

# **CSE 1201**

## **Data Structure**

### Chapter 3: Linked List

**Instructor**

**Md. Shahid Uz Zaman**

**Dept of CSE, RUET**

# Disadvantages of Array Processing

1. We must know in advance that how many elements are to be stored in array.
2. Array is static structure. It means that array is of fixed size. The memory which is allocated to array can not be increased or reduced
3. Since array is of fixed size, if we allocate more memory than requirement then the memory space will be wasted. And if we allocate less memory than requirement, then it will create problem.
4. The elements of array are stored in consecutive memory locations. So insertions and deletions are very difficult and time consuming.
5. Lot of shifting is required for insertion and deletion operation.

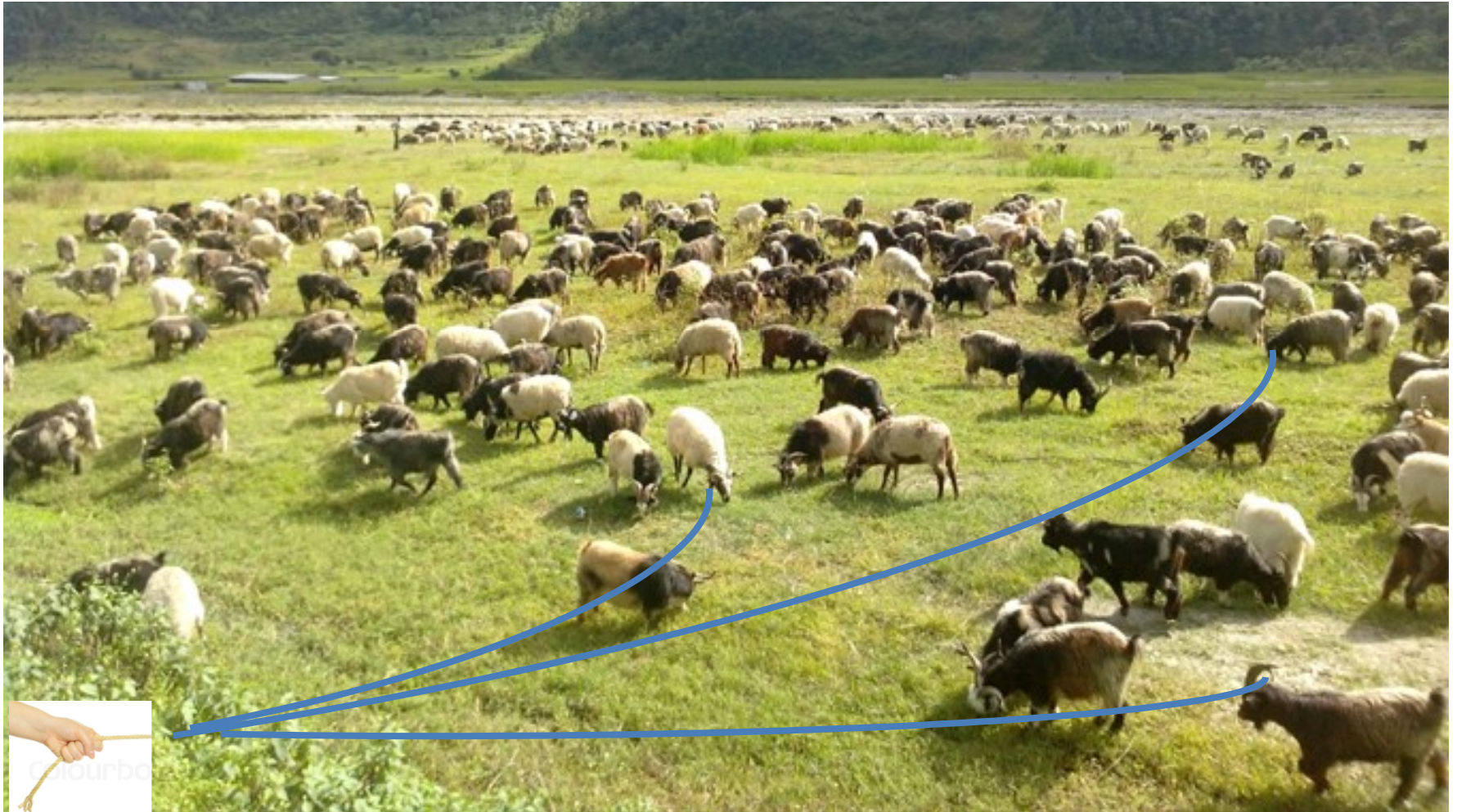
# Example: Goat Field



**How to identify a group of goats?**



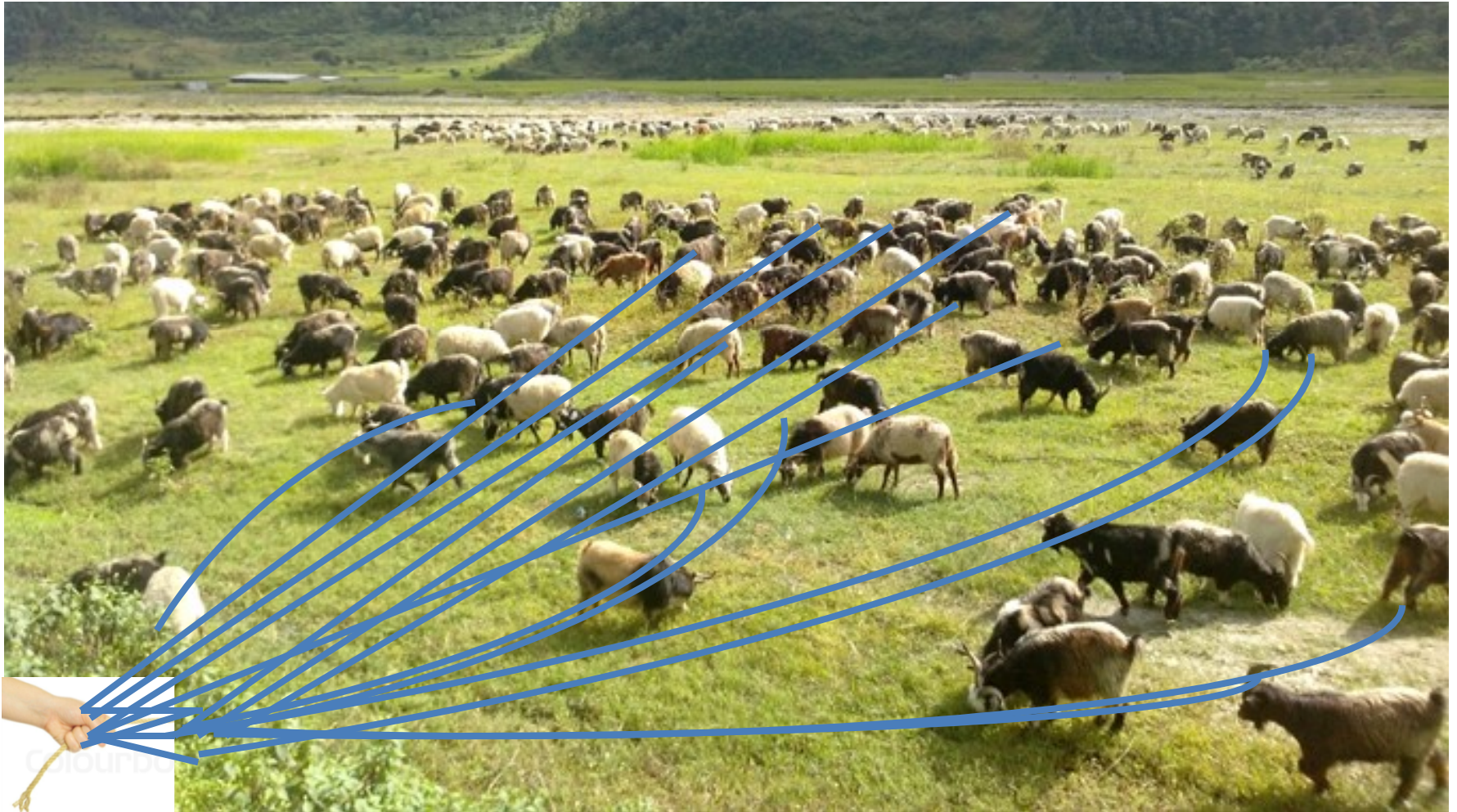
# Goat Field



**How to identify a group of goats? Using ropes**

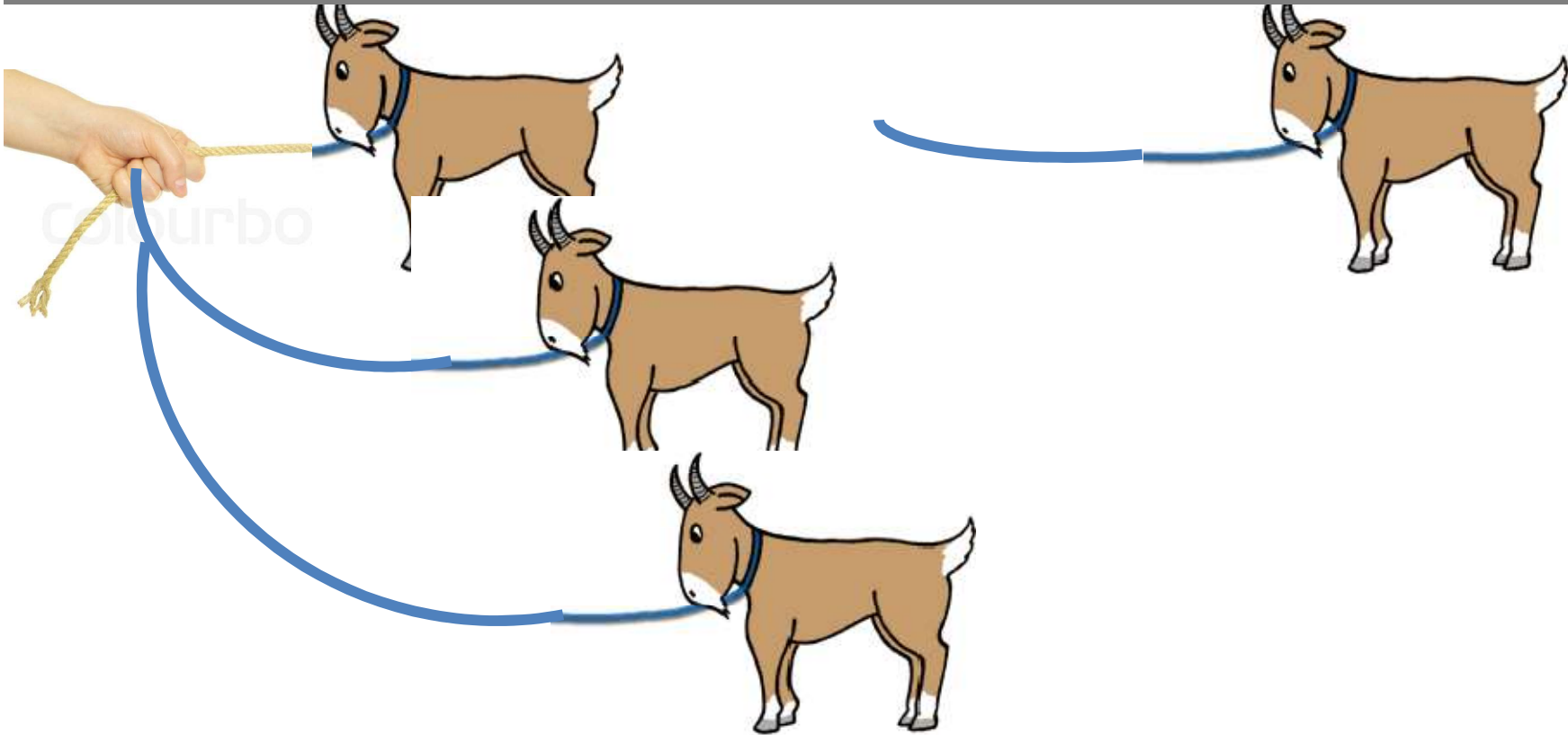


# Goat Field



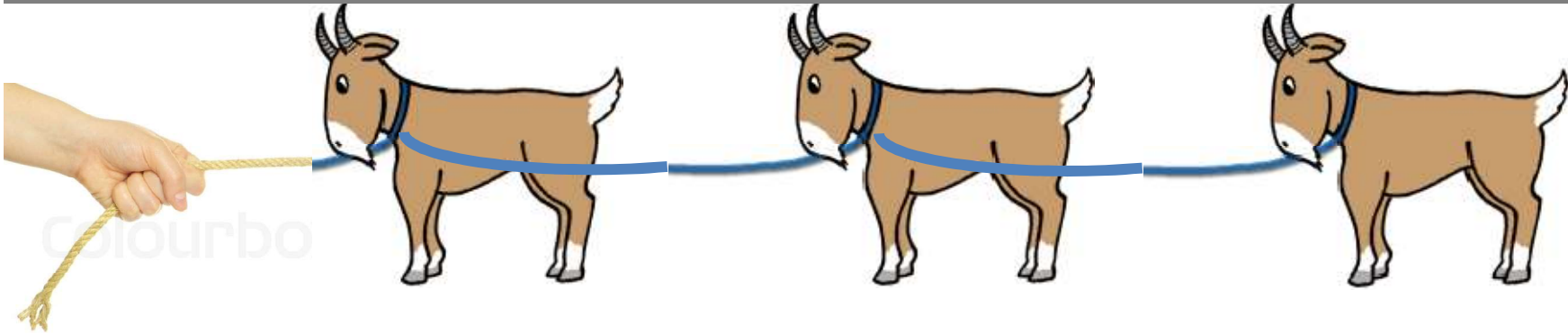
**If the group contains many goats?**

# Goat Field



- Suppose a group with 3 goats.
- Impossible to catch many ropes
- Then how to arrange those goats so that only one rope can be caught?

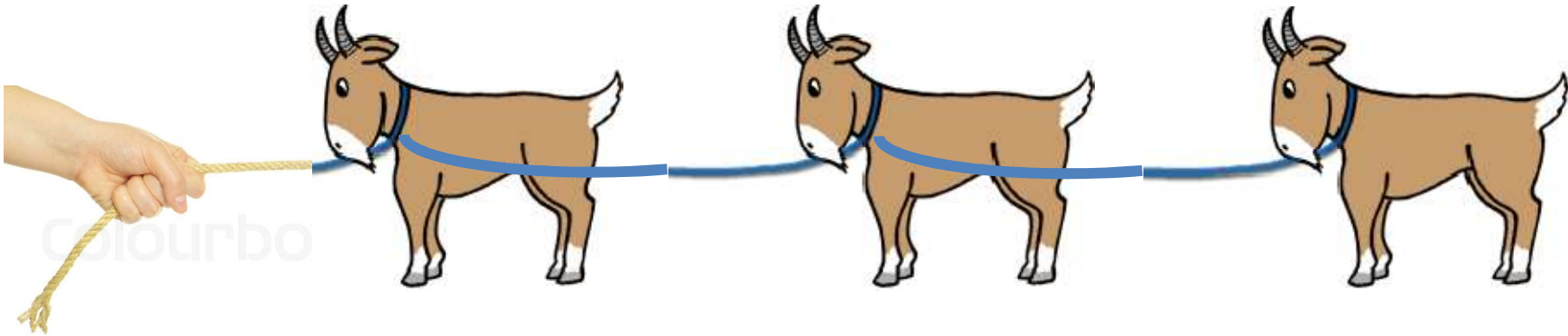
# Goat List



- Suppose a group with 3 goats.
- Impossible to catch many ropes
- Then how to arrange those goats so that only one rope can be caught?

Ans: Bind one with another.

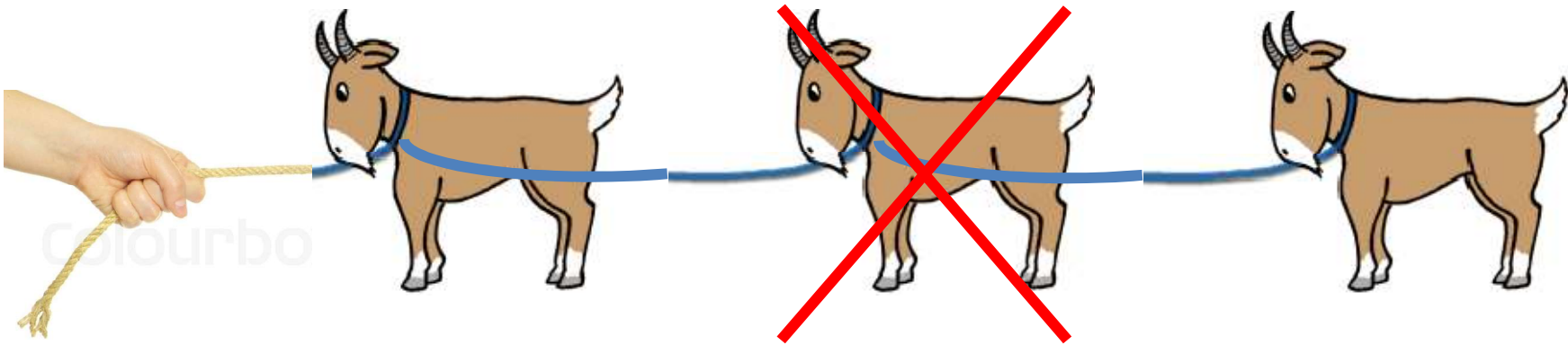
# Goat List : DELETE



- Suppose 2<sup>nd</sup> goat is needed to be deleted

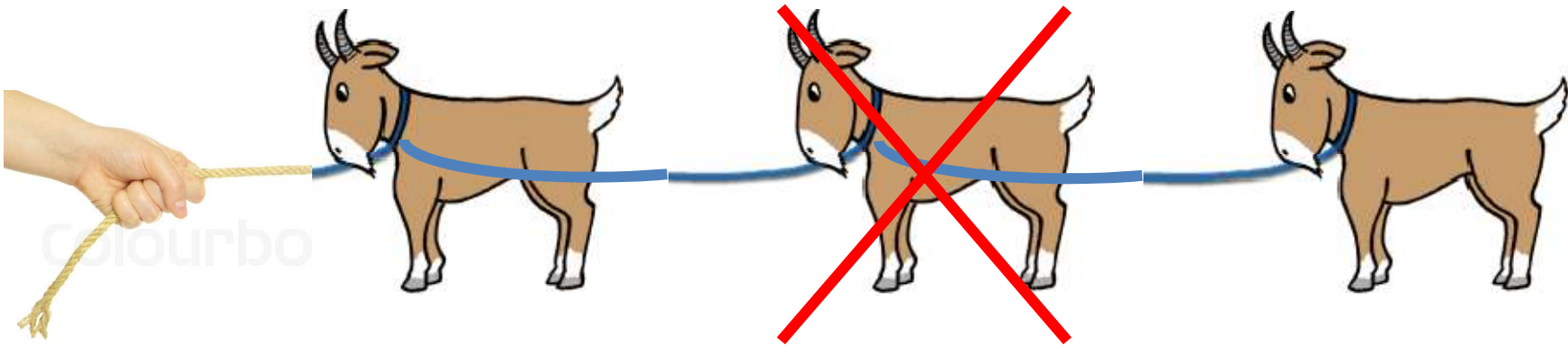


# Goat List : DELETE



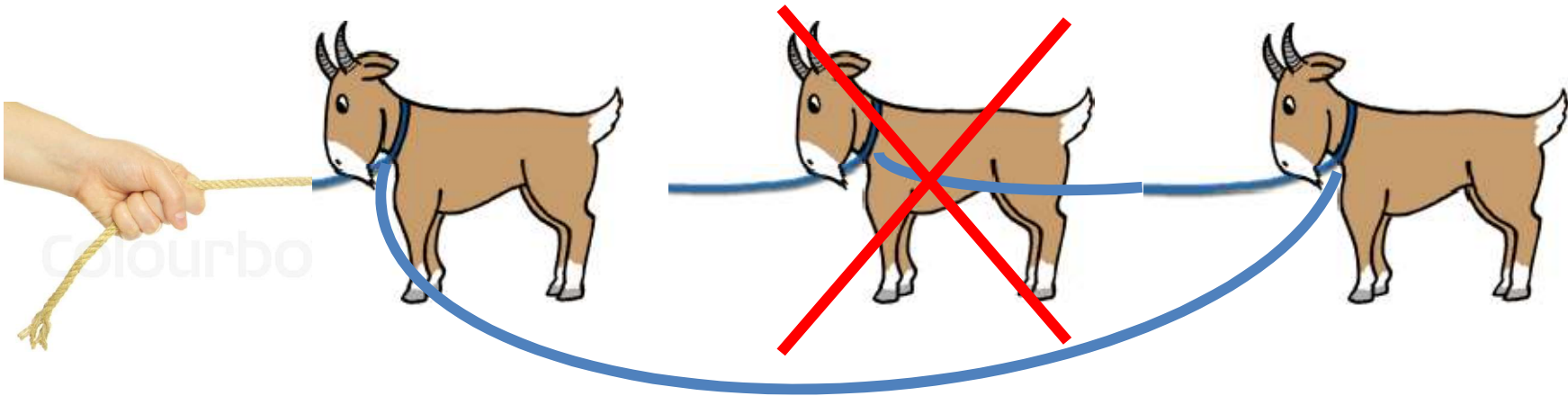
- Suppose 2<sup>nd</sup> goat is needed to be deleted

# Goat List : DELETE



- Suppose 2<sup>nd</sup> goat is needed to be deleted
- Step 1: bind 3<sup>rd</sup> goat to 1<sup>st</sup> one

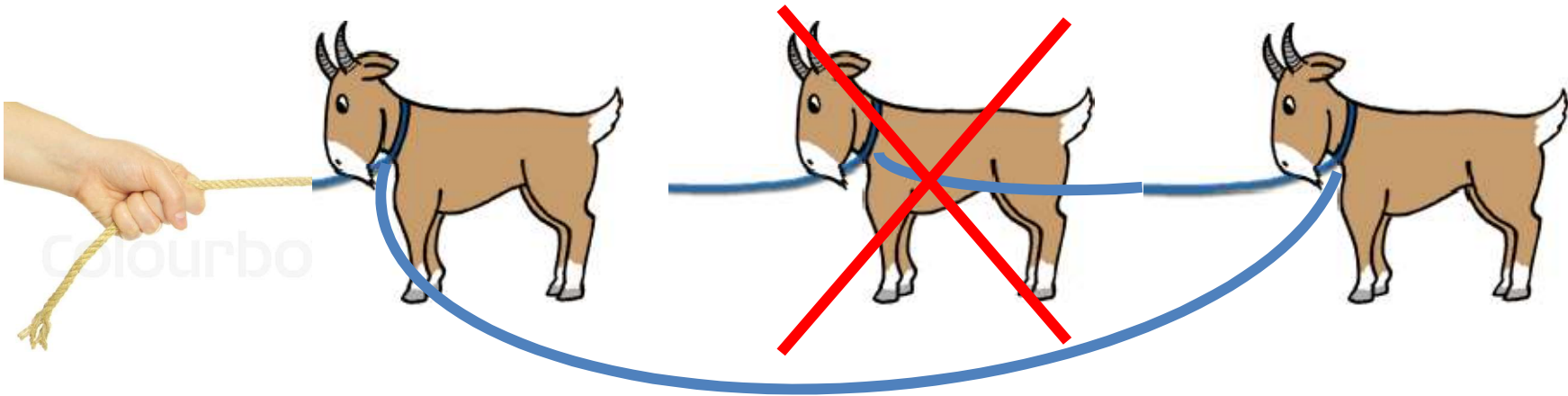
# Goat List : DELETE



- Suppose 2<sup>nd</sup> goat is needed to be deleted
- Step 1: bind 3<sup>rd</sup> goat to 1<sup>st</sup> one



# Goat List : DELETE



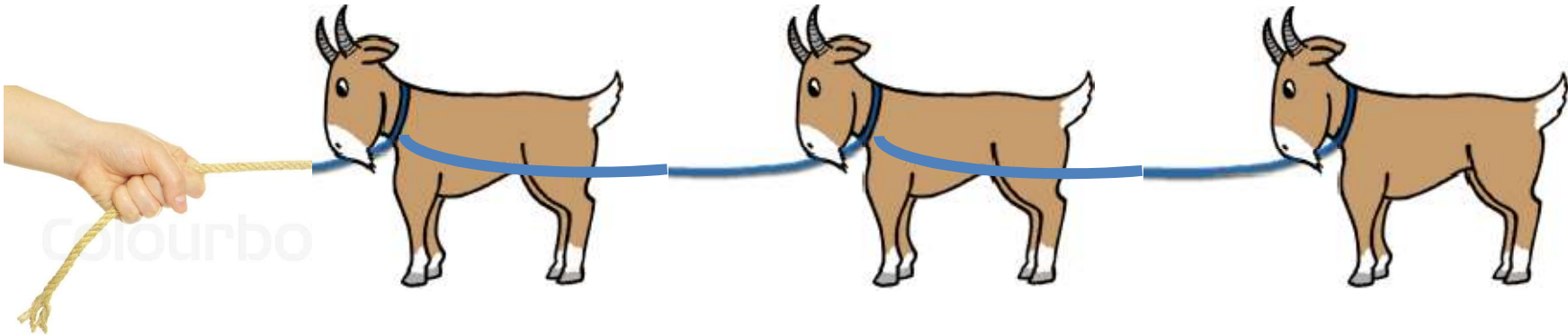
- Suppose 2<sup>nd</sup> goat is needed to be deleted
- Step 1: bind 3<sup>rd</sup> goat to 1<sup>st</sup> one
- Step 2: free 2<sup>nd</sup> goat

# Goat List : DELETE



- Suppose 2<sup>nd</sup> goat is needed to be deleted
- Step 1: bind 3<sup>rd</sup> goat to 1<sup>st</sup> one
- Step 2: free 2<sup>nd</sup> goat

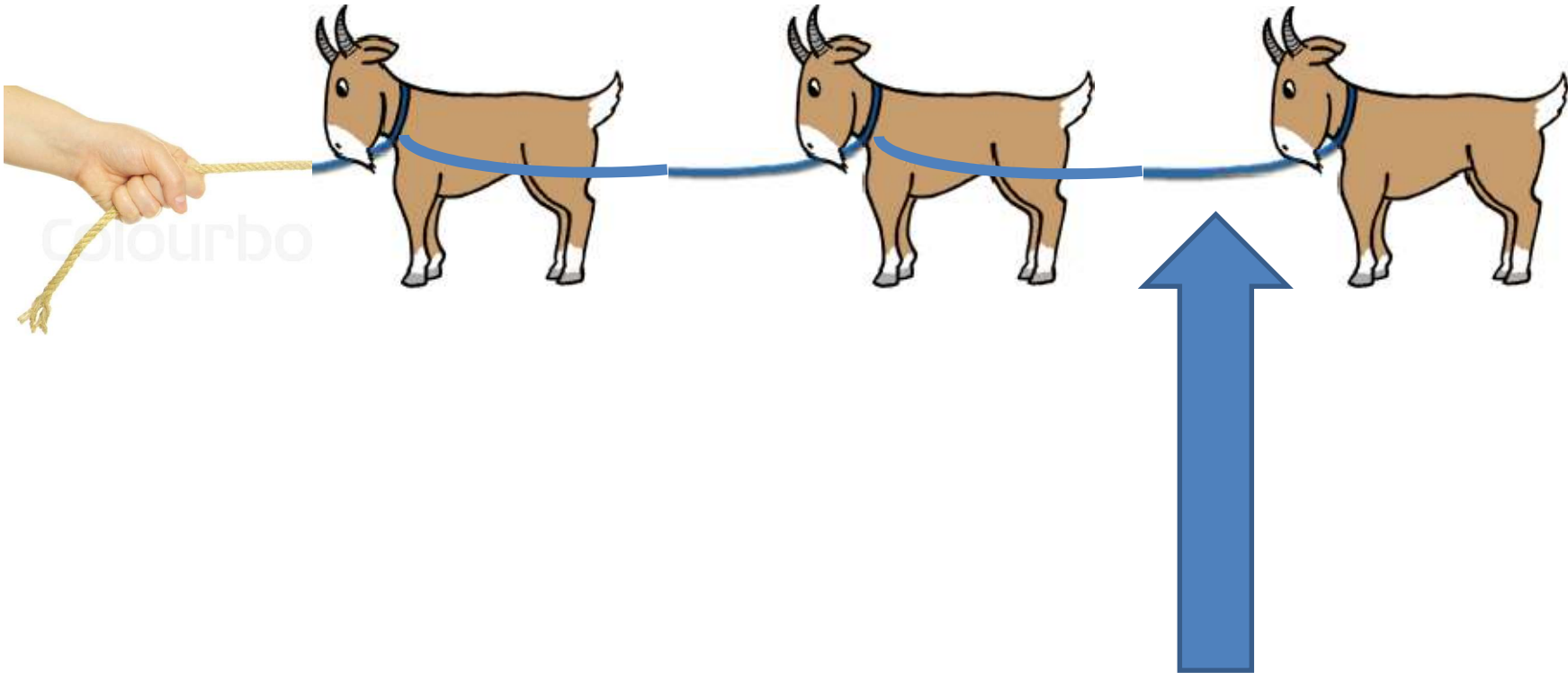
# Goat List : INSERT



- Suppose a new goat is needed to be Inserted between 2<sup>nd</sup> and 3<sup>rd</sup> goats.

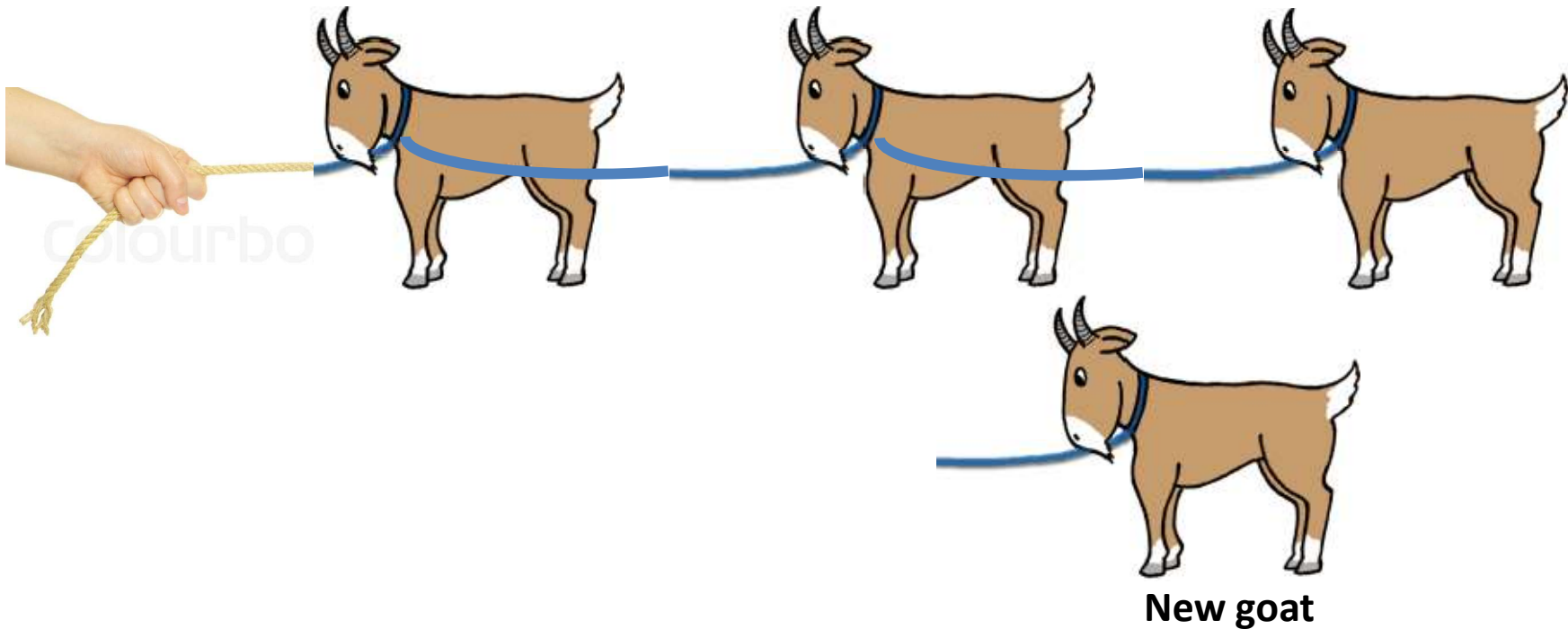


# Goat List : INSERT



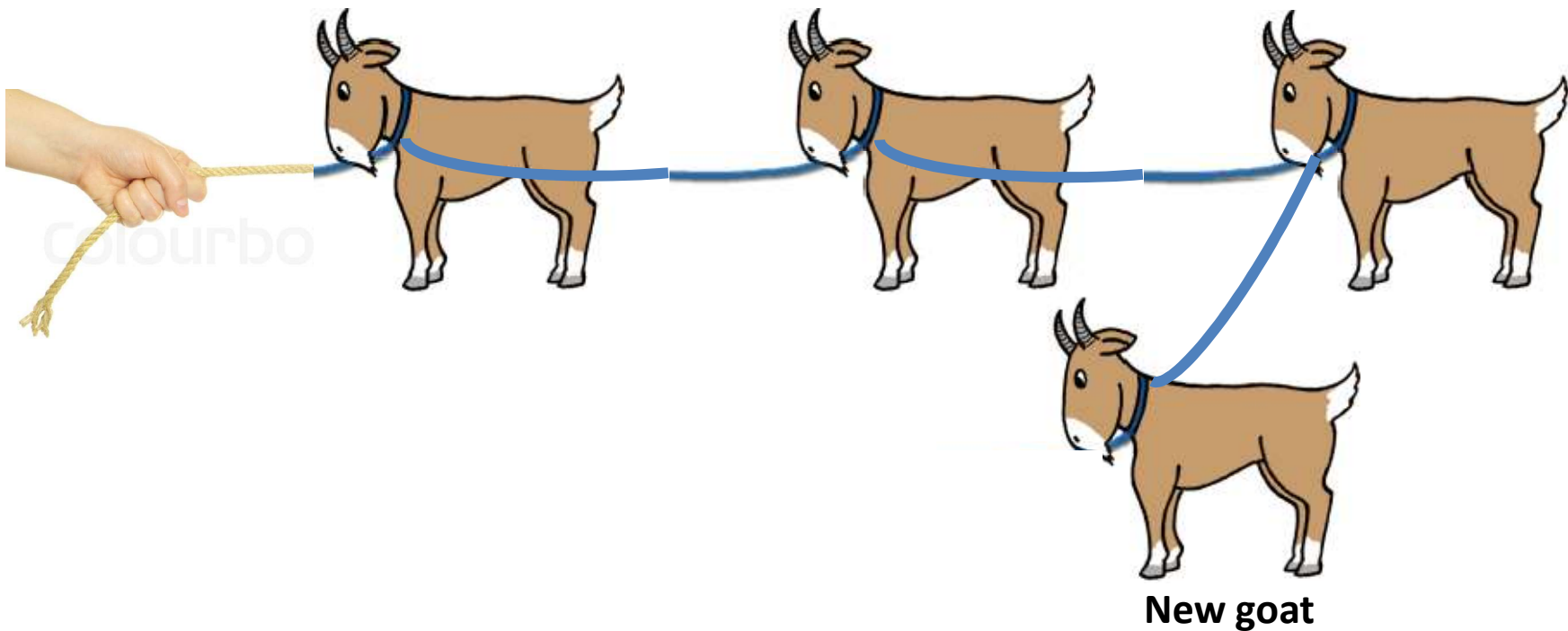
- Suppose a new goat is needed to be Inserted between 2<sup>nd</sup> and 3<sup>rd</sup> goats.
- Step 1: Bring a new goat
- Step 2:

# Goat List : INSERT



- Suppose a new goat is needed to be Inserted between 2<sup>nd</sup> and 3<sup>rd</sup> goats.
- Step 1: Bring a new goat
- Step 2: Bind 3<sup>rd</sup> one to new goat

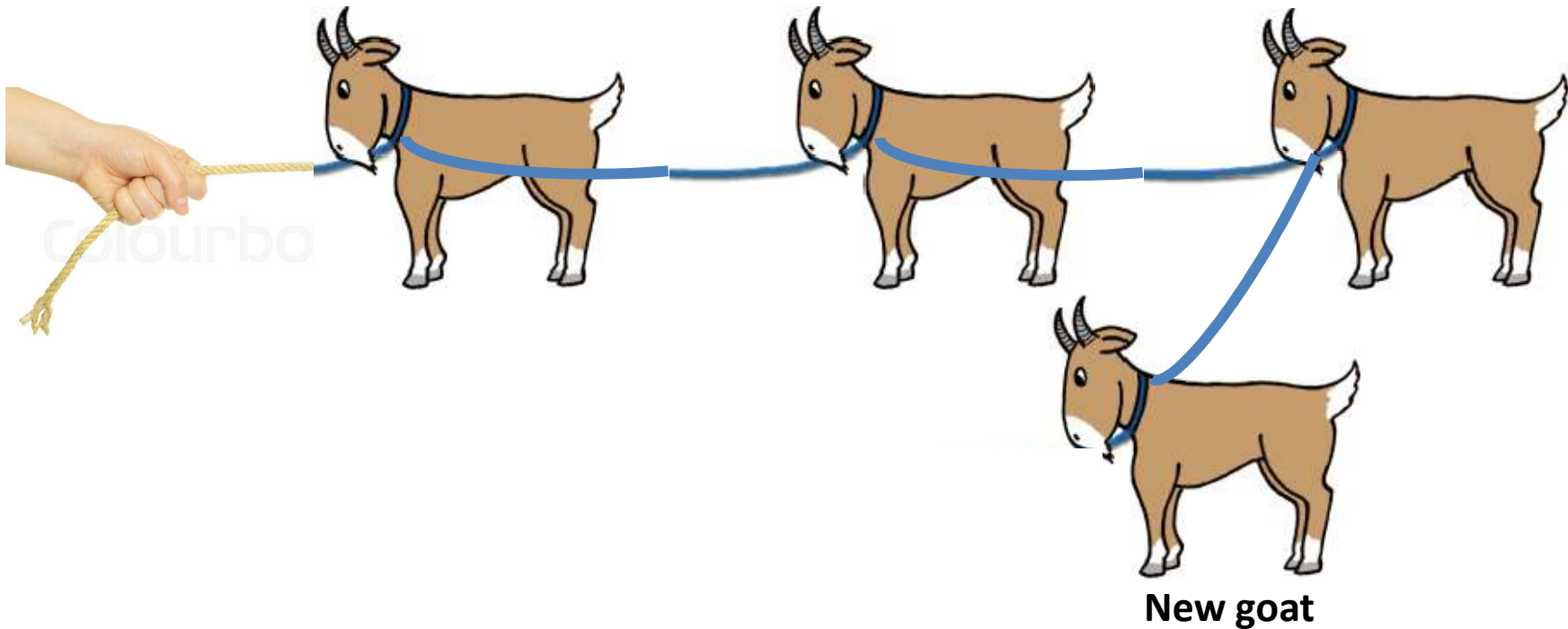
# Goat List : **INSERT**



- Suppose a new goat is needed to be Inserted between 2<sup>nd</sup> and 3<sup>rd</sup> goats.
- Step 1: Bring a new goat
- Step 2: Bind 3<sup>rd</sup> one to new goat

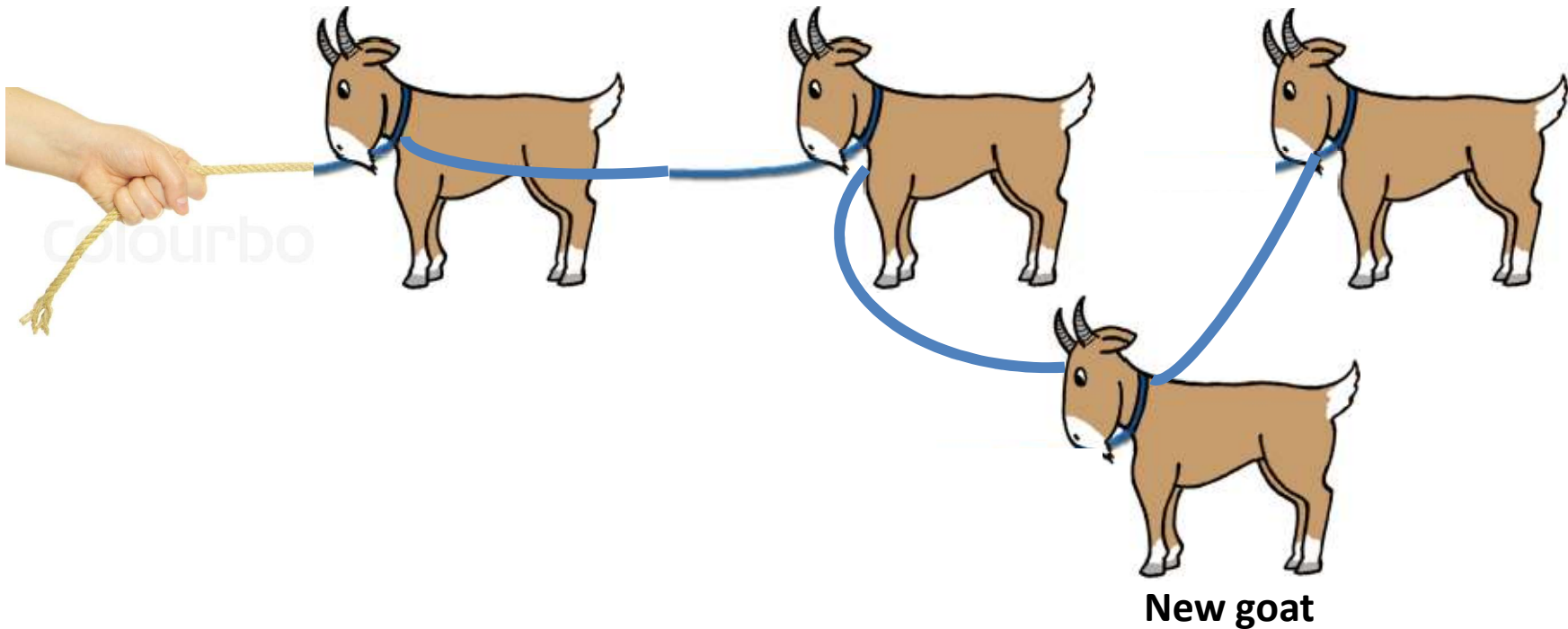


# Goat List : INSERT



- Suppose a new goat is needed to be Inserted between 2<sup>nd</sup> and 3<sup>rd</sup> goats.
  - Step 1: Bring a new goat
  - Step 2: Bind 3<sup>rd</sup> one to new goat
  - Step 3: Bind new one to 2<sup>nd</sup> one

# Goat List : **INSERT**



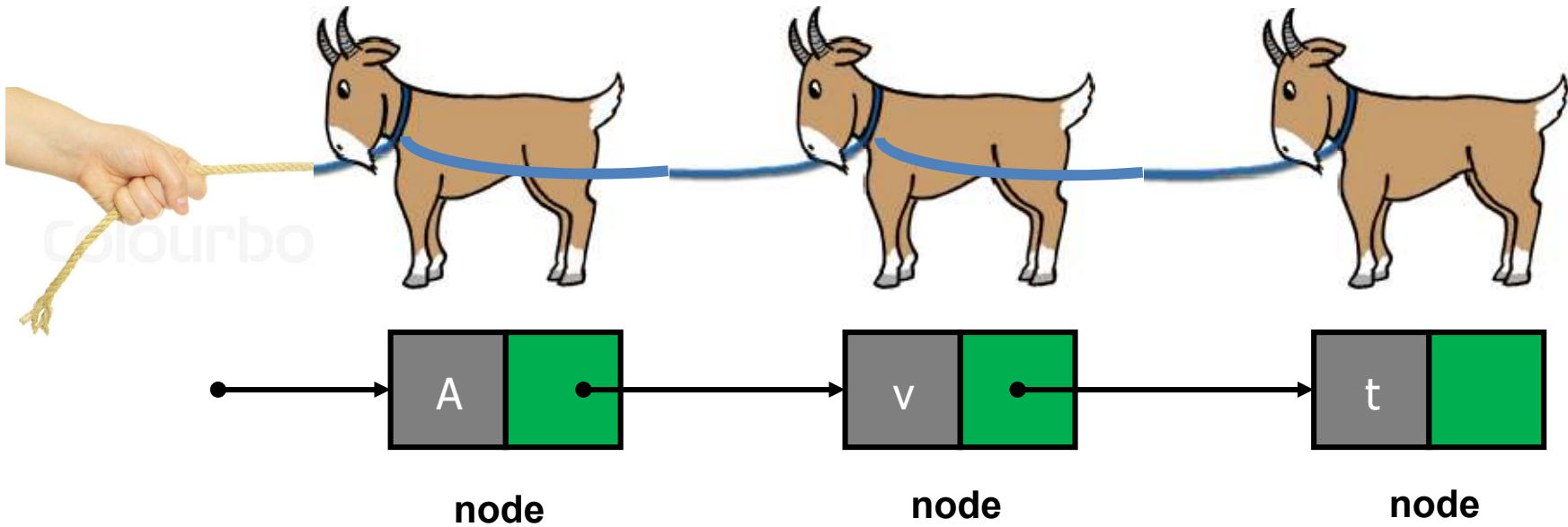
- Suppose a new goat is needed to be Inserted between 2<sup>nd</sup> and 3<sup>rd</sup> goats.
  - Step 1: Bring a new goat
  - Step 2: Bind 3<sup>rd</sup> one to new goat
  - Step 3: Bind new one to 2<sup>nd</sup> one

# Basics of Linked List

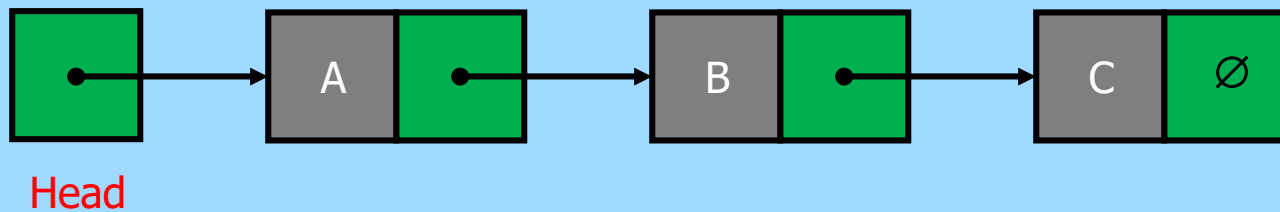
- Linked lists
    - Abstract data type (ADT)
  - Basic operations of linked lists
    - Insert, find, delete, print, etc.
  - Variations of linked lists
    - Circular linked lists
    - Doubly linked lists
- 
- A linked list can easily grow or shrink in size.
  - Insertion and deletion of nodes is quicker with linked lists than with arrays.



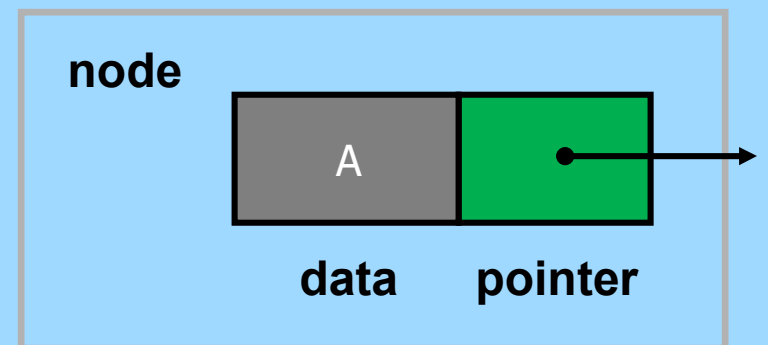
# Goat List : Link List



# Basics of Linked List

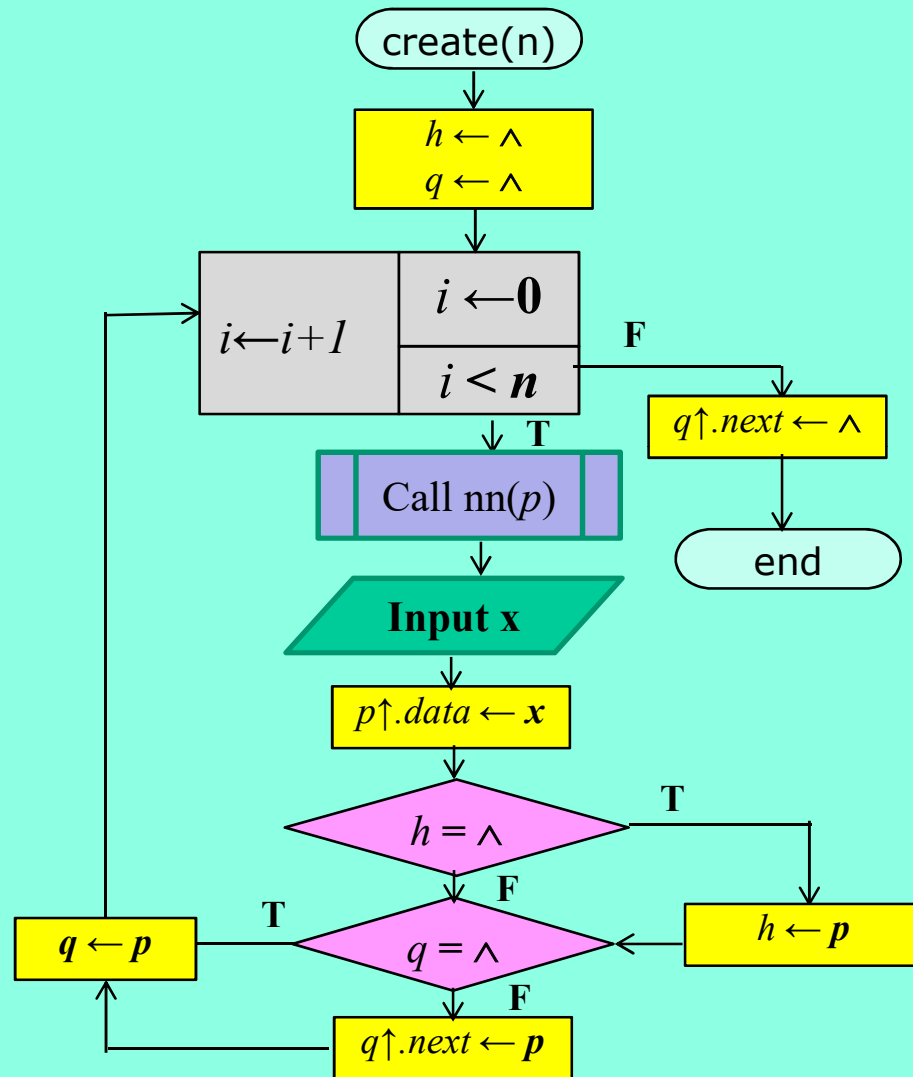


- A **linked list** is a series of connected **nodes**
- Each node contains at least
  - A piece of data (any type)
  - Pointer to the next node in the list
- **Head**: pointer to the first node
- The last node points to NULL

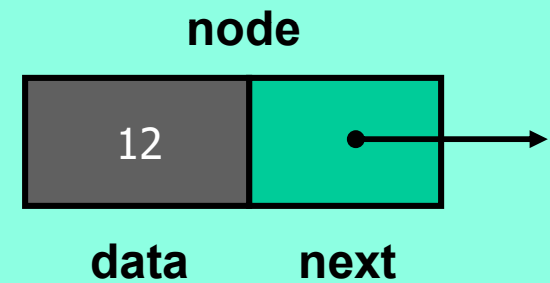


# Single Linked List

**Topic 1: Write an Algorithm to create a single linked list with  $n$  nodes.**



$n$ : total nodes  
 $x$ : input variable  
 $nn(p)$ : a function, call a new node pointed by  $p$   
 $\wedge$ : NULL

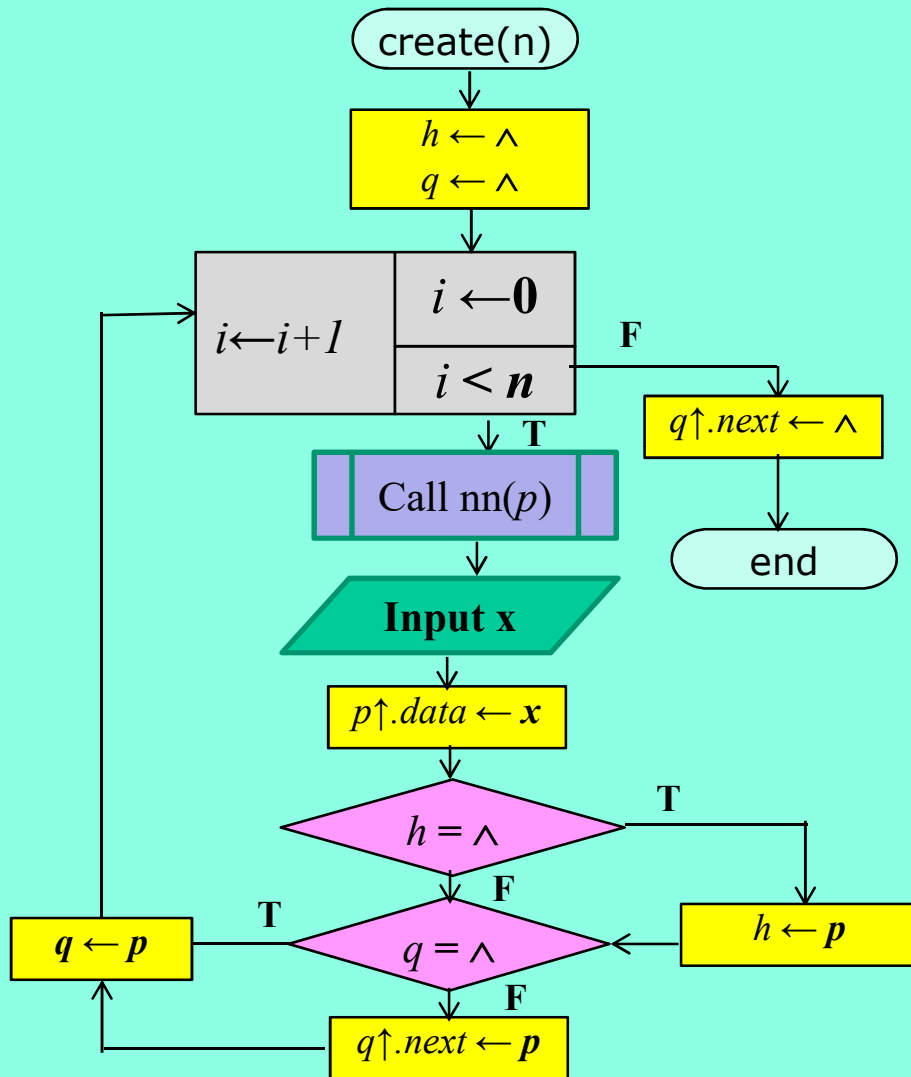


```

class node{
public:
    int data;
    node *next;
};
    
```

# Basics of Linked List

**Topic 1: Write an Algorithm to create a single linked list with n nodes.**

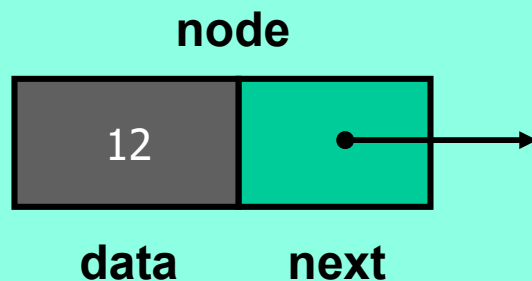
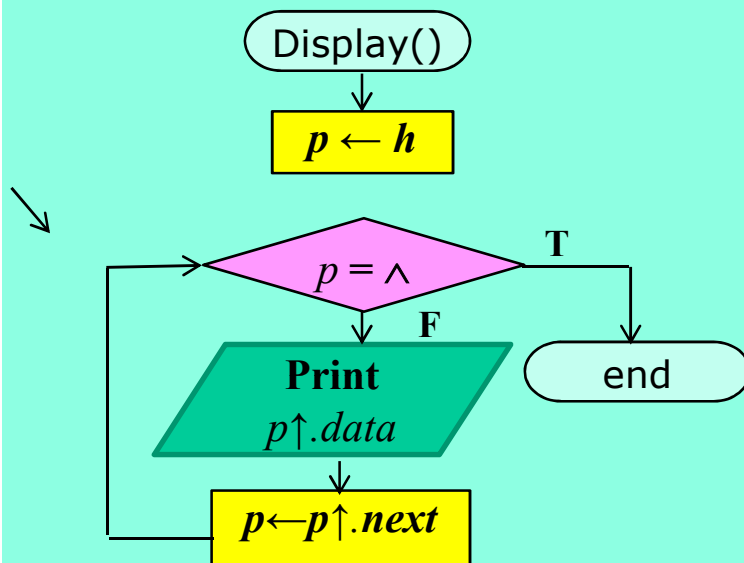


```

void create(int n){
    Node *p,*q;
    int i,x;
    h=NULL; q=NULL;
    for(i=0;i<n;i++){
        p= new Node();
        cout<<"Enter X:";
        cin>>x;
        p->data=x;
        if(h==NULL)
            h=p;
        if(q!=NULL)
            q->next=p;
        q=p;
    }
    q->next=0;
}
  
```

# Traversing a Single Linked List

**Topic 1: Write an Algorithm to traverse a single linked list.**



$h$ : address of first/head node  
 $p$ : pointer to a node  
 $p \uparrow .data$ : data field value of  $p$  pointed node  
 $p \uparrow .next$ : address of the next of  $p$  pointed node

```
void display(){
    Node *p;
    p=h;
    cout<<"\nThe list is..\n";
    while(p!=0){
        cout<<"-->"<<p->data;
        p=p->next;
    }
}
```



# C++ program for Linked List

Topic 1: Write an Algorithm to create a single linked list with n nodes.

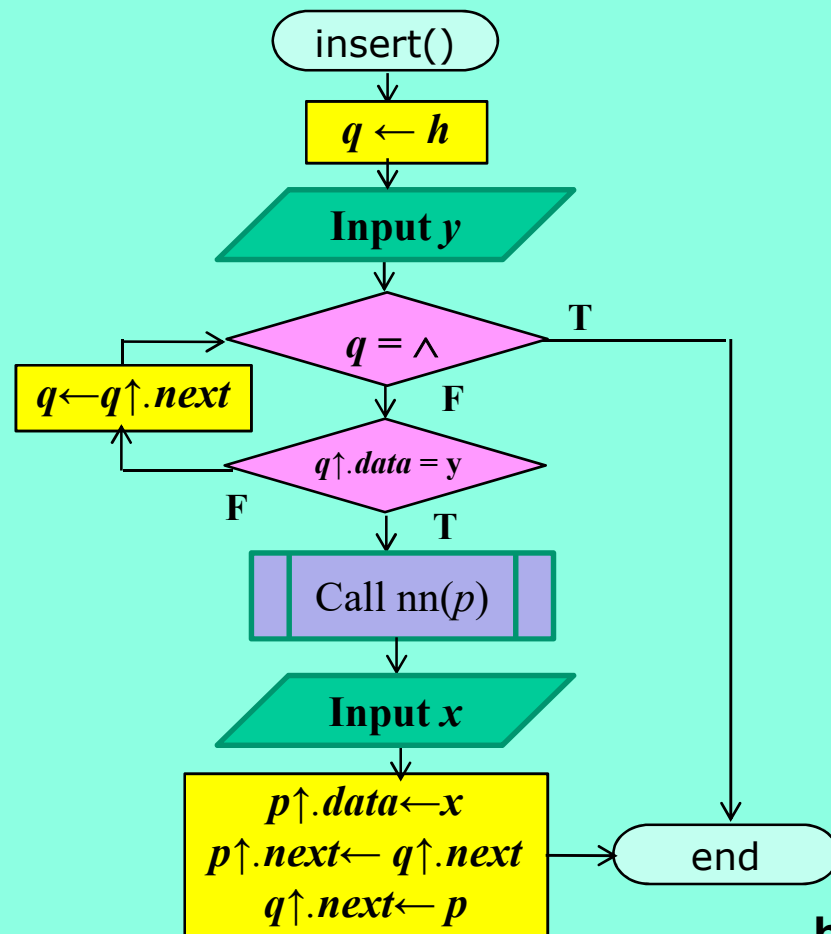
```
class Node{  
    public:  
        int data;  
        Node *next;  
};  
Node *h;
```

```
Node *h;  
class LinkedList{  
    public:  
        void create(int n){  
            //copy previous-----  
        }  
        void display(){  
            copy previous-----  
        }  
};
```

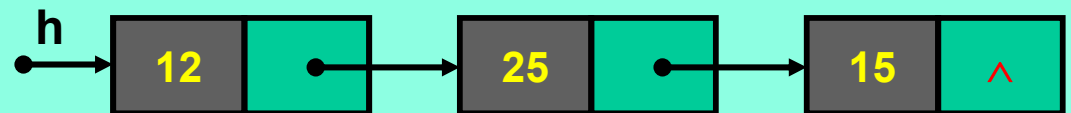
```
int main()  
{  
    LinkedList ll;  
    ll.create(4);  
    ll.display();  
}
```

# Inserting into a Single Linked List

**Topic 3: Write an Algorithm to insert a new node in a existing single linked list.**

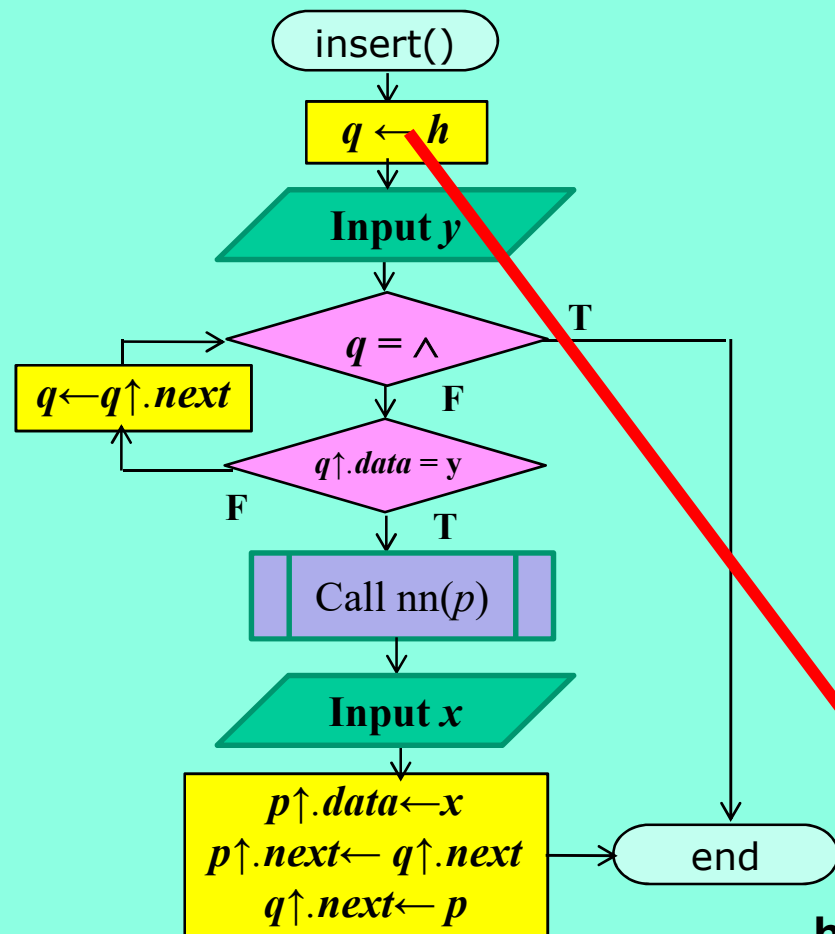


$h$ : address of first/head node  
 $p$ : pointer to a new node  
 $q \uparrow .data$ : address of the next of  $q$  pointed node  
 $q \uparrow .next$ : address of the next of  $q$  pointed node  
 $x$ : input data for new node  
 $y$ : target node, after  $y$  node  $x$  to be inserted

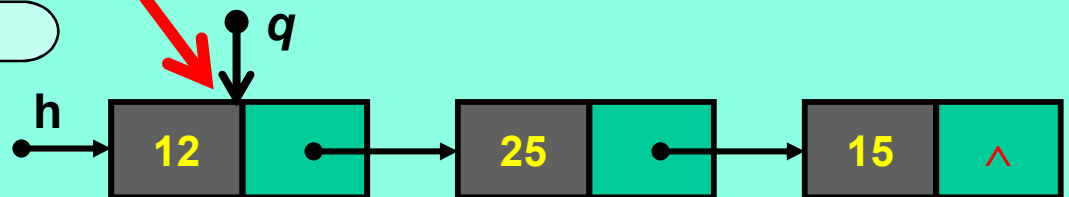


# Inserting into a Single Linked List

**Topic 3: Write an Algorithm to insert a new node in a existing single linked list.**

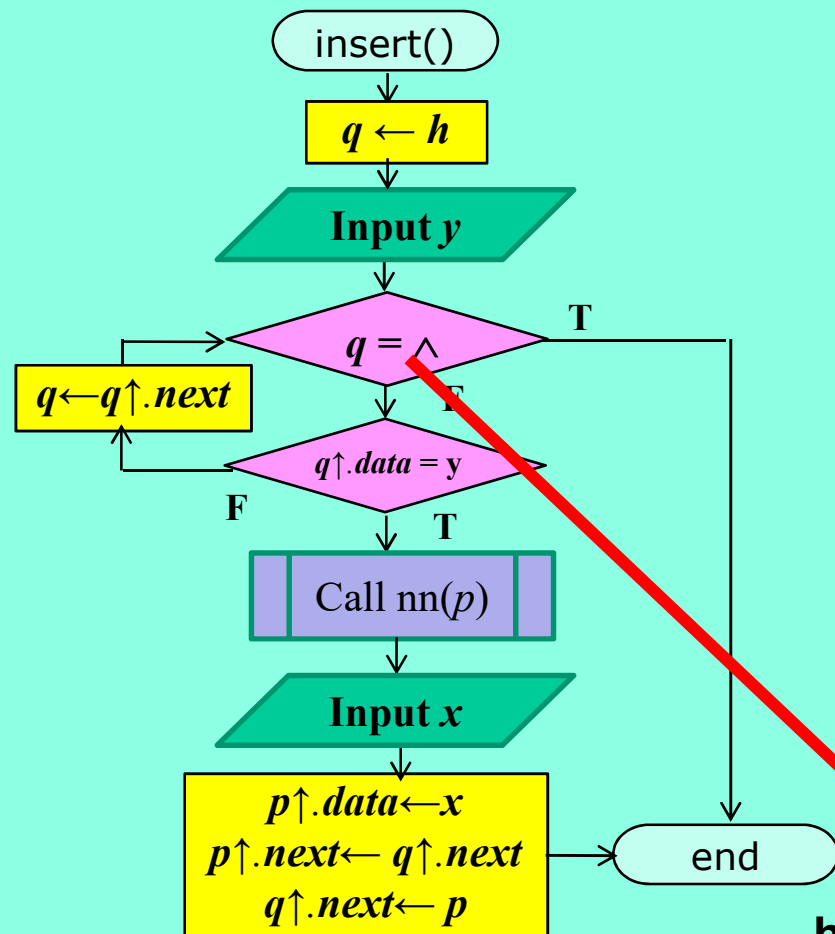


$h$ : address of first/head node  
 $p$ : pointer to a new node  
 $q \uparrow .data$ : address of the next of  $q$  pointed node  
 $q \uparrow .next$ : address of the next of  $q$  pointed node  
 $x$ : input data for new node  
 $y$ : target node, after  $y$  node  $x$  to be inserted

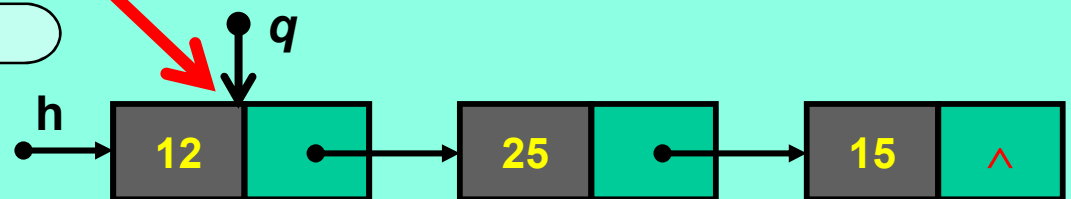


# Inserting into a Single Linked List

**Topic 3: Write an Algorithm to insert a new node in a existing single linked list.**

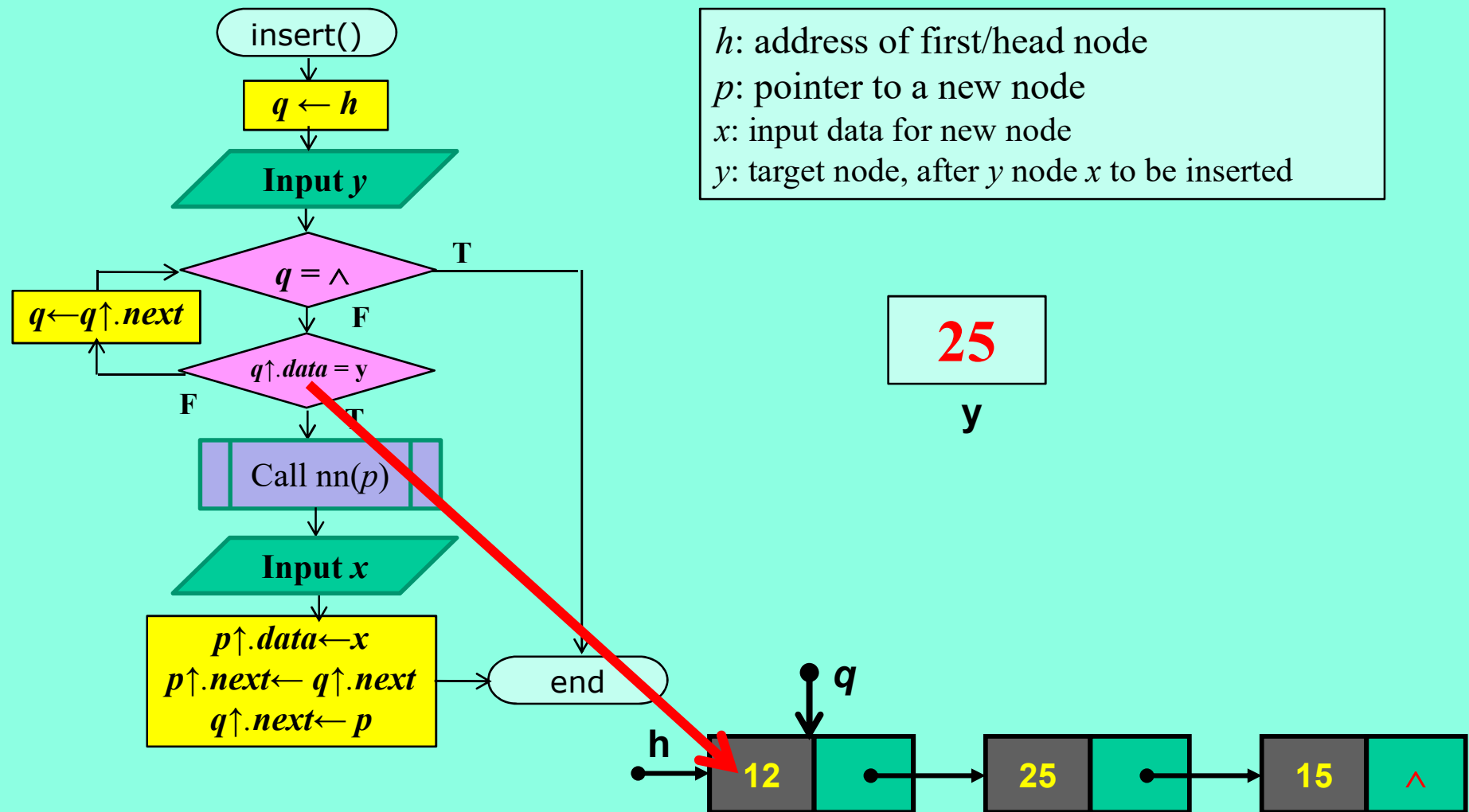


$h$ : address of first/head node  
 $p$ : pointer to a new node  
 $q \uparrow .data$ : address of the next of  $q$  pointed node  
 $q \uparrow .next$ : address of the next of  $q$  pointed node  
 $x$ : input data for new node  
 $y$ : target node, after  $y$  node  $x$  to be inserted



# Inserting into a Single Linked List

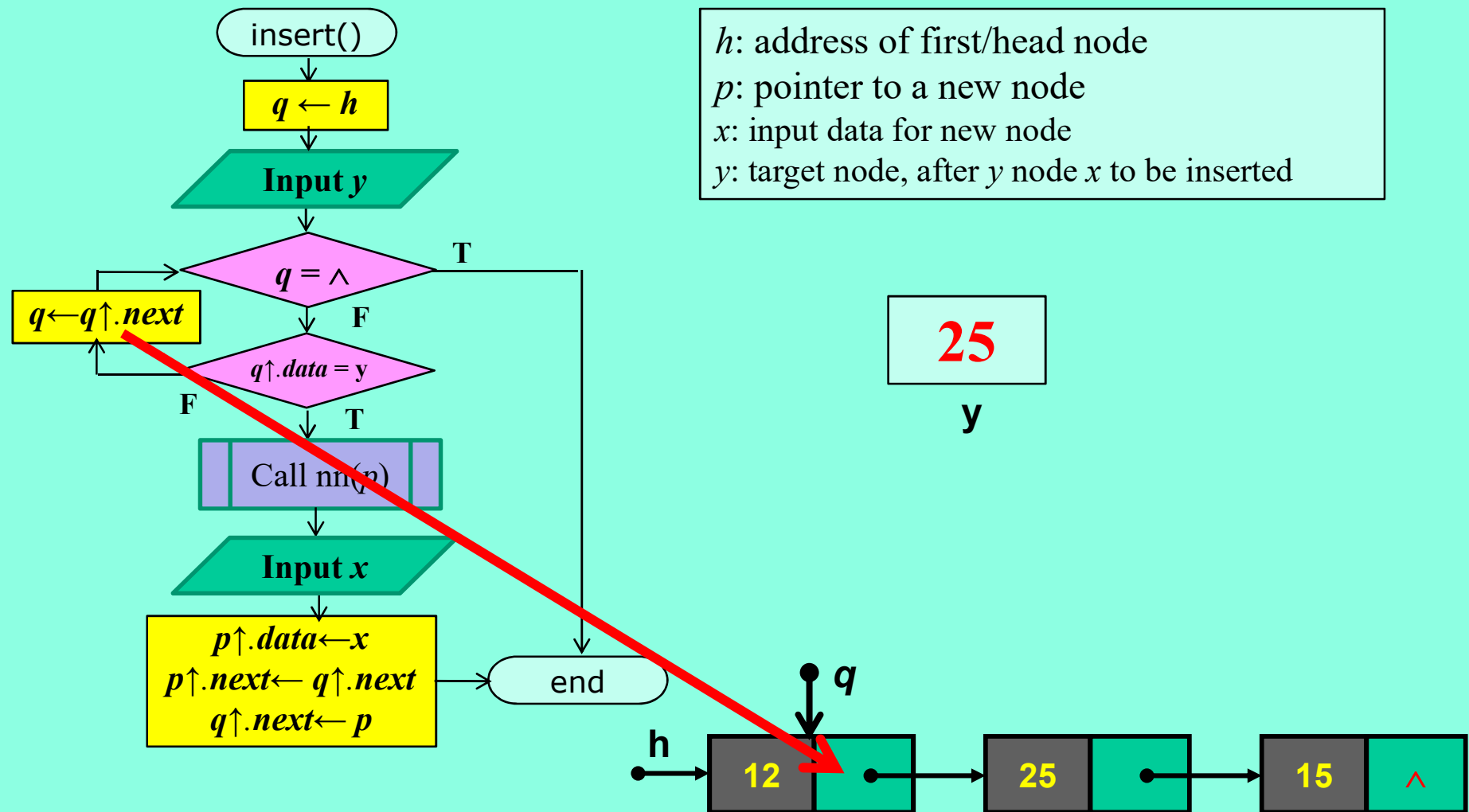
Topic 3: Write an Algorithm to insert a new node in a existing single linked list.





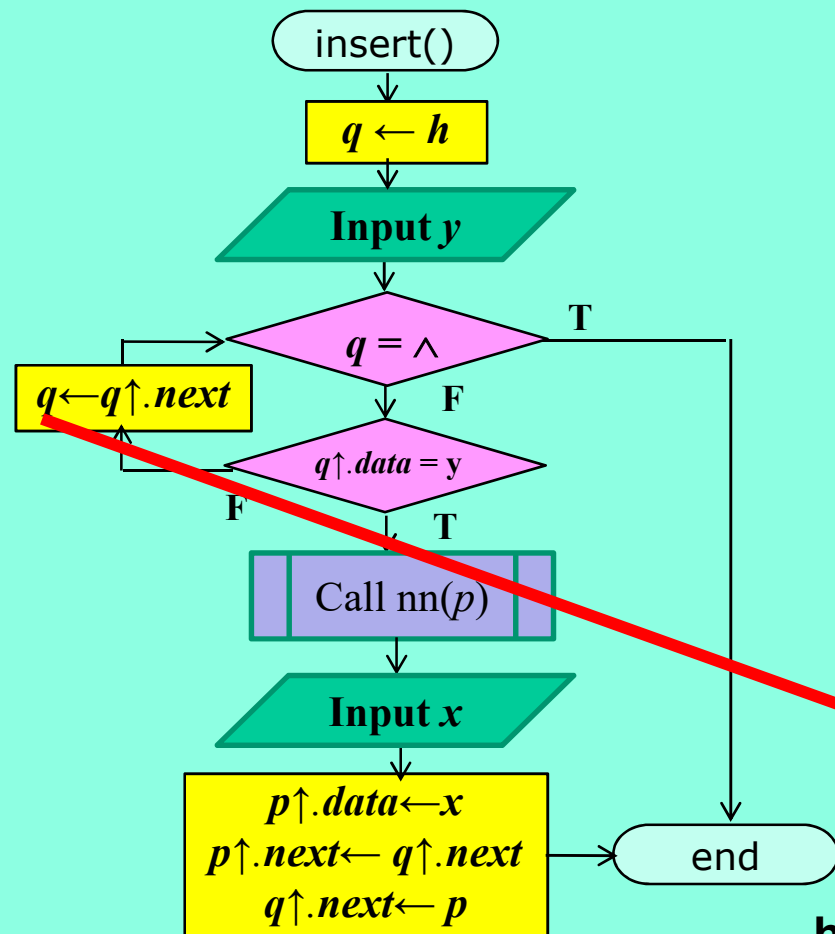
# Inserting into a Single Linked List

Topic 3: Write an Algorithm to insert a new node in a existing single linked list.



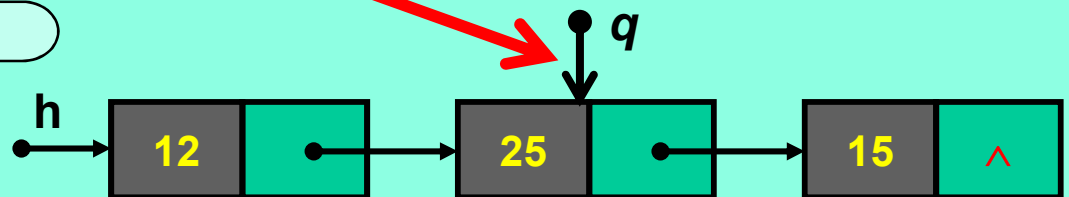
# Inserting into a Single Linked List

**Topic 3: Write an Algorithm to insert a new node in a existing single linked list.**



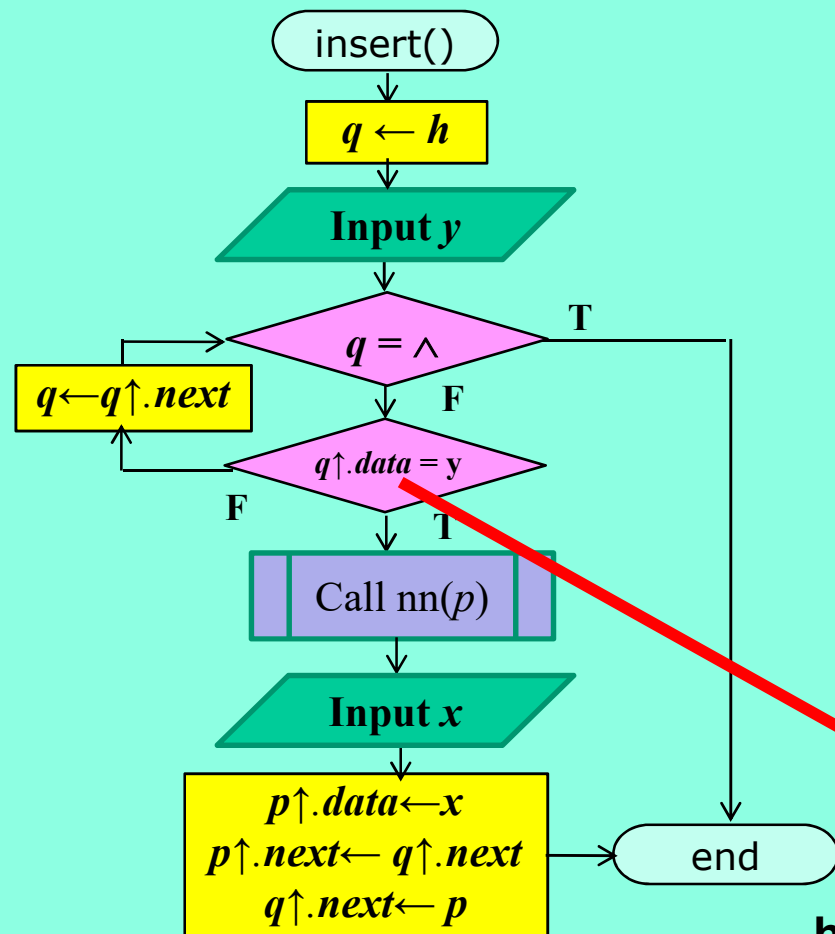
*h*: address of first/head node  
*p*: pointer to a new node  
*x*: input data for new node  
*y*: target node, after *y* node *x* to be inserted

25  
*y*

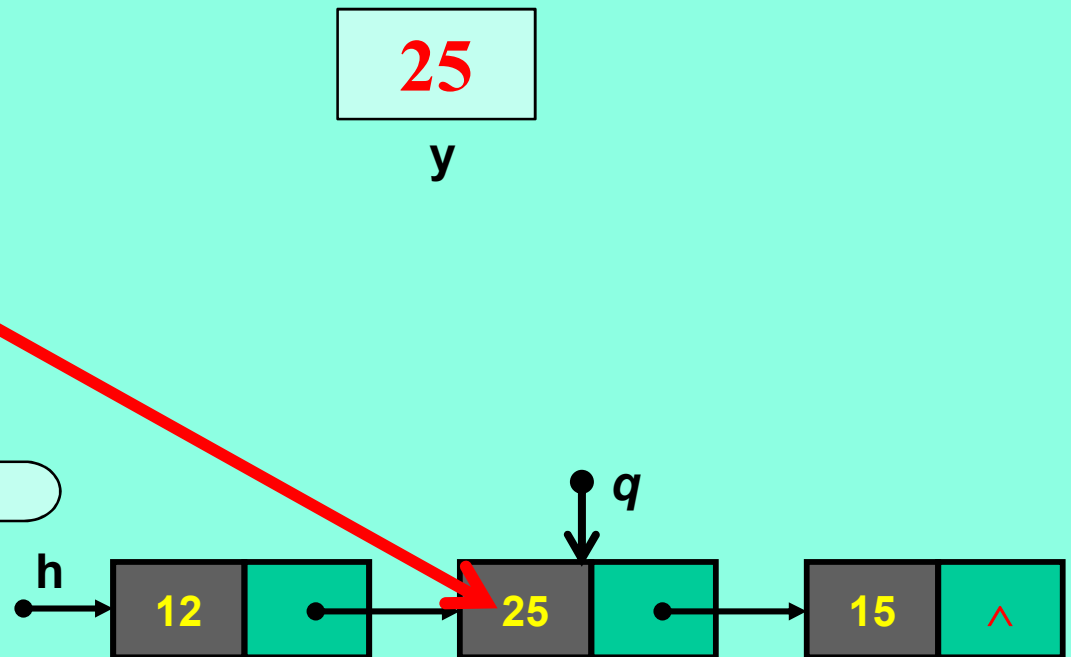


# Inserting into a Single Linked List

**Topic 3: Write an Algorithm to insert a new node in a existing single linked list.**

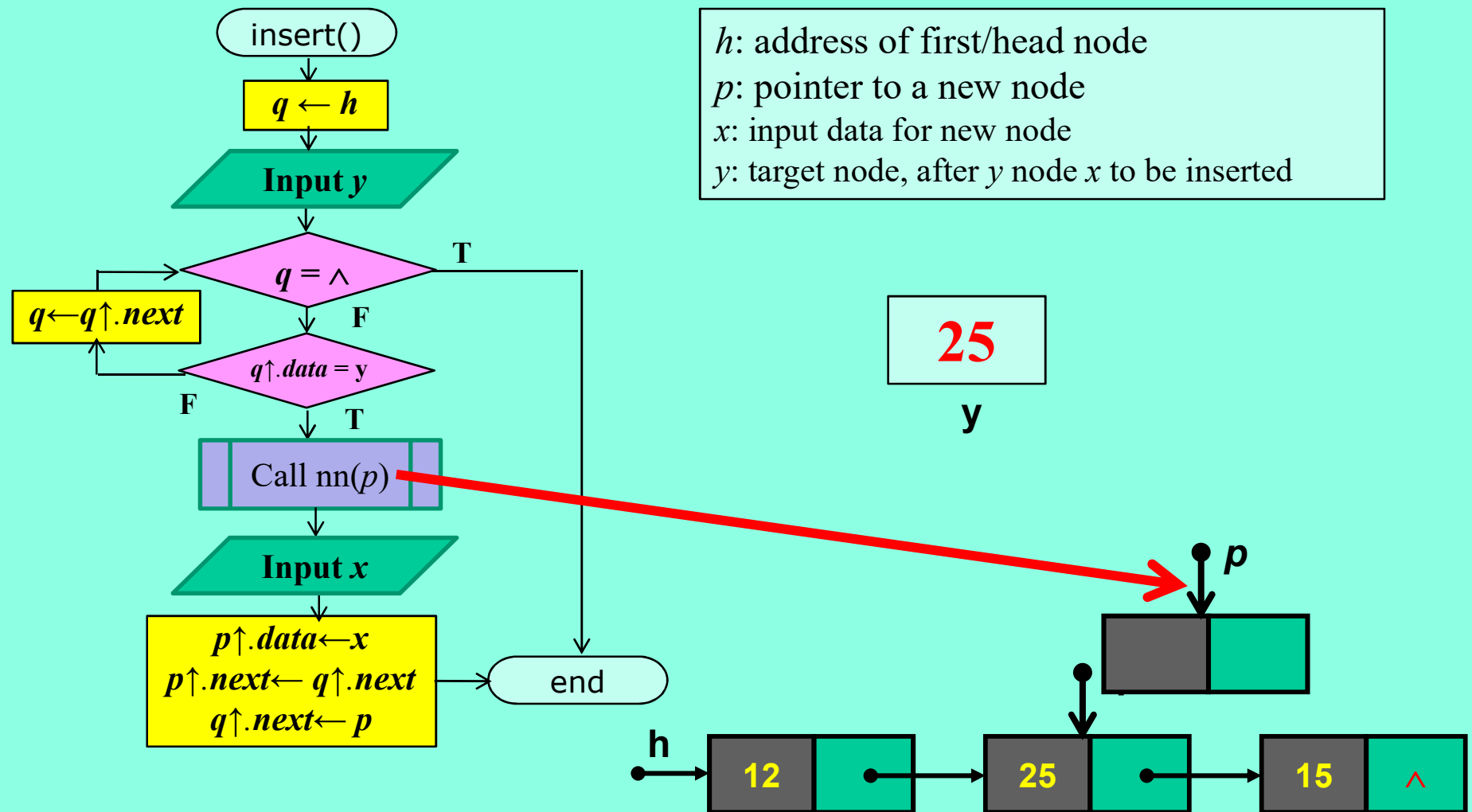


$h$ : address of first/head node  
 $p$ : pointer to a new node  
 $x$ : input data for new node  
 $y$ : target node, after  $y$  node  $x$  to be inserted



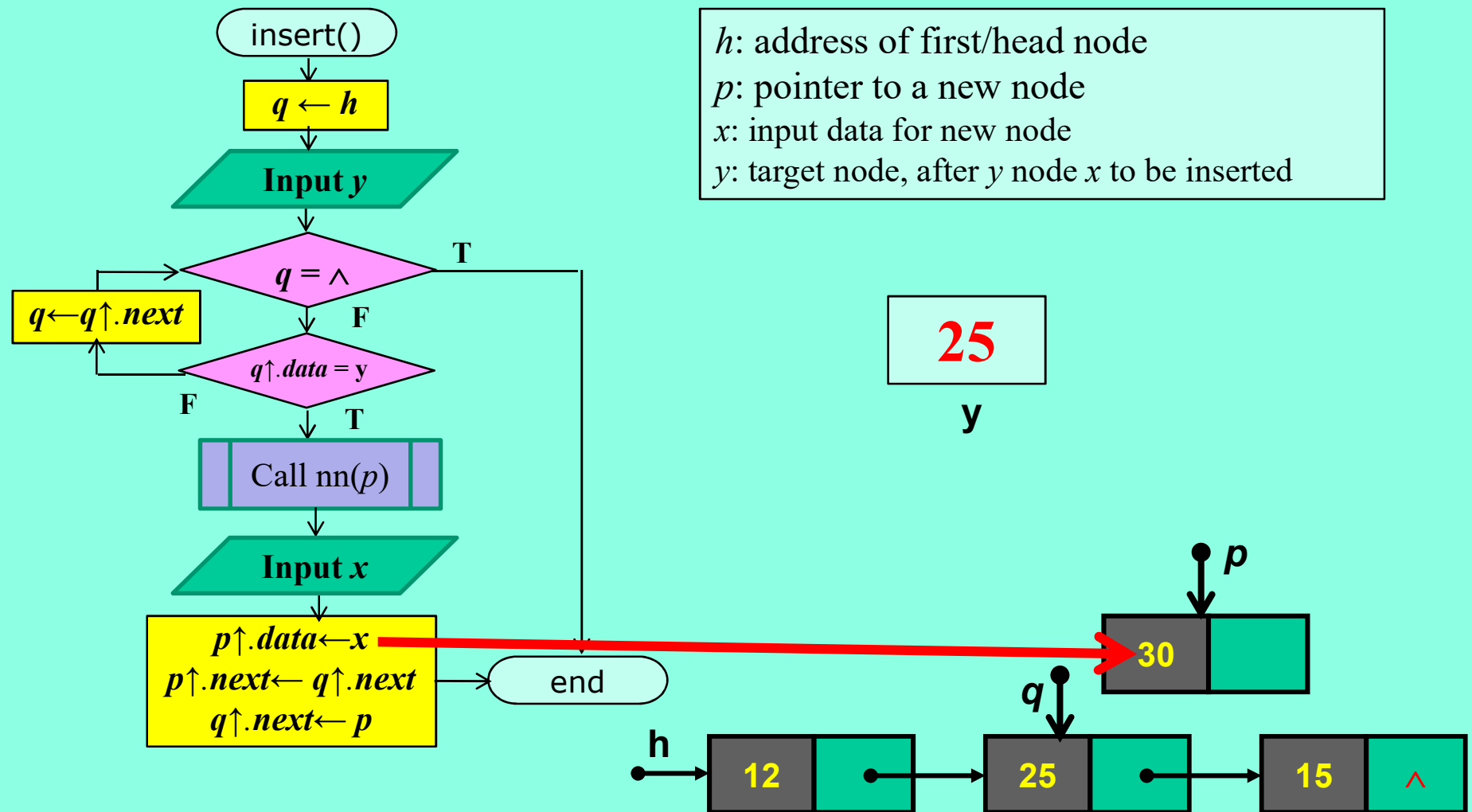
# Inserting into a Single Linked List

**Topic 3: Write an Algorithm to insert a new node in a existing single linked list.**



# Inserting into a Single Linked List

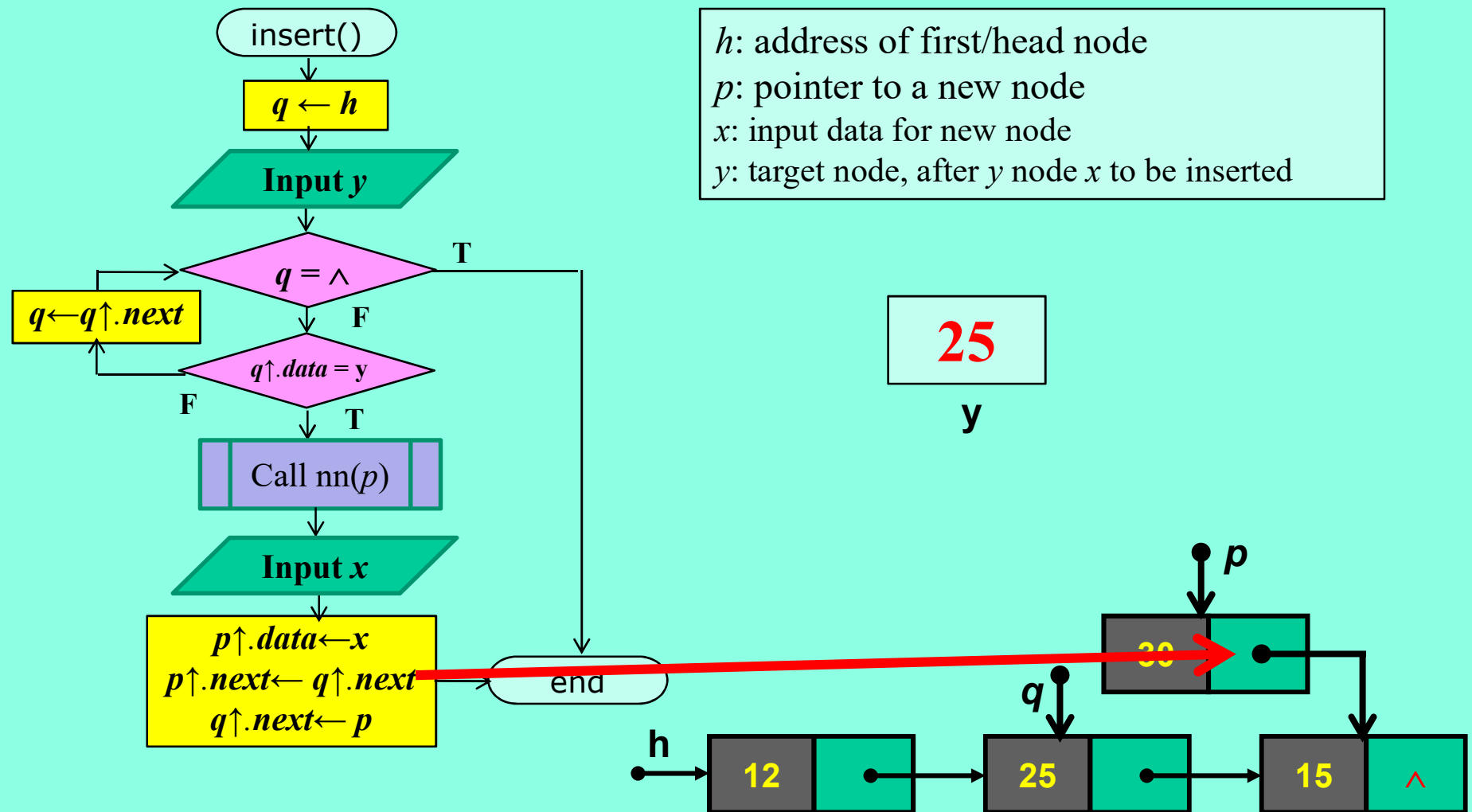
**Topic 3: Write an Algorithm to insert a new node in a existing single linked list.**





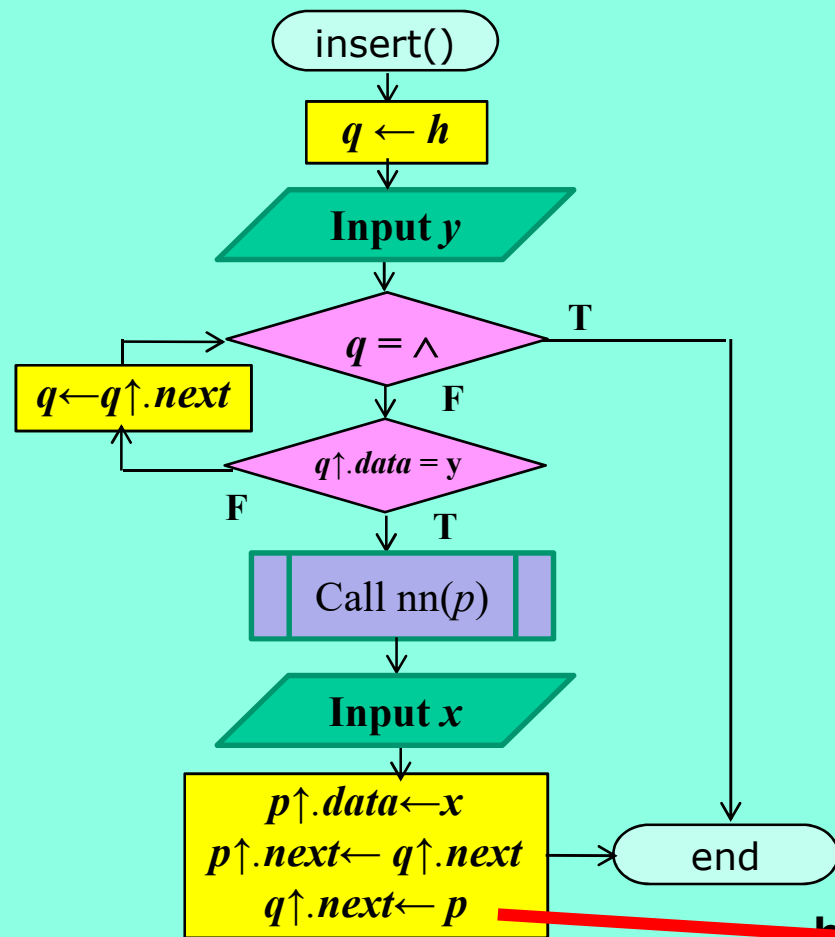
# Inserting into a Single Linked List

Topic 3: Write an Algorithm to insert a new node in a existing single linked list.



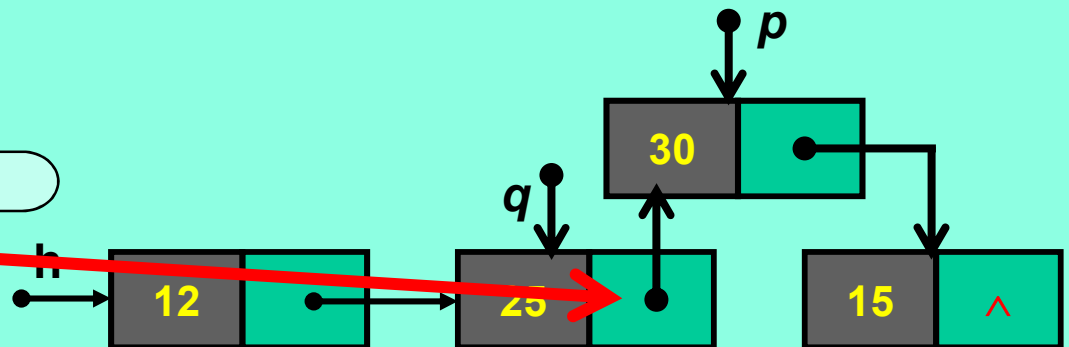
# Inserting into a Single Linked List

Topic 3: Write an Algorithm to insert a new node in a existing single linked list.



$h$ : address of first/head node  
 $p$ : pointer to a new node  
 $x$ : input data for new node  
 $y$ : target node, after  $y$  node  $x$  to be inserted

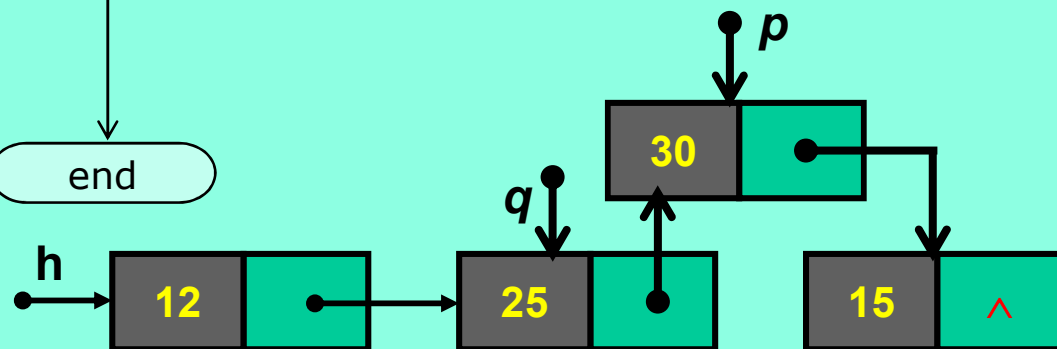
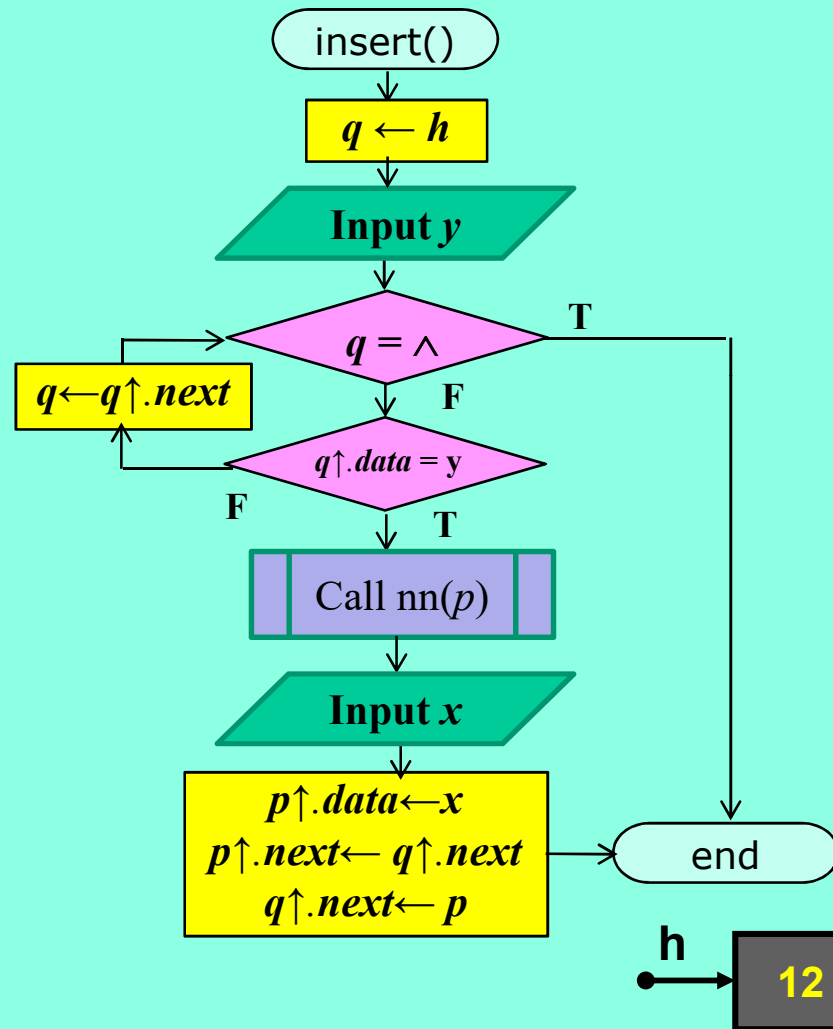
25  
 $y$



# Inserting into a Single Linked List

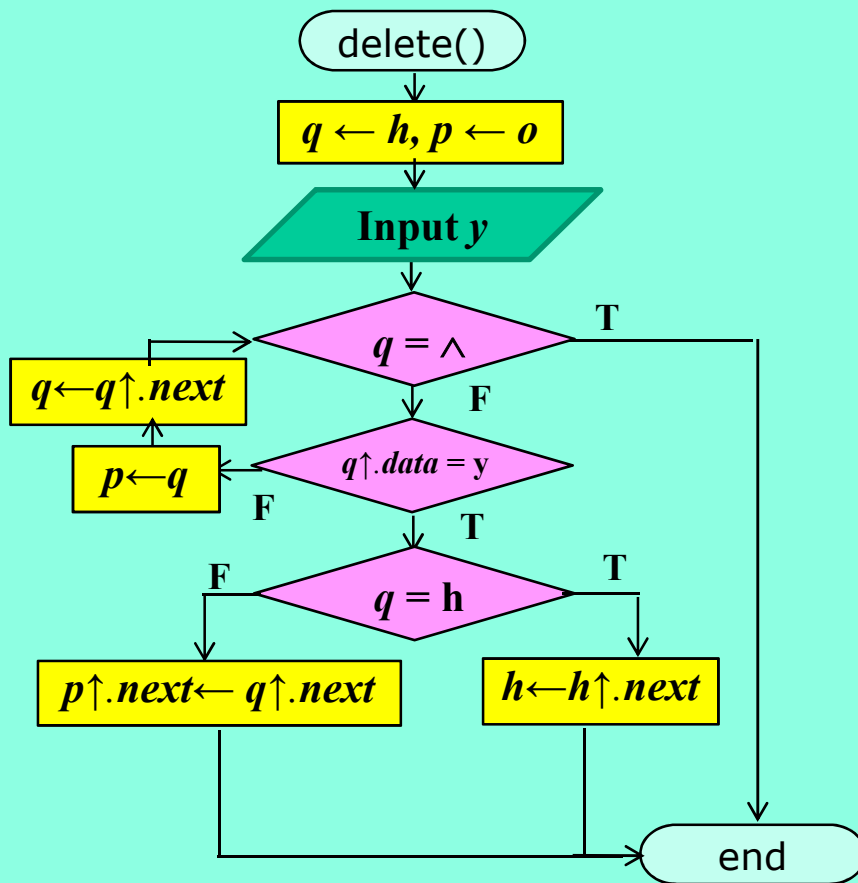
**Topic 3: Write an Algorithm to insert a new node in a existing single linked list.**

```
void Insert(){
    Node *q,*p;
    int x,y,flag=0;
    q=h;
    cout<<"\nAfter which element:";
    cin>>y;
    while(q!=0){
        if(q->data==y){
            p= new Node();
            cout<<"Enter data:";
            cin>>x;
            p->data=x; p->next=q->next; q->next=p;
            flag=1;
            break;
        }
        else
            q=q->next;
    }
    if(flag==0)
        cout<<"Sorry node NOT found"<<endl;
}
```

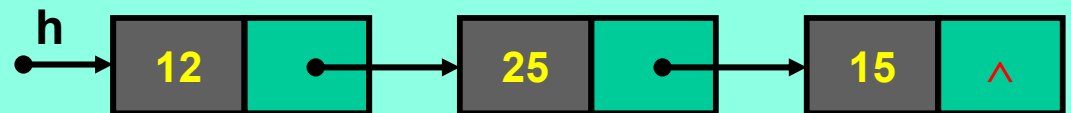


# Deleting in a Single Linked List

**Topic 3: Write an Algorithm to Delete an existing node in a single linked list.**

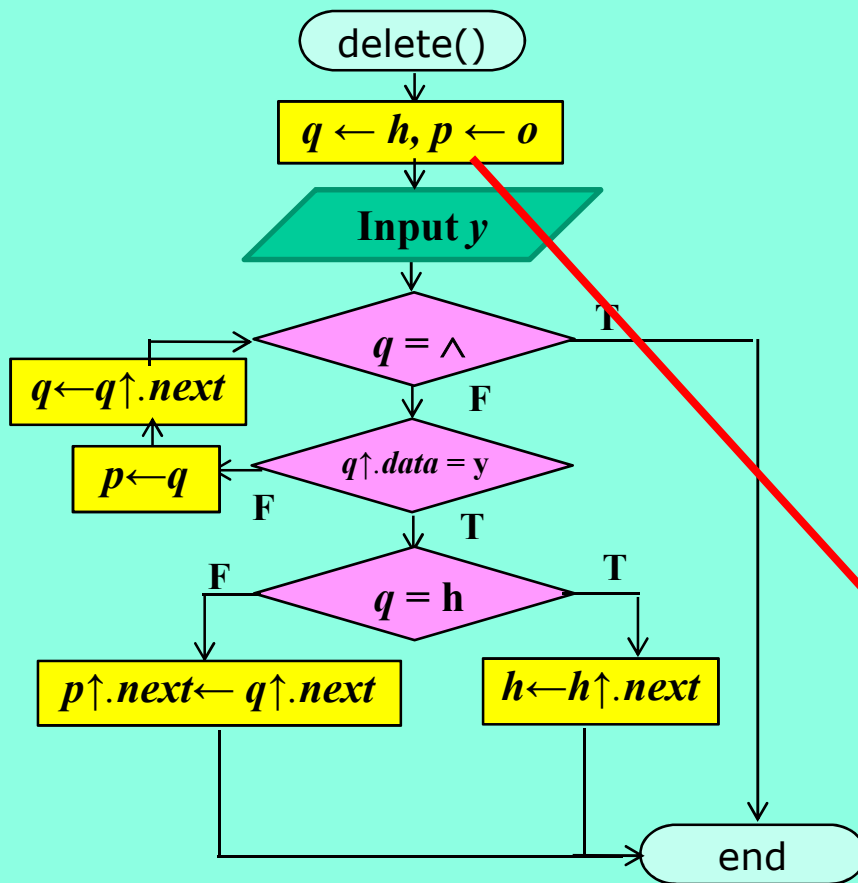


$h$ : address of first/head node  
 $q, p$ : pointer to a existing node  
 $y$ : target node to be deleted

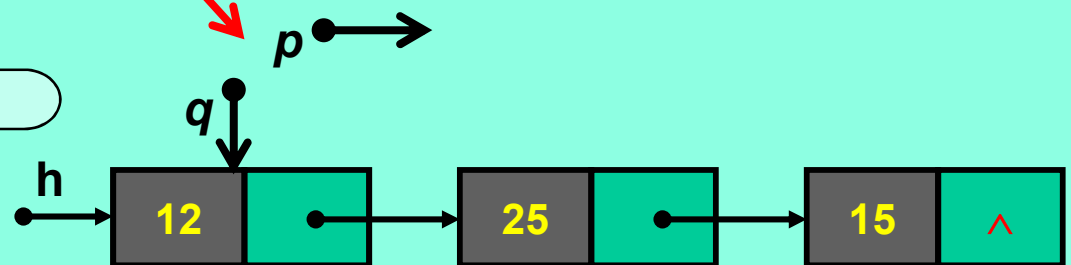


# Deleting in a Single Linked List

Topic 3: Write an Algorithm to Delete an existing node in a single linked list.

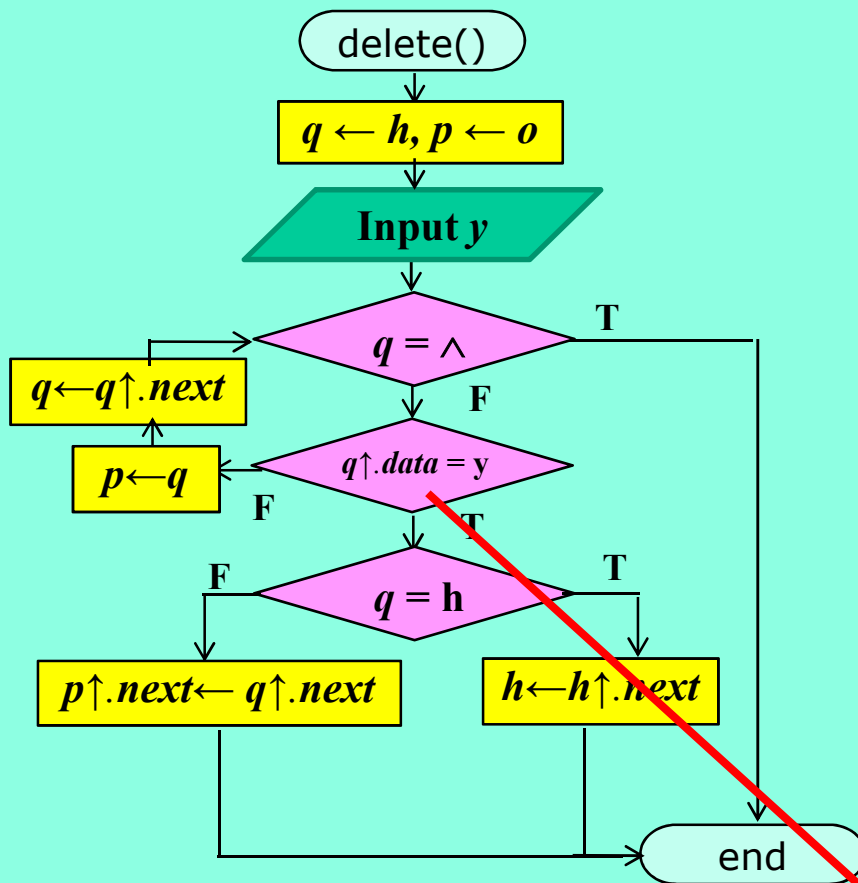


$h$ : address of first/head node  
 $q, p$ : pointer to a existing node  
 $y$ : target node to be inserted

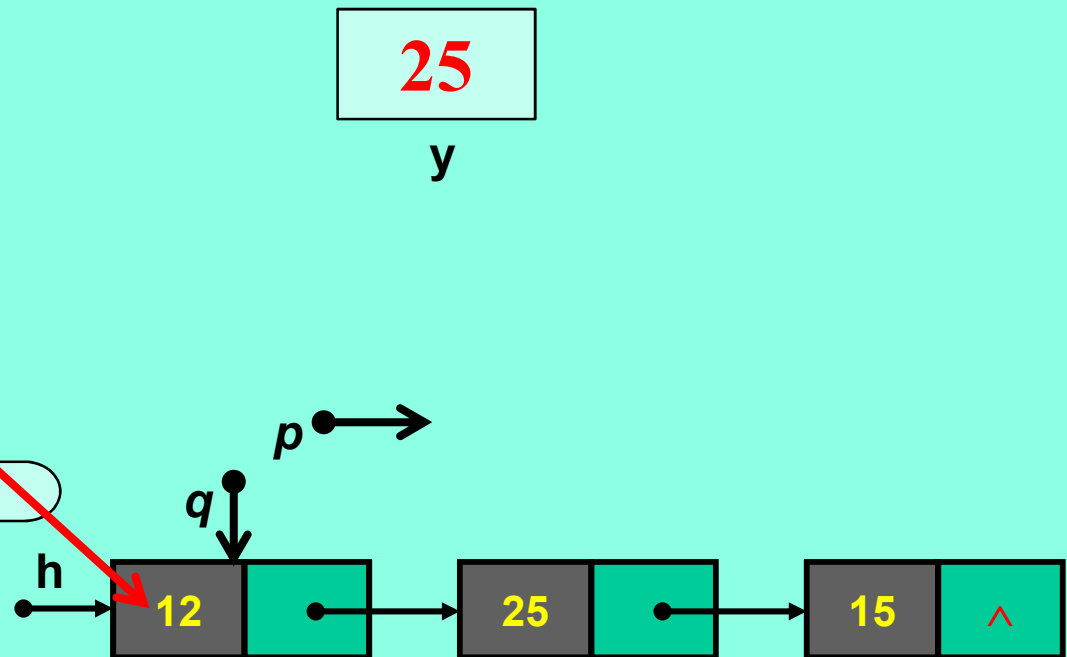


# Deleting in a Single Linked List

Topic 3: Write an Algorithm to Delete an existing node in a single linked list.



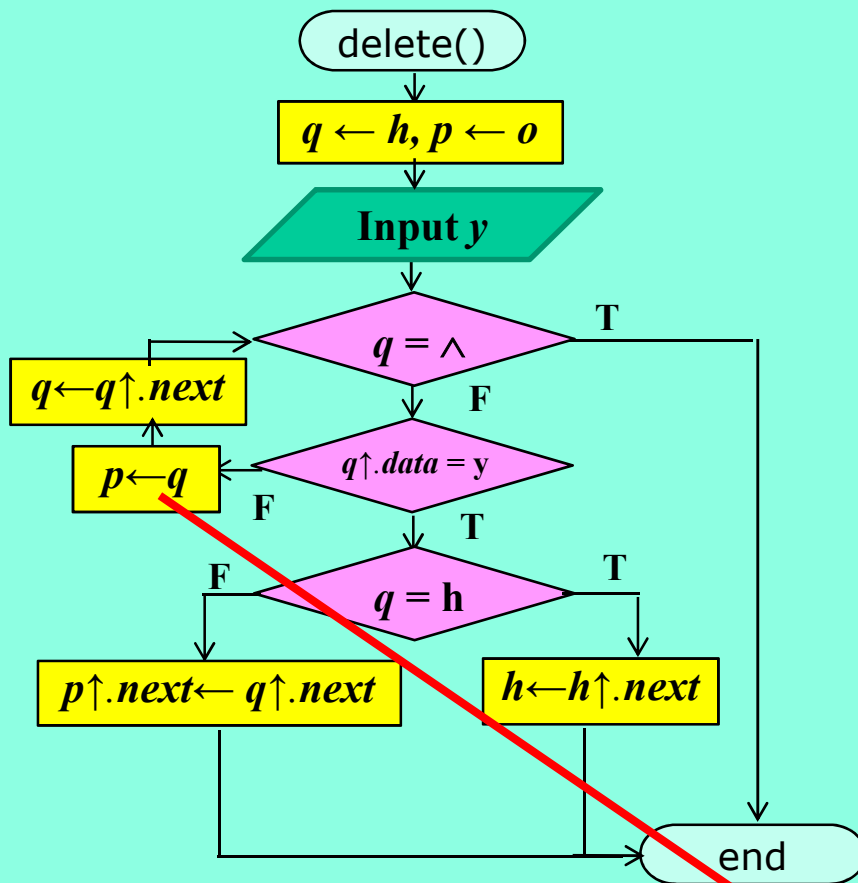
$h$ : address of first/head node  
 $q, p$ : pointer to a existing node  
 $y$ : target node to be inserted





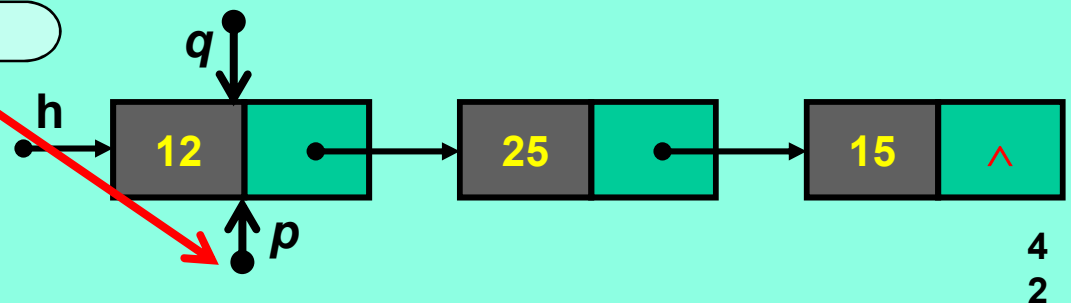
# Deleting in a Single Linked List

Topic 3: Write an Algorithm to Delete an existing node in a single linked list.



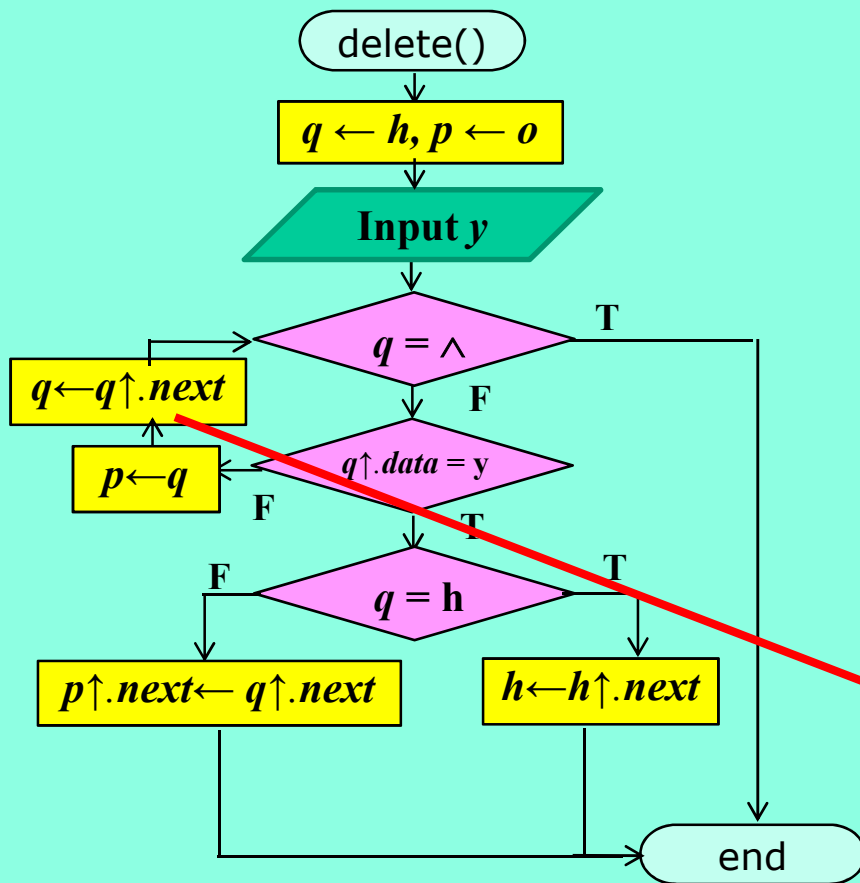
$h$ : address of first/head node  
 $q, p$ : pointer to a existing node  
 $y$ : target node to be inserted

25  
 $y$

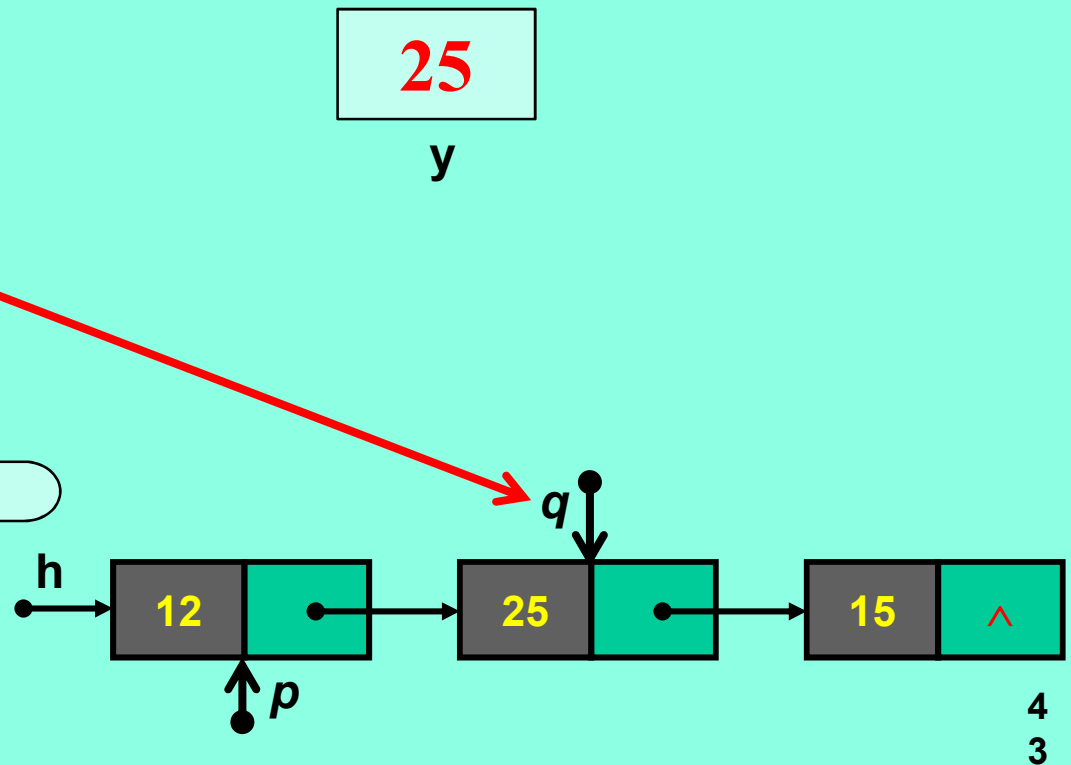


# Deleting in a Single Linked List

Topic 3: Write an Algorithm to Delete an existing node in a single linked list.

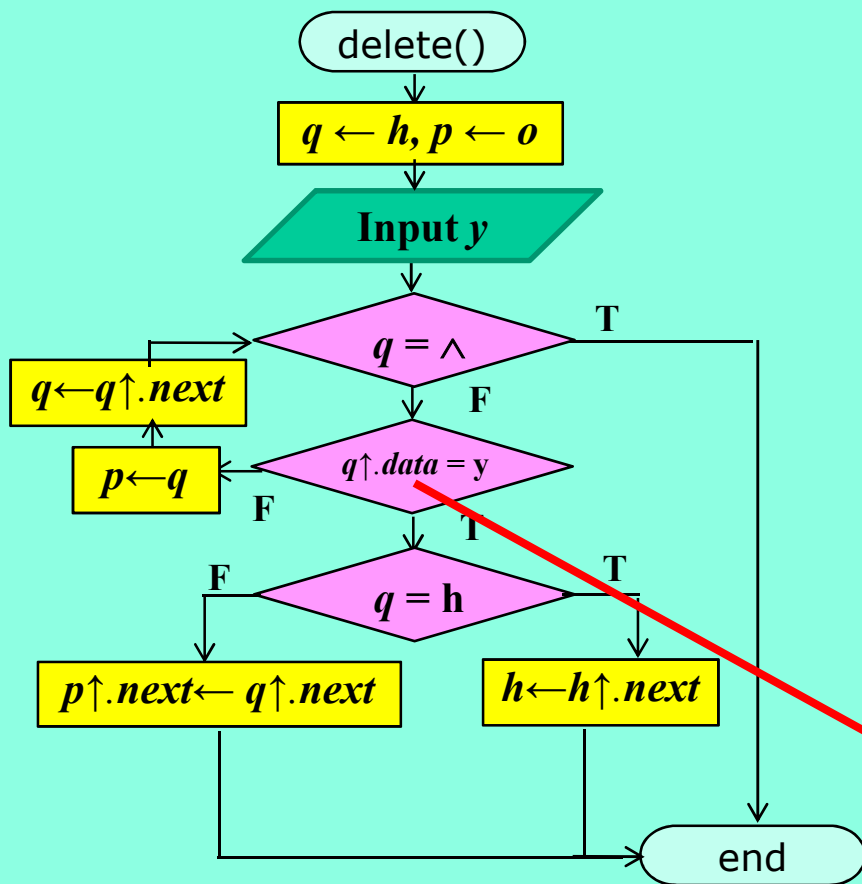


$h$ : address of first/head node  
 $q, p$ : pointer to a existing node  
 $y$ : target node to be inserted

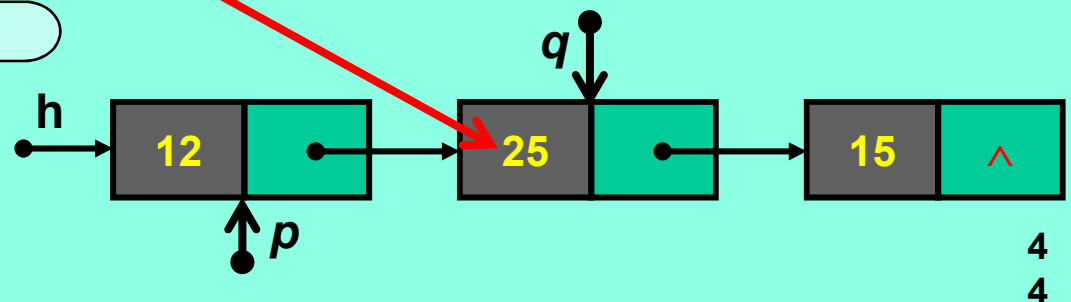


# Deleting in a Single Linked List

Topic 3: Write an Algorithm to Delete an existing node in a single linked list.

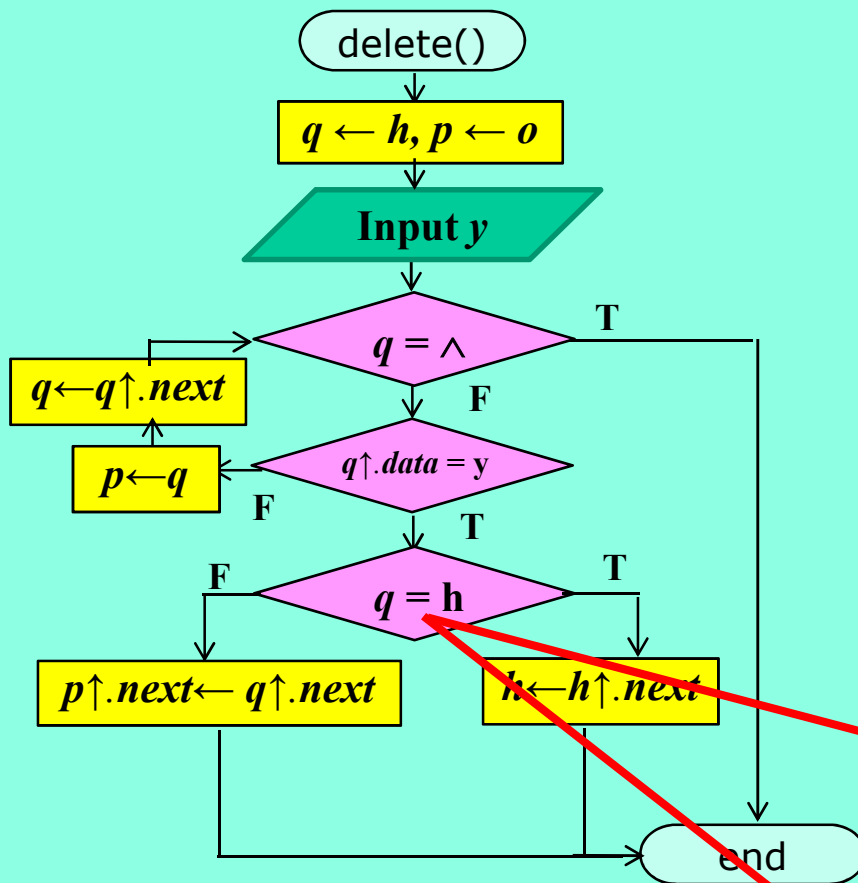


$h$ : address of first/head node  
 $q, p$ : pointer to a existing node  
 $y$ : target node to be inserted



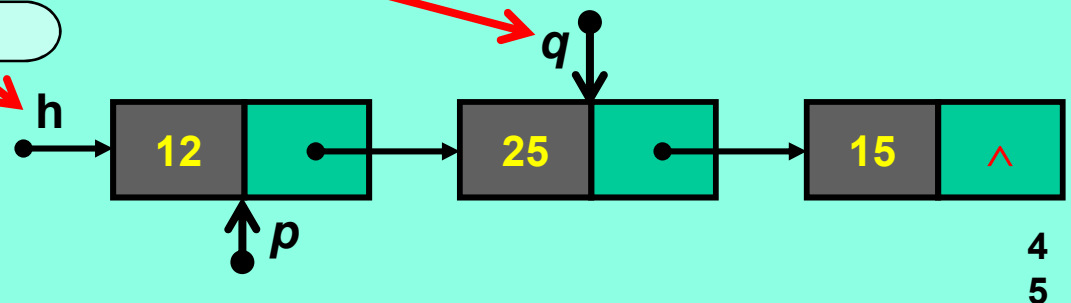
# Deleting in a Single Linked List

**Topic 3: Write an Algorithm to Delete an existing node in a single linked list.**



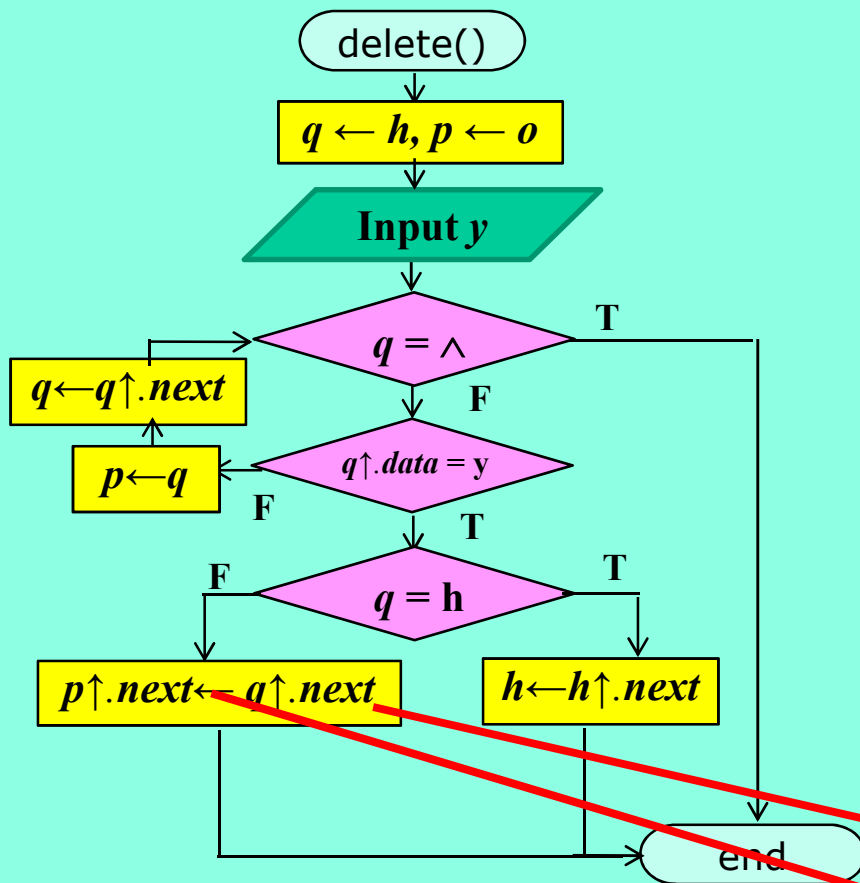
$h$ : address of first/head node  
 $q, p$ : pointer to a existing node  
 $y$ : target node to be inserted

25  
 $y$



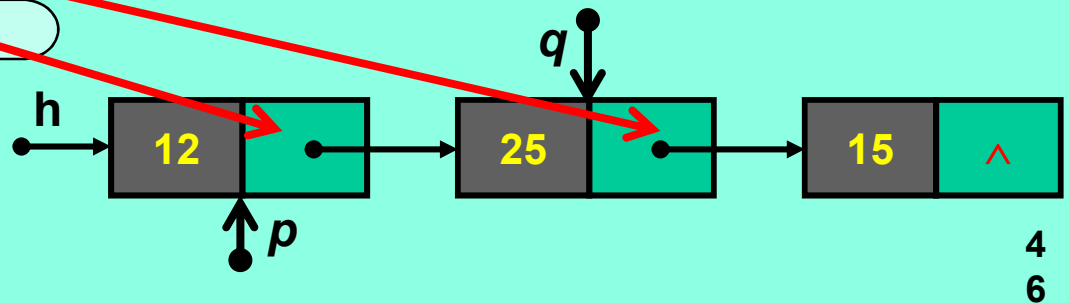
# Deleting in a Single Linked List

Topic 3: Write an Algorithm to Delete an existing node in a single linked list.



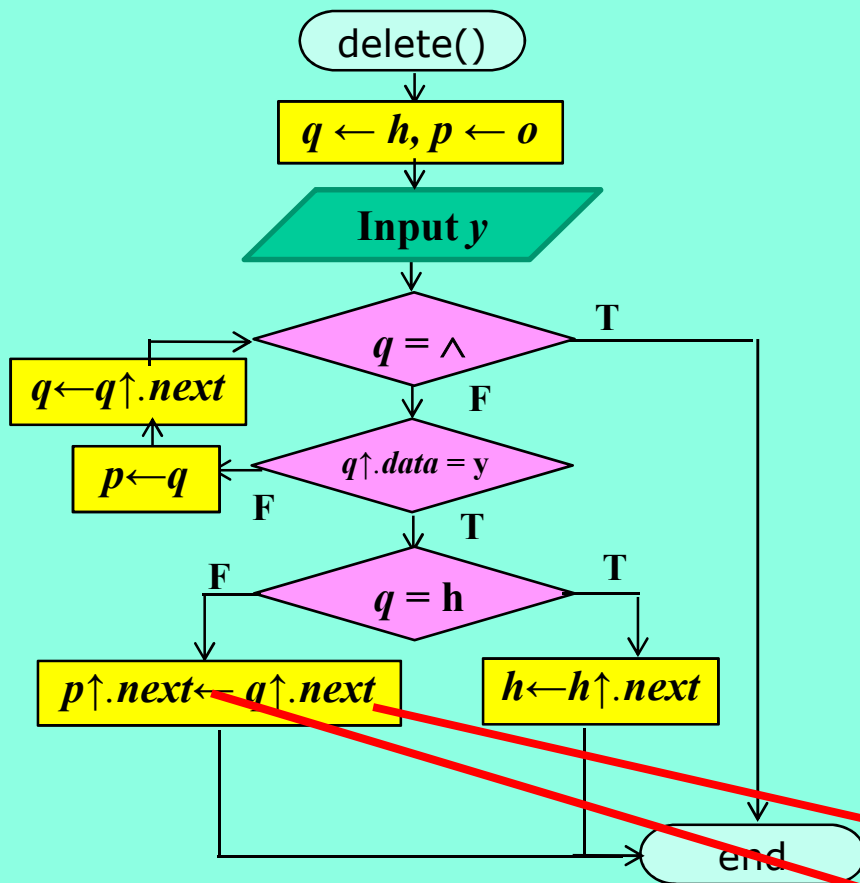
$h$ : address of first/head node  
 $q, p$ : pointer to a existing node  
 $y$ : target node to be inserted

25  
 $y$



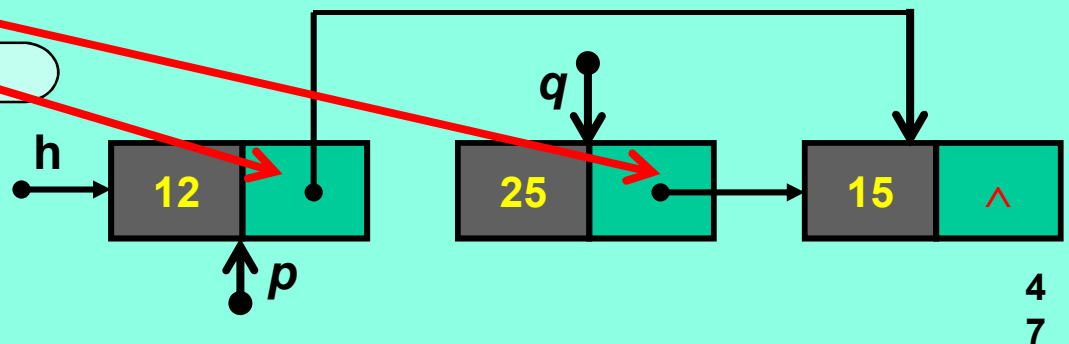
# Deleting in a Single Linked List

Topic 3: Write an Algorithm to Delete an existing node in a single linked list.



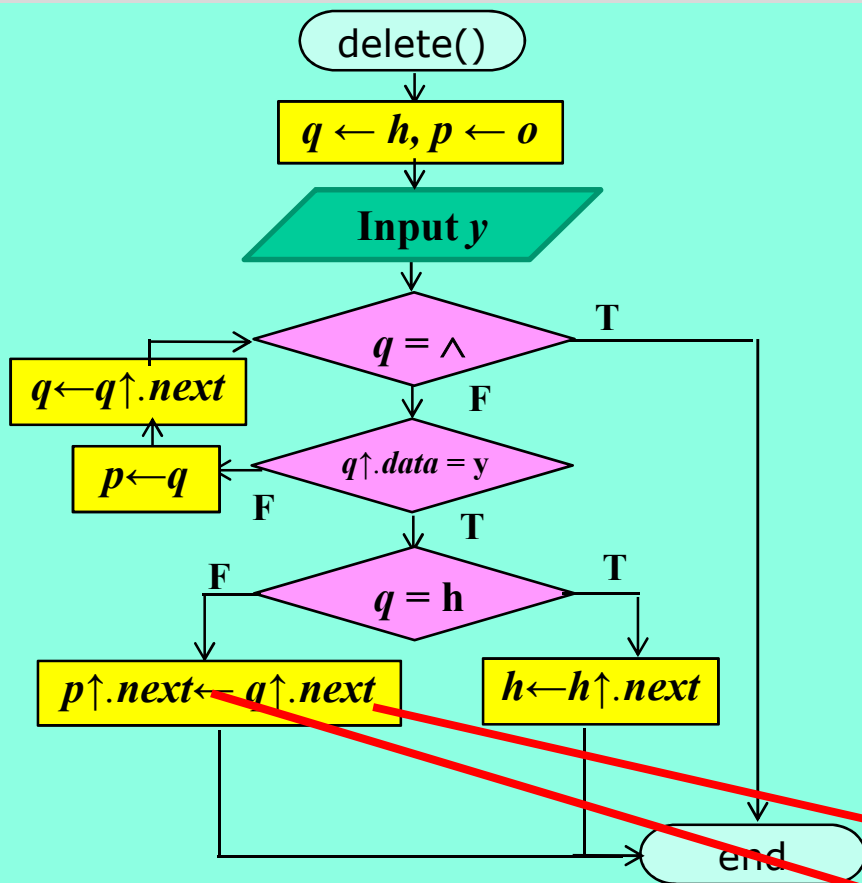
$h$ : address of first/head node  
 $q, p$ : pointer to a existing node  
 $y$ : target node to be inserted

25  
 $y$



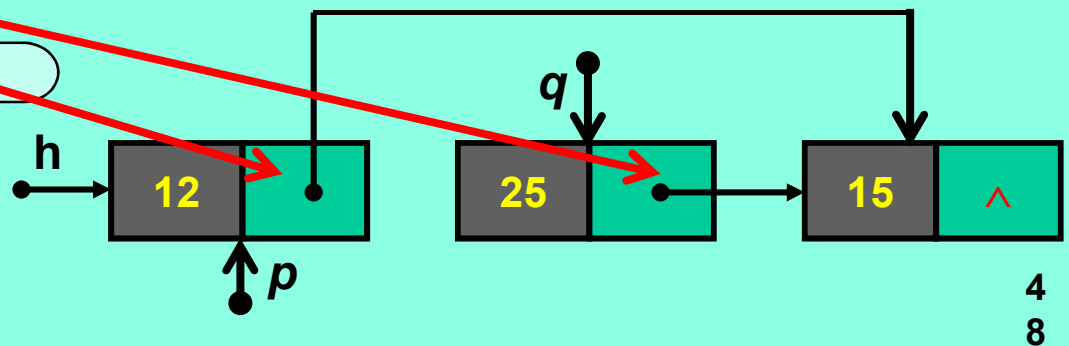
# Deleting in a Single Linked List

**Topic 3: Write an Algorithm to Delete an existing node in a single linked list.**



```

void Delete(){
    Node *q,*p;
    int y,flag=0;
    q=h; p=NULL;
    cout<<"\nDelete Which element:";
    cin>>y;
    while(q!=NULL){
        if(q->data==y){
            if(q==h)
                h=h->next;
            else
                p->next=q->next;
            flag=1;
            break;
        }
        else{
            p=q;
            q=q->next;
        }
    }
    if(flag==0)
        cout<<"Sorry Node NOT found"<<endl;
}
  
```

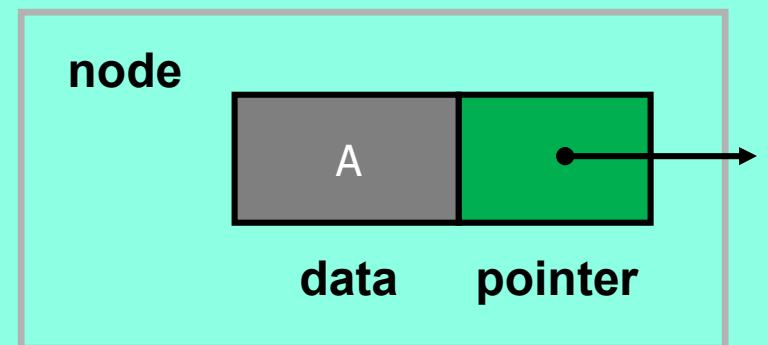
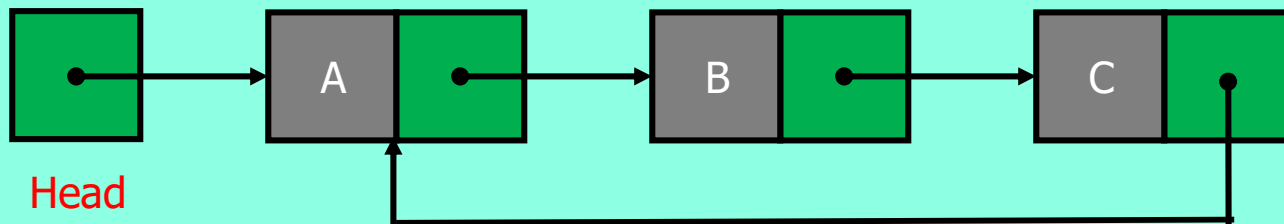




# Circular Link List

# Circular Linked List

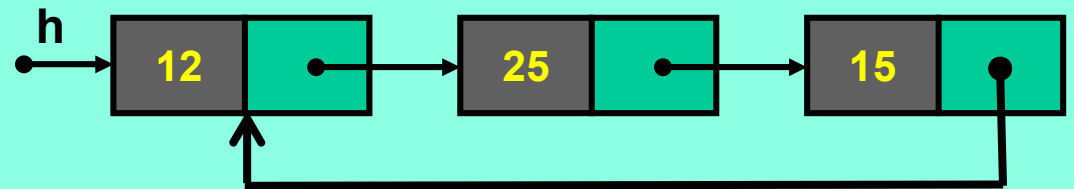
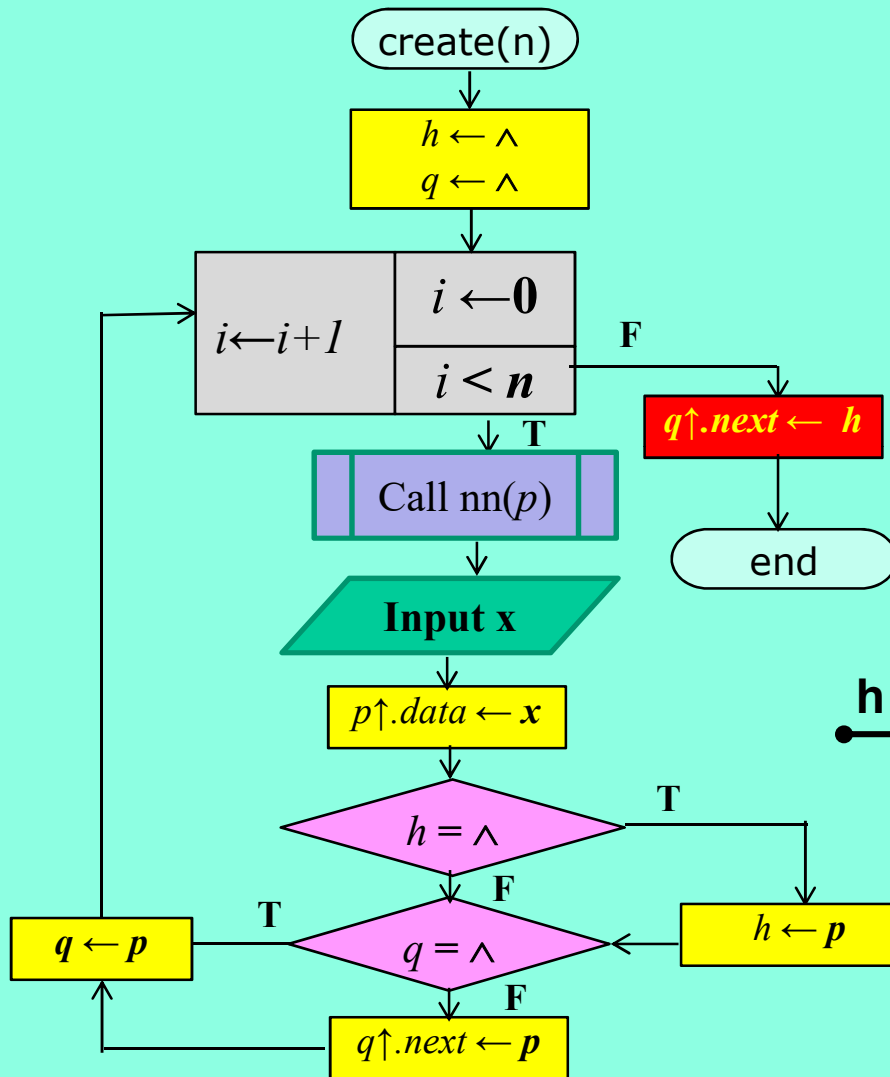
- A *Circular linked list* is a simple linked list except the last connected to the head/first node.



# Single Linked List

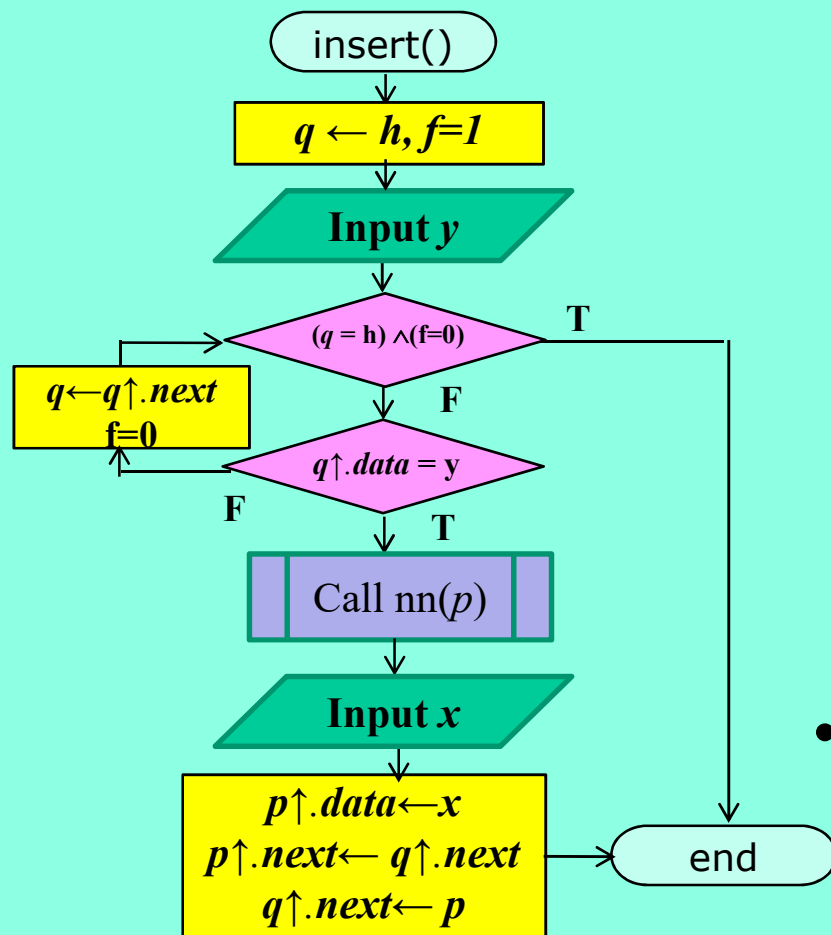
**Topic 1: Write an Algorithm to create a single circular linked list with n nodes.**

$n$ : total nodes  
 $x$ : input variable  
 $nn(p)$ : a function, call a new node pointed by  $p$   
 $\wedge$ : NULL

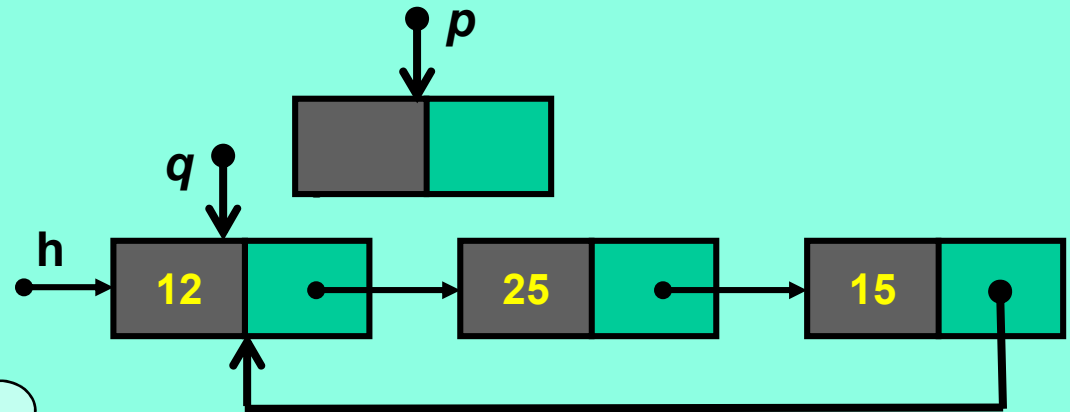


# Circular Single Linked List

**Topic 2:** Write an Algorithm to insert a new node after a existing node in circular single linked list.

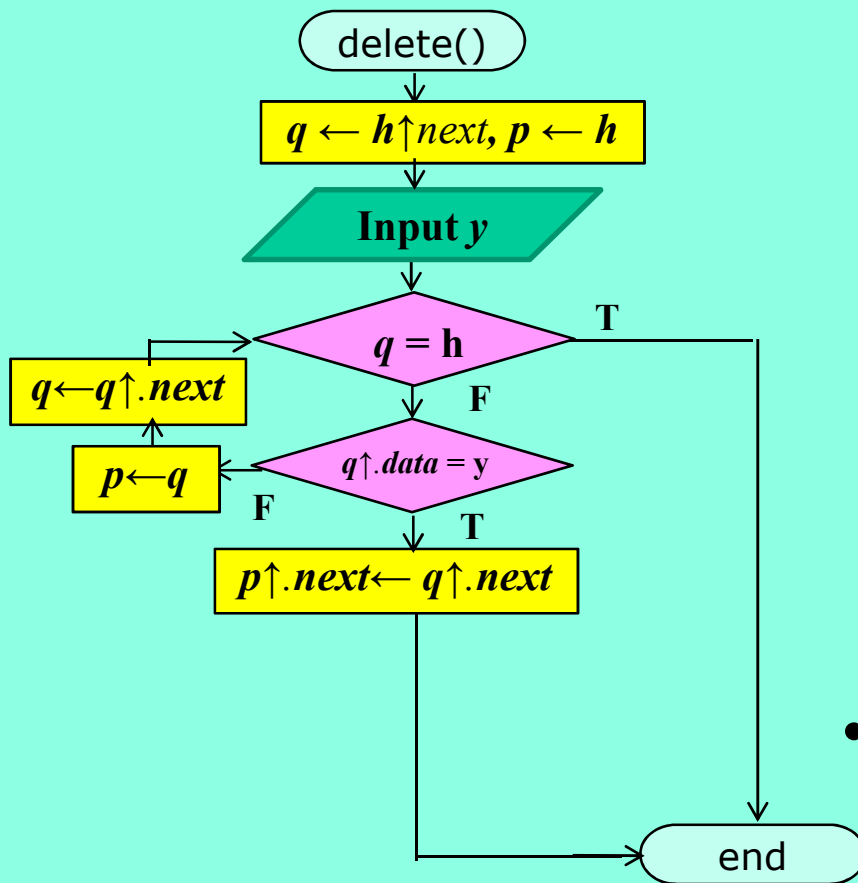


*h*: address of first/head node  
*p*: pointer to a new node  
*x*: input data for new node  
*y*: target node, after *y* node *x* to be inserted



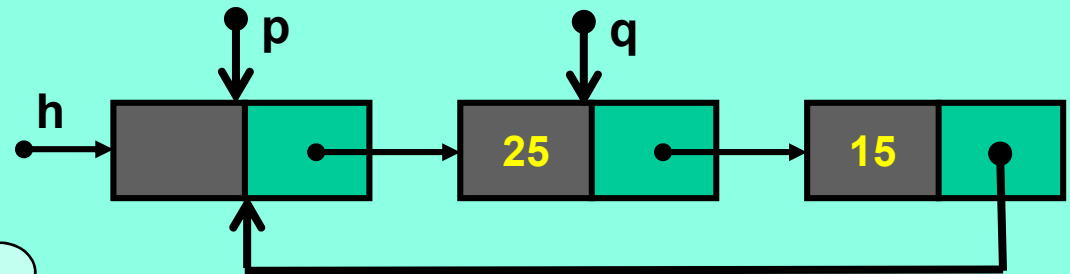
# Circle Single Linked List

**Topic 3: Write an Algorithm to Delete an existing node in a single circular linked list.**



$h$ : address of first/head node  
 $q, p$ : pointer to a existing node  
 $y$ : target node to be inserted

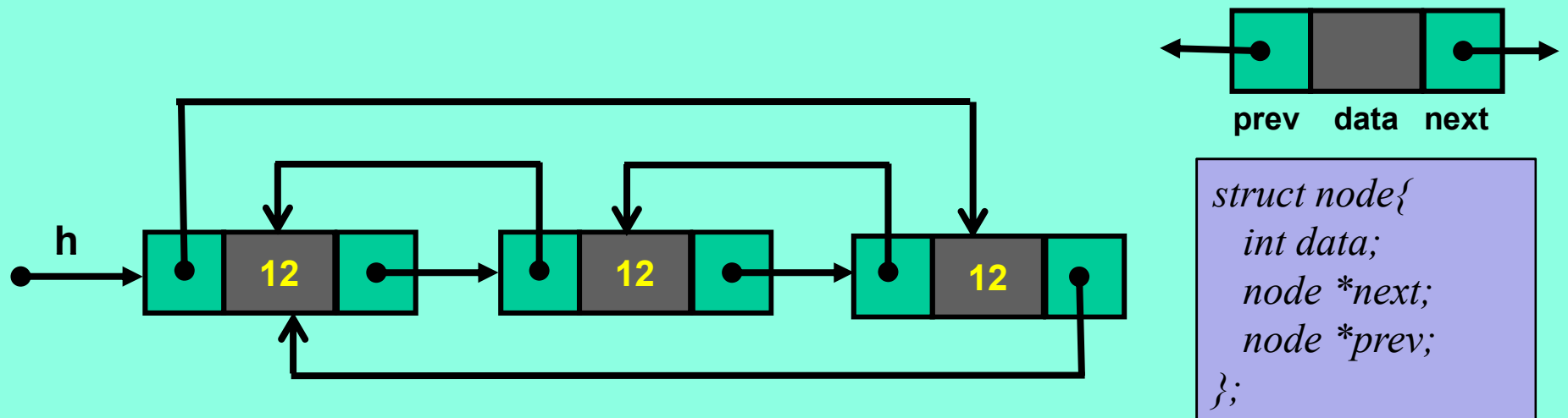
*First Node is the head node can not be deleted*



# **Double Circular Link List**

# Double (Two Way) Linked List

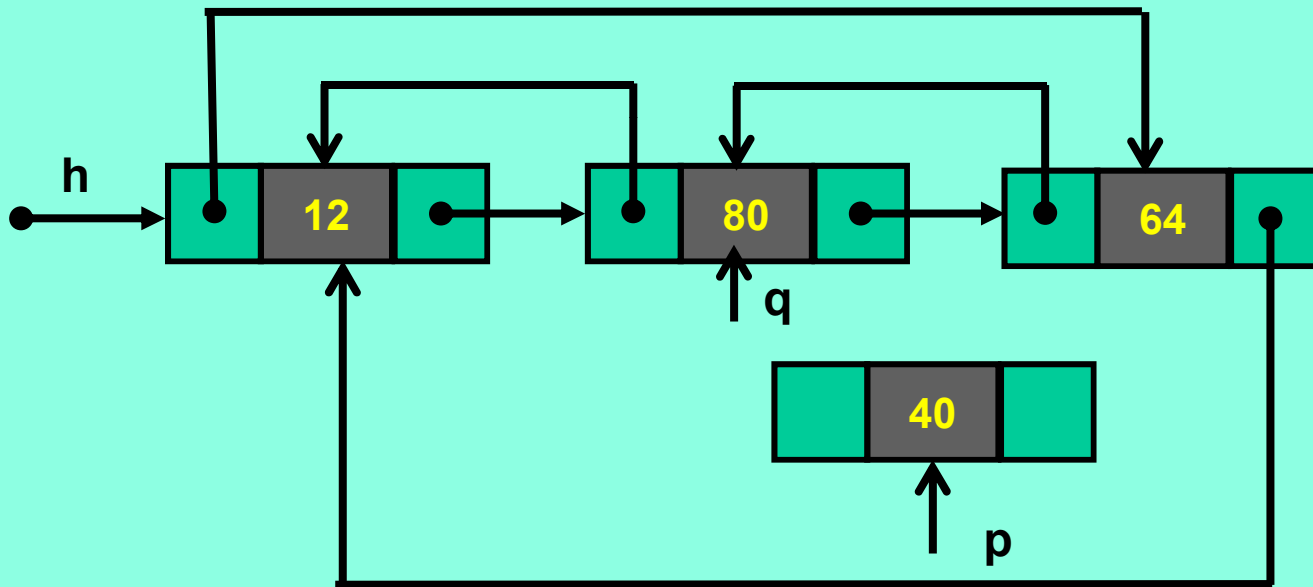
Here each node stores its previous and next node addresses



# Double (Two Way) Linked List

Topic 1: Write an Algorithm to insert a new node in a double linked list.

Suppose we need to Insert a new node after node 80.  
Node 80 is pointed by q.

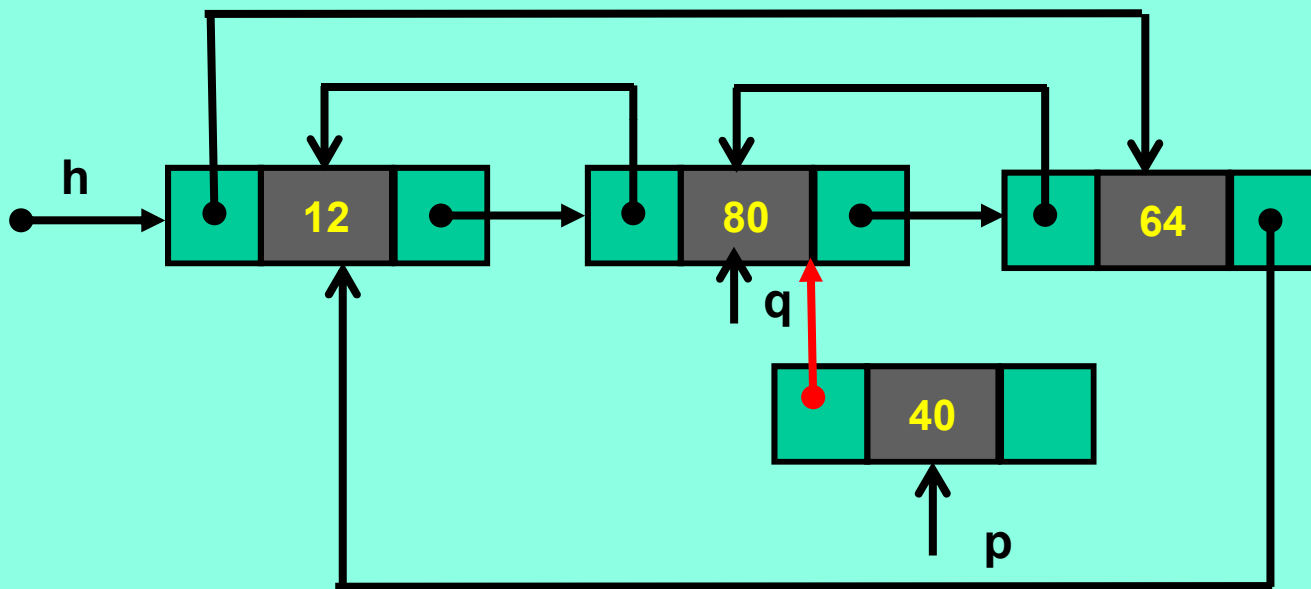




# Double (Two Way) Linked List

Topic 1: Write an Algorithm to insert a new node in a double linked list.

Suppose we need to Insert a new node after node 80.  
Node 80 is pointed by q.

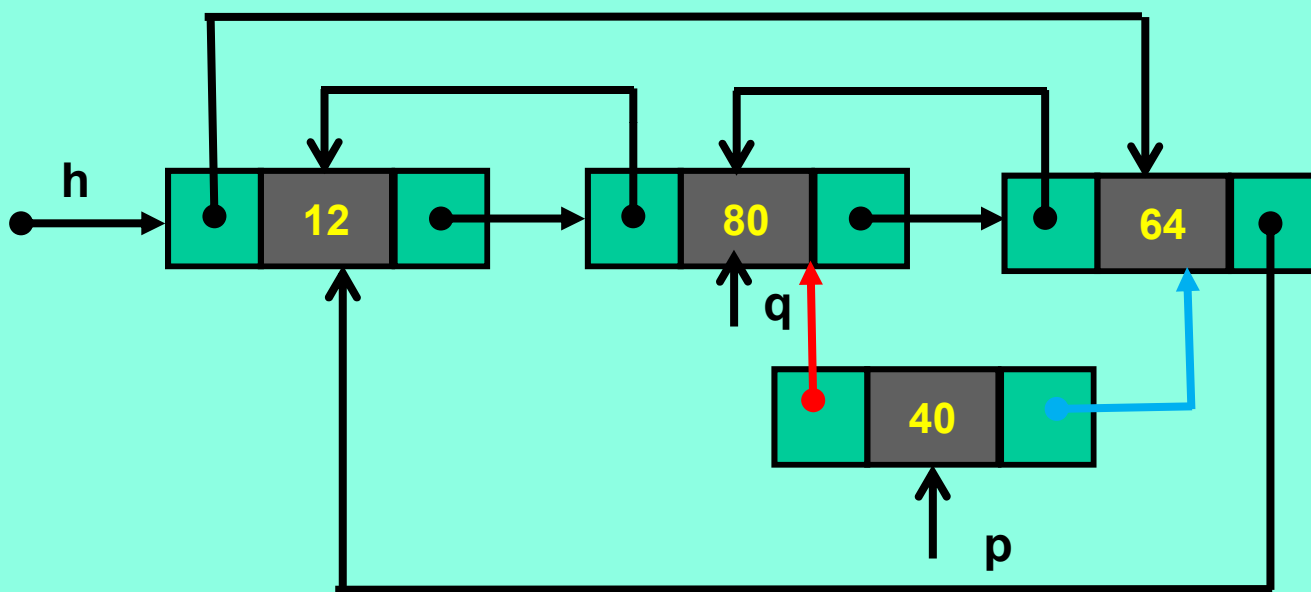


$p \uparrow .prev \leftarrow q$

# Double (Two Way) Linked List

Topic 1: Write an Algorithm to insert a new node in a double linked list.

Suppose we need to Insert a new node after node 80.  
Node 80 is pointed by q.



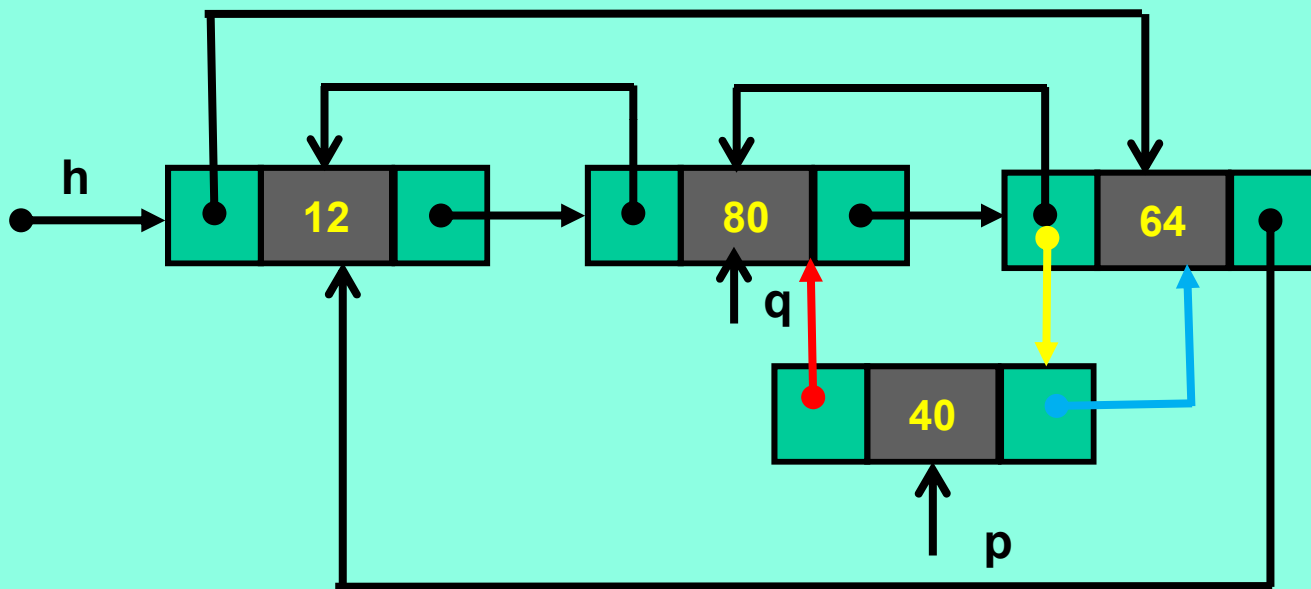
$p \uparrow .prev \leftarrow q$

$p \uparrow .next \leftarrow q \uparrow .next$

# Double (Two Way) Linked List

Topic 1: Write an Algorithm to insert a new node in a double linked list.

Suppose we need to Insert a new node after node 80.  
Node 80 is pointed by q.



$p \uparrow .prev \leftarrow q$

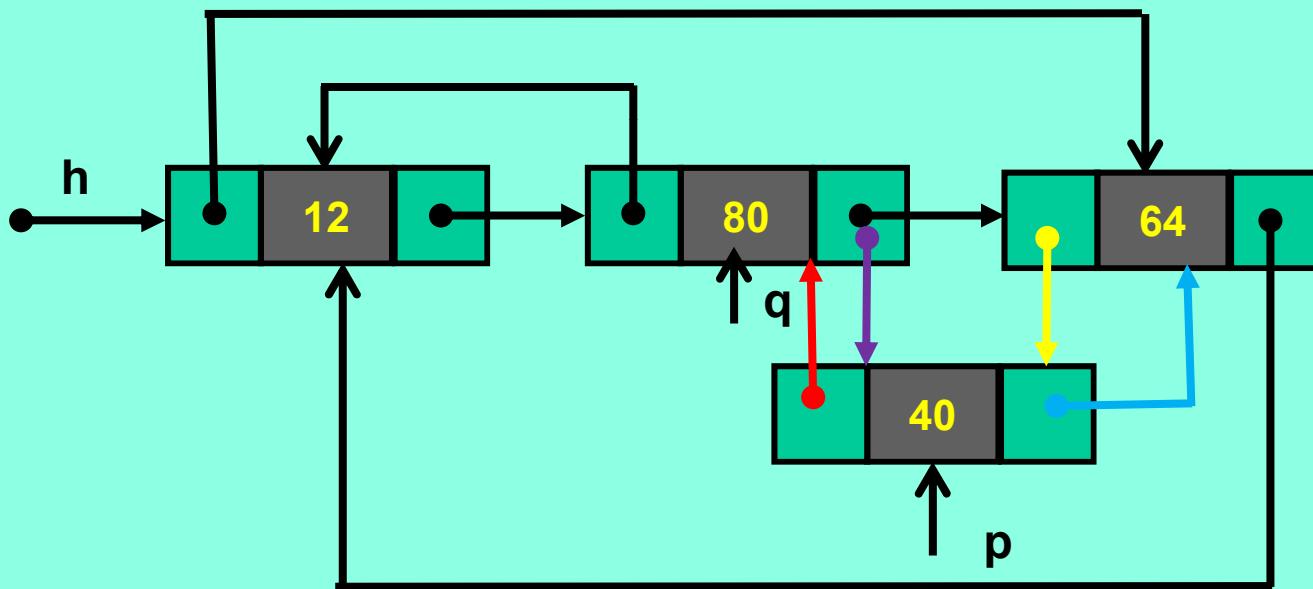
$p \uparrow .next \leftarrow q \uparrow .next$

$(q \uparrow .next) \uparrow .prev \leftarrow p$

# Double (Two Way) Linked List

Topic 1: Write an Algorithm to insert a new node in a double linked list.

Suppose we need to Insert a new node after node 80.  
Node 80 is pointed by q.



$p \uparrow .prev \leftarrow q$

$p \uparrow .next \leftarrow q \uparrow .next$

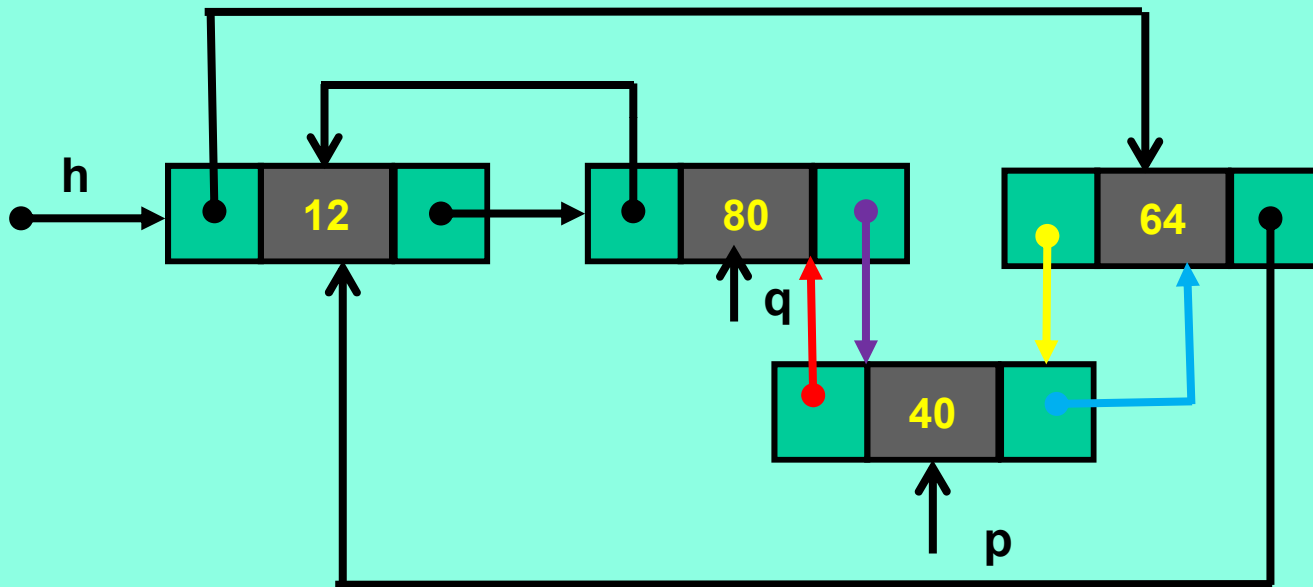
$(q \uparrow .next) \uparrow .prev \leftarrow p$

$(q \uparrow .next) \leftarrow p$

# Double (Two Way) Linked List

Topic 1: Write an Algorithm to insert a new node in a double linked list.

Suppose we need to Insert a new node **AFTER** node 80.  
Node 80 is pointed by q.



$p \uparrow .prev \leftarrow q$

$p \uparrow .next \leftarrow q \uparrow .next$

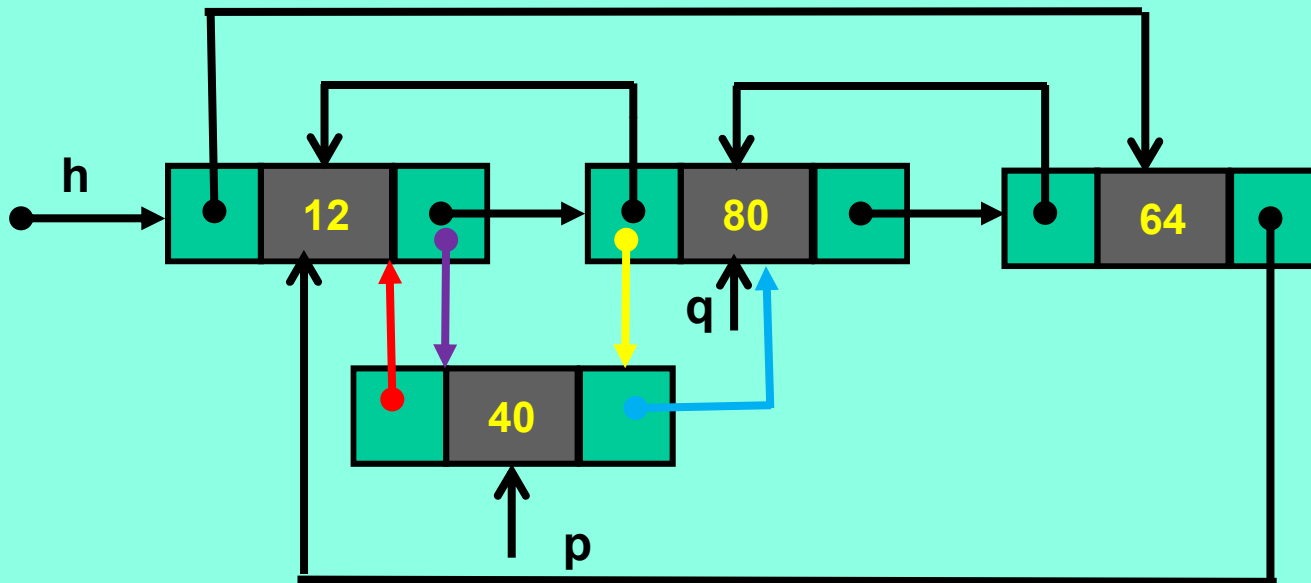
$(q \uparrow .next) \uparrow .prev \leftarrow p$

$(q \uparrow .next) \uparrow .prev \leftarrow p$

# Double (Two Way) Linked List

Topic 1: Write an Algorithm to insert a new node in a double linked list.

Suppose we need to Insert a new node after node 80.  
Node 80 is pointed by q.



$p \uparrow .prev \leftarrow q \uparrow .prev$

$p \uparrow .next \leftarrow p$

$(q \uparrow .prev) \uparrow .next \leftarrow p$

$(q \uparrow .prev) \leftarrow p$

# Assignment (submit all after mid break)

**Prob 1: Write an Algorithm to Insert a new node at the beginning of an existing circular single linked list. Also write a function to implement the above.**

**Prob 2: Write an Algorithm to Insert a new node at the end of an existing single circular linked list. Also write a function to implement the above.**

**Prob 3: Write an Algorithm to join two existing single circular linked lists into one single circular linked list. Also write a function to implement the above.**

**Prob 4: Repeat 2-3 for double circular linked list.**

**Prob 5: Write an Algorithm to replace a particular data searching all nodes of an existing single circular linked list. Also write a function to implement the above.**

**Prob 6: Write an Algorithm to delete all nodes containing a particular data of an existing double circular linked list. Also write a function to implement the above.**

# Assignment (submit after mid break)

**Prob 1: Write an Algorithm to Insert a new node at the beginning of an existing single linked list. Also write a function to implement the above.**

**Prob 2: Write an Algorithm to Insert a new node at the end of an existing single linked list. Also write a function to implement the above.**

**Prob 3: Write an Algorithm to join two existing single linked lists into one single link list. Also write a function to implement the above.**

**Prob 4: Implement STACK and QUEUE using link list.**

**Prob 5. Write an algorithm using link list to delete all the nodes with value of y.**

**Prob 6: Write an algorithm to find the highest value.**

**Prob 7: Write an algorithm to merge two sorted link list pointed by h1 and h2 respectively into one sorted link list. Use h1 or h2 as the output list. Do not use the third link list.**