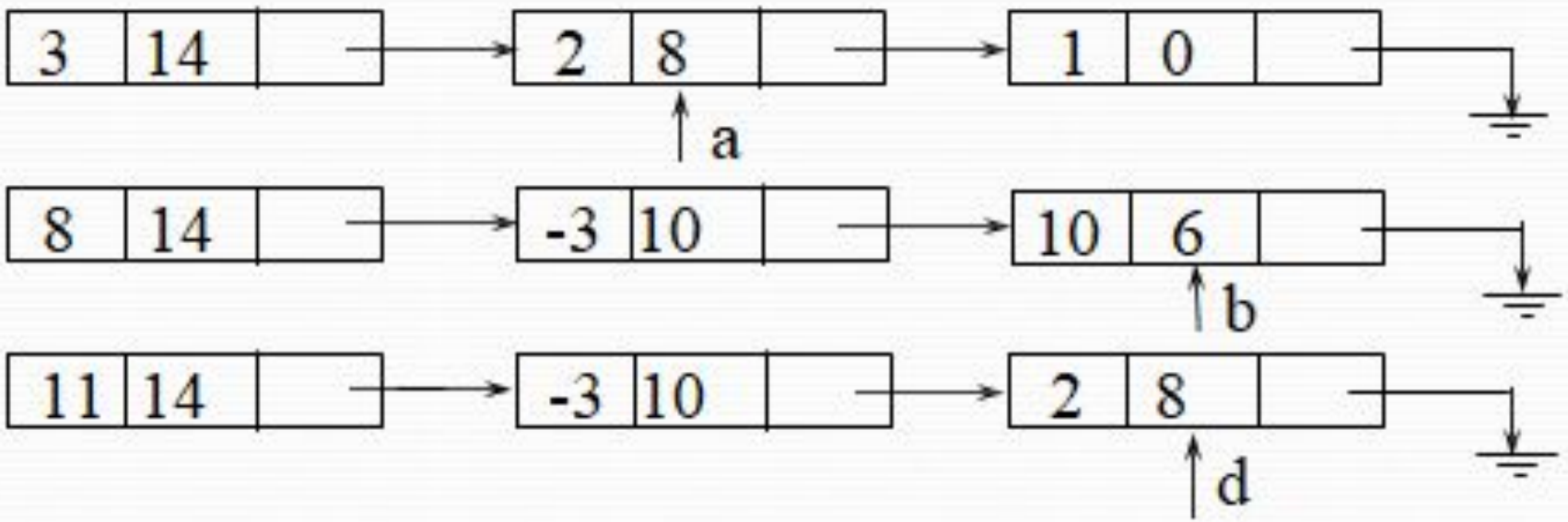


$$b = 8x^{14} - 3x^{10} + 10x^{6}$$



Adding Polynomials





a->expon > b->expon

Algorithm:

- pHead1, pHead2, pHead3 are three head nodes for three Lists.

```
p1 = phead1;
p2 = phead2;
p3 = phead3;

/* now traverse the lists till one list gets exhausted */
while ((p1 != NULL) || (p2 != NULL))
{
while (p1 ->exp > p2 -> exp )
{
p3 -> exp = p1 -> exp;
p3 -> coff = p1 -> coff ;
append (p3, phead3);
/* now move to the next term in list 1*/
p1 = p1 -> next;
}
      //similarly perform if p2->exp > p1->exp
            // Continues on next page
```

```
while (p1 ->exp = p2 -> exp )
{
p3-> exp = p1-> exp;
p3->coff = p1->coff + p2-> coff ;
append (p3, phead3) ;
p1 = p1->next ;
p2 = p2->next ;
}
}
```

/* now consider the possibility that list2 gets exhausted , and there are terms remaining only in list1. So all those terms have to be appended to end of list3. However, you do not have to do it term by term, as p1 is already pointing to remaining terms, so simply append the pointer p1 to phead3 */

```
if ( p1 != NULL)
append (p1, phead3) ;
else
append (p2, phead3);
```

Thank You