CSE 1201
Data Structure

Chapter 5: Linked List (Part 1)

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.

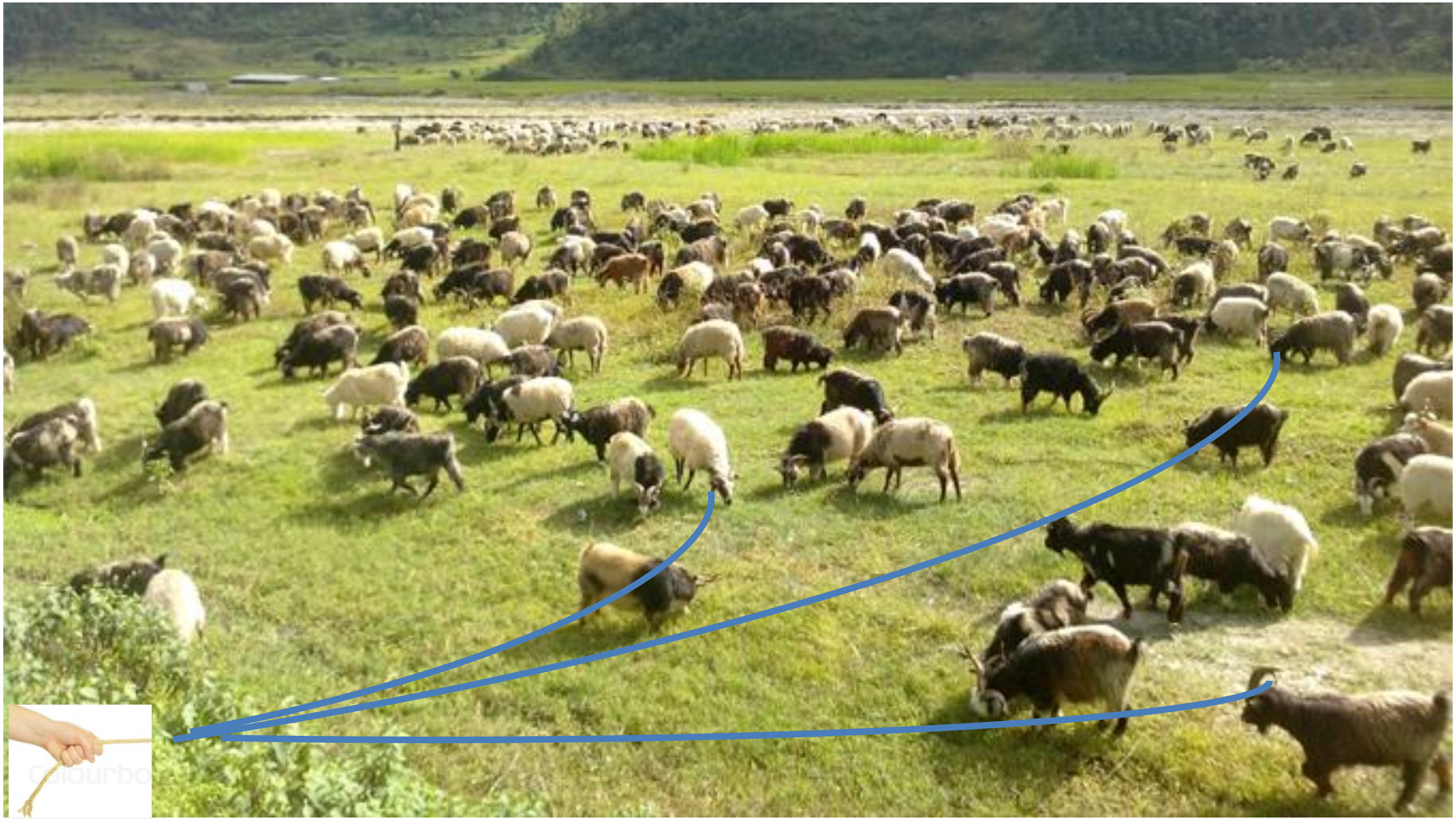
4. The elements of array are stored in consecutive memory locations. So insertions and deletions are very difficult and time consuming.

5. Lot of shifting is required for insertion and deletion operation.

Example: Goat Field

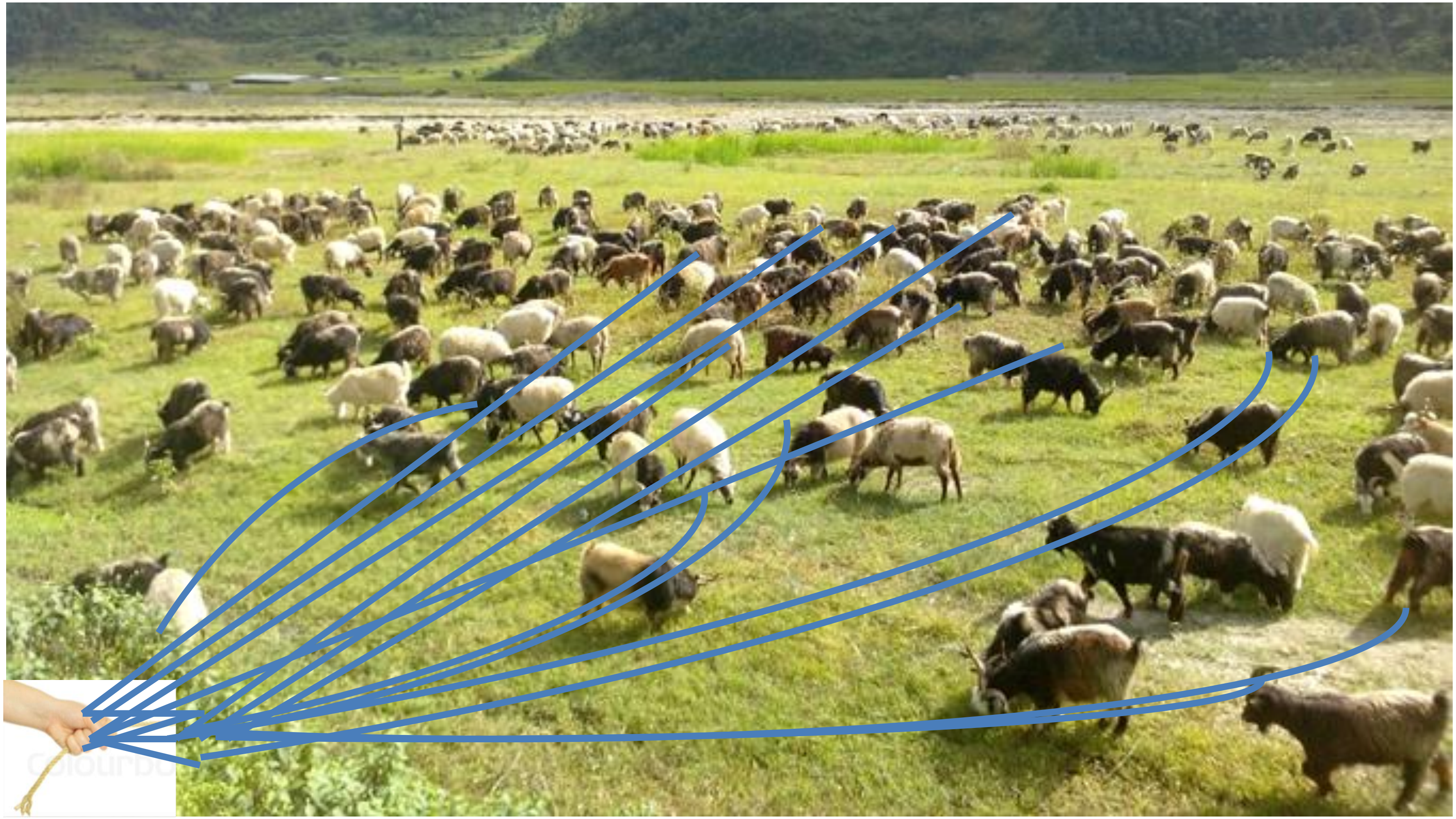
How to identify a group of goats?

Goat Field



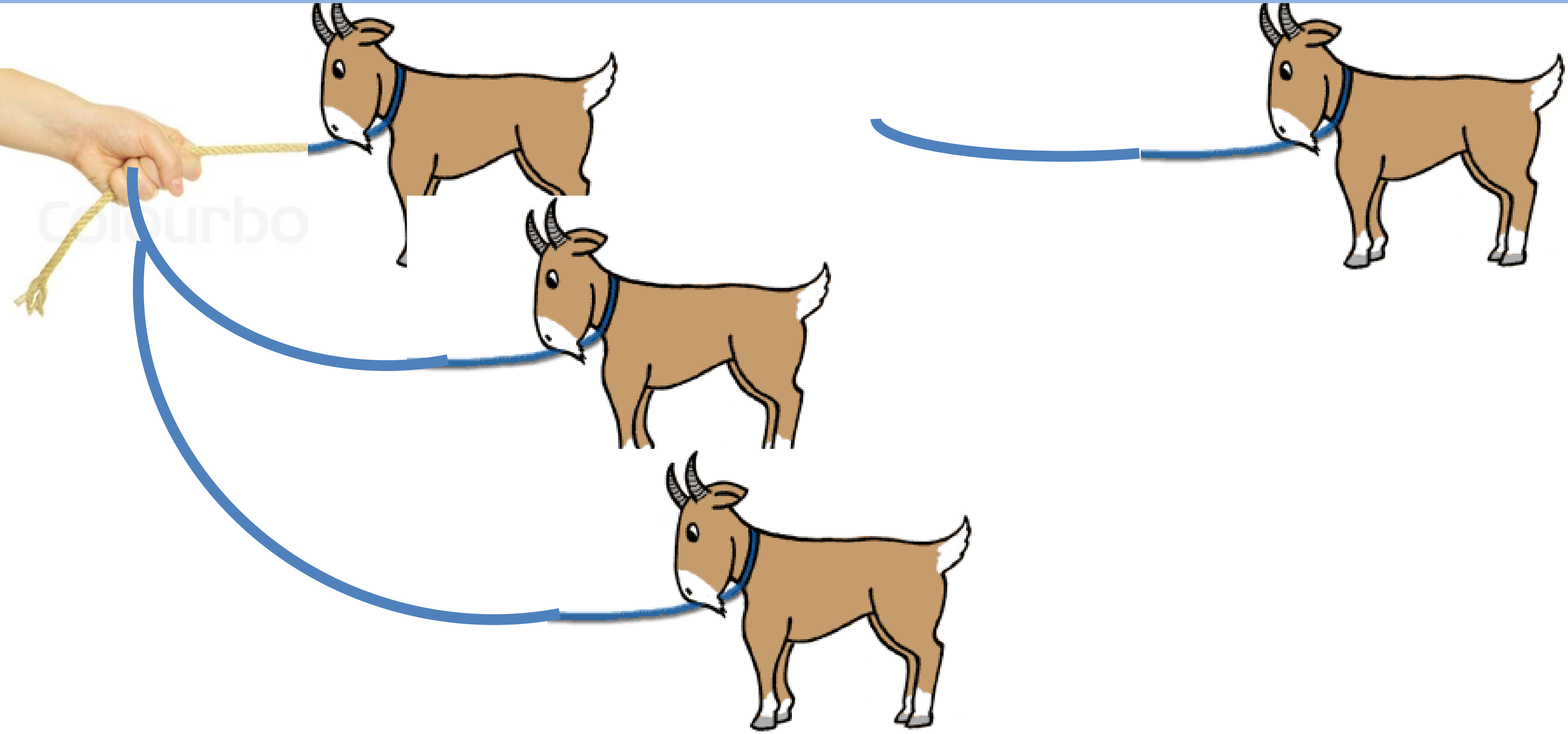
How to identify a group of goats? Using ropes

Goat Field



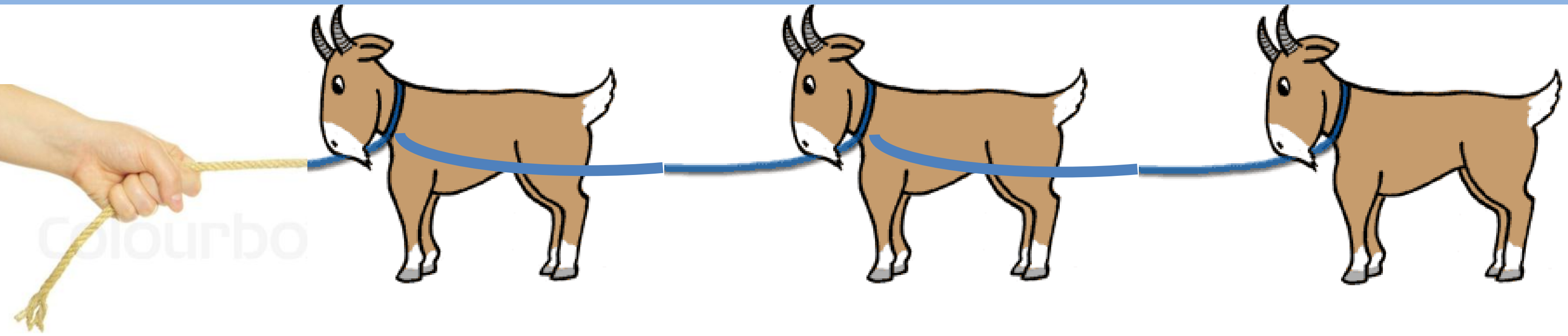
If the group contains many goats?

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?

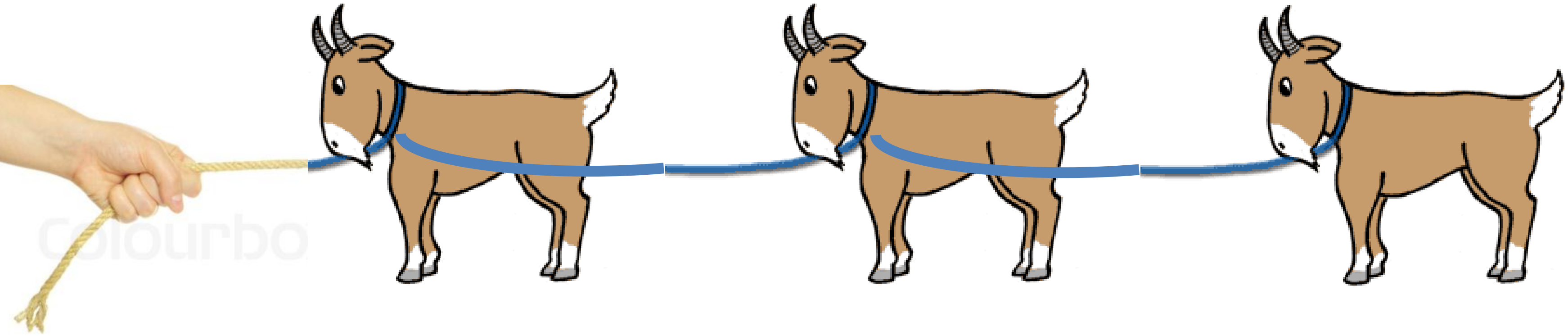
Goat List



- 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?

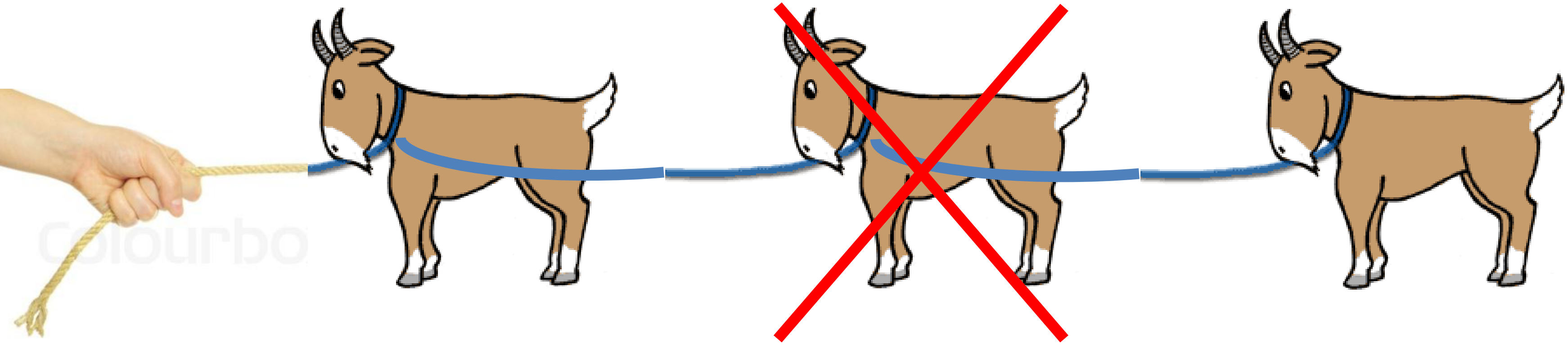
Ans: Bind one with another.

Goat List : DELETE



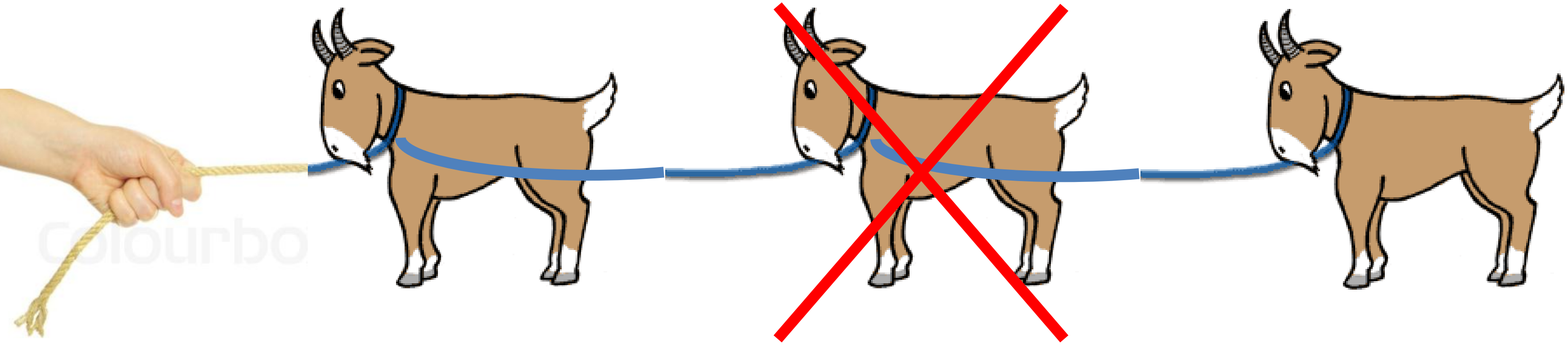
- Suppose 2nd goat is needed to be deleted

Goat List : DELETE



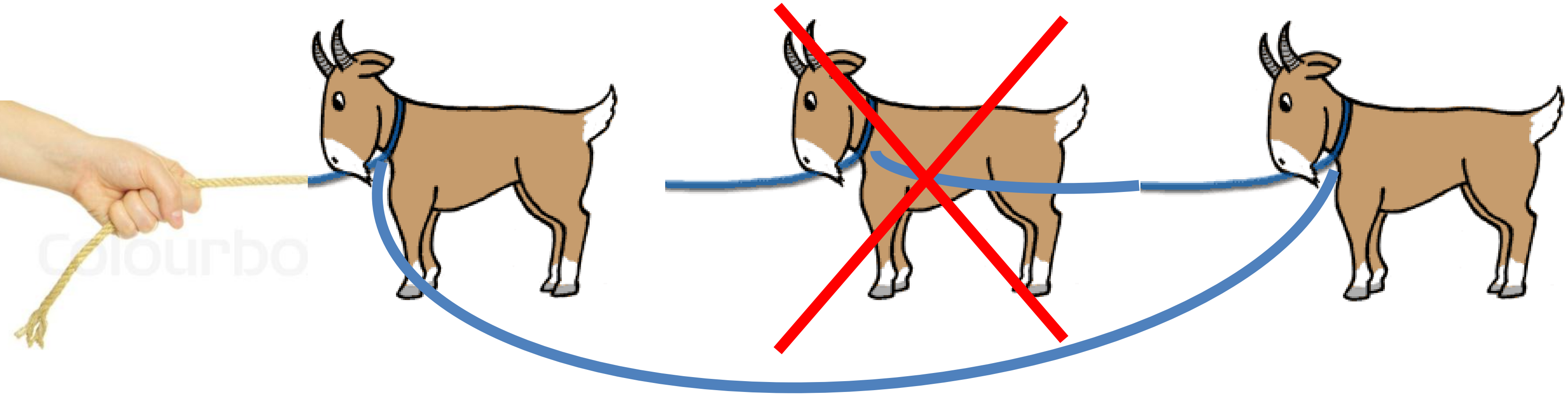
- Suppose 2nd goat is needed to be deleted

Goat List : DELETE



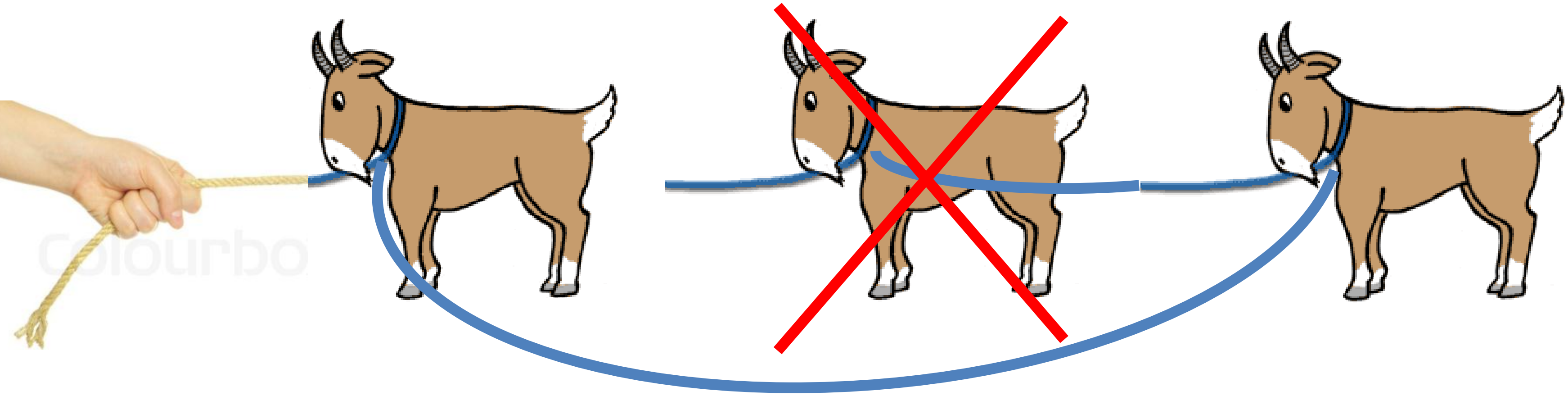
- Suppose 2nd goat is needed to be deleted
- Step 1: bind 3rd goat to 1st one

Goat List : DELETE



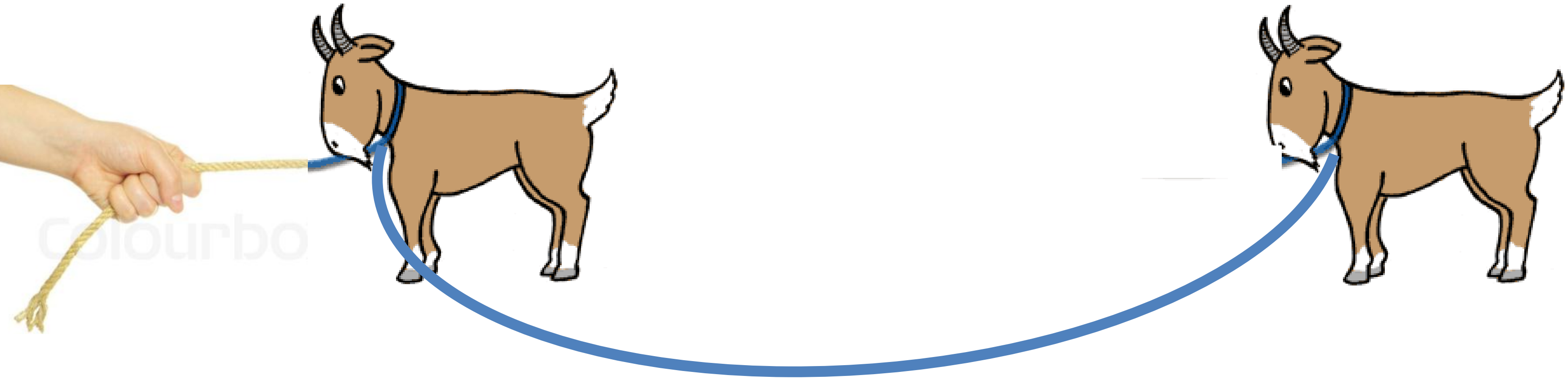
- Suppose 2nd goat is needed to be deleted
- Step 1: bind 3rd goat to 1st one

Goat List : DELETE



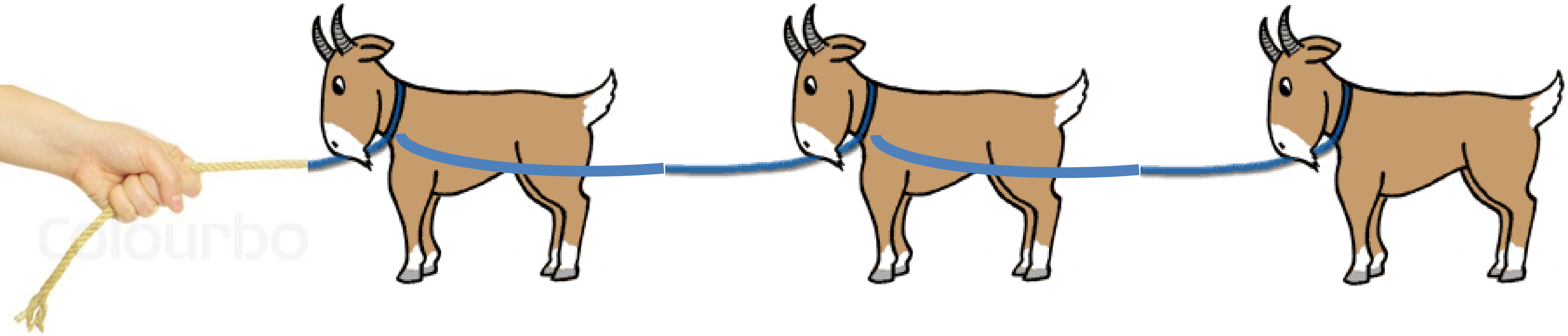
- Suppose 2nd goat is needed to be deleted
- Step 1: bind 3rd goat to 1st one
- Step 2: free 2nd goat

Goat List : DELETE



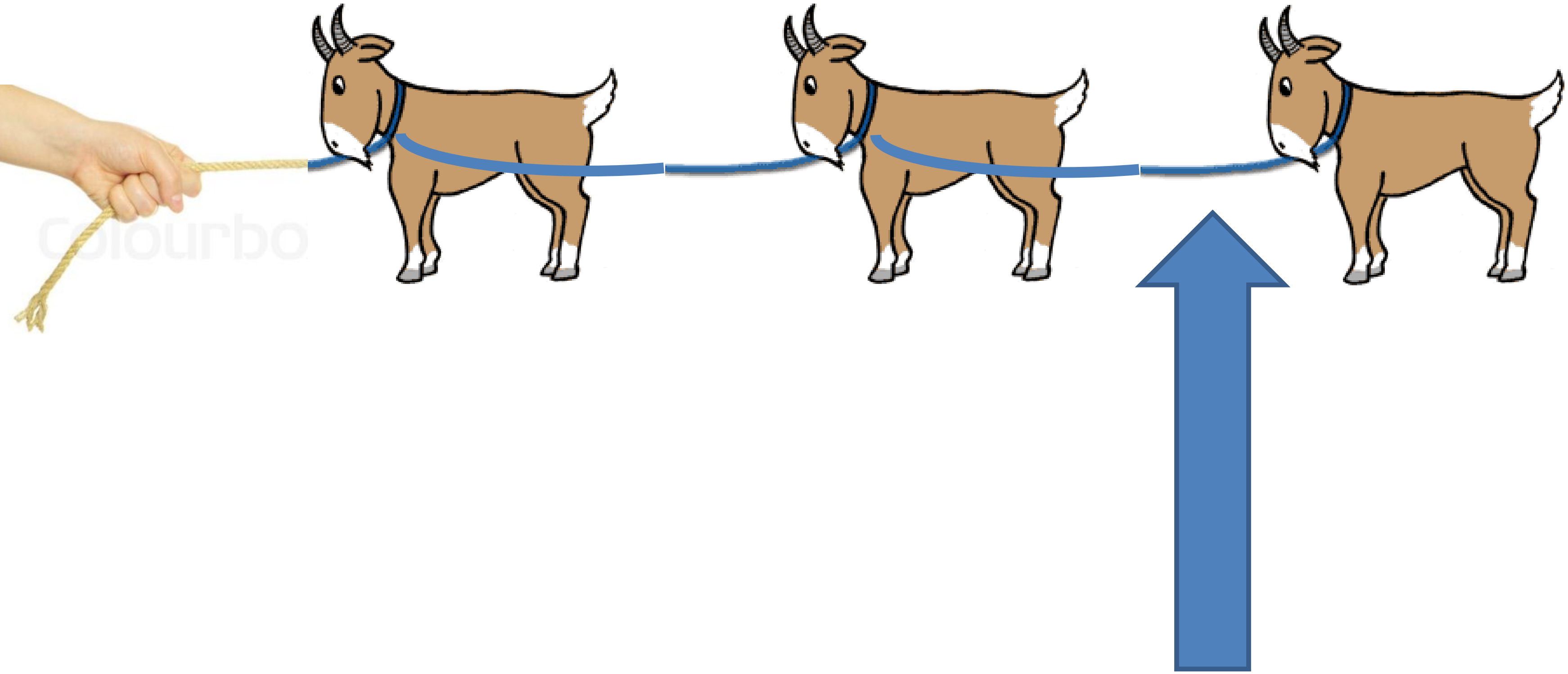
- Suppose 2nd goat is needed to be deleted
- Step 1: bind 3rd goat to 1st one
- Step 2: free 2nd goat

Goat List : INSERT



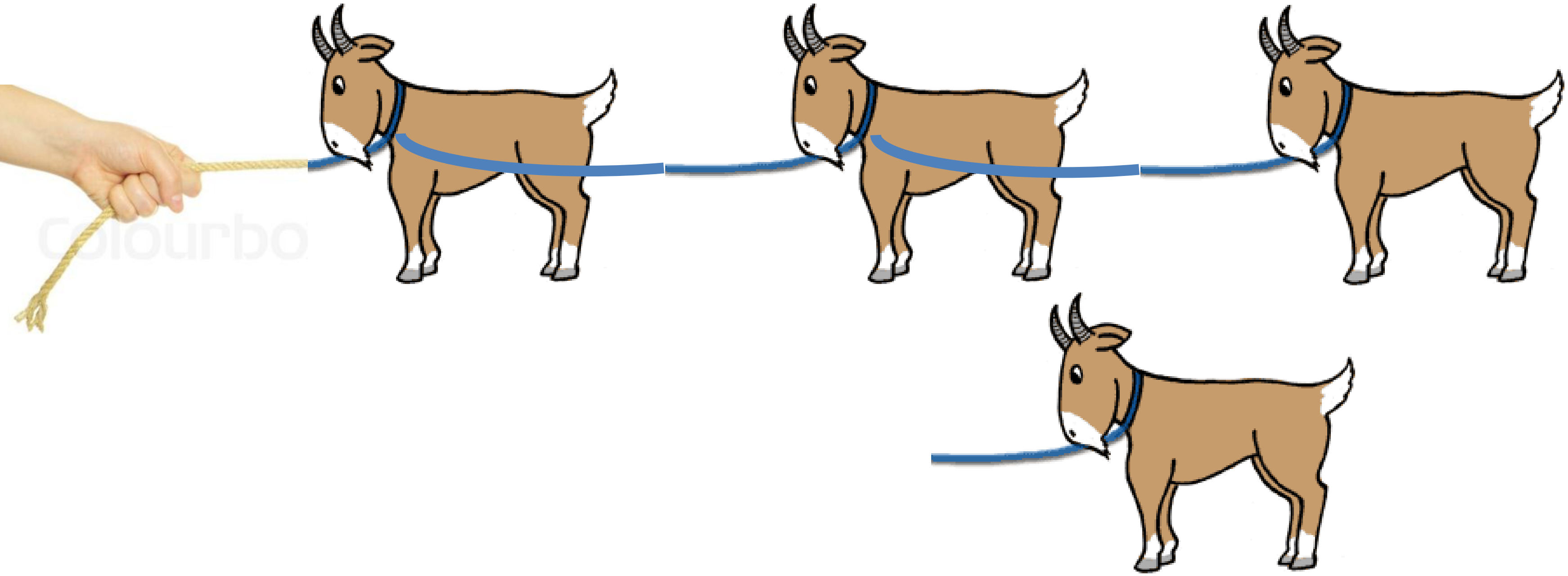
- Suppose a new goat is needed to be Inserted between 2nd and 3rd goats.

Goat List : INSERT



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

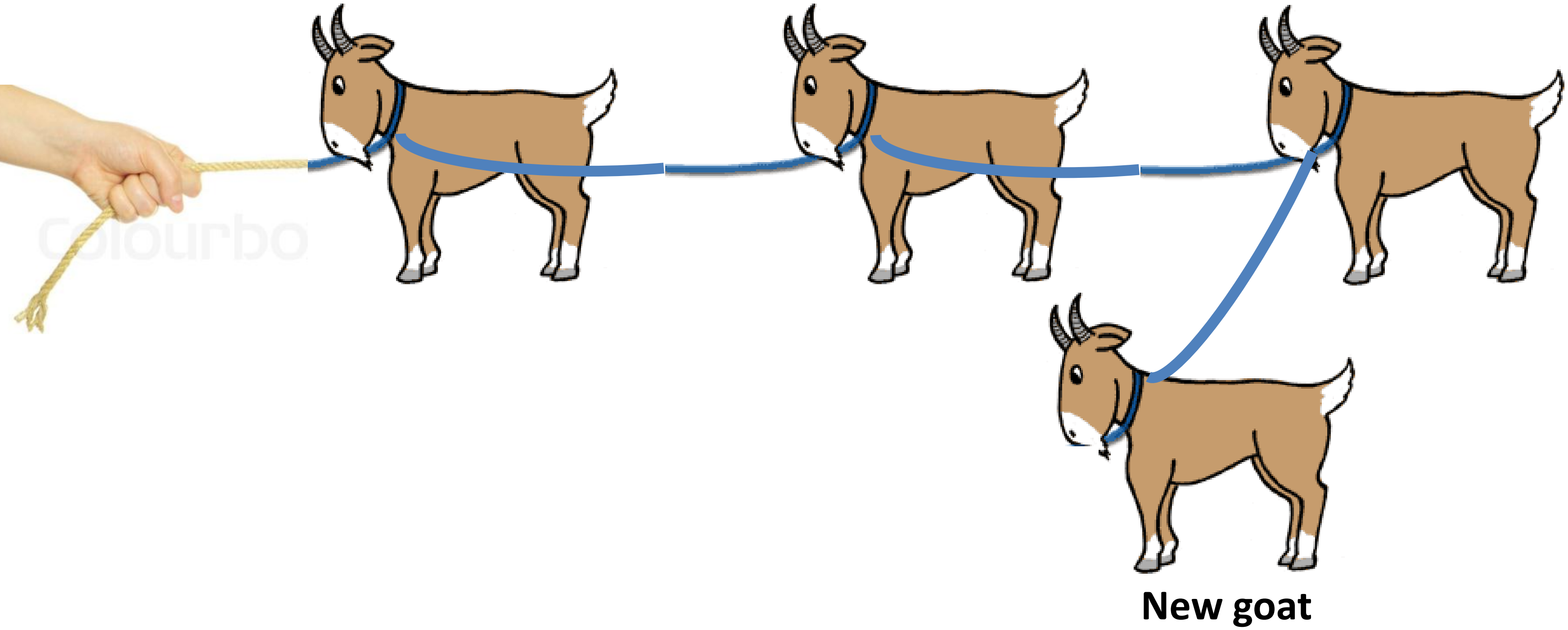
Goat List : INSERT



New goat

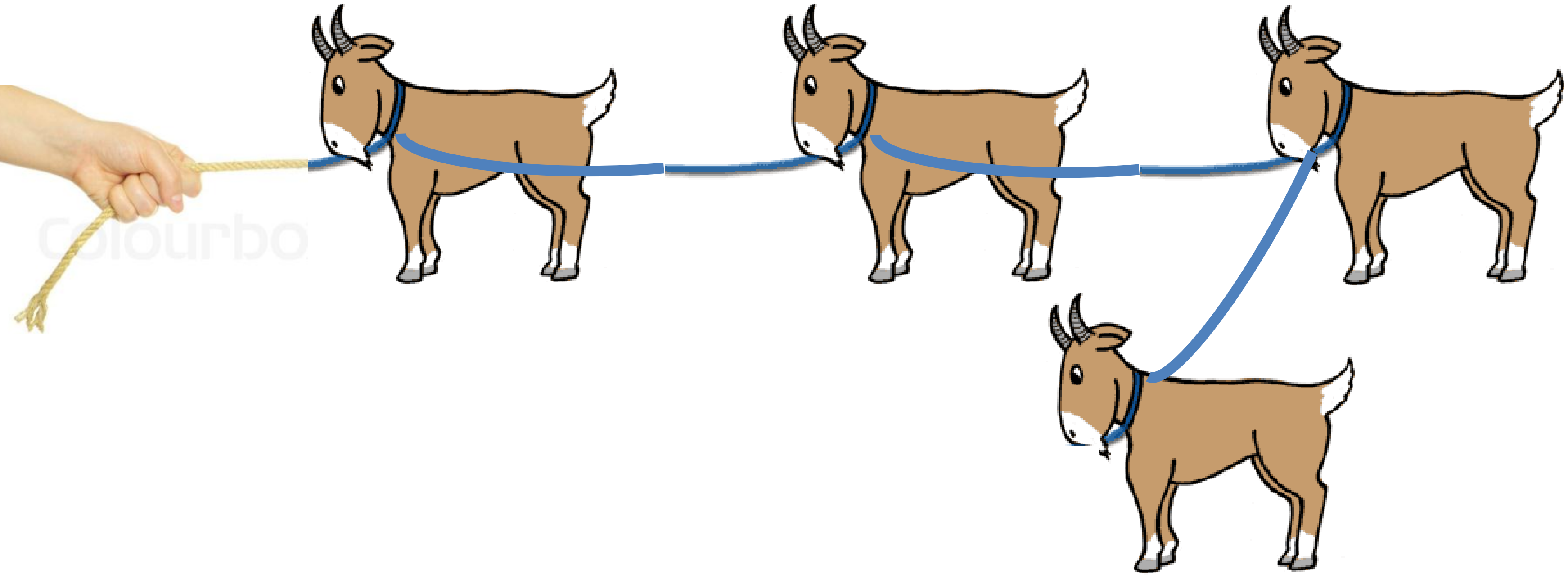
- Suppose a new goat is needed to be Inserted between 2nd and 3rd goats.
    - Step 1: Bring a new goat
    - Step 2: Bind 3rd one to new goat

Goat List : INSERT



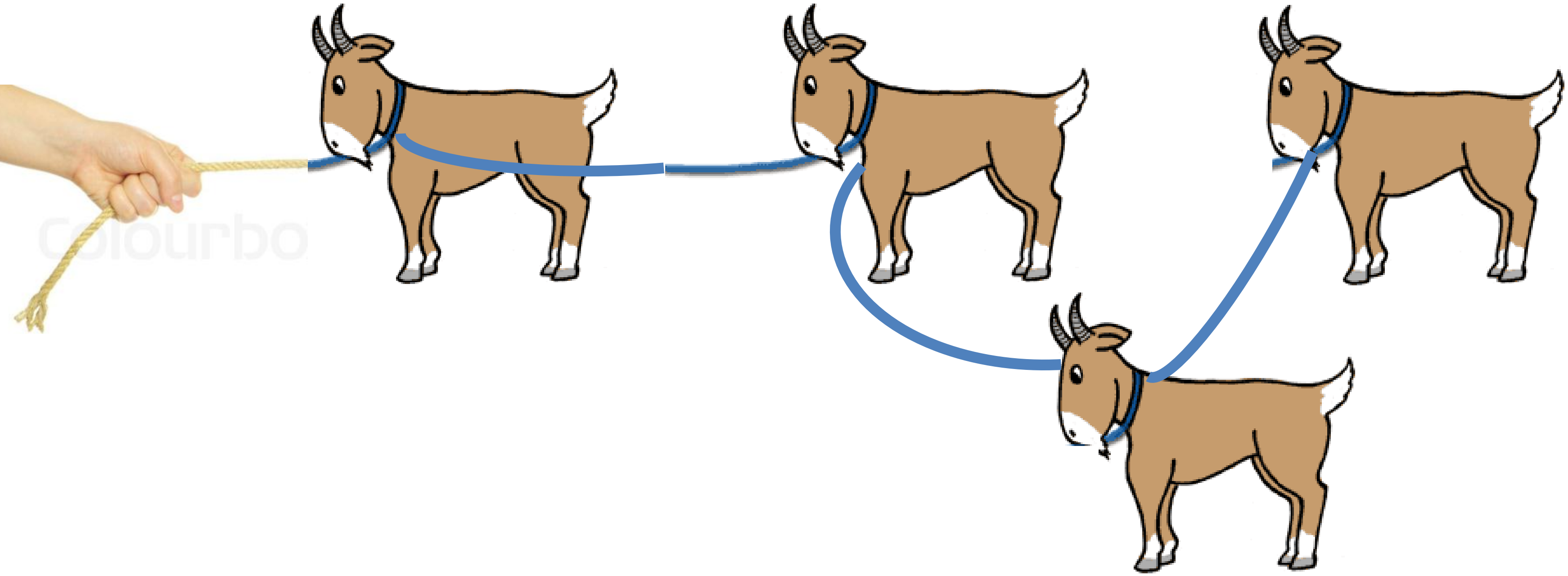
- Suppose a new goat is needed to be Inserted between 2nd and 3rd goats.
  - Step 1: Bring a new goat
  - Step 2: Bind 3rd one to new goat

Goat List : INSERT



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

Goat List : INSERT



New goat

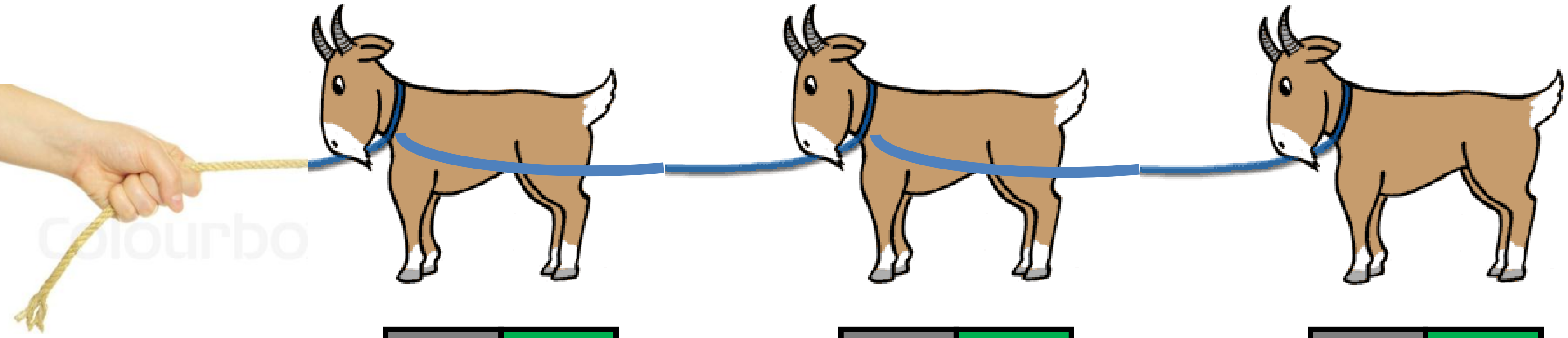
- Suppose a new goat is needed to be Inserted between 2nd and 3rd goats.
  - Step 1: Bring a new goat
  - Step 2: Bind 3rd one to new goat
  - Step 3: Bind new one to 2nd one

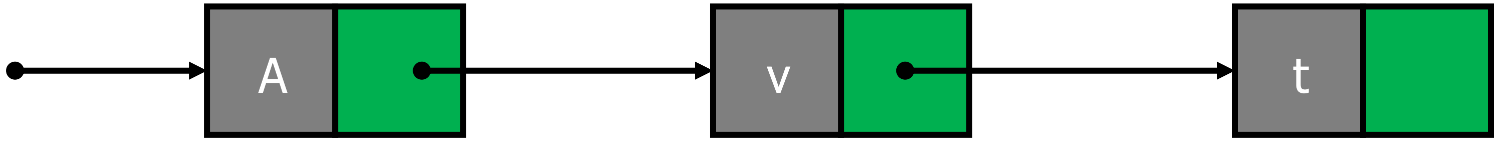
Basics of Linked List

- 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.

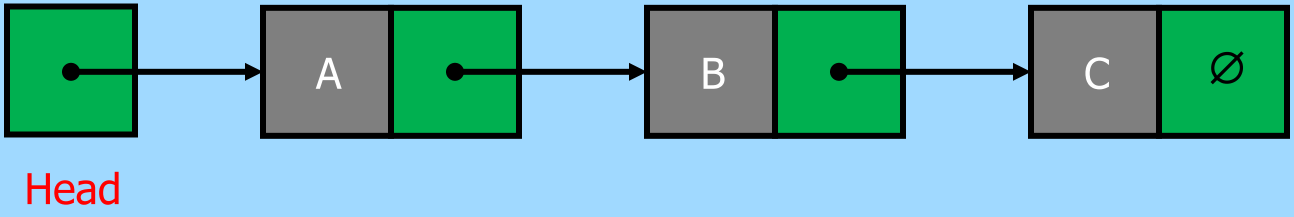
Goat List : Link List



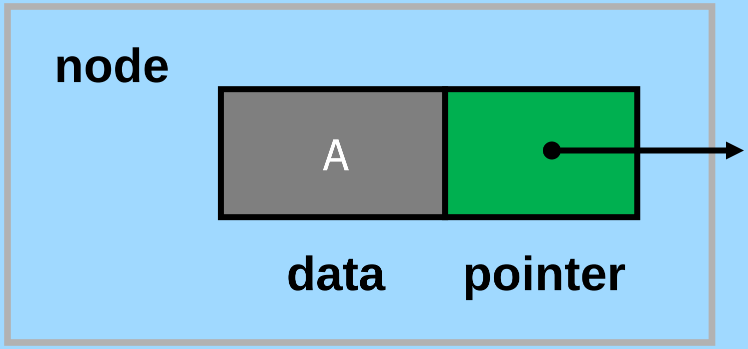


node node node

Basics of Linked List

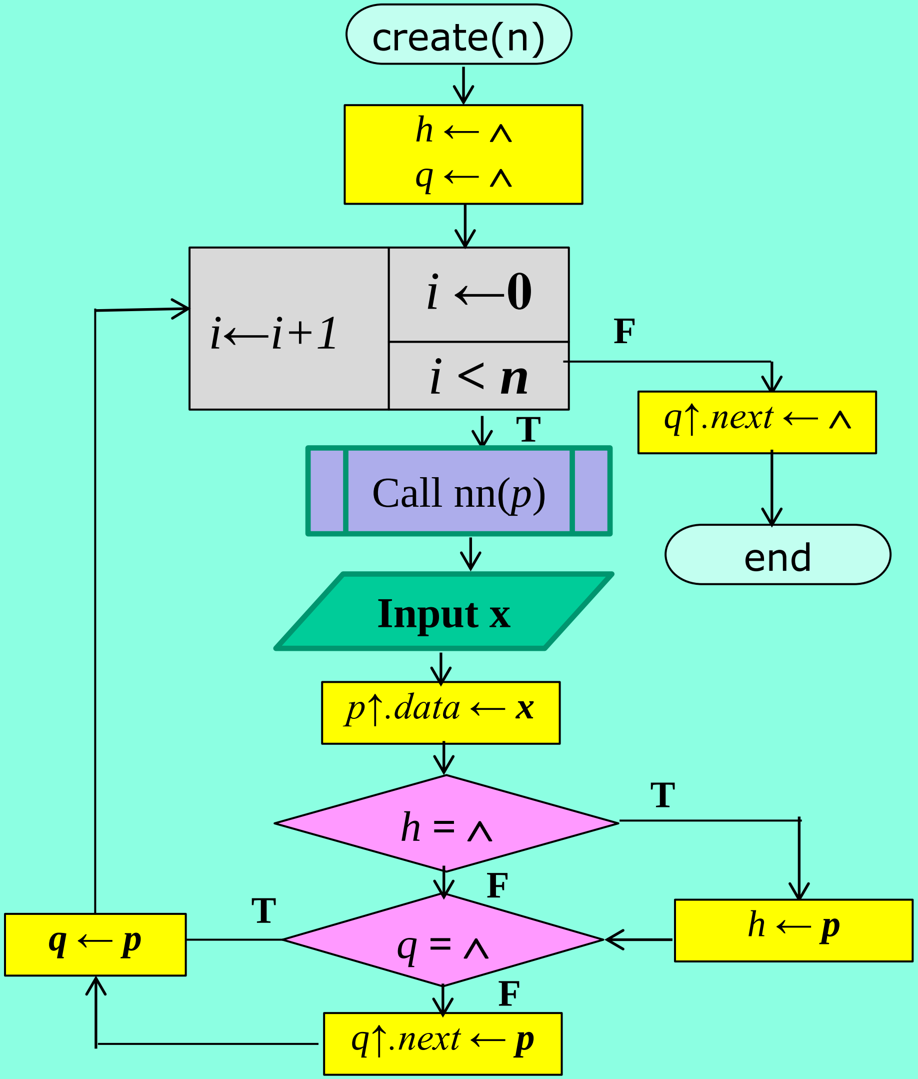


- A ***linked list*** is a series of connected ***nodes***
- Each node contains at least
  - A piece of data (any type)
  - Pointer to the next node in the list
- ***Head***: pointer to the first node
- The last node points to `NULL`

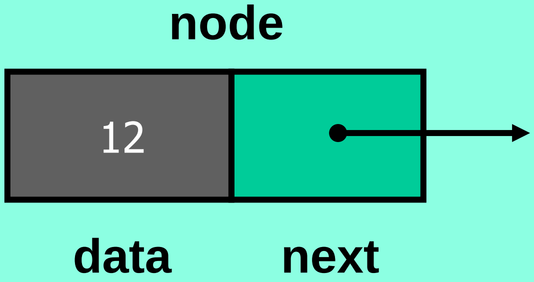


Single Linked List

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


create(n)
$h \leftarrow \wedge$
$q \leftarrow \wedge$
$i \leftarrow i+1$
$i \leftarrow \mathbf{0}$
$i < \boldsymbol{n}$
F
T
$q\uparrow.next \leftarrow \wedge$
end
Call nn(*p*)
Input x
$p\uparrow.data \leftarrow \mathbf{x}$
$h = \wedge$
T
$h \leftarrow \boldsymbol{p}$
F
$q = \wedge$
T
$\boldsymbol{q} \leftarrow \boldsymbol{p}$
F
$q\uparrow.next \leftarrow \boldsymbol{p}$

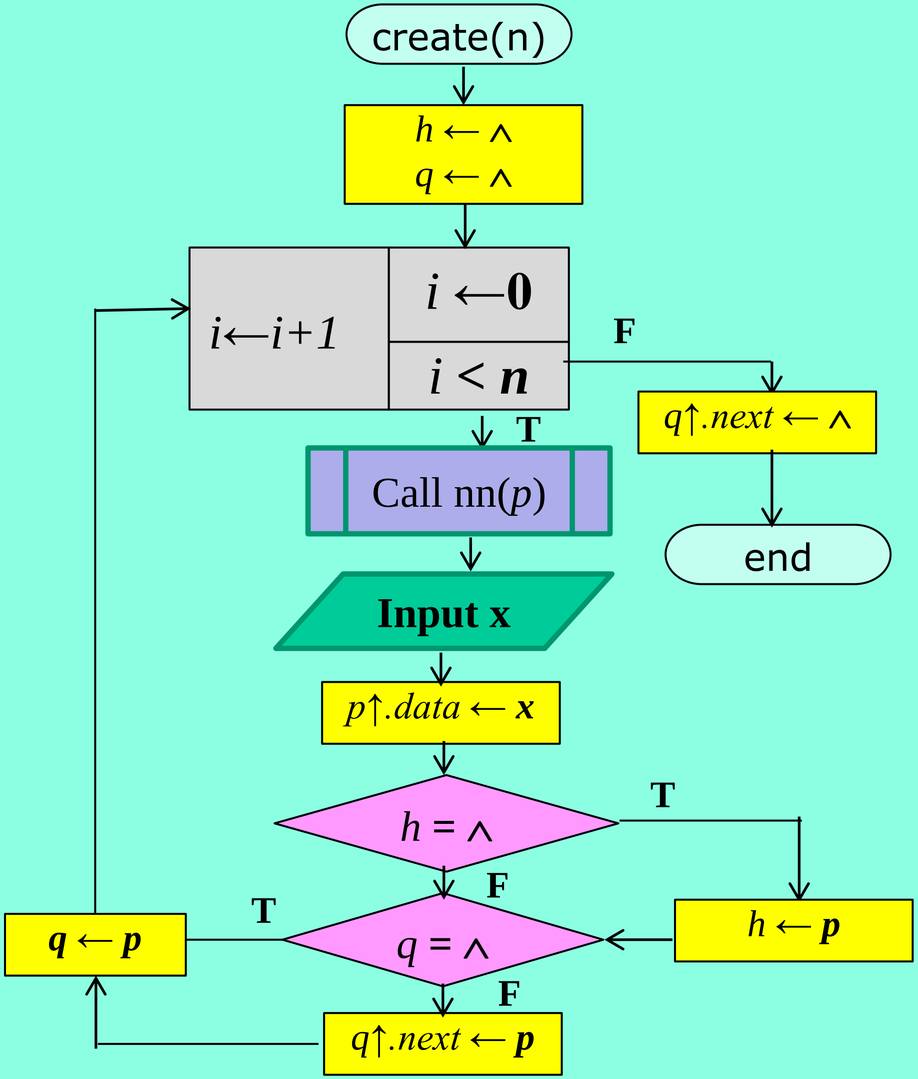
n: total nodes
x: input variable
nn(p): a function, call a new node pointed by *p*
∧ : NULL



```
struct node{
   int data;
   node *next;
};
```

Basics of Linked List

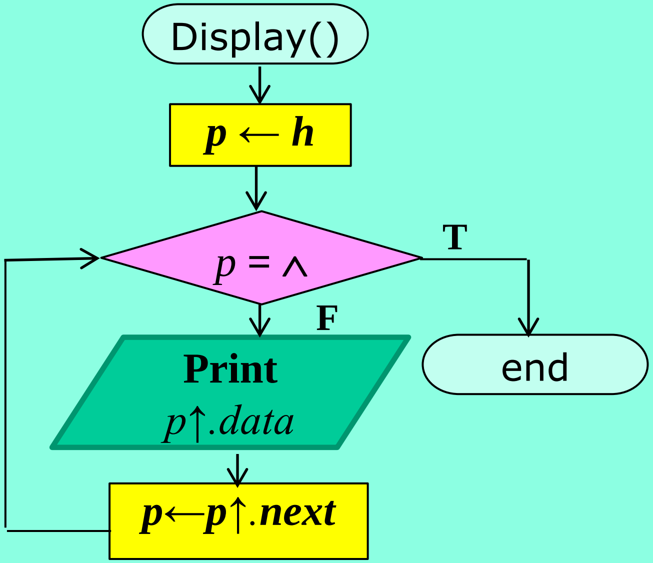
Topic 1: Write an Algorithm to create a single linked list with n nodes.



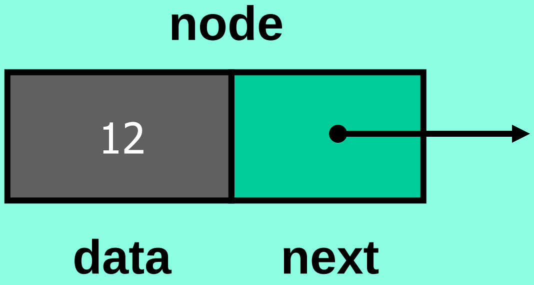
```
void create(int n){
    struct node *p,*q;
    int i,x;
    h=0; q=0;
    for(i=0;i<n;i++){
      p= (struct node *)malloc(sizeof(struct node));
      printf("Enter X:");
      scanf("%d",&x);
      p->data=x;
      if(h==0)
        h=p;
      if(q!=0)
        q->next=p;

      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↑.data: data field value of p pointed node
p↑.next: address of the next of **p** pointed node



```
void display(){
   struct node *p;
   p=h;
   printf("\nThe list is..\n");
   while(p!=0){
     printf("-->%d ",p->data);
     p=p->next;
   }
}
```

C program forLinked List

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

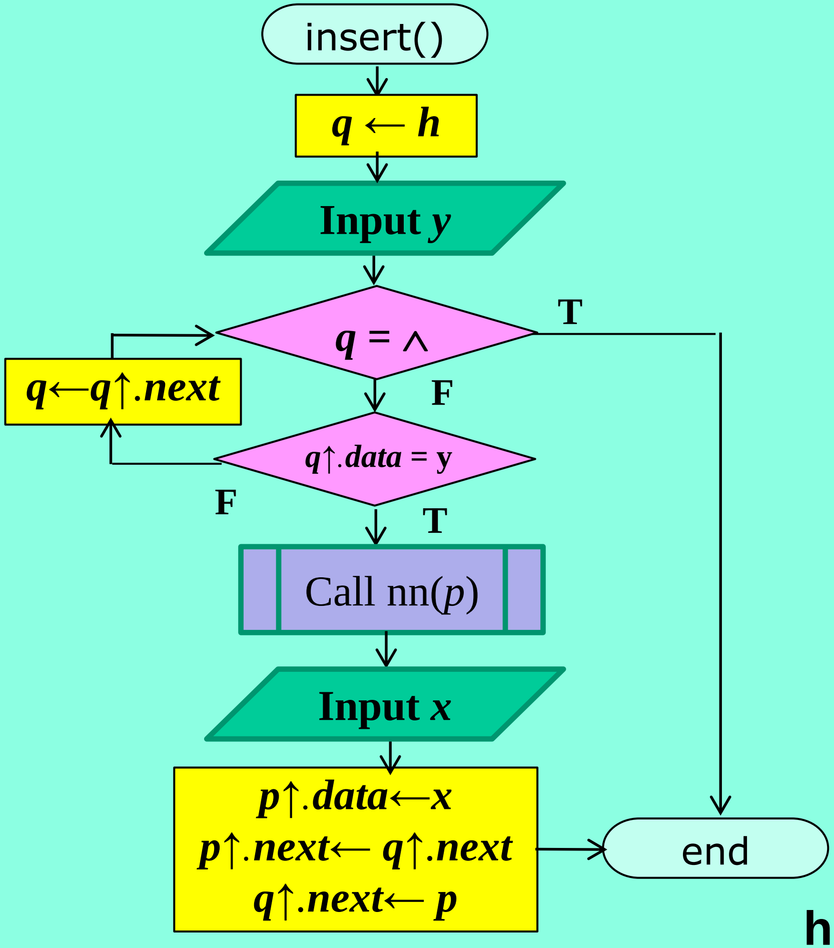
```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *h;
int main()
{
    void insert(int n);
    void display();
    insert(5);
    display();
    return 0;
}
```

```
void insert(int n){
    struct node *p,*q;
    int i,x;
    h=0; q=0;
    for(i=0;i<n;i++){
      p= (struct node *)malloc(sizeof(struct node));
      printf("Enter X:");
      scanf("%d",&x);
      p->data=x;
      if(h==0)
        h=p;
      if(q!=0)
        q->next=p;
      q=p;
    }
   q->next=0;

}
```

```
void display(){
   struct node *p;
   p=h;
   printf("\nThe list is..\n");
   while(p!=0){
     printf("-->%d ",p->data);
     p=p->next;
   }
}
```

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

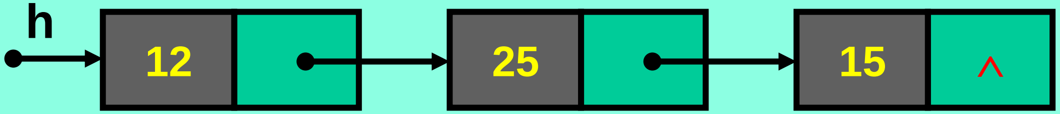
p: pointer to a new node

q↑.*data*: address of the next of ***q*** pointed node

q↑.*next*: address of the next of ***q*** pointed node

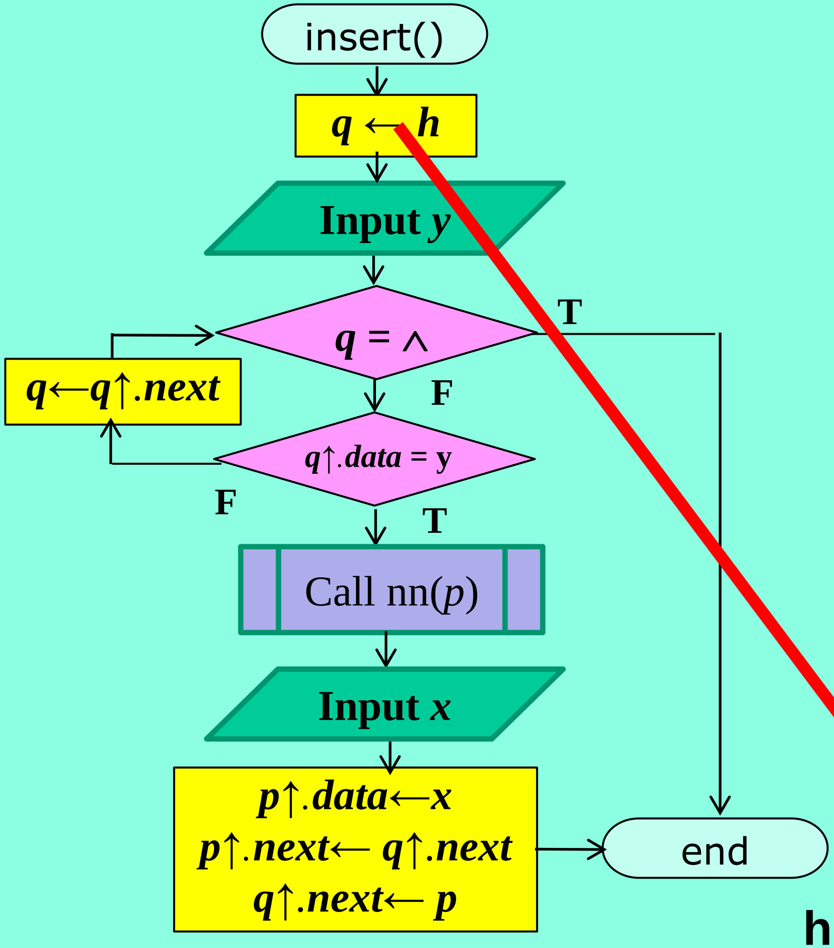
x: input data for new node

y: target node, after *y* node *x* to be inserted

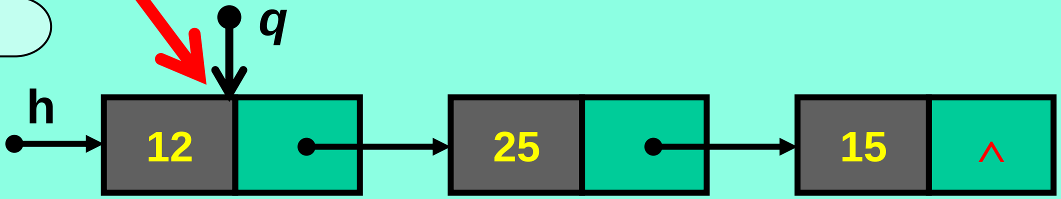


Inserting into a Single Linked List

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

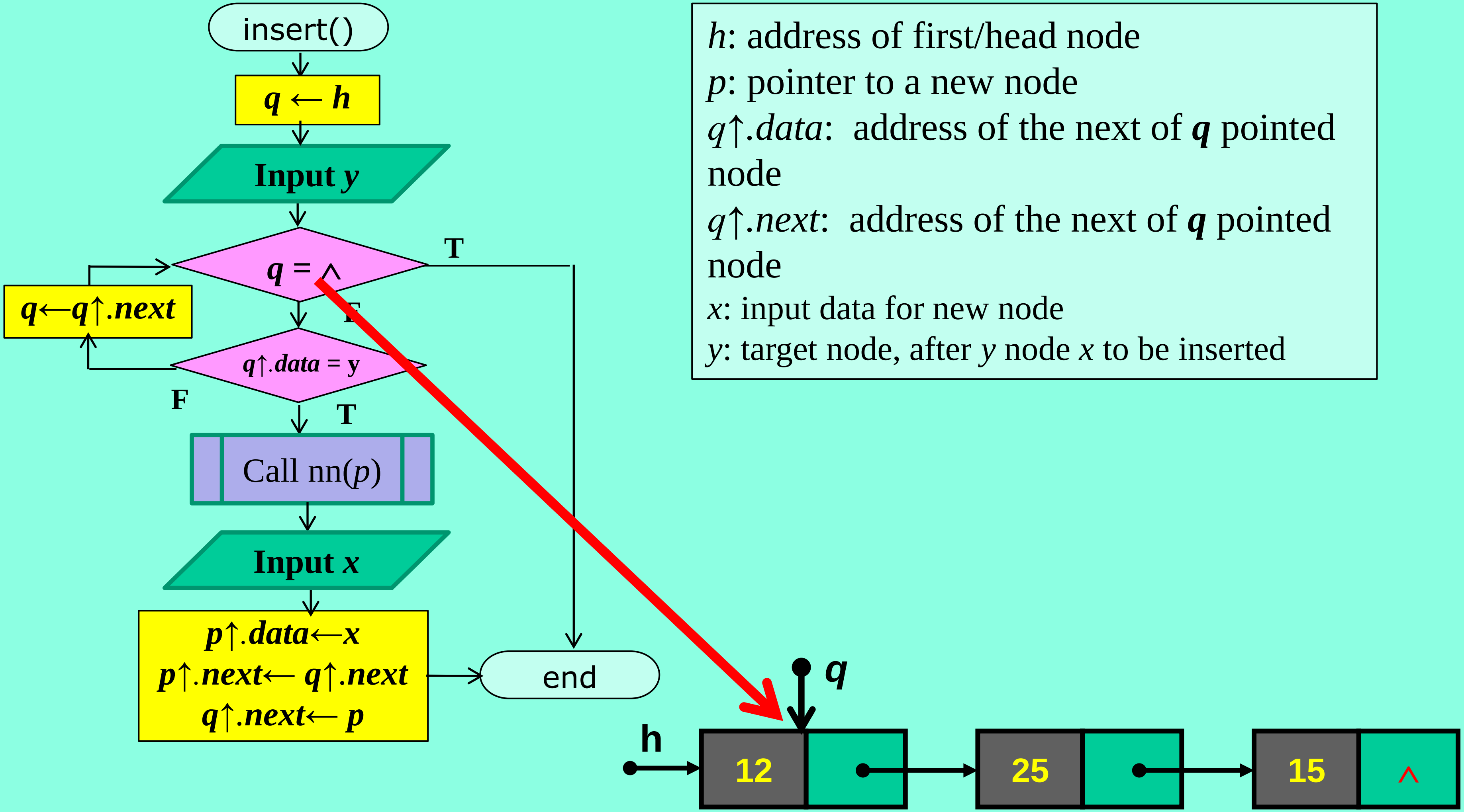


h: address of first/head node
p: pointer to a new node
q↑.*data*: address of the next of ***q*** pointed node
q↑.*next*: address of the next of ***q*** pointed node
x: input data for new node
y: target node, after *y* node *x* to be inserted



Inserting into a Single Linked List

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



insert()

q ← *h*

Input *y*

q = ^

T

F

q←*q*↑.*next*

q↑.*data* = y

F

T

Call nn(*p*)

Input *x*

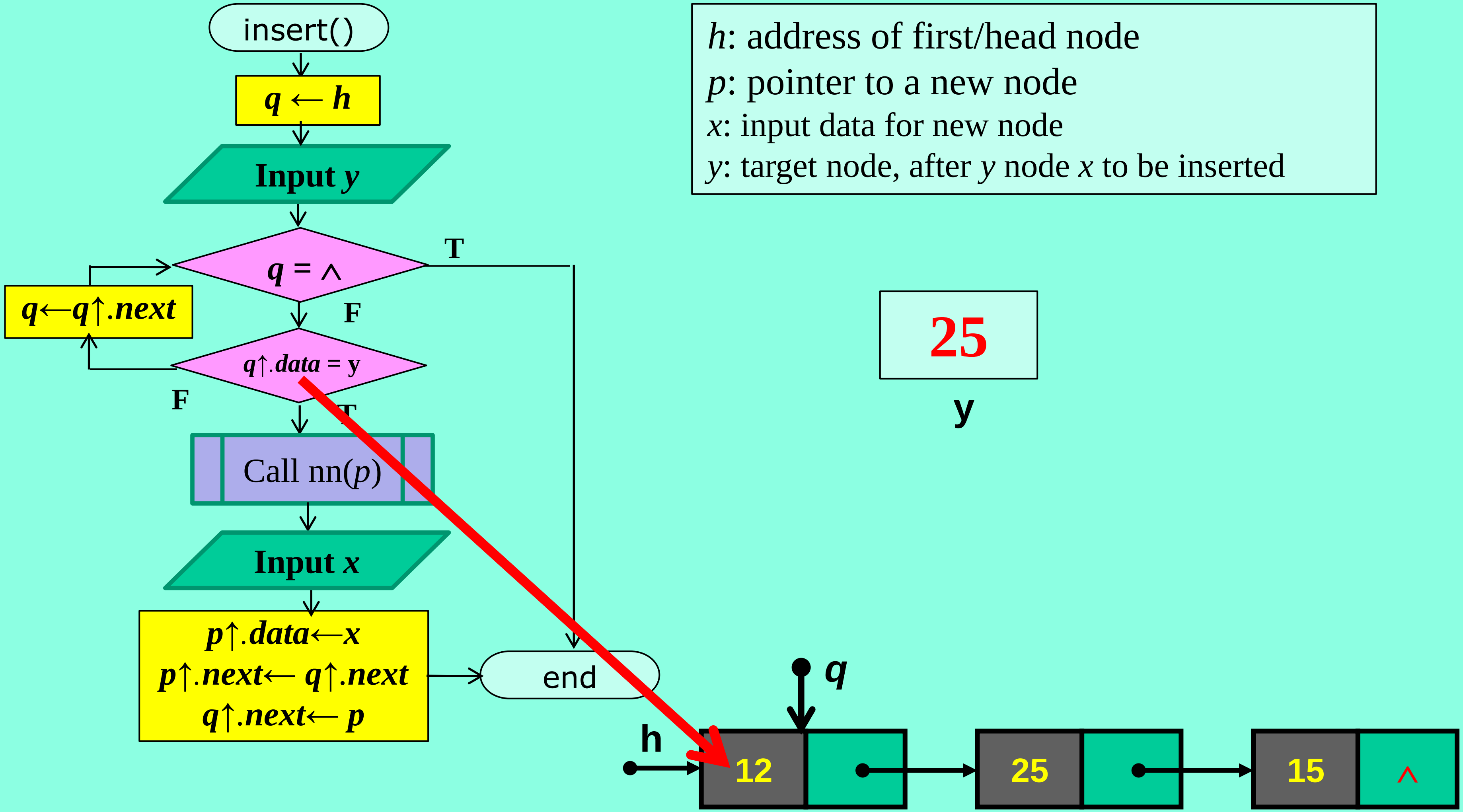
p↑.*data*←*x*
p↑.*next*← *q*↑.*next*
q↑.*next*← *p*

end

h: address of first/head node
p: pointer to a new node
q↑.*data*: address of the next of ***q*** pointed node
q↑.*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

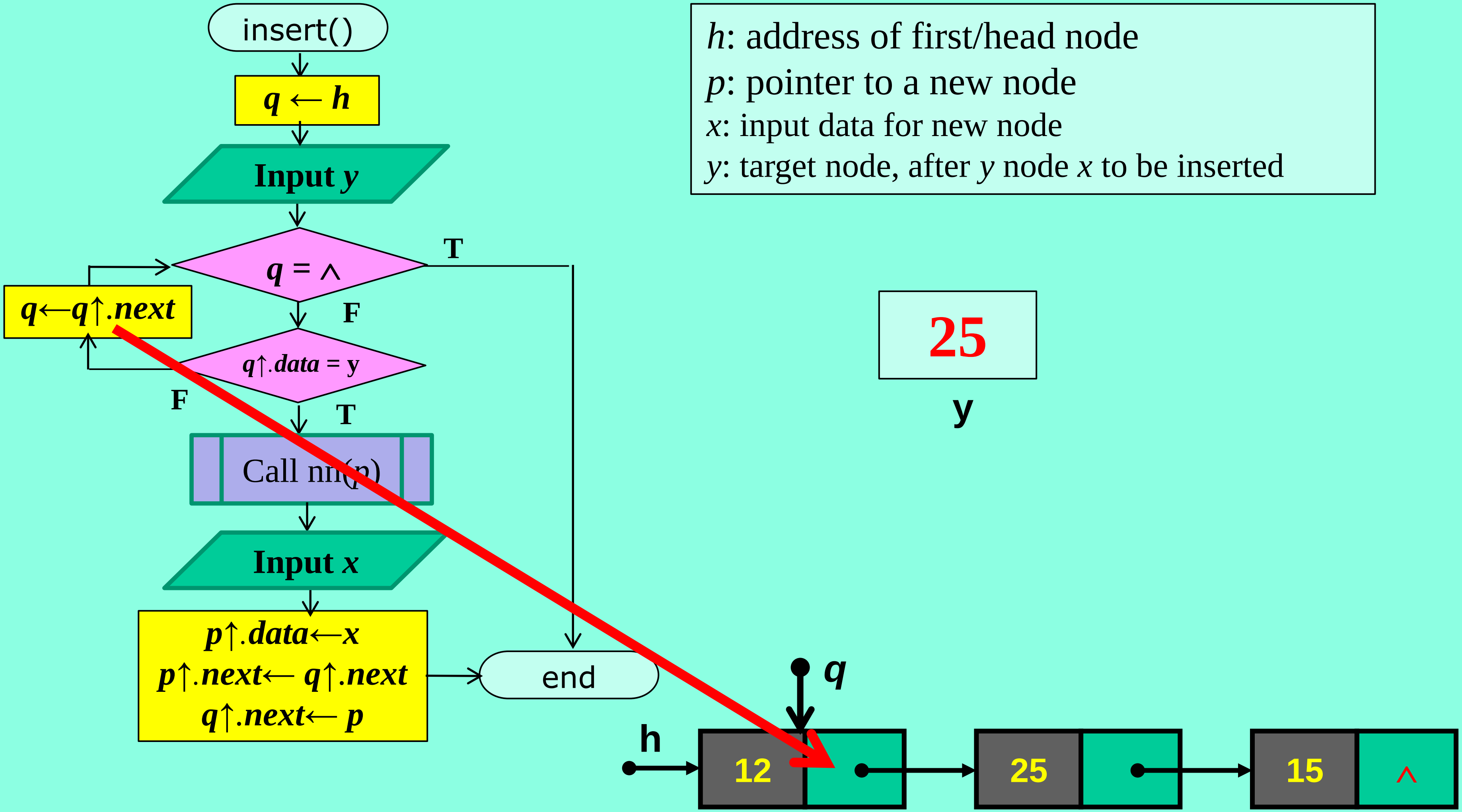
x: input data for new node

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
x: input data for new node
y: target node, after *y* node *x* to be inserted



insert()

q* ← *h

Input *y*

***q* = ^**

T

F

q*←*q*↑.*next

***q*↑.*data* = y**

F

T

Call nn(p)

Input *x*

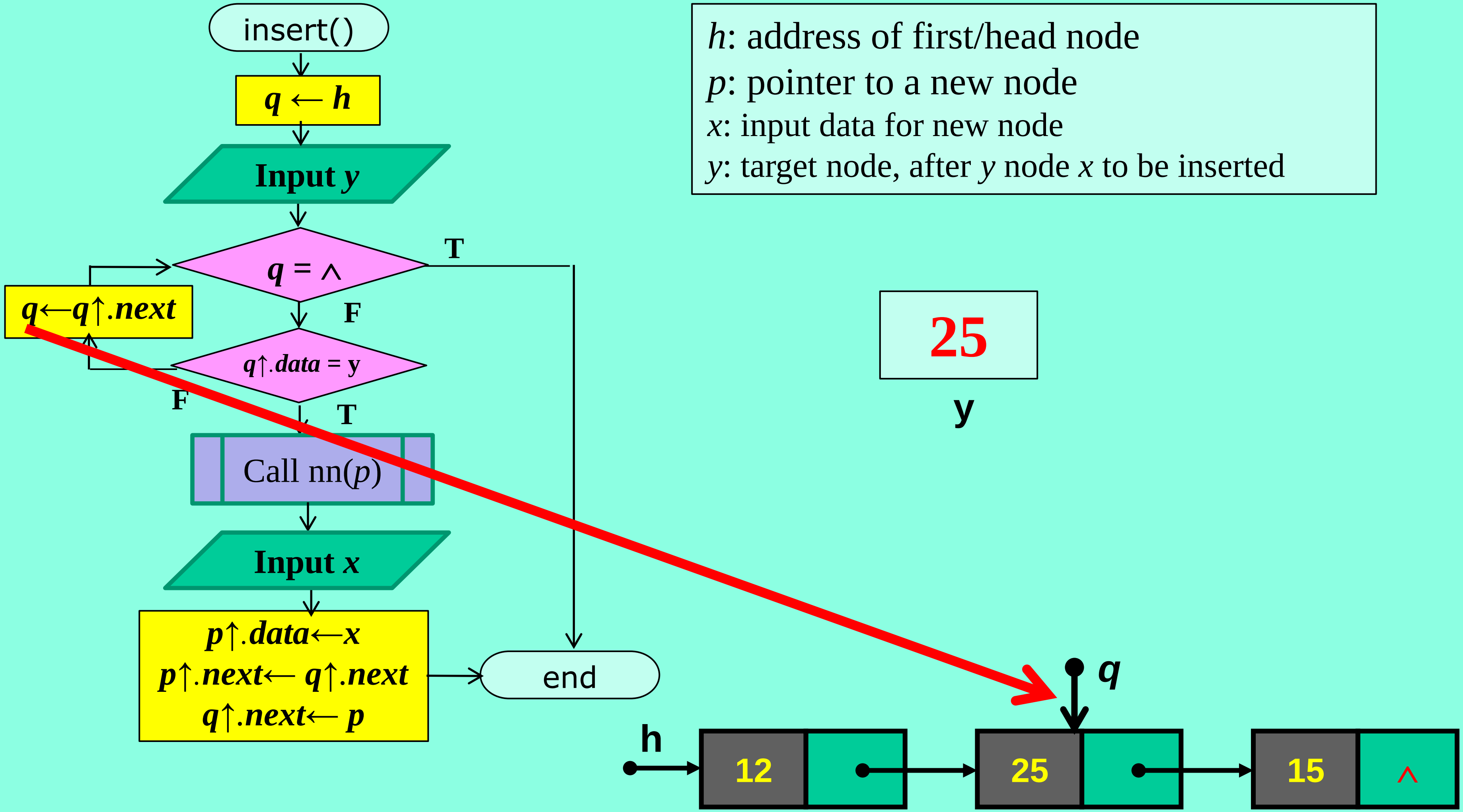
p*↑.*data*←*x
p*↑.*next*← *q*↑.*next
q*↑.*next*← *p

end

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



insert()

q* ← *h

Input *y*

***q* = ^**

T

F

q*←*q*↑.*next

q↑.*data* = y

F

T

Call nn(*p*)

Input *x*

p*↑.*data*←*x
p*↑.*next*← *q*↑.*next
q*↑.*next*← *p

end

Inserting into a Single Linked List

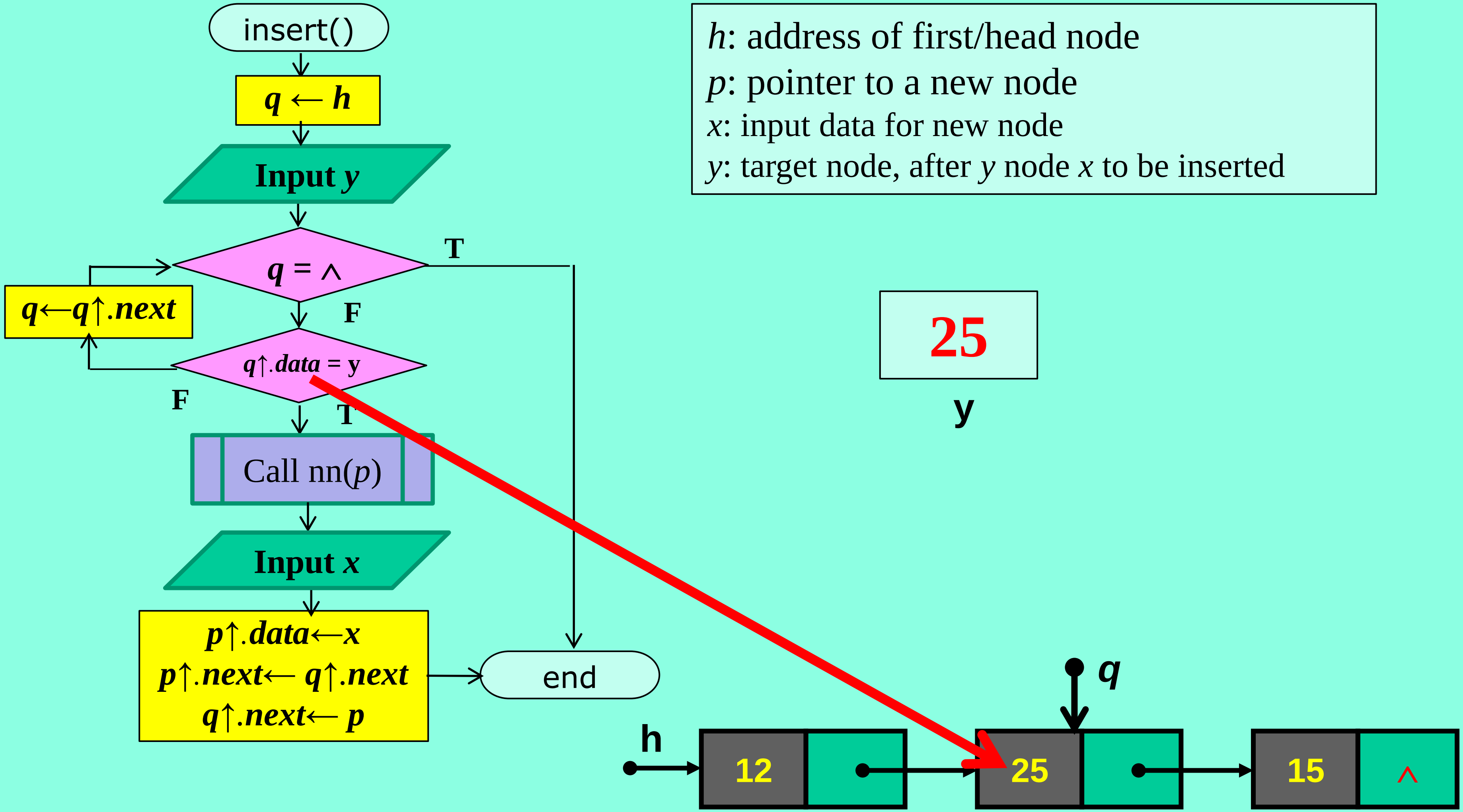
Topic 3: Write an Algorithm to insert a new node in a existing single linked list.

h: address of first/head node

p: pointer to a new node

x: input data for new node

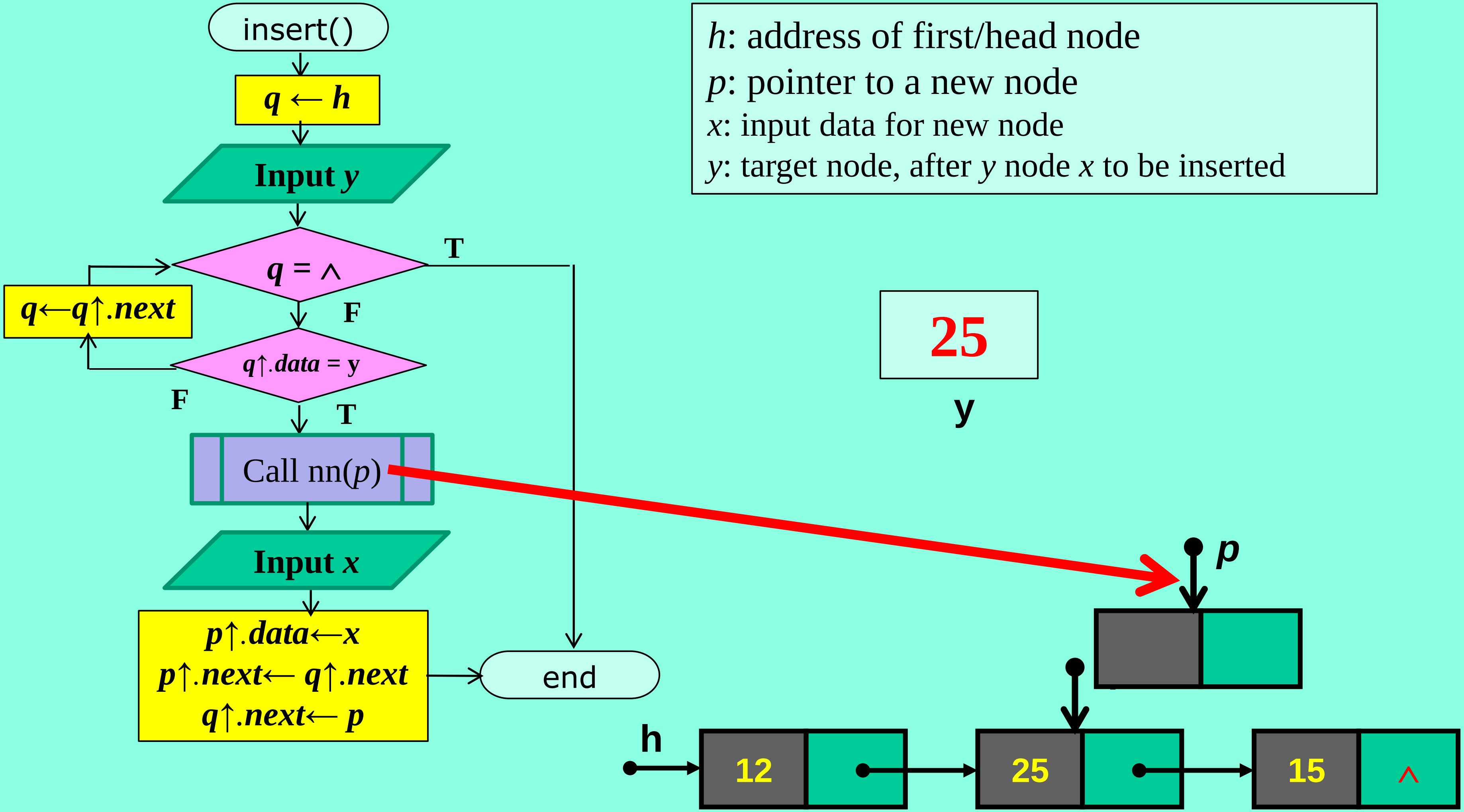
y: target node, after *y* node *x* to be inserted



Inserting into a Single Linked List

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

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



insert()

q* ← *h

Input *y*

***q* = ^**

T

F

q*←*q*↑.*next

***q*↑.*data* = y**

F

T

Call nn(*p*)

Input *x*

p*↑.*data*←*x
p*↑.*next*← *q*↑.*next
q*↑.*next*← *p

end

25

y

p

h

12

25

15

^

Inserting into a Single Linked List

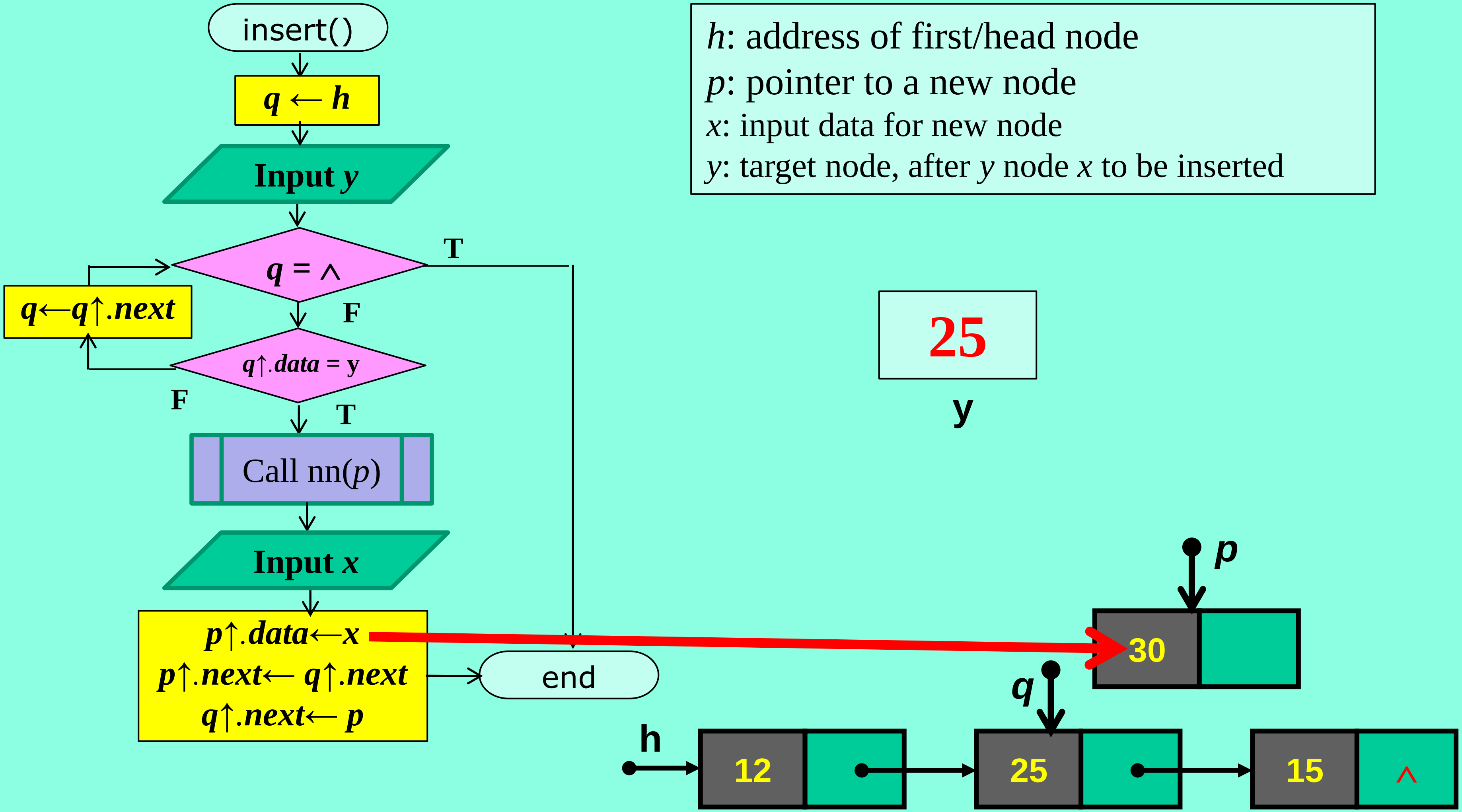
Topic 3: Write an Algorithm to insert a new node in a existing single linked list.

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



insert()

q ← h

Input *y*

***q* = ^**

T

F

q←q↑.next

q↑.data = y

F

T

Call nn(*p*)

Input *x*

p↑.data←x

p↑.next← q↑.next

q↑.next← p

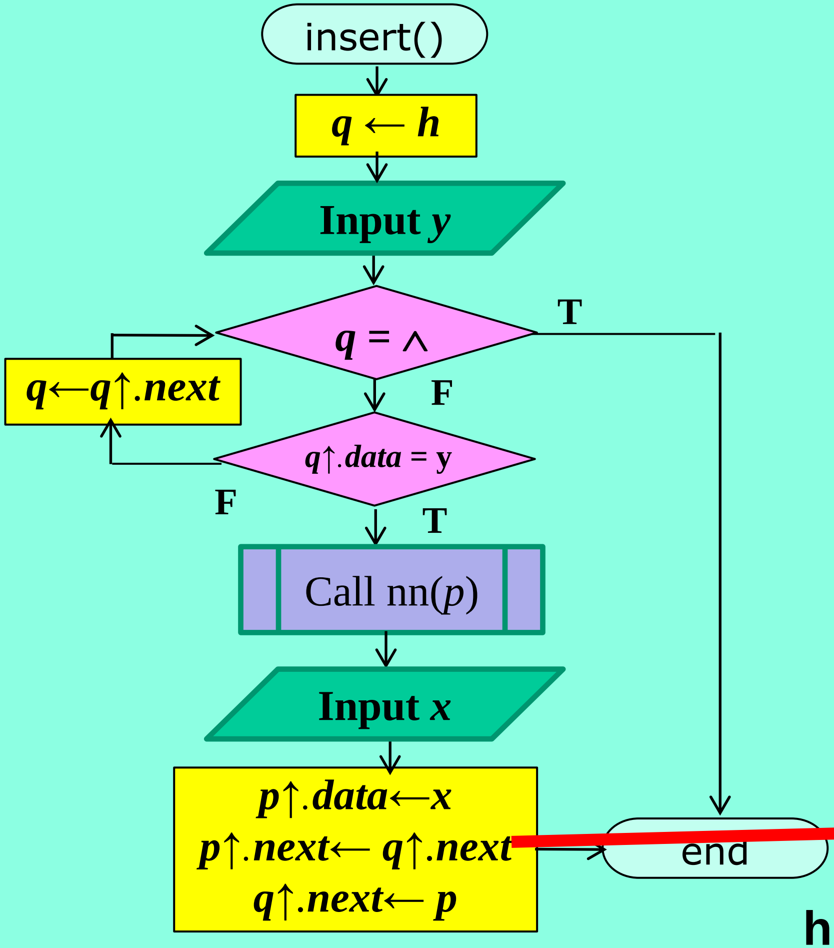
end

25

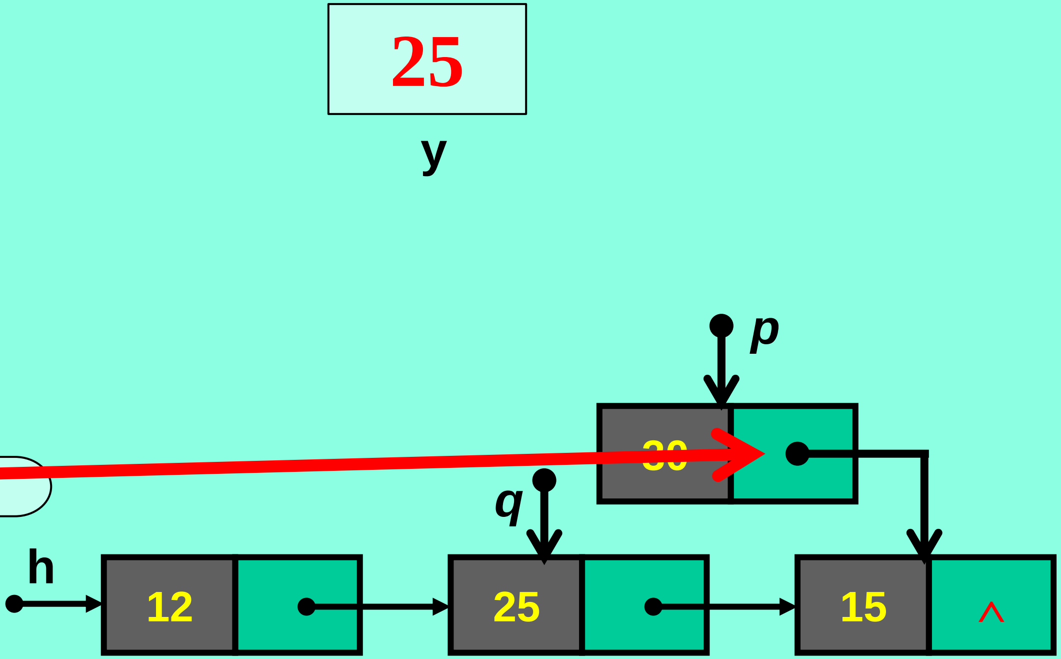
y

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

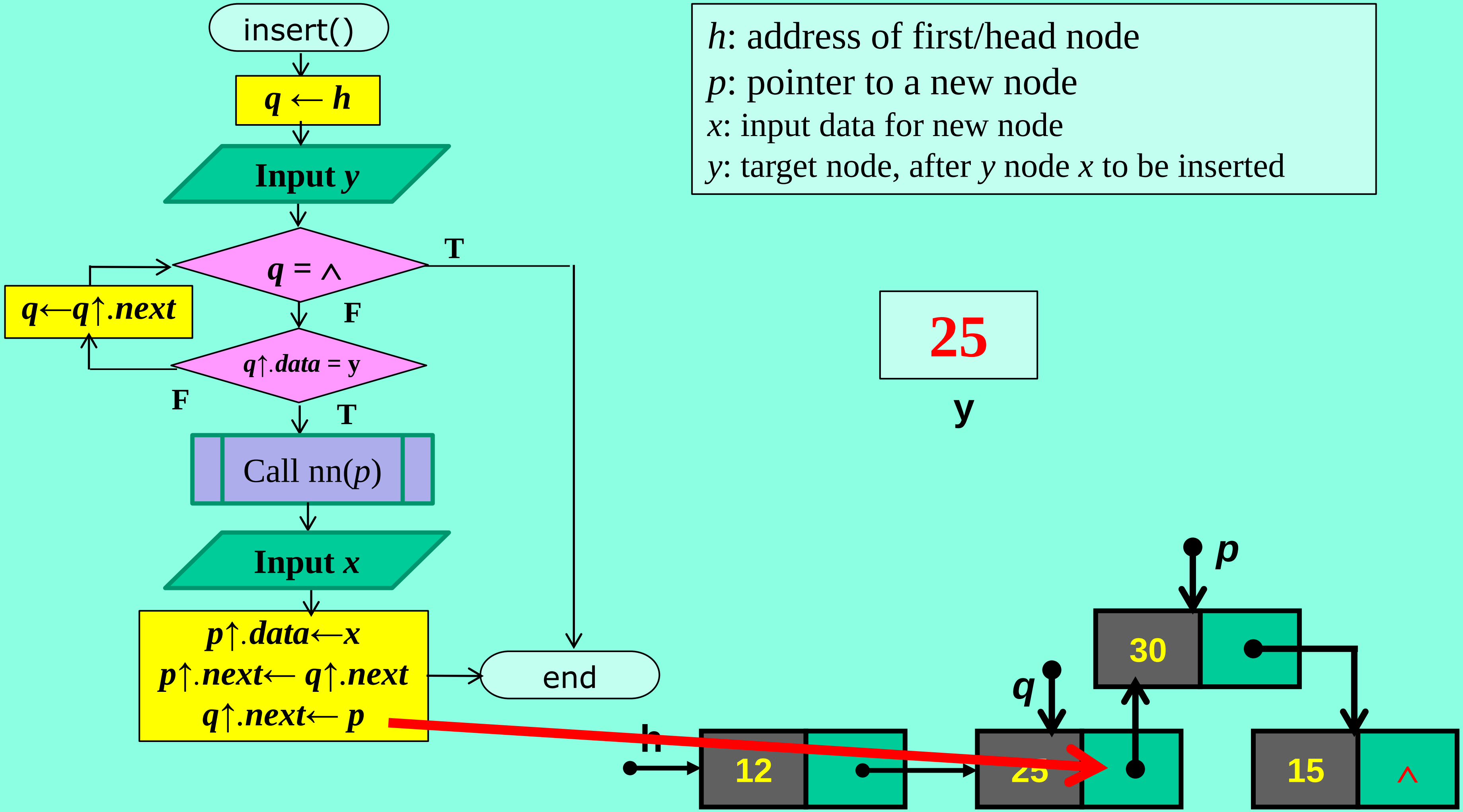
Topic 3: Write an Algorithm to insert a new node in a existing single linked list.

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



insert()

q ← *h*

Input *y*

q = ^

T

F

q←*q*↑.*next*

q↑.*data* = y

F

T

Call nn(*p*)

Input *x*

p↑.*data*←*x*

p↑.*next*← *q*↑.*next*

q↑.*next*← *p*

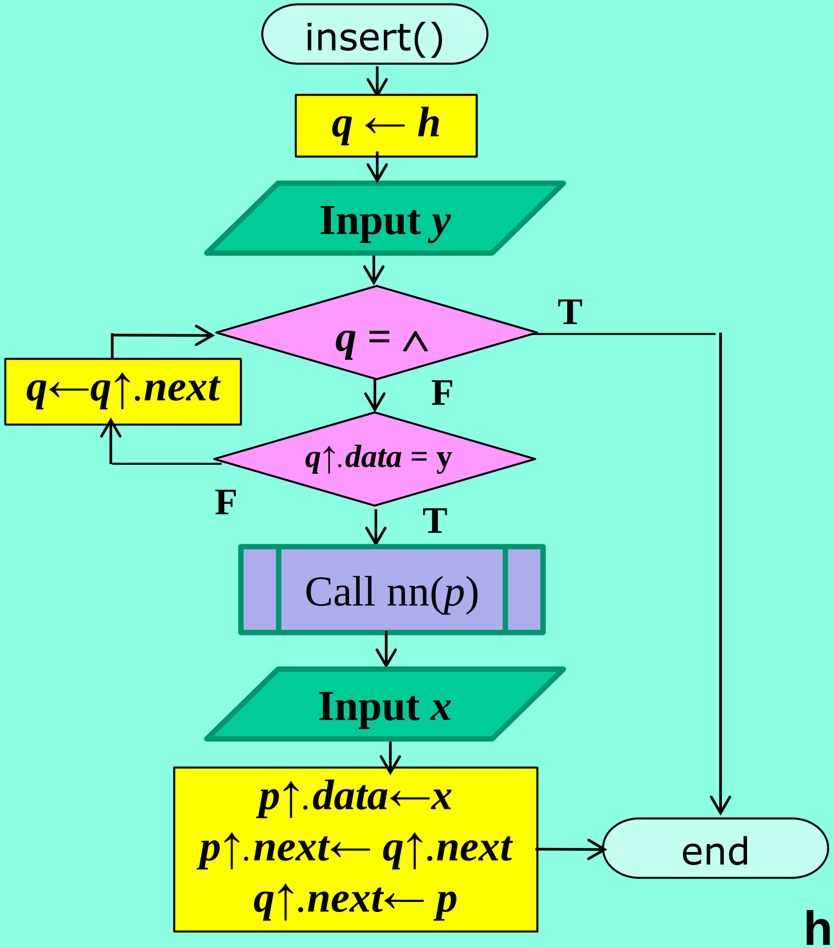
end

25

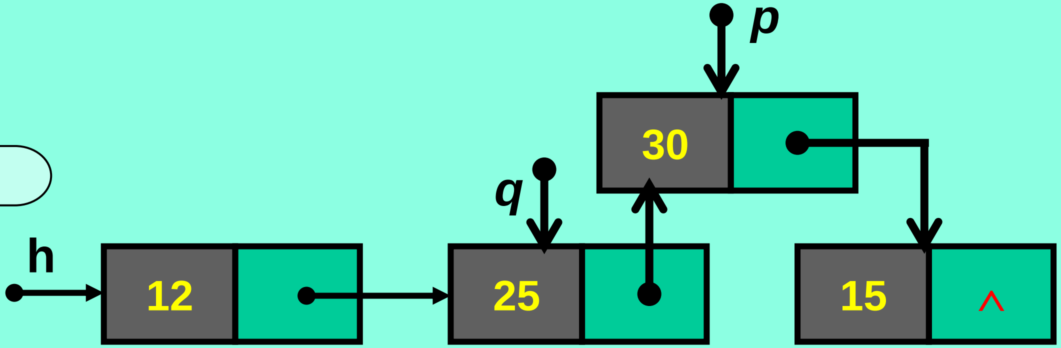
y

Inserting into a Single Linked List

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

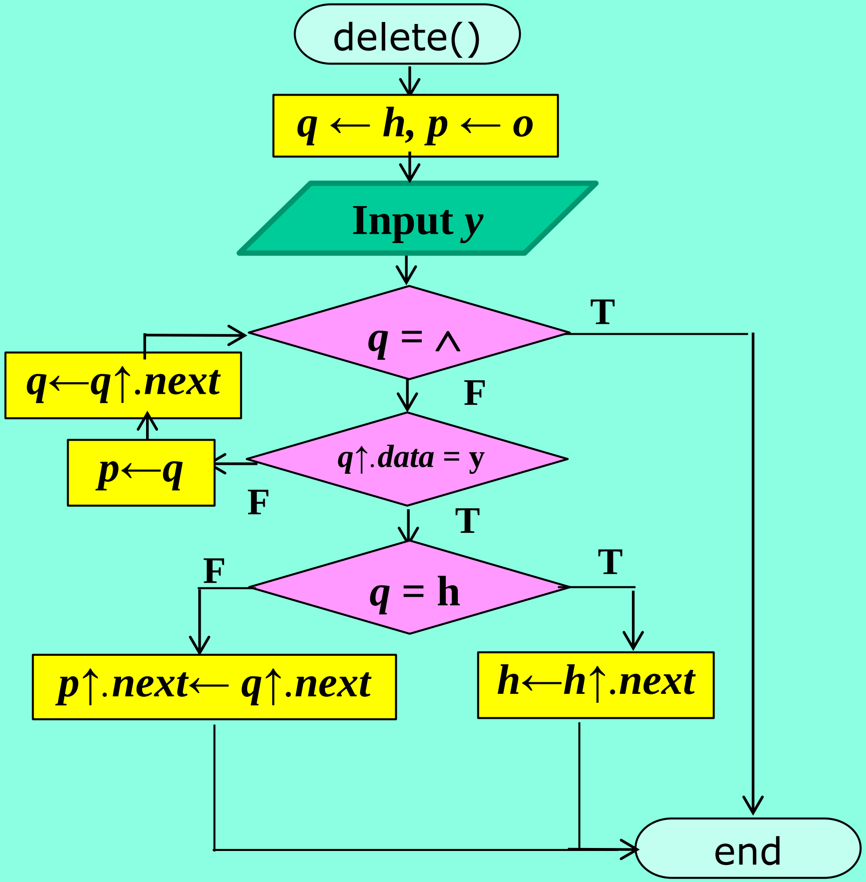


```
void insert(){
 struct node *q,*p;
 int x,y;
 q=h;
 printf("\nAfter which element:");
 scanf("%d",&y);
 while(p!=0){
   if(q->data==y){
     p= (struct node *)malloc(sizeof(struct node));
     printf("Enter data:");
     scanf("%d",&x);
     p->data=x;   p->next=q->next;  q->next=p;    break;
   }
   else
       q=q->next;
  }
}
```



Deleting in a Single Linked List

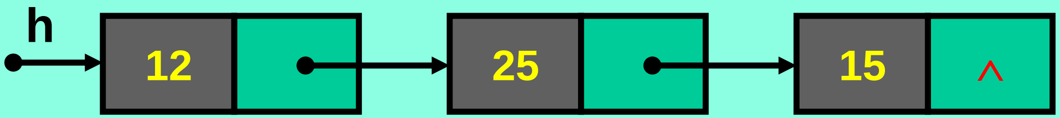
Topic 3: Write an Algorithm to Delete an existing node in a single linked list.



h: address of first/head node

q,*p*: pointer to a existing node

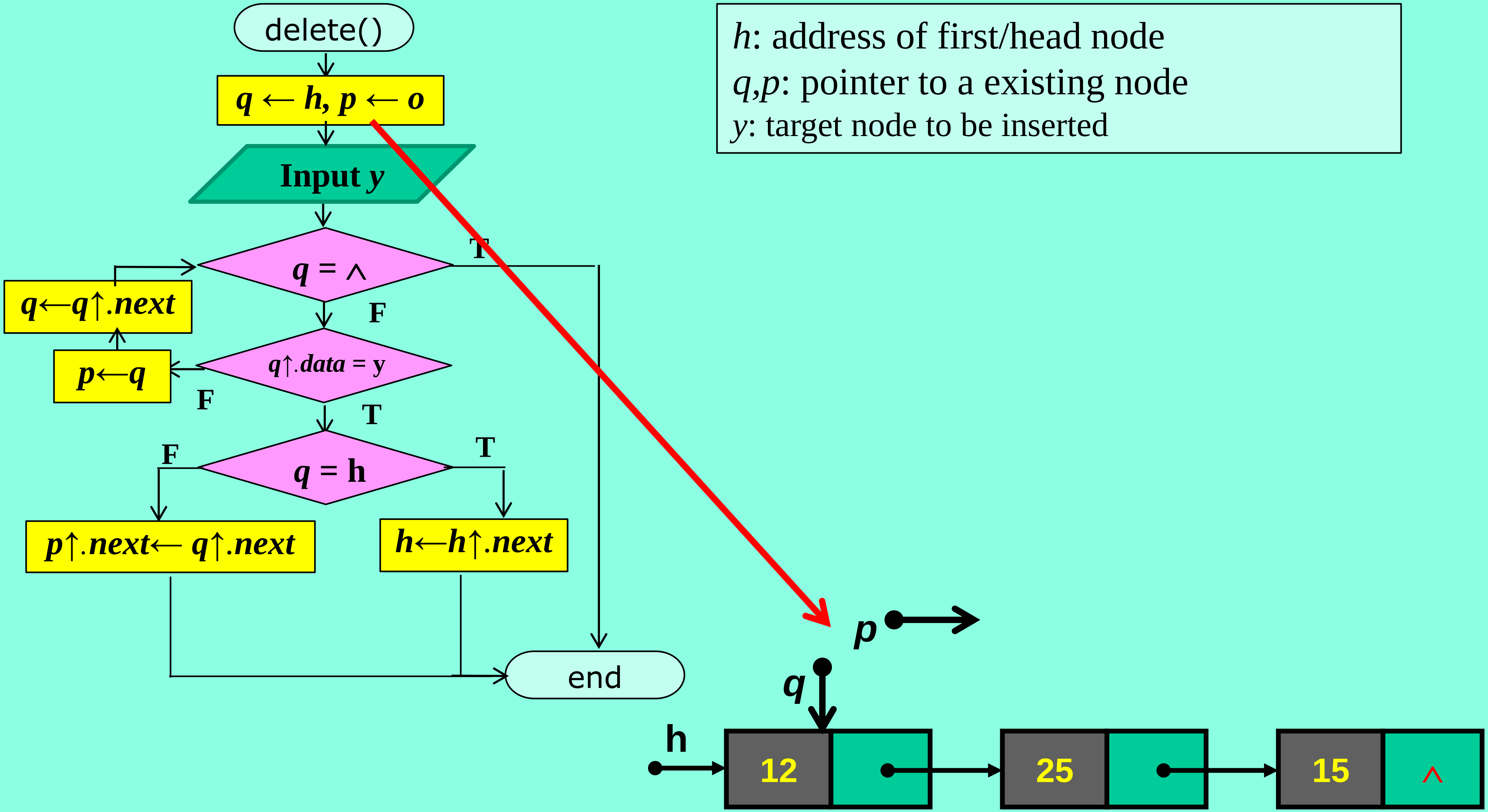
y: target node to be deleted



Deleting in a Single Linked List

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

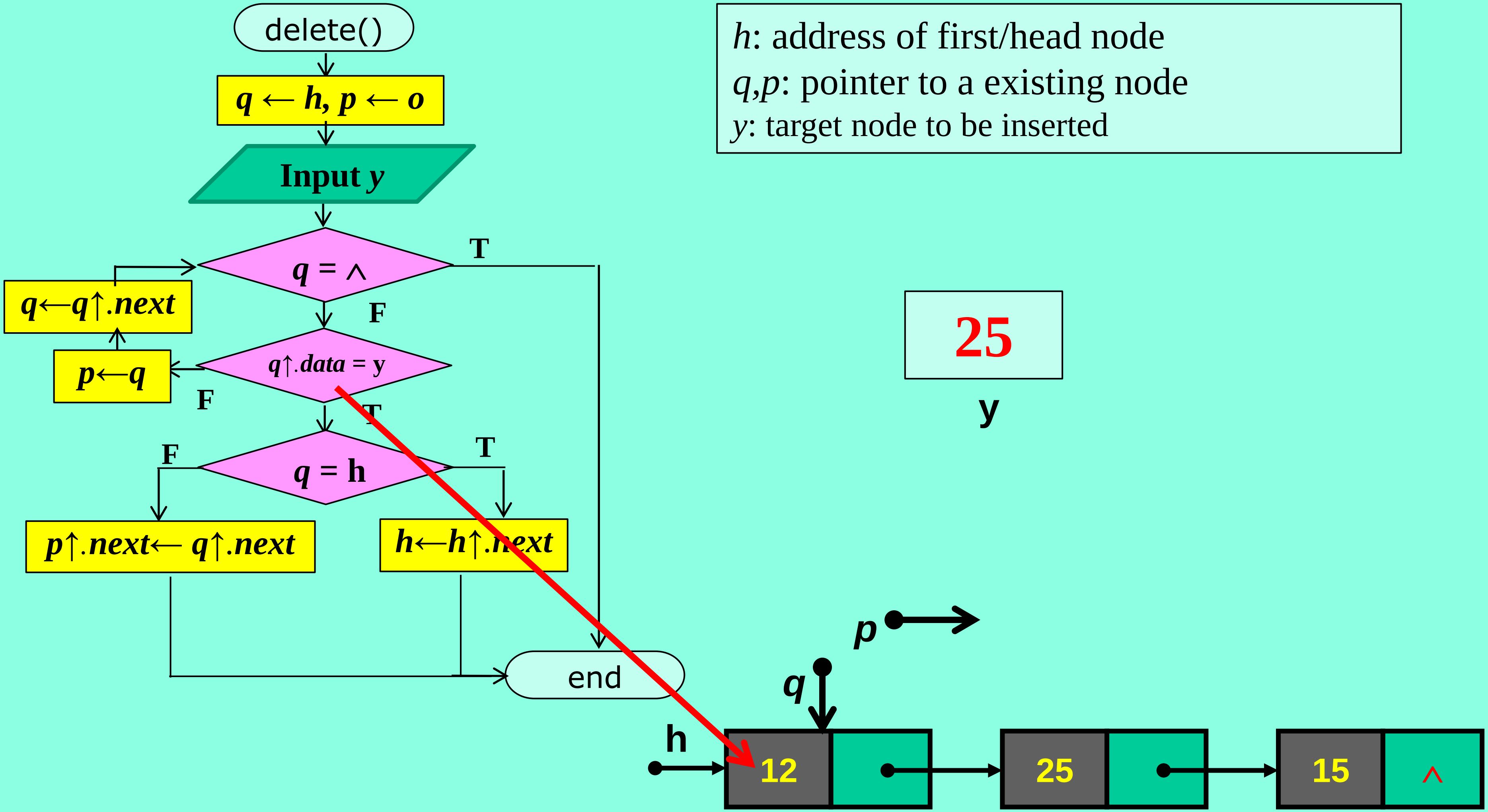
h: address of first/head node
q,*p*: pointer to a existing node
y: target node to be inserted



Deleting in a Single Linked List

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

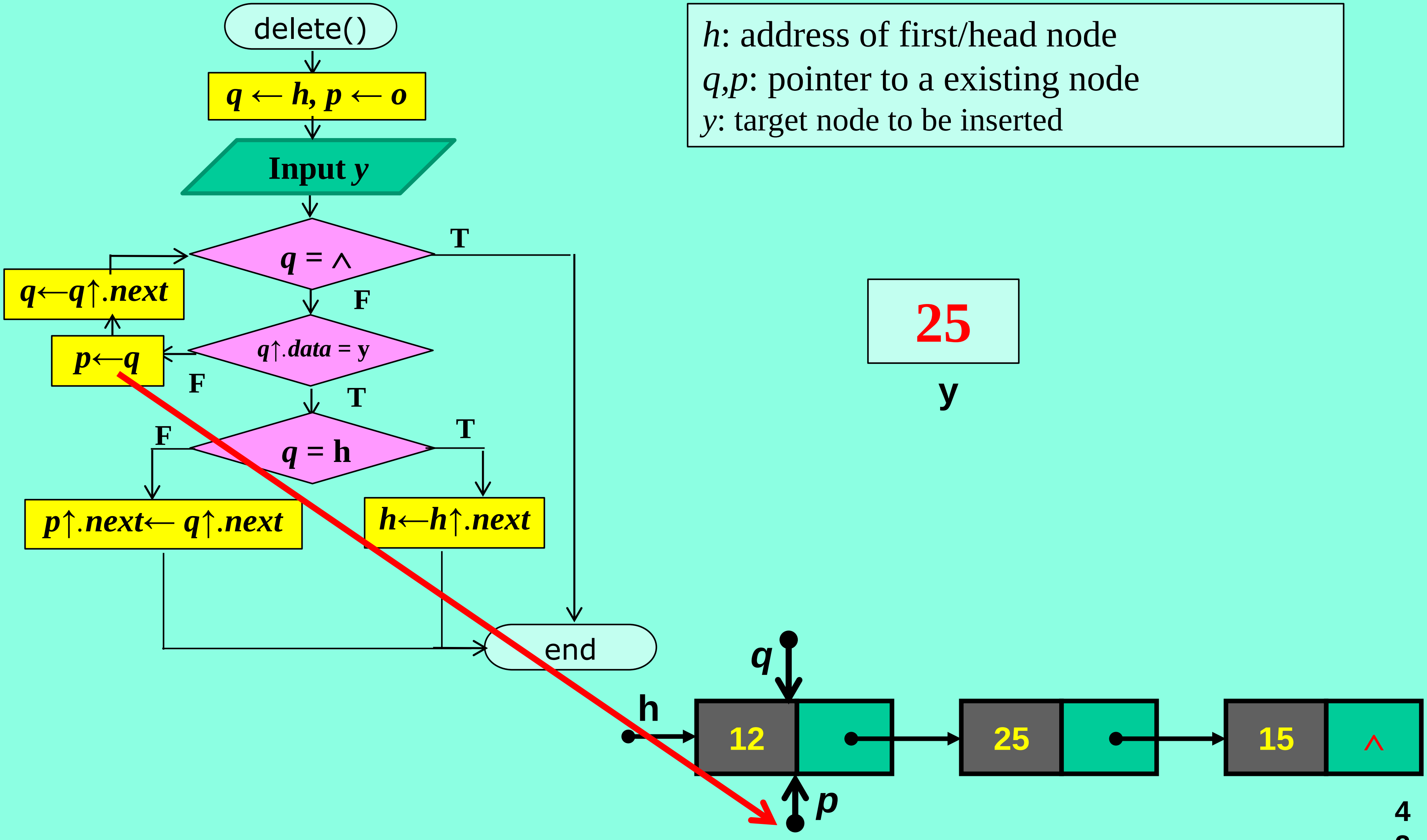
h: address of first/head node
q,*p*: pointer to a existing node
y: target node to be inserted



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



delete()

q ← *h*, *p* ← *o*

Input *y*

q = ^

T

F

q←*q*↑.*next*

p←*q*

q↑.*data* = y

F

T

F

q = h

T

p↑.*next*← *q*↑.*next*

h←*h*↑.*next*

end

25

y

q

h

12

25

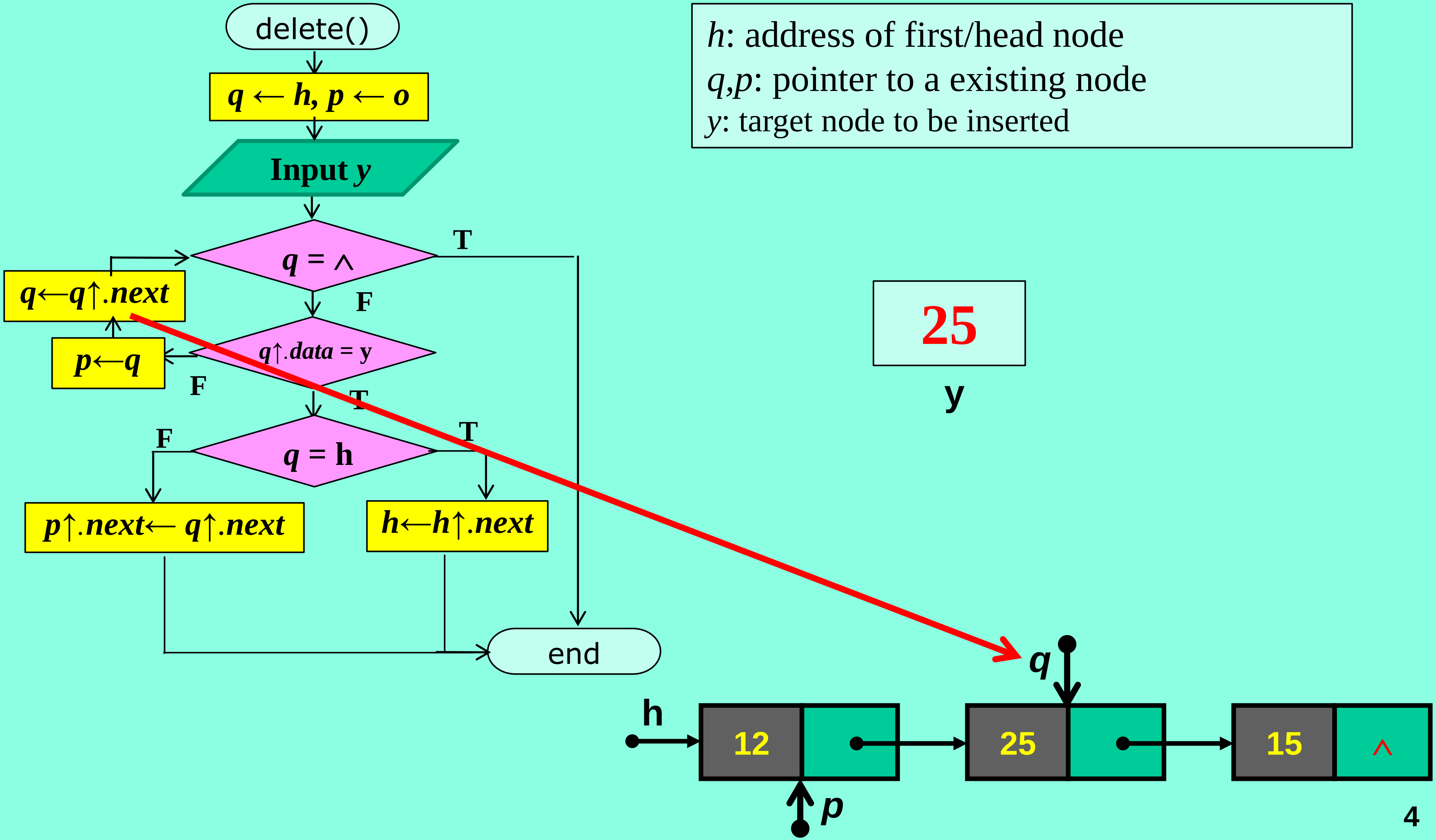
15

^

p

Deleting in a Single Linked List

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

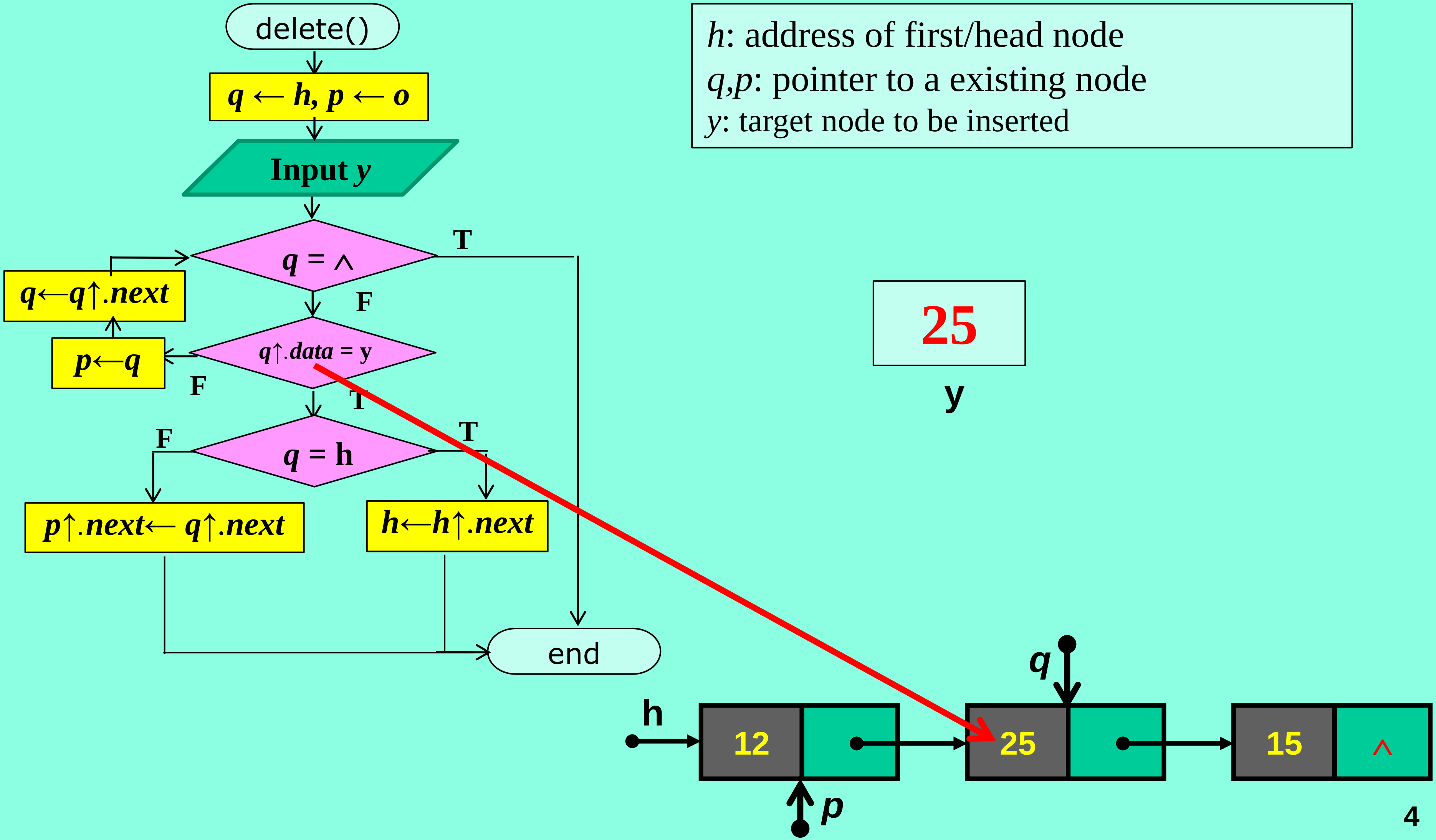


h: address of first/head node
q,p: pointer to a existing node
y: target node to be inserted

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



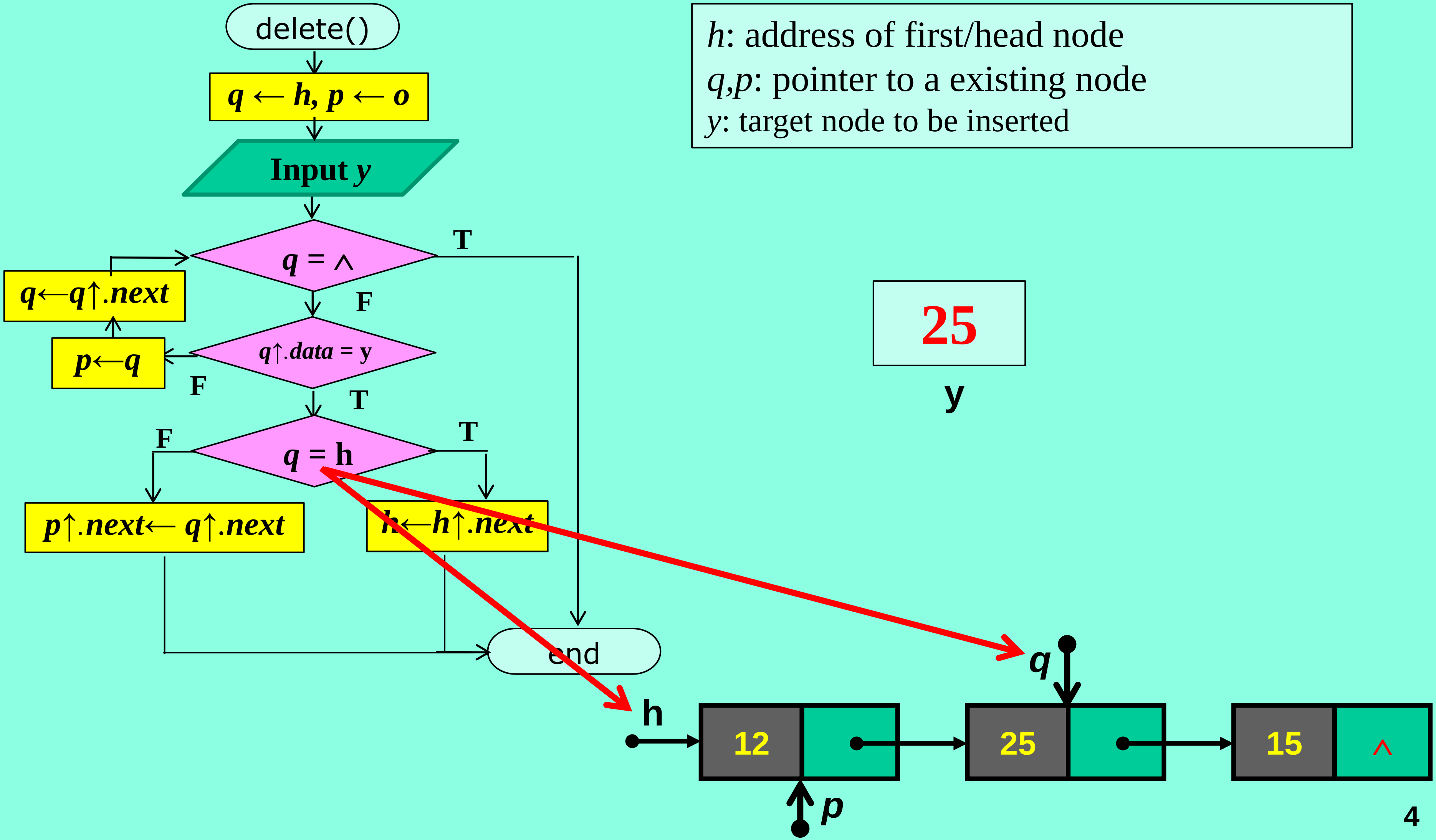
Deleting in a Single Linked List

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

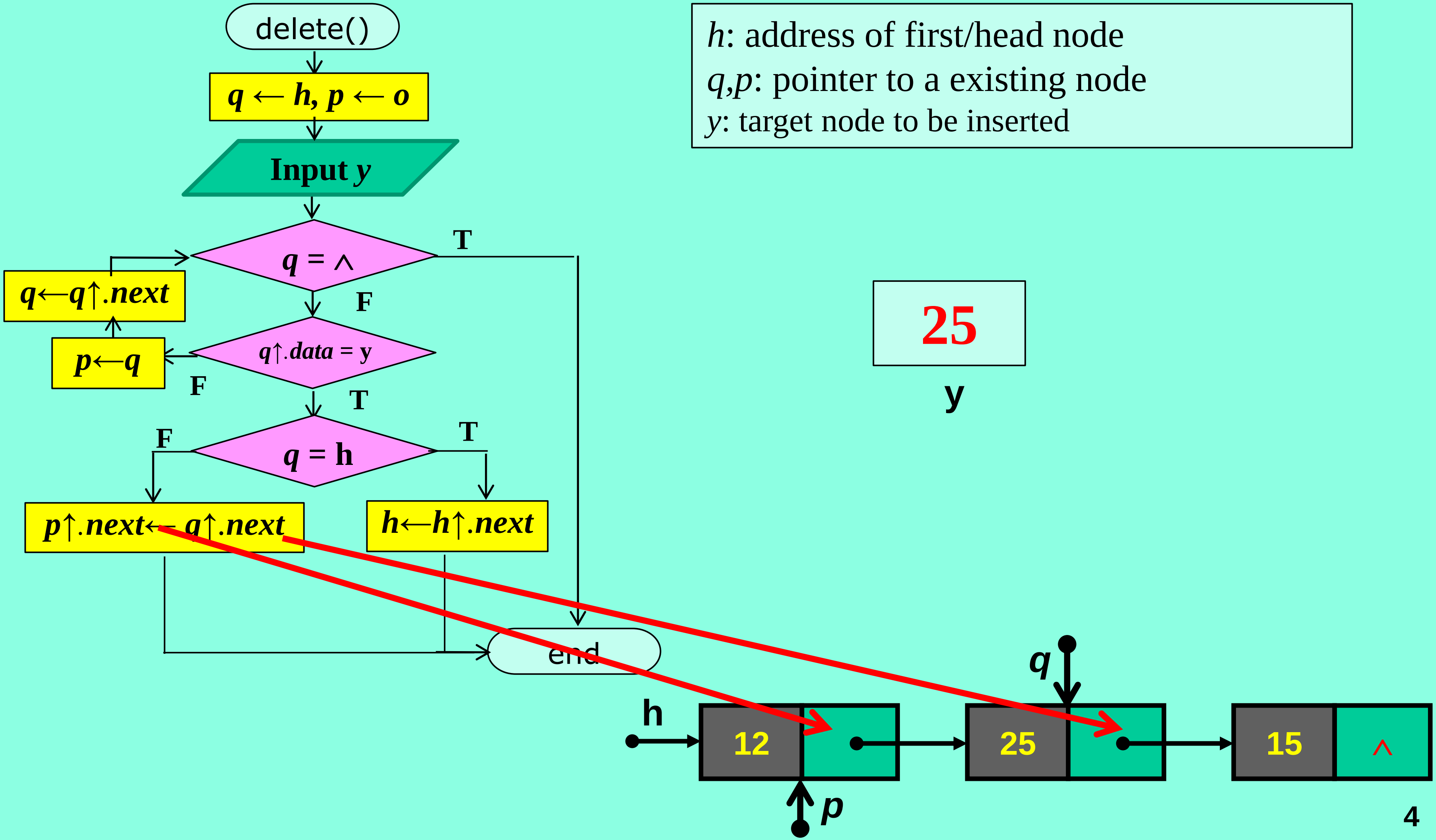
y: target node to be inserted



Deleting in a Single Linked List

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

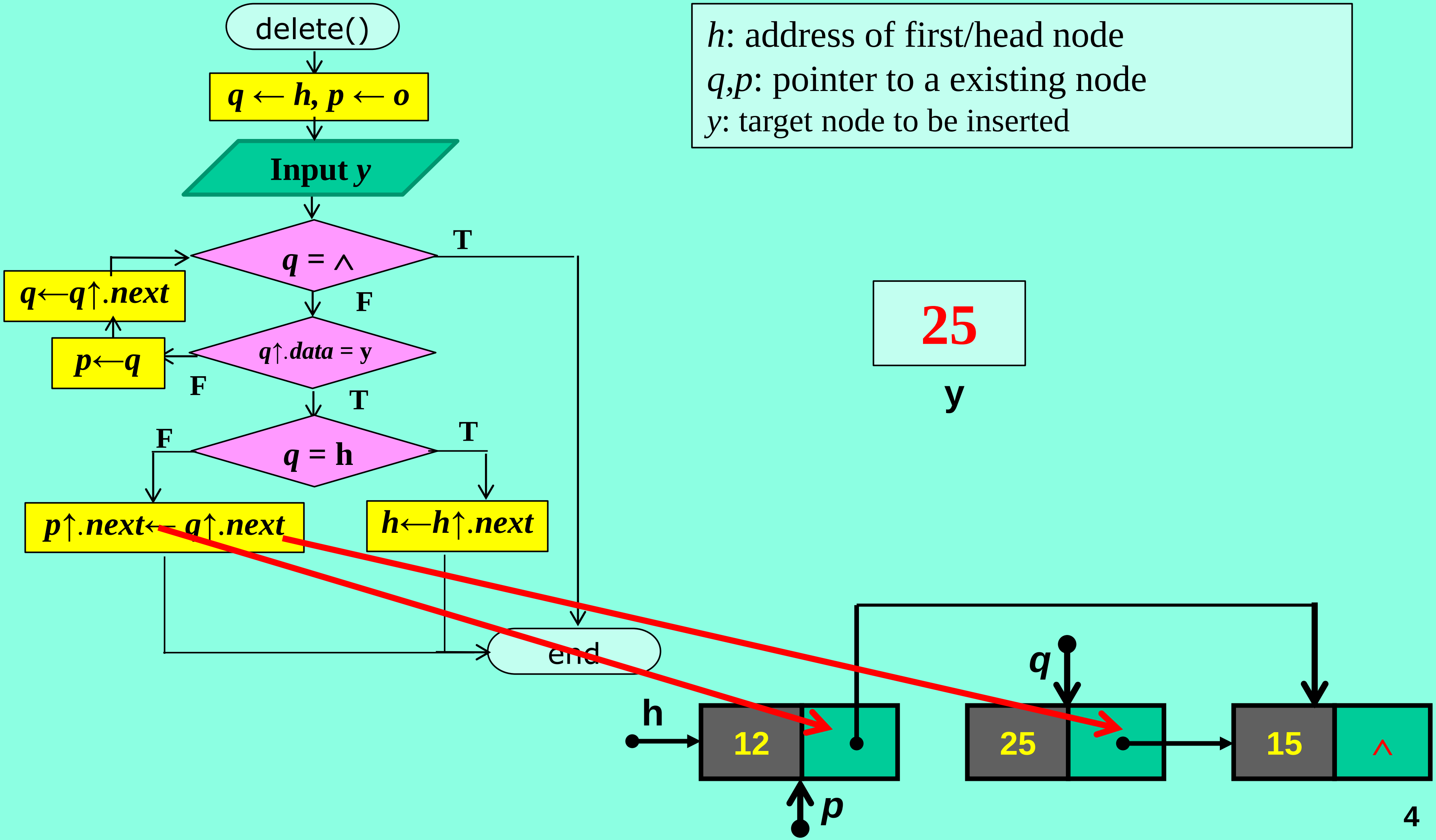
h: address of first/head node
q,*p*: pointer to a existing node
y: target node to be inserted



Deleting in a Single Linked List

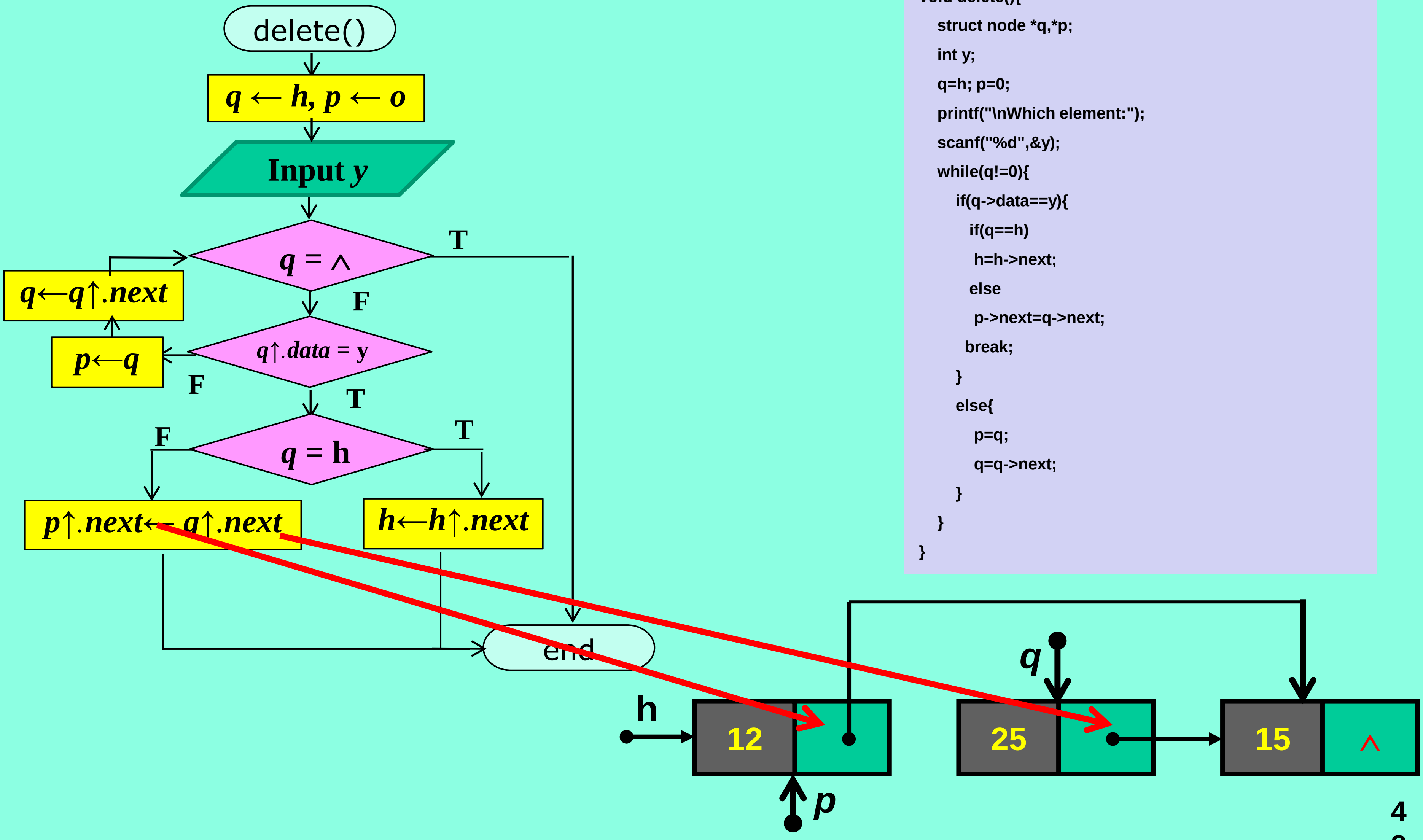
Topic 3: Write an Algorithm to Delete an existing node in a single linked list.

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.



```
void delete(){
    struct node *q,*p;
    int y;
    q=h; p=0;
    printf("\nWhich element:");
    scanf("%d",&y);
    while(q!=0){
        if(q->data==y){
           if(q==h)
             h=h->next;
           else
             p->next=q->next;
          break;
        }
        else{
            p=q;
            q=q->next;
        }
    }
}
```

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: Implemtn 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.