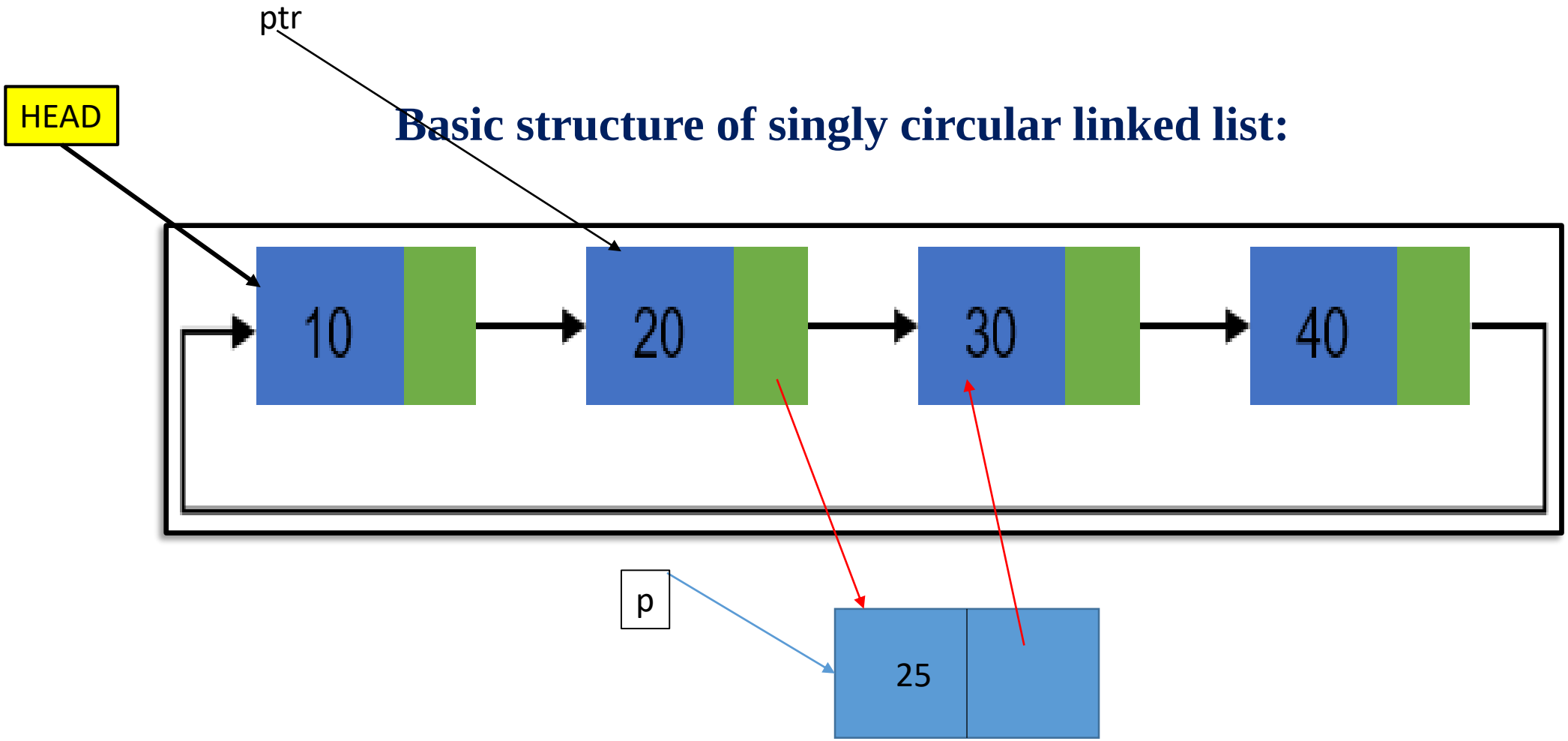
Circular linked list

Circular linked list:



Circular Linked Lists

- Circular linked lists are the most complex type of linked list to implement, but they can be very efficient for certain operations, such as traversal and queueing.
- In a circular linked list, the last node in the list points back to the first node in the list, forming a loop.
- Example: Circular buffer, circular queue, circular linked list implementation of a hash table.

Circular linked list:

Advantages of a Circular linked list

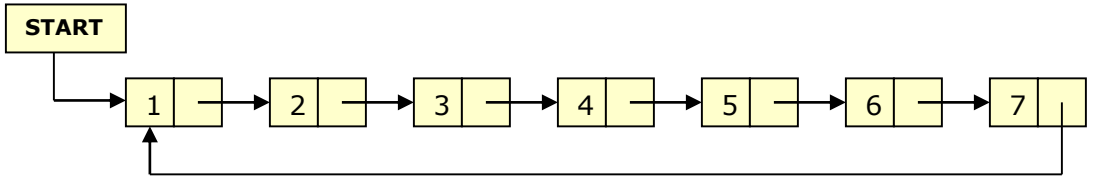
- Entire list can be traversed from any node.
- Circular lists are the required data structure when we want a list to be accessed in a circle or loop.
- Despite of being singly circular linked list we can easily traverse to its previous node, which is not possible in singly linked list.

Disadvantages of Circular linked list

- Circular list are complex as compared to singly linked lists.
- Reversing of circular list is a complex as compared to singly or doubly lists.
- If not traversed carefully, then we could end up in an infinite loop.
- Like singly and doubly lists circular linked lists also doesn’t supports direct accessing of elements.

Circular Linked List

- In a circular linked list, the last node contains a pointer to the first node of the list. We can have a circular singly listed list as well as circular doubly linked list. While traversing a circular linked list, we can begin at any node and traverse the list in any direction forward or backward until we reach the same node where we had started. Thus, a circular linked list has no beginning and no ending.



Operations on circular linked list

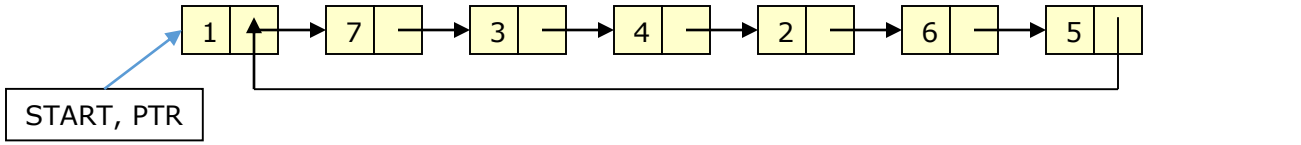
- Creation of list
- Traversal of list
- Insertion of node
  - At the beginning of the list
  - At the end of the list
  - At any position in the list
- Deletion of node
  - Deletion of first node
  - Deletion of node from middle of the list
  - Deletion of last node
- Counting total number of nodes
- Reversing of list

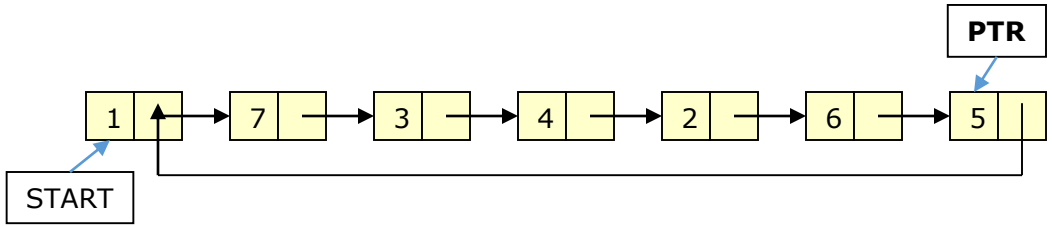
Circular Linked List

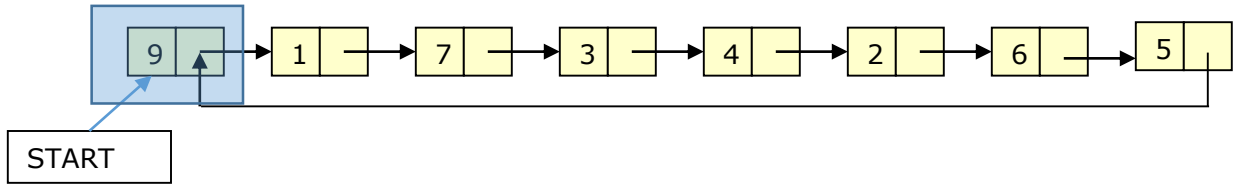
Algorithm to insert a new node in the beginning of circular the linked list

```
Step 1:SET New_Node = CREATE NEW NODE
Step 2:SET New_Node->DATA = VAL
Step 3:SET PTR = START
Step 4:Repeat Step 5 while PTR->NEXT != START
Step 5:       PTR = PTR->NEXT
Step 6:SET New_Node->Next = START
Step 7:SET PTR->NEXT = New_Node
Step 8:SET START = New_Node
Step 9:EXIT
```

Circular Linked List





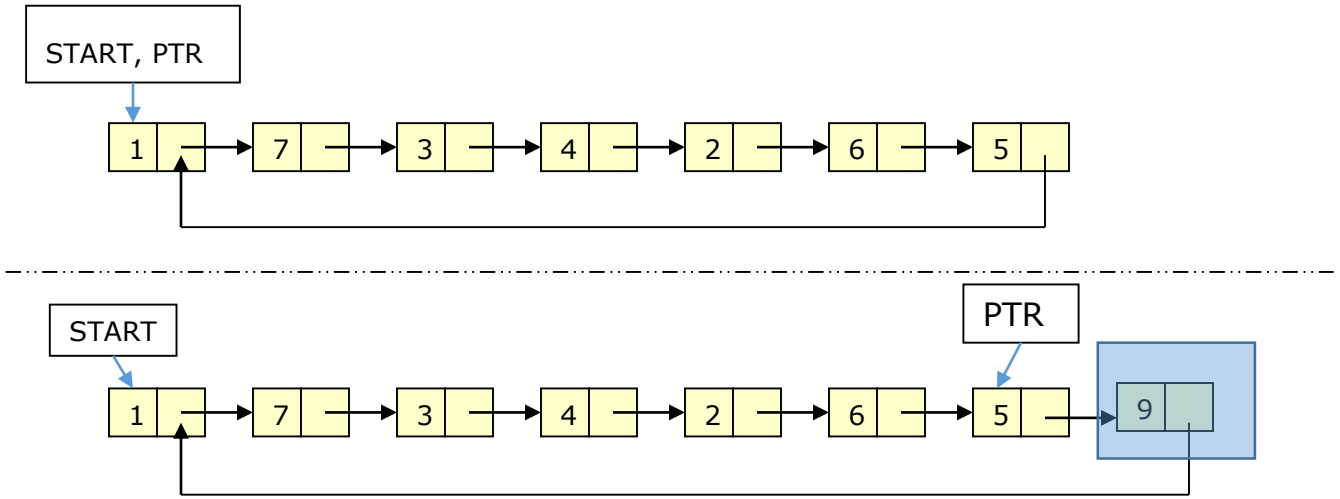


Circular Linked List

Algorithm to insert a new node at the end of the circular linked list

```
Step 1: SET New_Node = CREATE NEW NODE
Step 2: SET New_Node->DATA = VAL
Step 3: SET New_Node->Next = START
Step 4: SET PTR = START
Step 5: Repeat Step 6 while PTR->NEXT != START
Step 6:       SET PTR = PTR ->NEXT
        [END OF LOOP]
Step 7: SET PTR ->NEXT = New_Node
Step 8: EXIT
```

Circular Linked List



Circular Linked List

```
Algorithm to insert a new node after a node that has value NUM

Step 1: SET New_Node = CREATE NEW NODE
Step 2: SET New_Node->DATA = VAL
Step 3: SET PTR = START
Step 4: SET PREPTR = PTR
        [START OF LOOP]
Step 5: SET PREPTR = PTR
Step 6: SET PTR = PTR->NEXT
Step 7:Repeat Step 5 and 6 while PTR->DATA != NUM AND PTR!=START
       [END OF LOOP]
Step 8: IF PTR != START
Step 9:       PREPTR->NEXT = New_Node
Step 9:       SET New_Node->NEXT = PTR
       [END OF IF]
Step 10:EXIT
```

Circular Linked List

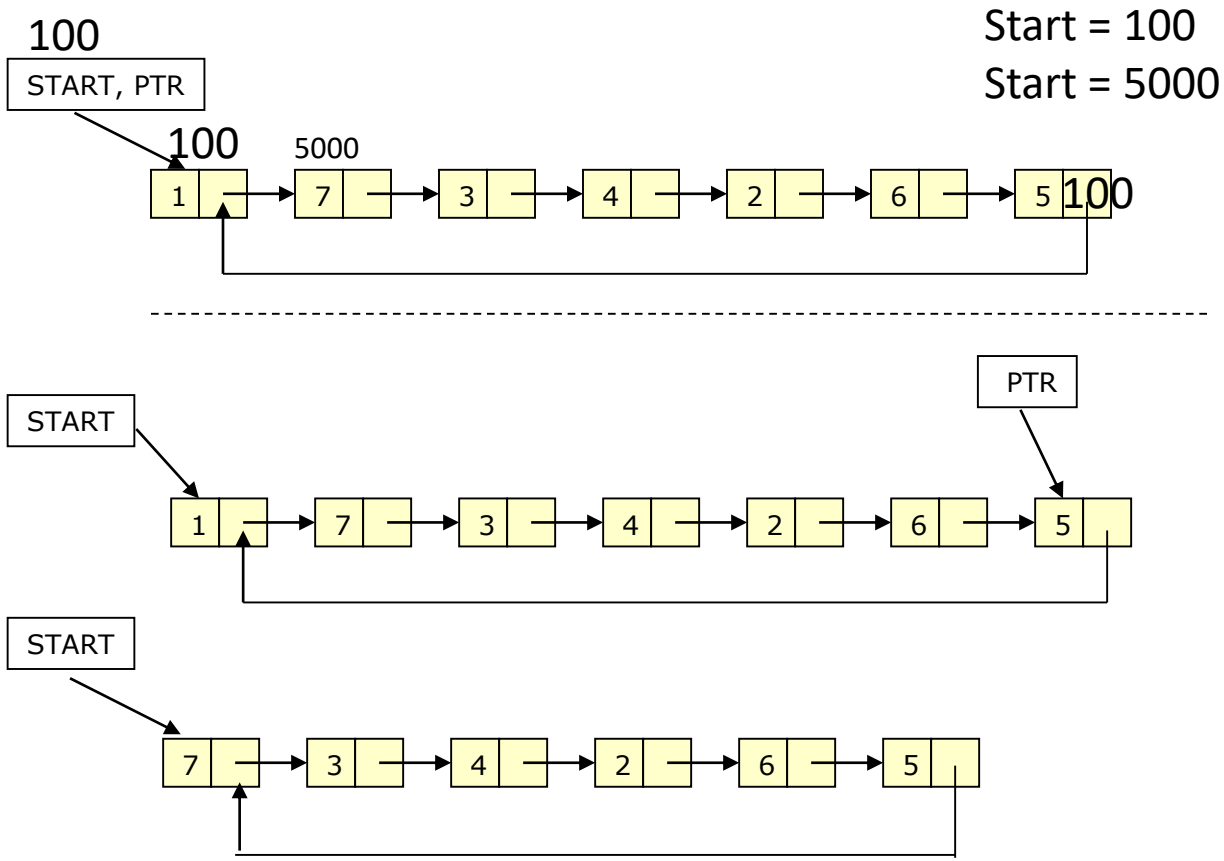
Algorithm to delete the first node from the circular linked list

```
Step 1: IF START = NULL, then
                Write UNDERFLOW
                Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR->NEXT != START
Step 4:                 SET PTR = PTR->NEXT
        [END OF IF]
Step 5: SET PTR->NEXT = START->NEXT
Step 6: FREE START
Step 7: SET START = PTR->NEXT
Step 8: EXIT
```

Circular Linked List

```
delnode_beg()
{
        struct Node *ptr=head;
        if(head==NULL)
        { printf("UNDERFLOW");
        return;
        }
        if(head->next == head)
        {
                printf("%d DELETED",head->data);
                free(head);
                head=NULL;
                return;
        }
        while(ptr->next !=head)
        { ptr=ptr->next;
        }
        ptr->next=head->next;
        printf("%d DELETED",head->data);
        free(head);
        head=ptr->next;
}
```

Circular Linked List

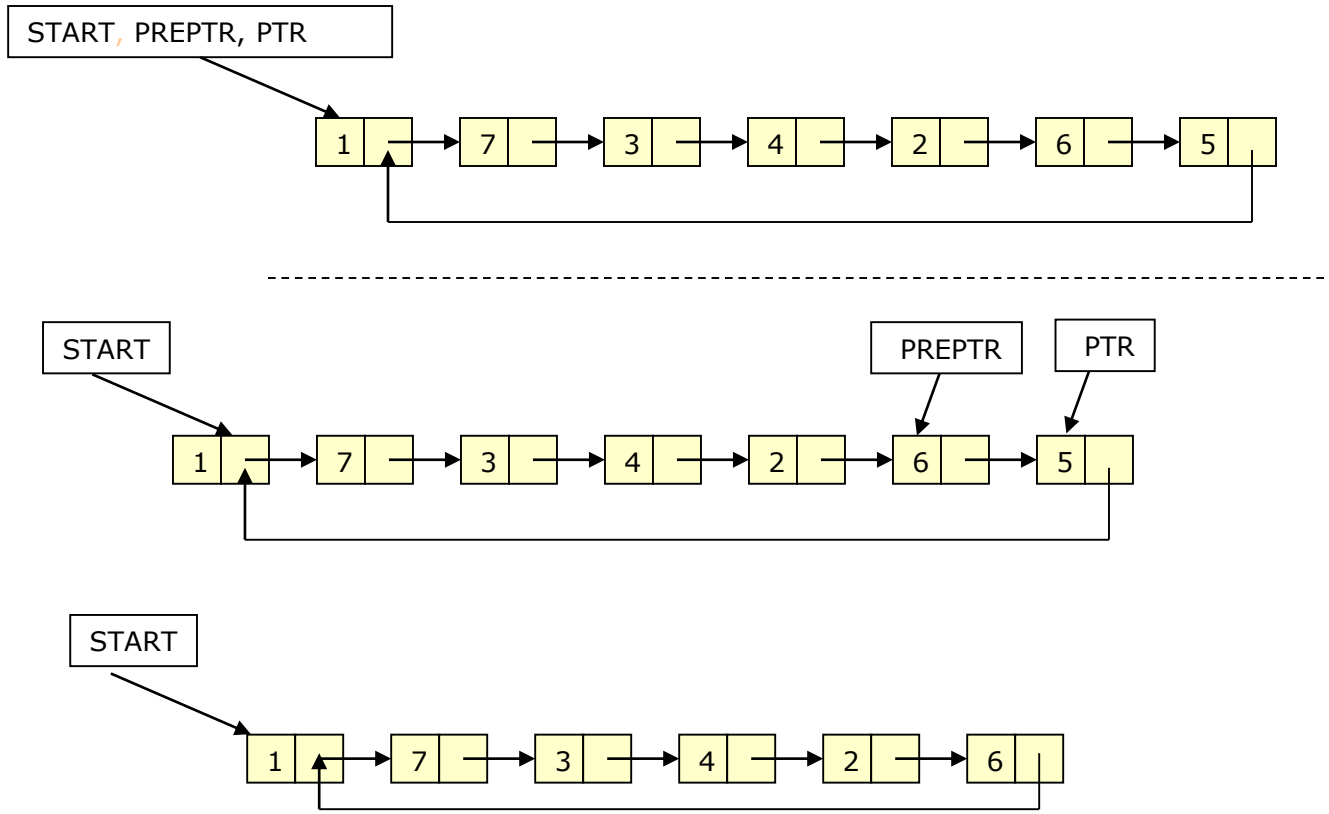


Circular Linked List

Algorithm to delete the last node of the circular linked list

```
Step 1: IF START = NULL, then
              Write UNDERFLOW
              Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 and 5 while PTR->NEXT != START
Step 4:              SET PREPTR = PTR
Step 5:              SET PTR = PTR->NEXT
        [END OF LOOP]
Step 6: SET PREPTR->NEXT = START
Step 7: FREE PTR
Step 8: EXIT
```

Circular Linked List

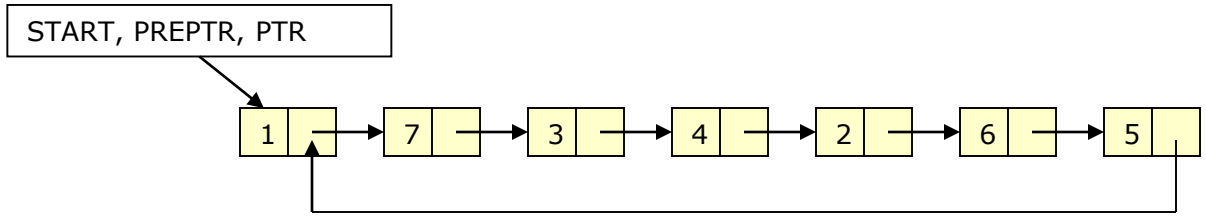


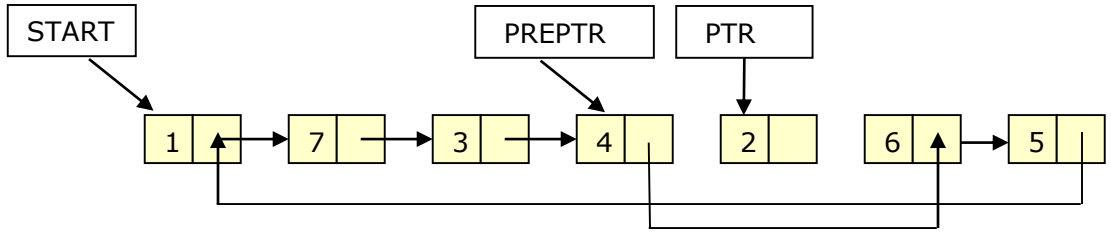
Circular Linked List

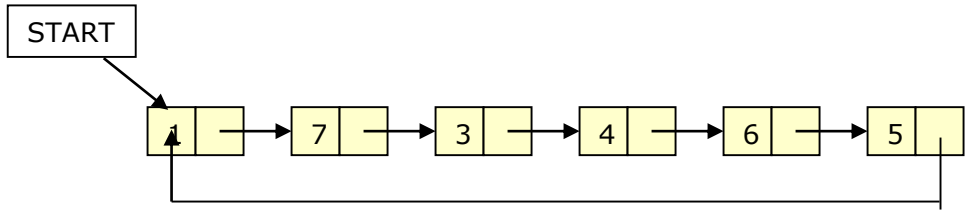
Algorithm to delete the node after a given node from the circular linked list

Step 1: IF START = NULL, then
Write UNDERFLOW
Go to Step 9
[END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Step 5 and 6 while PTR->DATA != NUM and PTR != START
Step 5: SET PREPTR = PTR
Step 6: SET PTR = PTR->NEXT
[END OF LOOP]
Step 7: SET PREPTR->NEXT = PTR->NEXT
Step 8: FREE PTR
Step 9: EXIT

Circular Linked List







Circular Linked List

Try

- Counting total number of nodes in Circular Linked List
- Searching an element in the list
- Frequency count of an element in the list
- Finding MAX, MIN, SUM of values in the list
- Reversing of list

Applications of linked lists

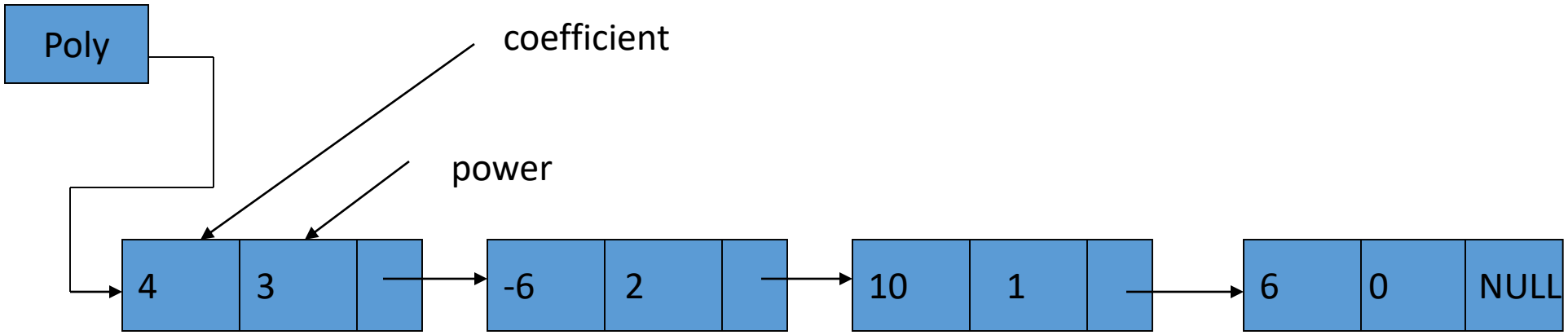
- To implement the other data structures such as stacks, queues, trees and graphs.
- To maintain a directory of names.
- To perform arithmetic operation on long integers.
- To manipulate polynomial.
- To represent sparse matrices.

Polynomial Manipulation

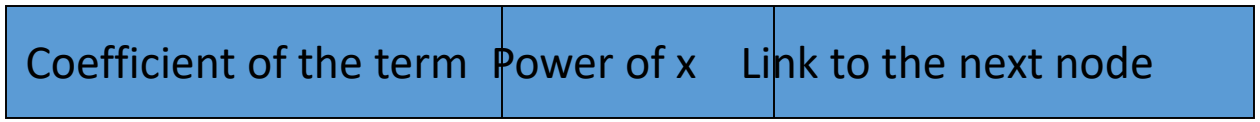
A polynomial of type

$$4x^3-6x^2+10x+6$$

can be represented using following linked list



In the above list, each node has the following structure



Polynomial Manipulation

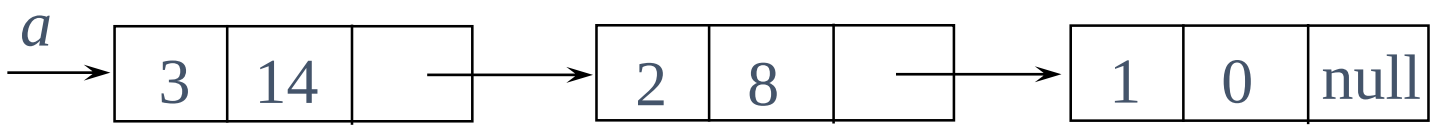
The required memory declarations for the representation of a polynomial with integer coefficients are

```
typedef struct nodetype
        {
                int coeff;
                int power;
                struct nodetype *next;
        }node;
        node *poly;
```

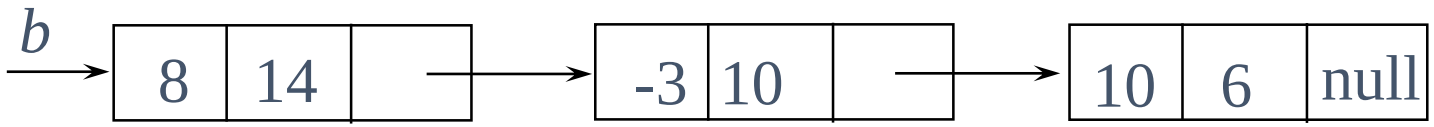
Polynomial Representation: Example

- **Examples**

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



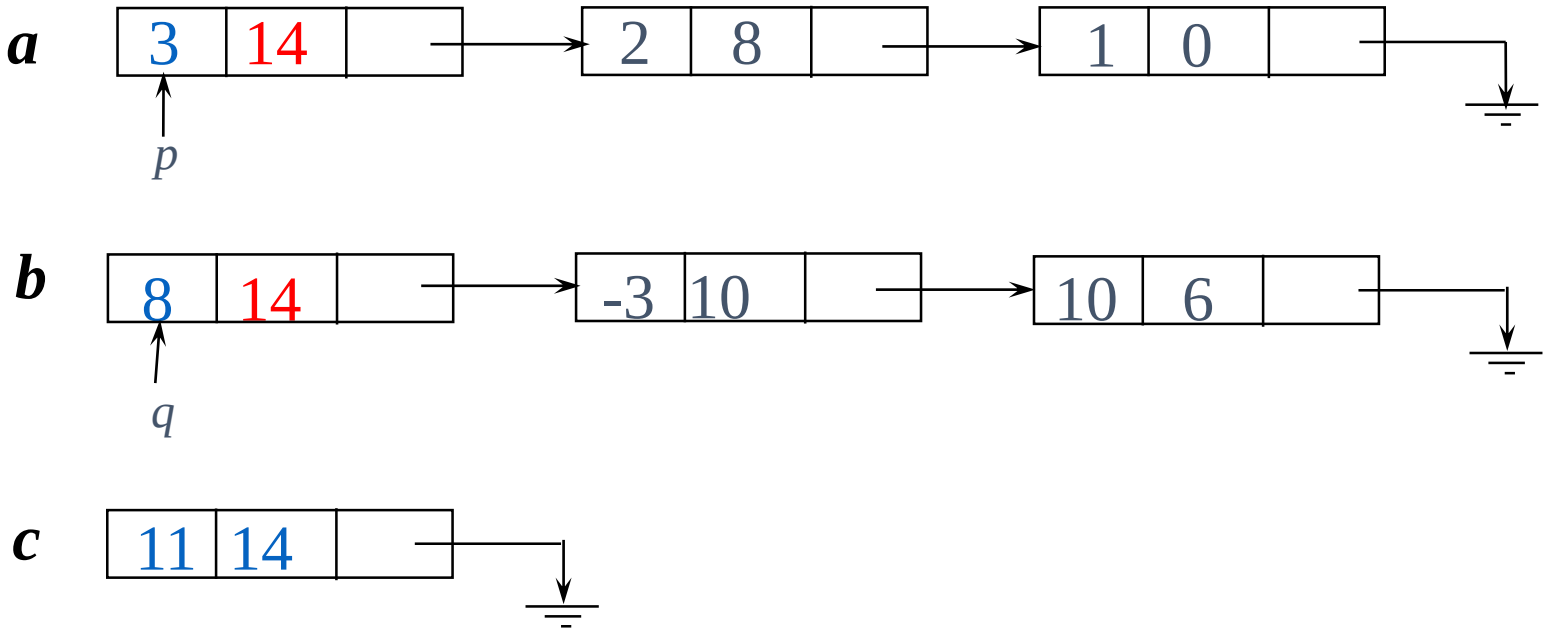
$$b = 8x^{14} - 3x^{10} + 10x^{6}$$



Adding Polynomials: $c = a + b$

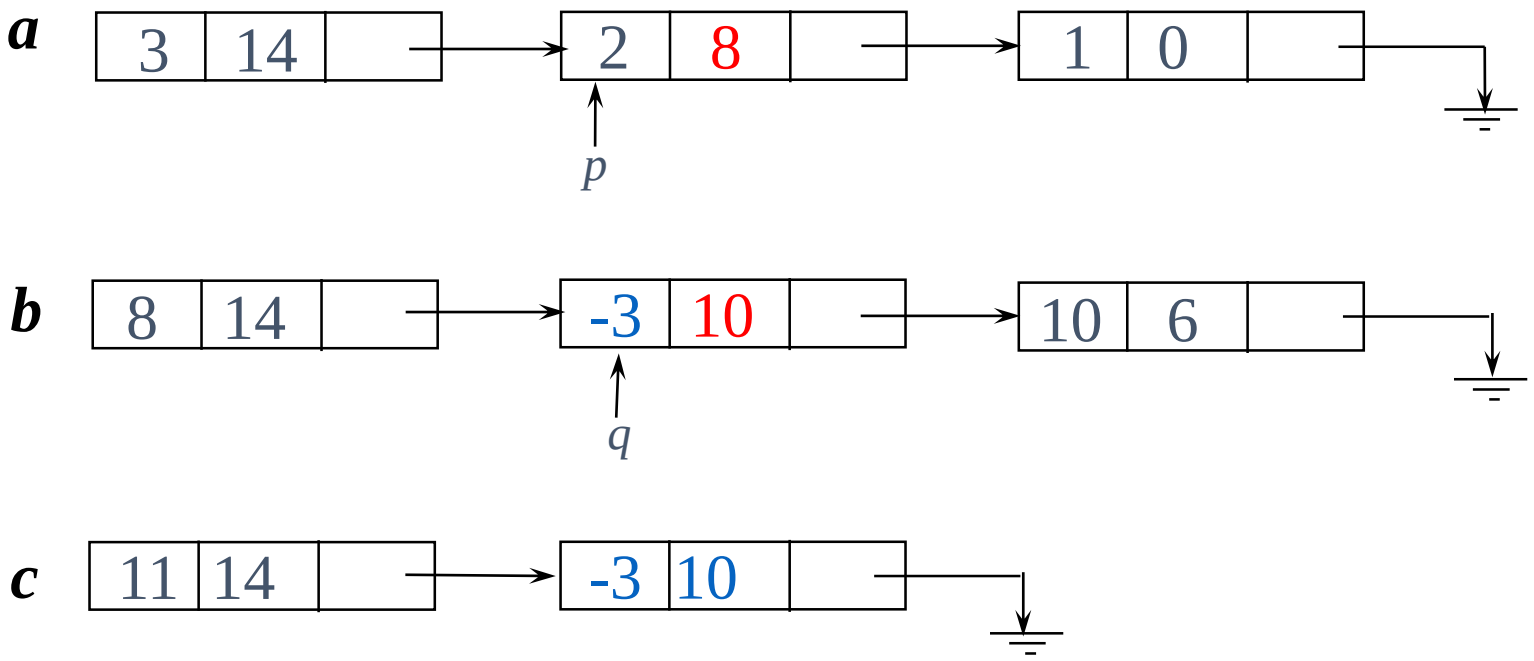
- **Adding Polynomials: $c = a + b$**

Case 1: *p->power == q->power then c->coeff = p->coeff + q->coeff*
c->power = p->power



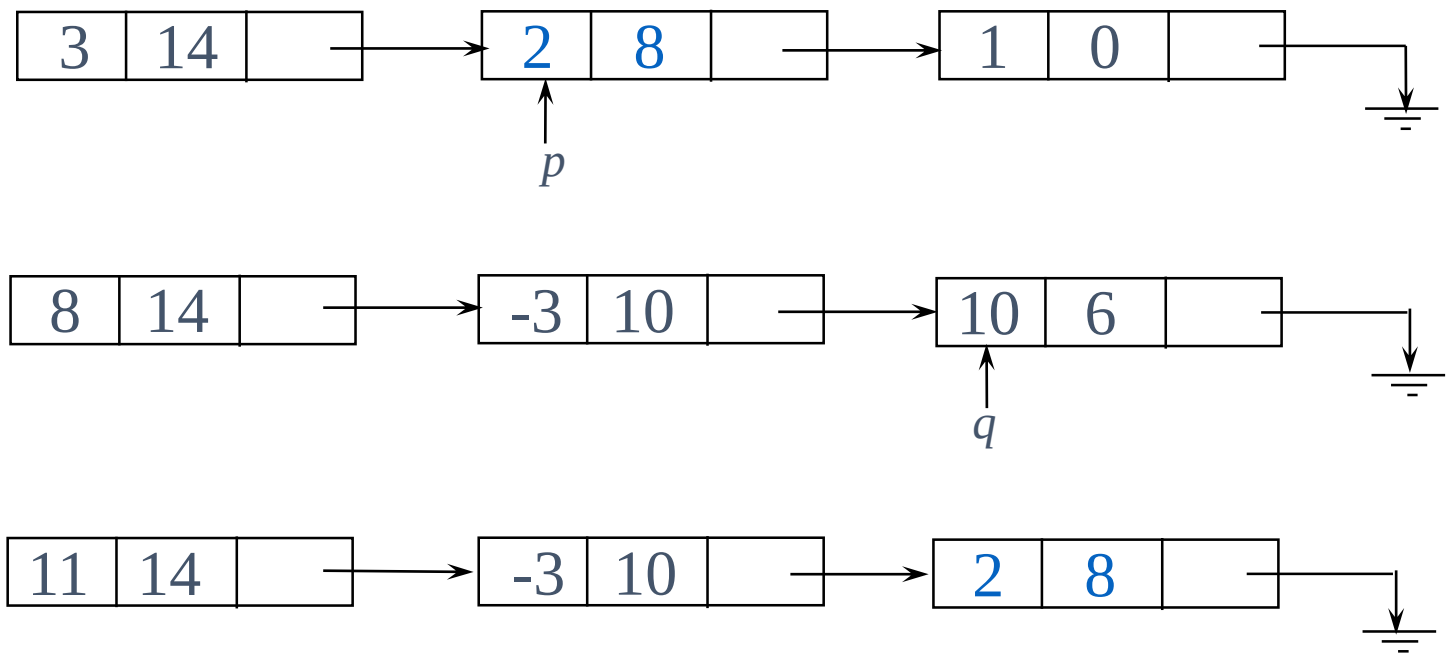
Adding Polynomials : $c = a + b$ (cont.)

Case 2: *p->power < q->power then c->coeff = q->coeff*
c->power = q->power



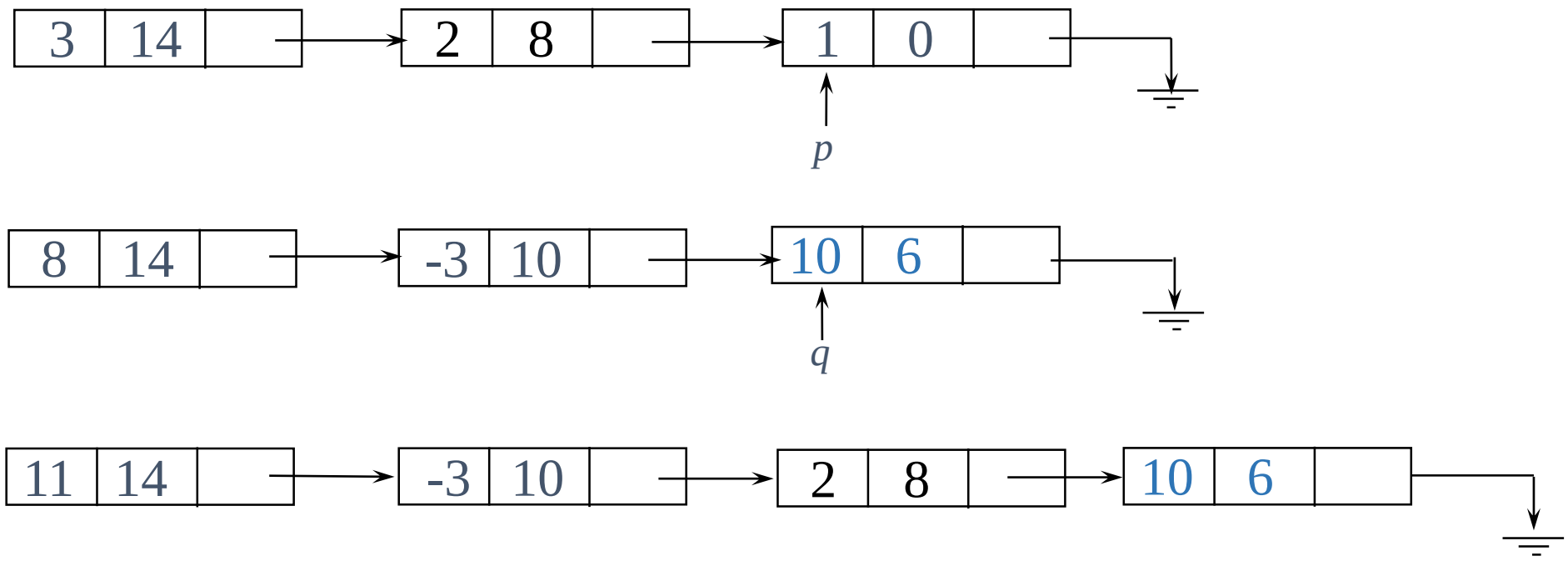
Adding Polynomials: $c = a + b$ (cont.)

Case 3: *p->power > q->power then c->coeff = p->coeff*
c->power = p->power



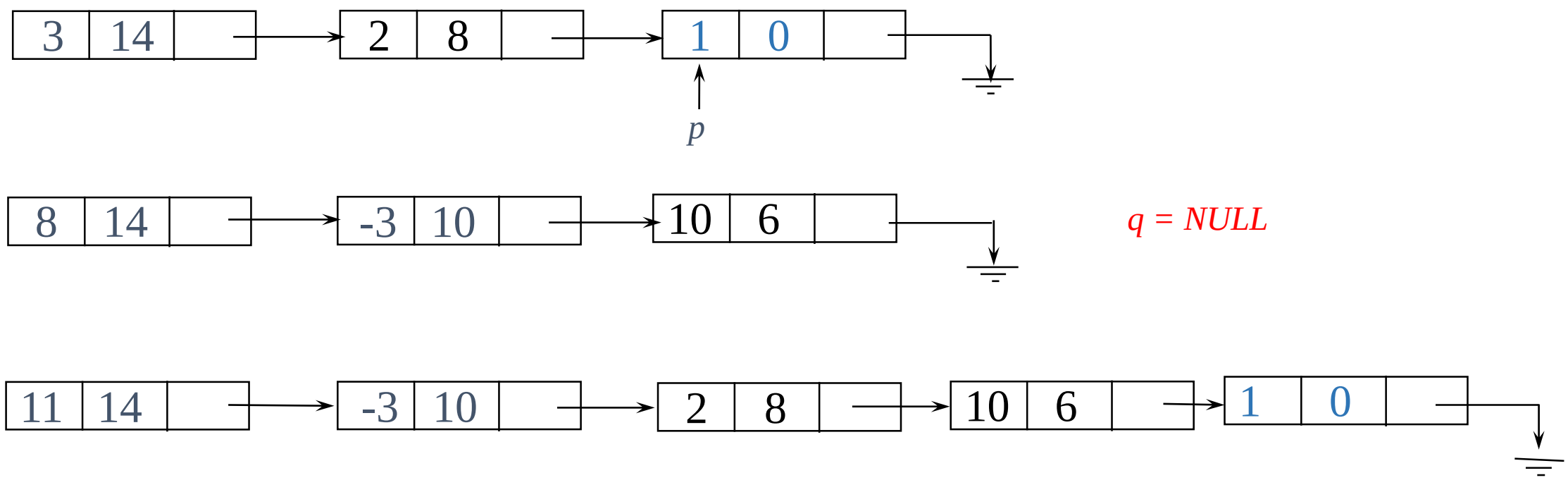
Adding Polynomials: $c = a + b$ (cont.)

Case 2: *p->exp < q->exp*

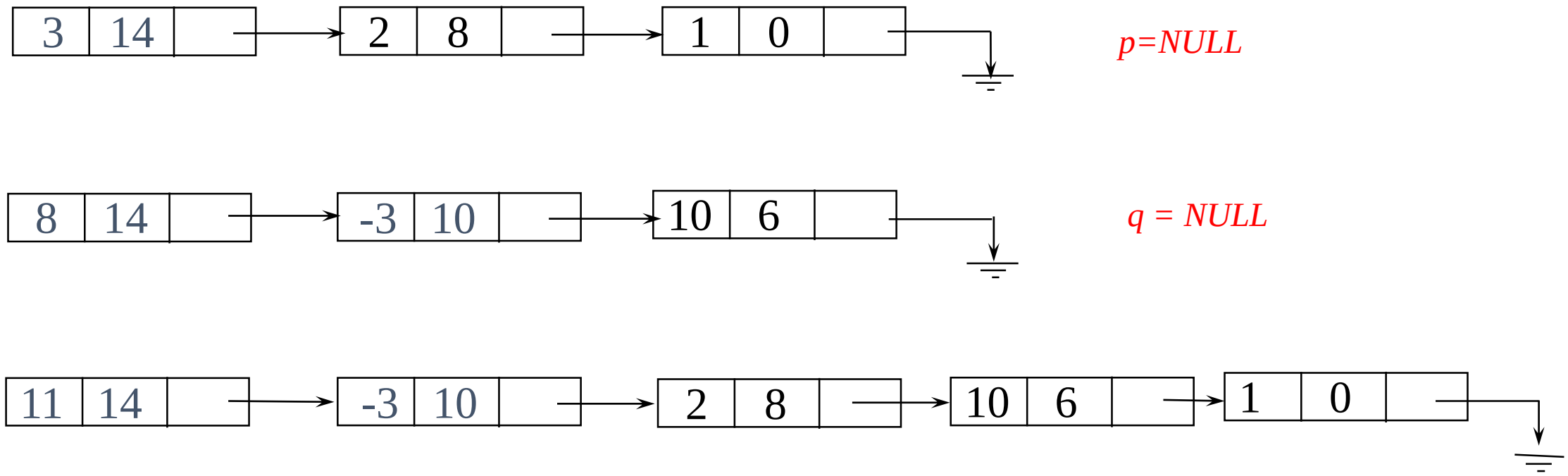


Adding Polynomials: $c = a + b$ (cont.)

Case 4: ***q==NULL*** *then while p != NULL insert all nodes of p at the end of c*



Adding Polynomials: $c = a + b$ (cont.)



Lab Assignment 2:

Implement addition of Two polynomials using Linked Lists

Next Topic:

Doubly Linked List

Header Linked List