CS & IT ENGINEERING

Data Structure

Linked List
Chapter- 3
Lec- 05

By- Pankaj Sharma sir

TOPICS TO BE COVERED

- Linked List-V

The following C function takes a simply linked list p as an argument.

```
struct item
{
                int data;
                struct item * next;
};
int f (struct item *p)
{
        return((p==NULL) || (p→next==NULL)) ||
                ((p → data <= p →next →data) && f(p →next)));

}
```

Template

False True

False

a||b||c && d

0||1||——

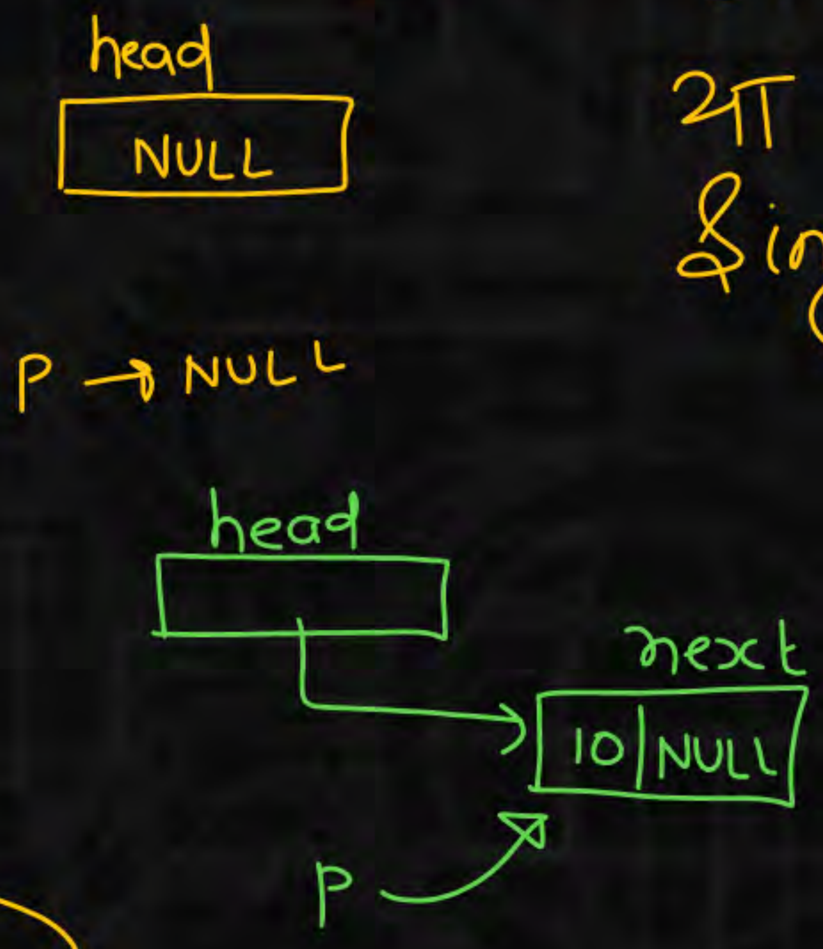
→ 1

Empty

या

Single node

⇒ $\boxed{1}$



The function returns 1 if and only if

(A) The list is empty or has exactly one element

(B) The elements in the list are sorted in non-decreasing order of data value

(C) The elements in the list are sorted in non-increasing order of data value

(D) not all elements in the list have the same data value

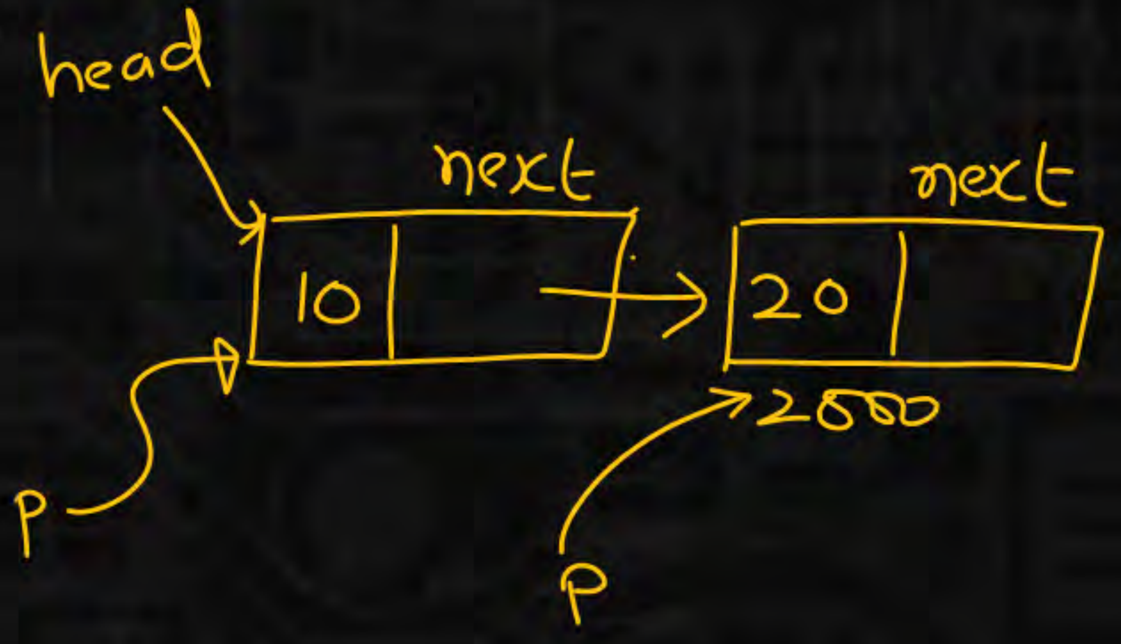
The following C function takes a simply linked list p as an argument.

```
struct item
{
            int data;
            struct item * next;
};
int f (struct item *p)
{

        return((p==NULL) || (p→next==NULL)) ||
              ((p → data <= p →next →data) && f(p →next)));

}
```

Template



The function returns 1 if and only if $\boxed{10<=20}$ True

$a||b|| c \&\& d$

F F

$0||0|| c \&\& d$

$c \&\& d$

True && f(2000)

(A) The list is empty or has exactly one element

(B) The elements in the list are sorted in non-decreasing order of data value

(C) The elements in the list are sorted in non-increasing order of data value

(D) not all elements in the list have the same data value

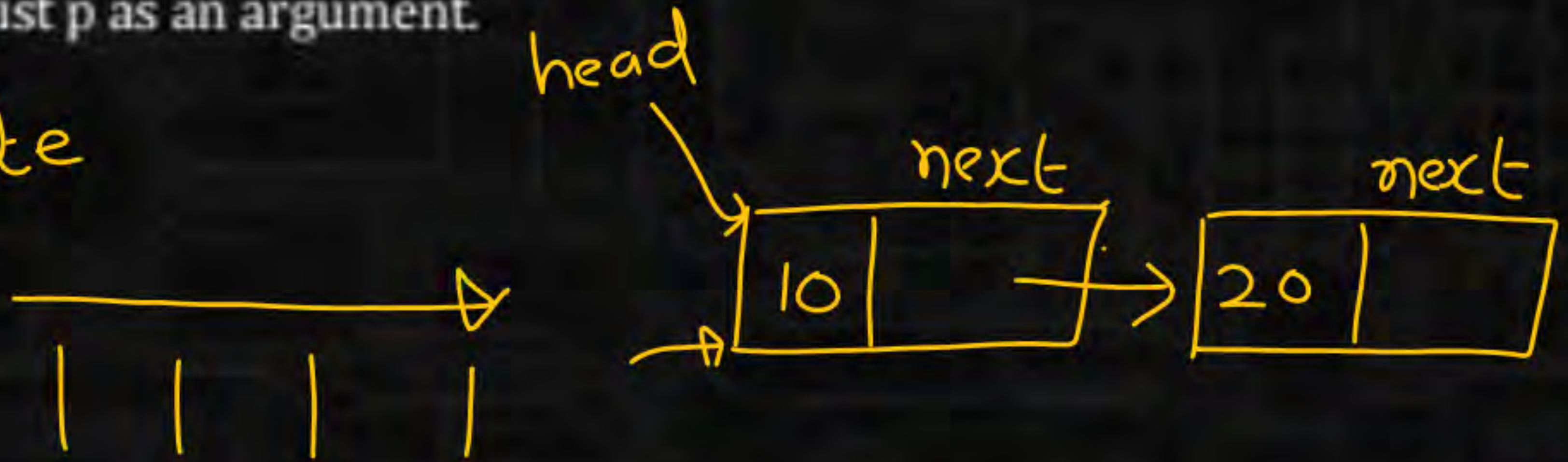
The following C function takes a simply linked list p as an argument.

```
struct item
{
            int data;
            struct item * next;
};
int f (struct item *p)
{

        return((p==NULL) || (p→next==NULL)) ||
                ((p → data <= p →next →data) && f(p →next)));

}
```

Template



increasing order

10 → 20 → 30

Non-decreasing order

10 → 10 → 20

10 → 9 → 40

p

10 < 9

The function returns 1 if and only if

A The list is empty or has exactly one element

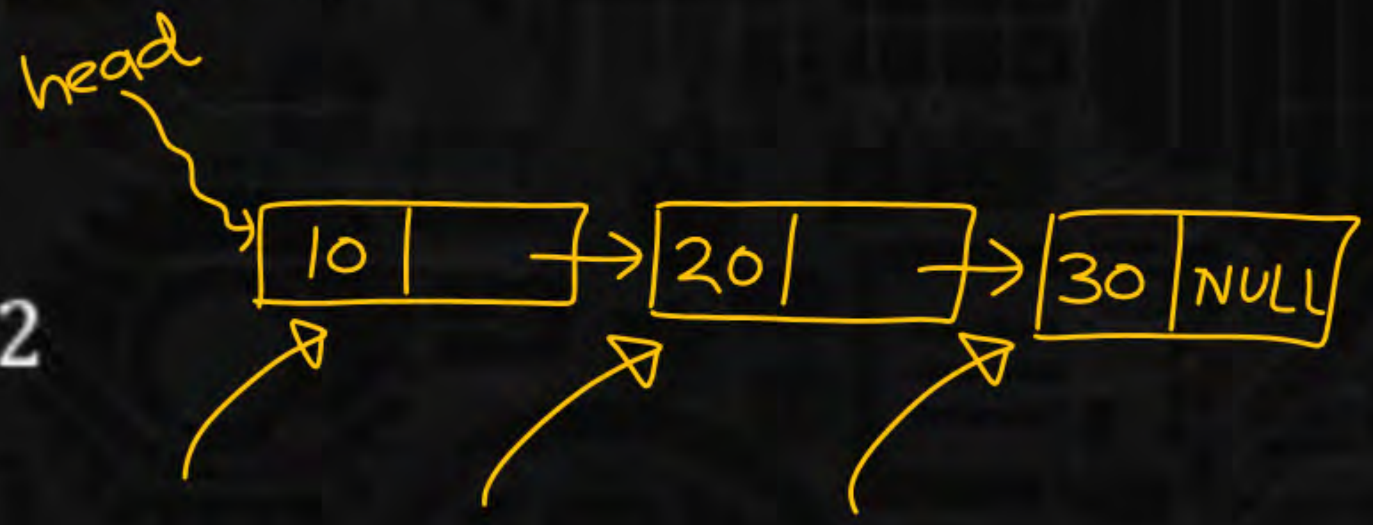
[✓ B] The elements in the list are sorted in non-decreasing order of data value

C The elements in the list are sorted in non-increasing order of data value

D not all elements in the list have the same data value

In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is-

- A log n
- B n/2
- C (log n)-1
- [✓ D] n



100

What is the worst case time complexity to reverse a singly-linked list in O(1) space?

constant

A $\Theta(n^2)$

B $\Theta(n \log n)$

[✓ C] $\Theta(n)$

D Not possible

What is the worst case time complexity of inserting n elements into an empty linked list, if the linked list needs to be maintained in sorted order?

A $\Theta(n^2)$

B $\Theta(1)$

C $\Theta(n)$

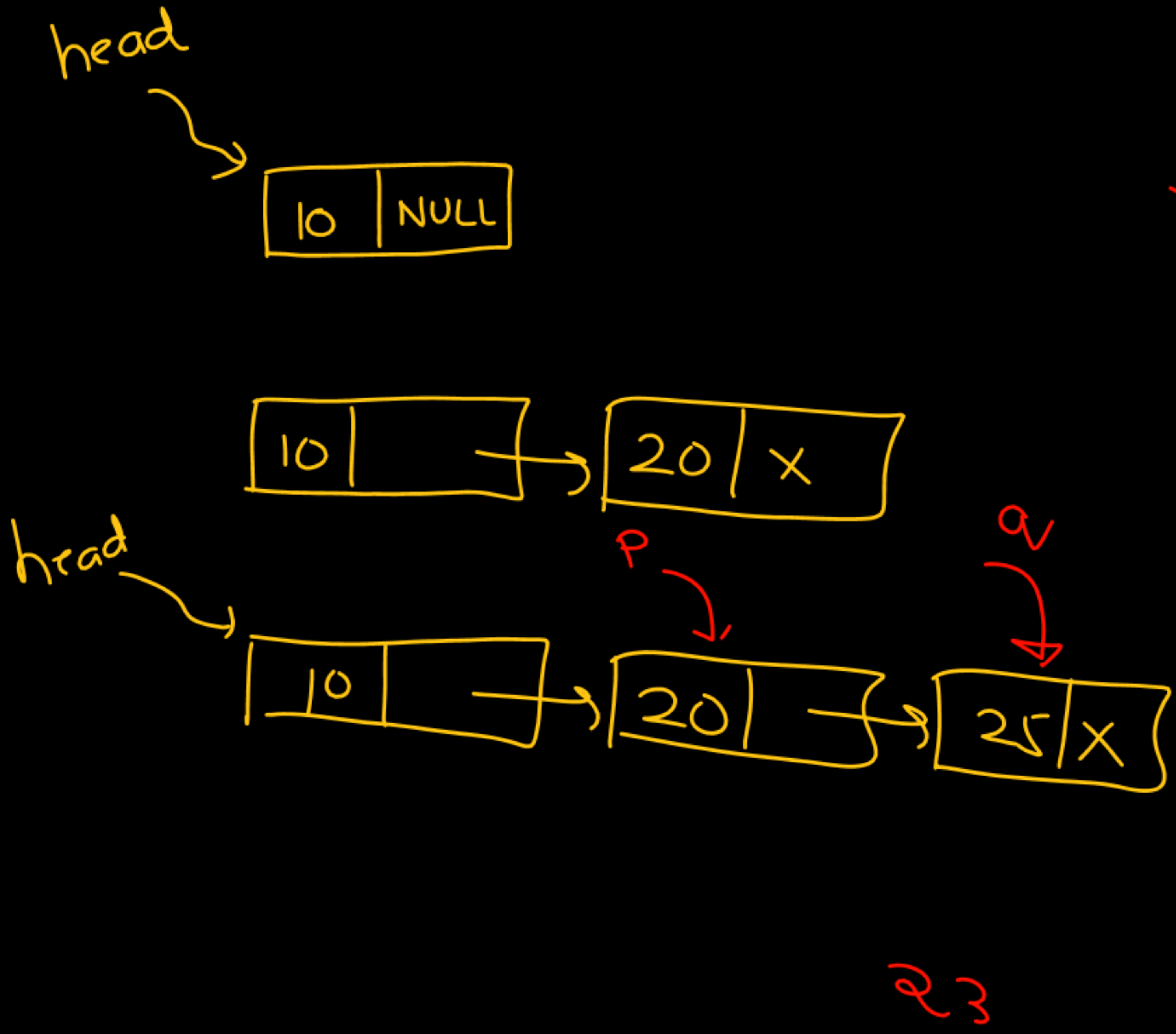
D $\Theta(n \log n)$

$$\frac{n(n-1)}{\cancel{2}}$$

$$\boxed{n^2 - n}$$

$\Rightarrow$

10, 20, 25, 23, 6, 8



worst case

Let P be a singly linked list and q be the pointer to an intermediate node x in the list. What is the time complexity of the best known algorithm to delete the node x from the list?

A $\Theta(\log^2 n)$

B $\Theta(1)$

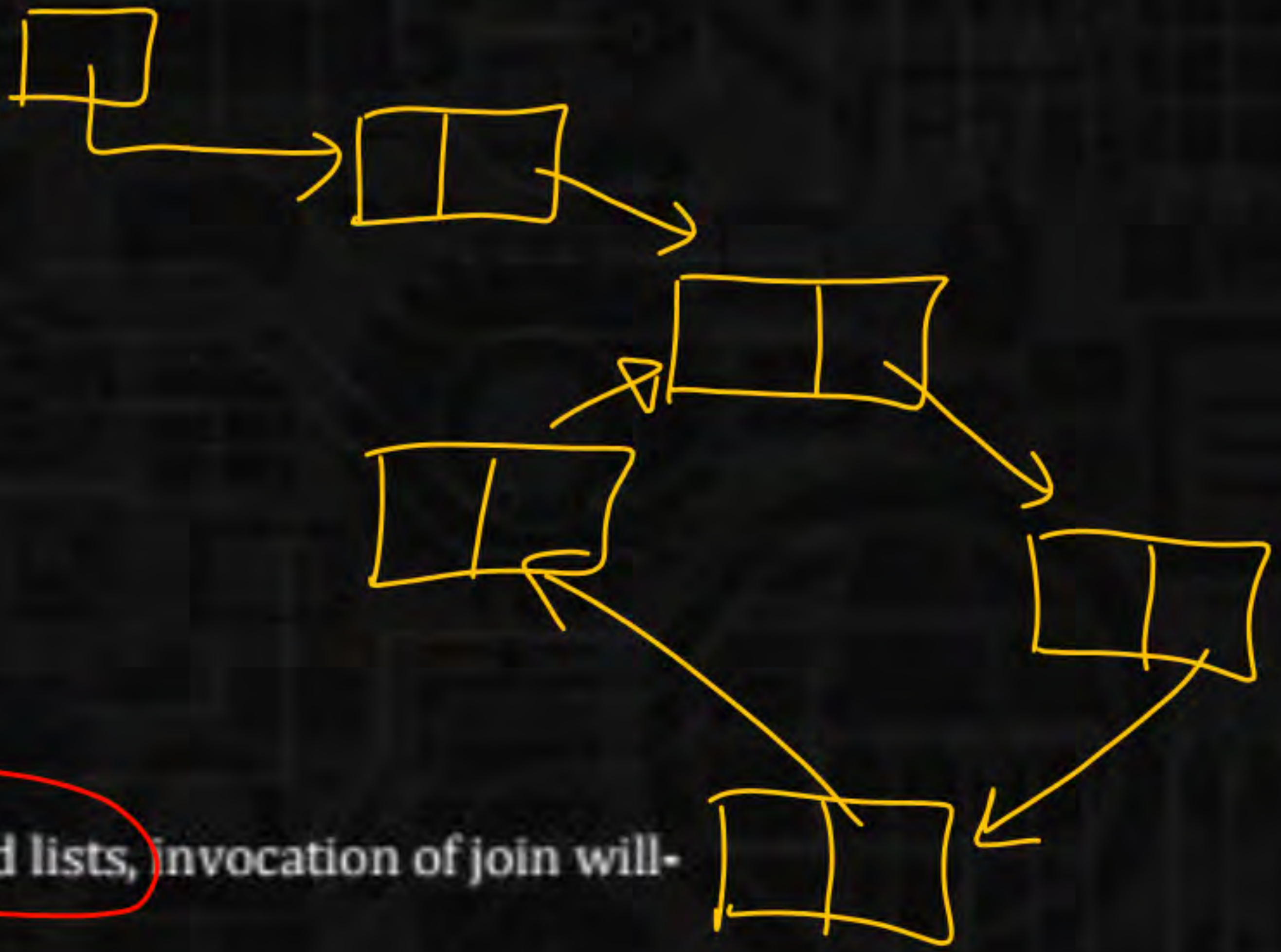
C $\Theta(n)$

D $\Theta(\log n)$

Consider the C code fragment given below.

```
typedef struct node
{
                int data;
                node * next;
} node;
void join (node * m, node * n)
{
           node * p = n;
           while (p → next! = NULL)
           {
                p = p → next;
            }
            p → next = m;
}
```

Assuming that m and n point to valid NULL-terminated linked lists, invocation of join will-



A append list m to the end of list n for all inputs.

B either cause a null pointer dereference or append list m to the end of list n.

C cause a null pointer dereference for all inputs.

D append list n to the end of list m for all inputs.

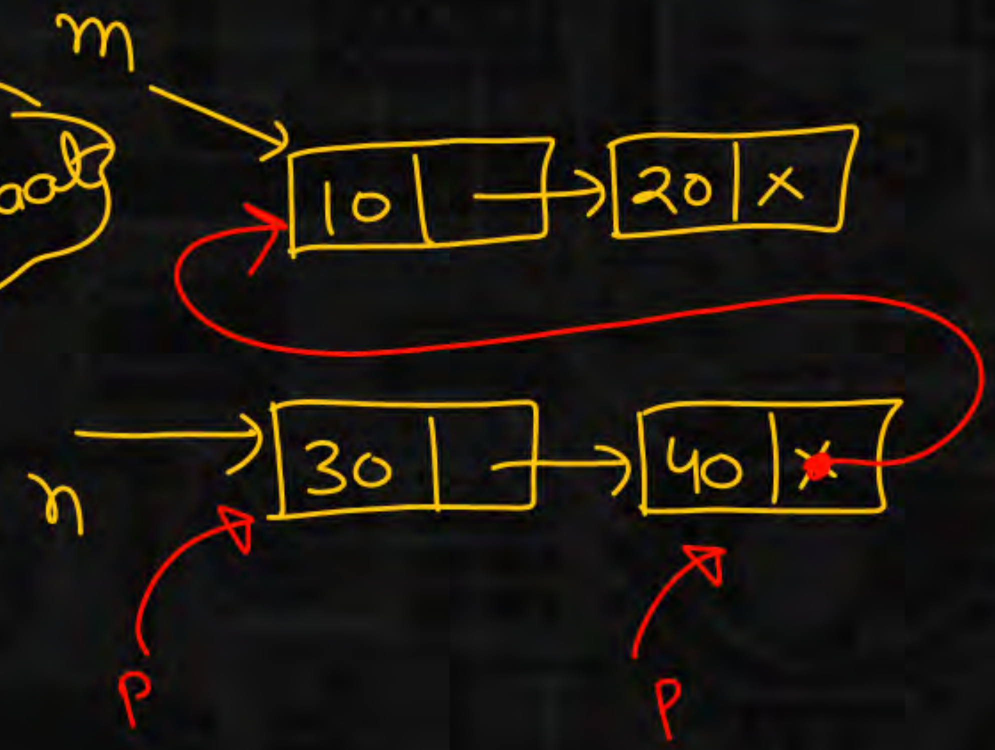
Consider the C code fragment given below.

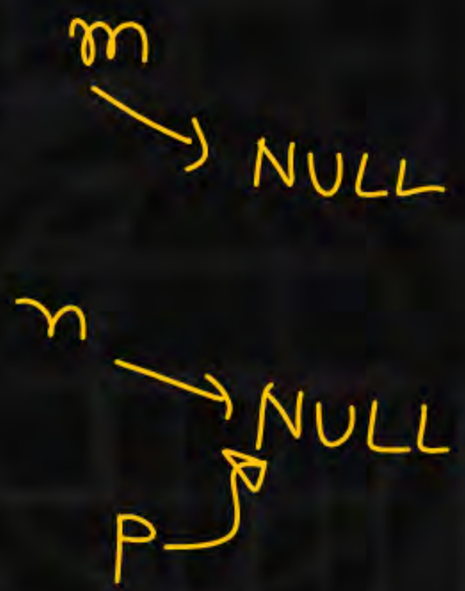
```
typedef struct node
{
            int data;
            node * next;
} node;
void join (node * m, node * n)
{
        node * p = n;
        while (p → next! = NULL)
        {
            p = p → next;
        }
        p → next = m;
}
```

Uske baad

NULL → next

n ⇒ end m





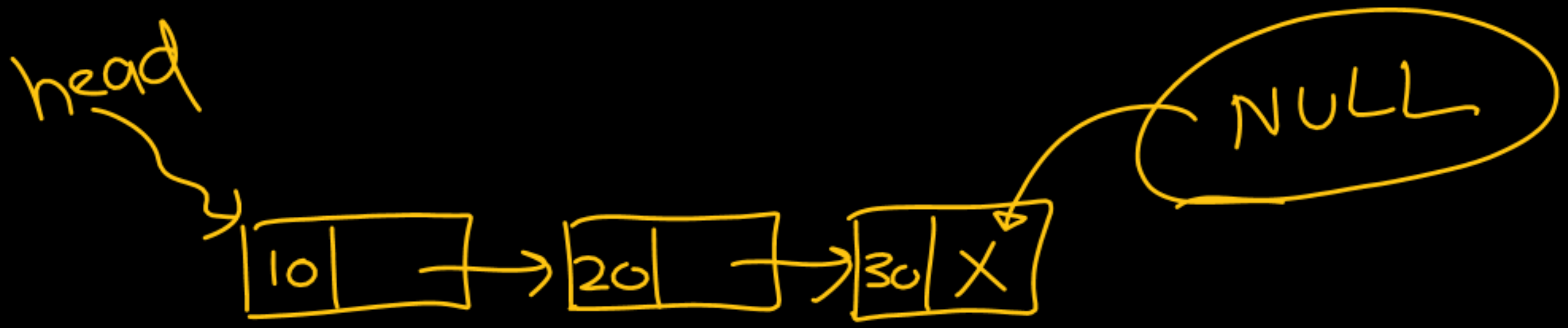
Assuming that m and n point to valid NULL-terminated linked lists, invocation of join will-

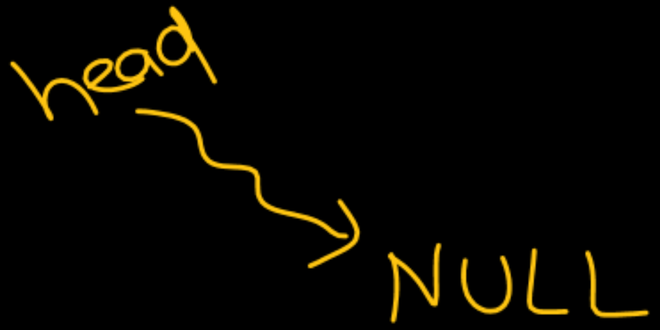
(A) append list m to the end of list n for all inputs.

[✓ (B) either cause a null pointer dereference or append list m to the end of list n.]

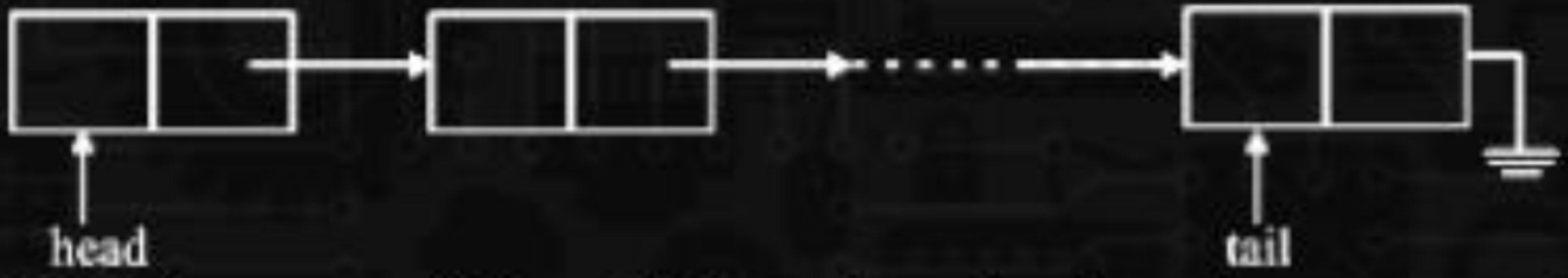
(C) cause a null pointer dereference for all inputs.

(D) append list n to the end of list m for all inputs.


head
10
20
30
X
NULL


head
NULL

A queue is implemented using a non-circular singly linked list. The queue has a head pointer and a tail pointer, as shown in the figure. Let n denote the number of nodes in the queue. Let enqueue be implemented by inserting a new node at the head, and dequeue be implemented by deletion of a node from the tail.



Which one of the following is the time complexity of the most time-efficient implementation of enqueue and dequeue, respectively, for this data structure?

A $\Theta(1), \Theta(1)$

B $\Theta(1), \Theta(n)$

C $\Theta(n), \Theta(1)$

D $\Theta(n), \Theta(n)$

The following C function takes a singly-linked list of integers as a parameter and rearranges the elements of the list. The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7, in the given order. What will be the contents of the list after the function completes execution?

```
struct node
{
        int value;
        struct node *next;
};
void rearrange (struct node *list)
{
        struct node *p, *q;
        int temp;
        if (! list ||! list → next) return;
        p = list; q = list → next;
        while (q)
        {
                temp = p → value;
                p → value = q → value;
                q → value = temp;
                p = q → next;
                q = p ? p → next: 0 ;
        }
}
```

A 1,2, 3,4, 5, 6,7

[✓ B 2, 1,4,3,6,5,7]

C 1, 3, 2, 5,4, 7, 6

D 2, 3, 4, 5, 6, 7, 1

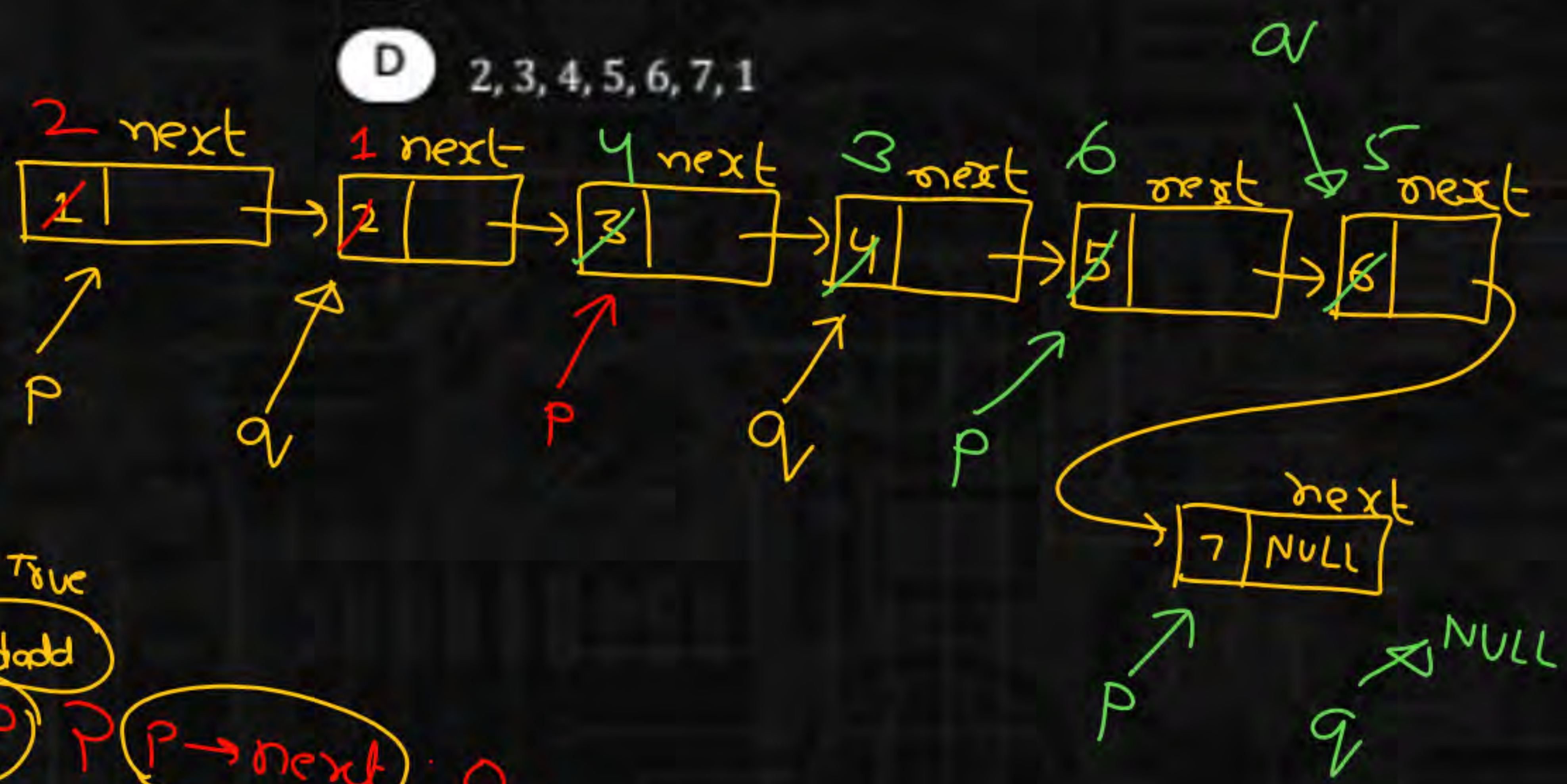
if (list = = NULL || list → next = = NULL)

NULL → false

swap

list

! list



True

validadd

q = p ? p → next : 0 ;

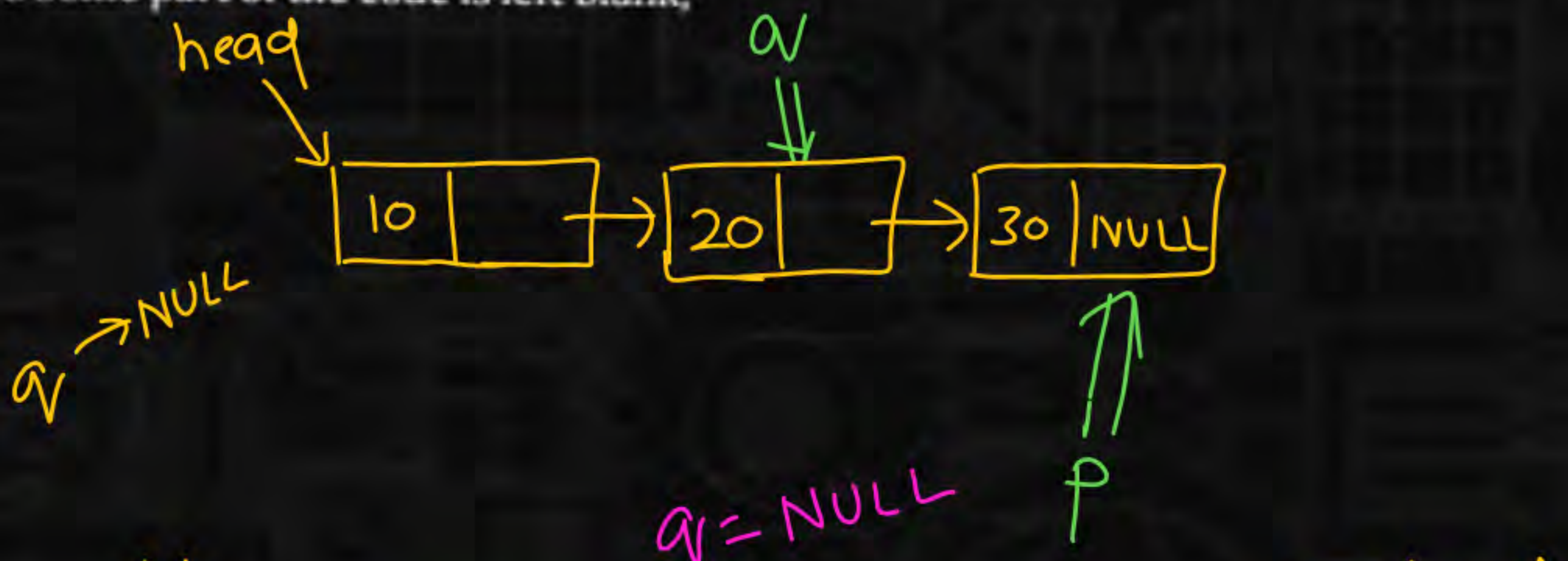
q = p → next ;

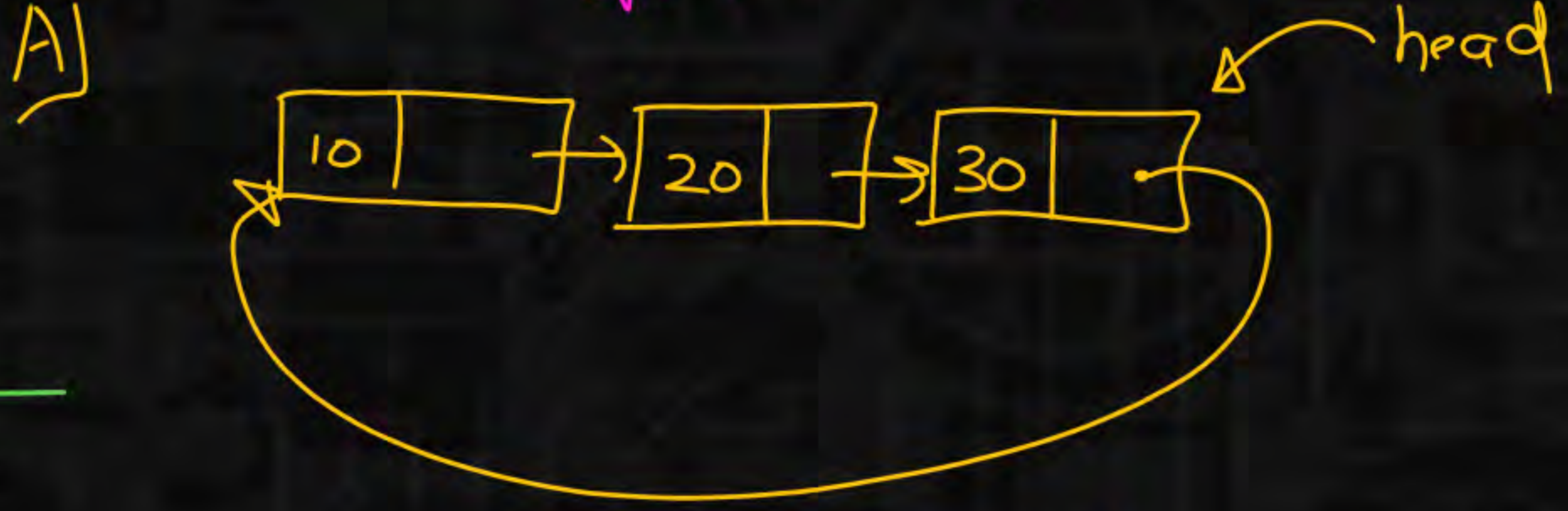
The following C function takes a singly linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank.

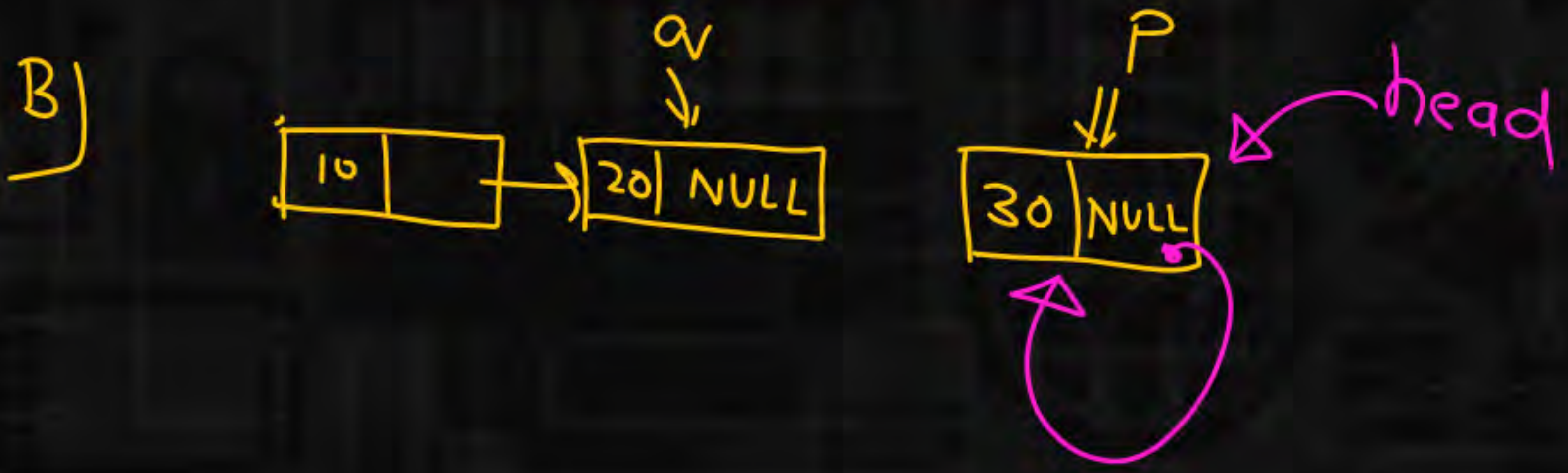
```
typedef struct node
{
          int value;
          struct node * next;
}         node;
node *move_to_front(Node *head)
{
          node *p, *q;
if((head= =NULL) ||{(head → next = = NULL))
                    return head;
q = NULL; p = head;
while (p → next! = NULL){
         q = p;
         p = p → next;}
         return head;}                ________________
```

Choose the correct alternative to replace the blank line.

- **A** q = NULL; p → next = head; head = p;
- **B** q → next = NULL; head = p ; p → next = head;
- **C** head = p ; p → next = q; q → next = NULL;
- **D** q → next = NULL;p → next = head; head = p;

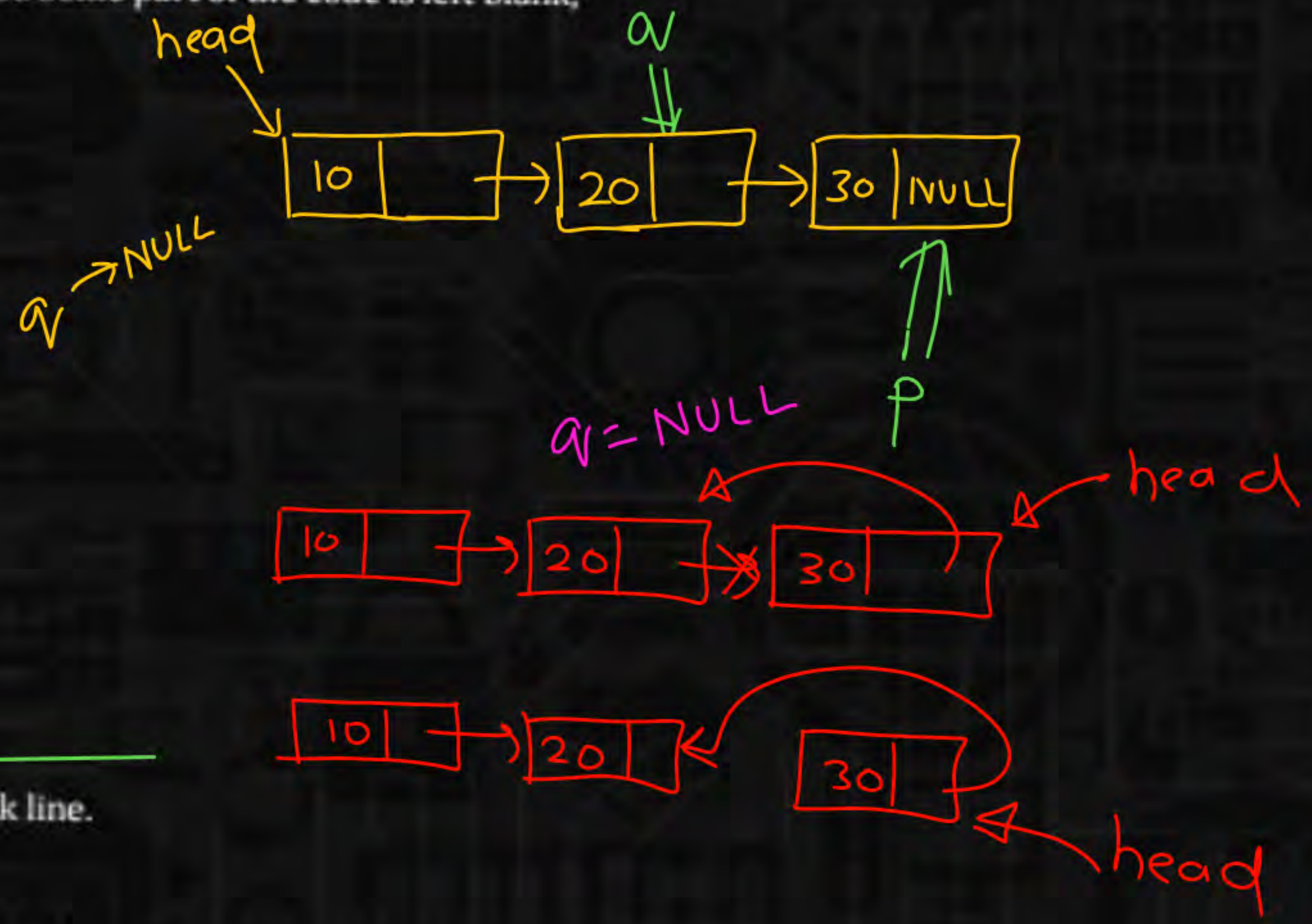






The following C function takes a singly linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank.

```
typedef struct node
{
            int value;
            struct node * next;
}           node;
node *move_to_front(Node *head)
{
            node *p, *q;
if((head= =NULL) ||{(head → next = = NULL))
                        return head;
q = NULL; p = head;
while (p → next! = NULL){
            q = p;
            p = p → next;}
            return head;}
```



Choose the correct alternative to replace the blank line.

A q = NULL; p → next = head; head = p;

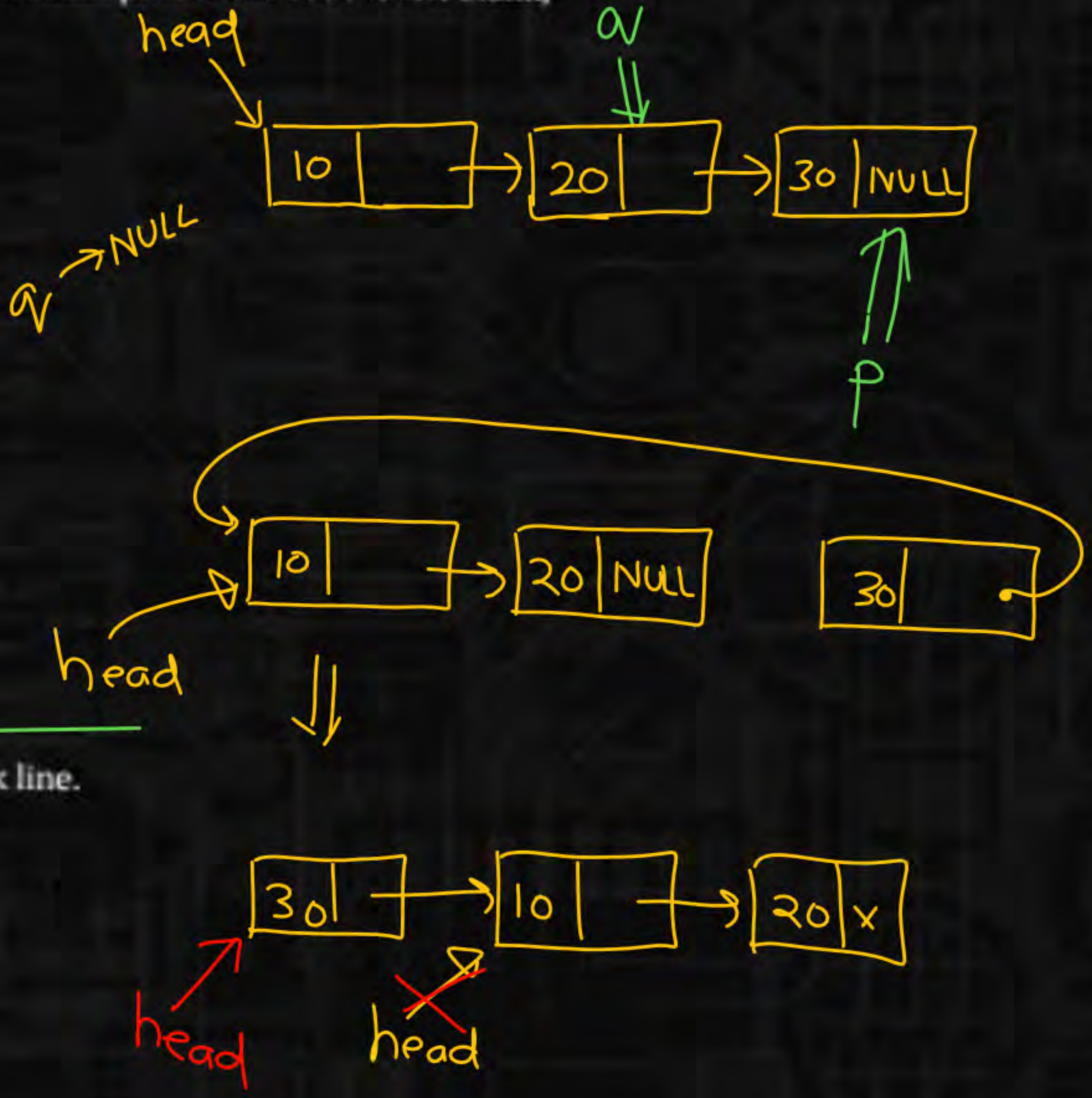
B q → next = NULL; head = p ; p → next = head;

C head = p ; p → next = q; q → next = NULL;

D q → next = NULL;p → next = head; head = p;

The following C function takes a singly linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank.

```
typedef struct node
{
            int value;
            struct node * next;
}           node;
node *move_to_front(Node *head)
{
            node *p, *q;
if((head= =NULL) ||((head → next = = NULL))
                        return head;
q = NULL; p = head;
while (p → next! = NULL){
        q = p;
        p = p → next;}
        return head;}
```



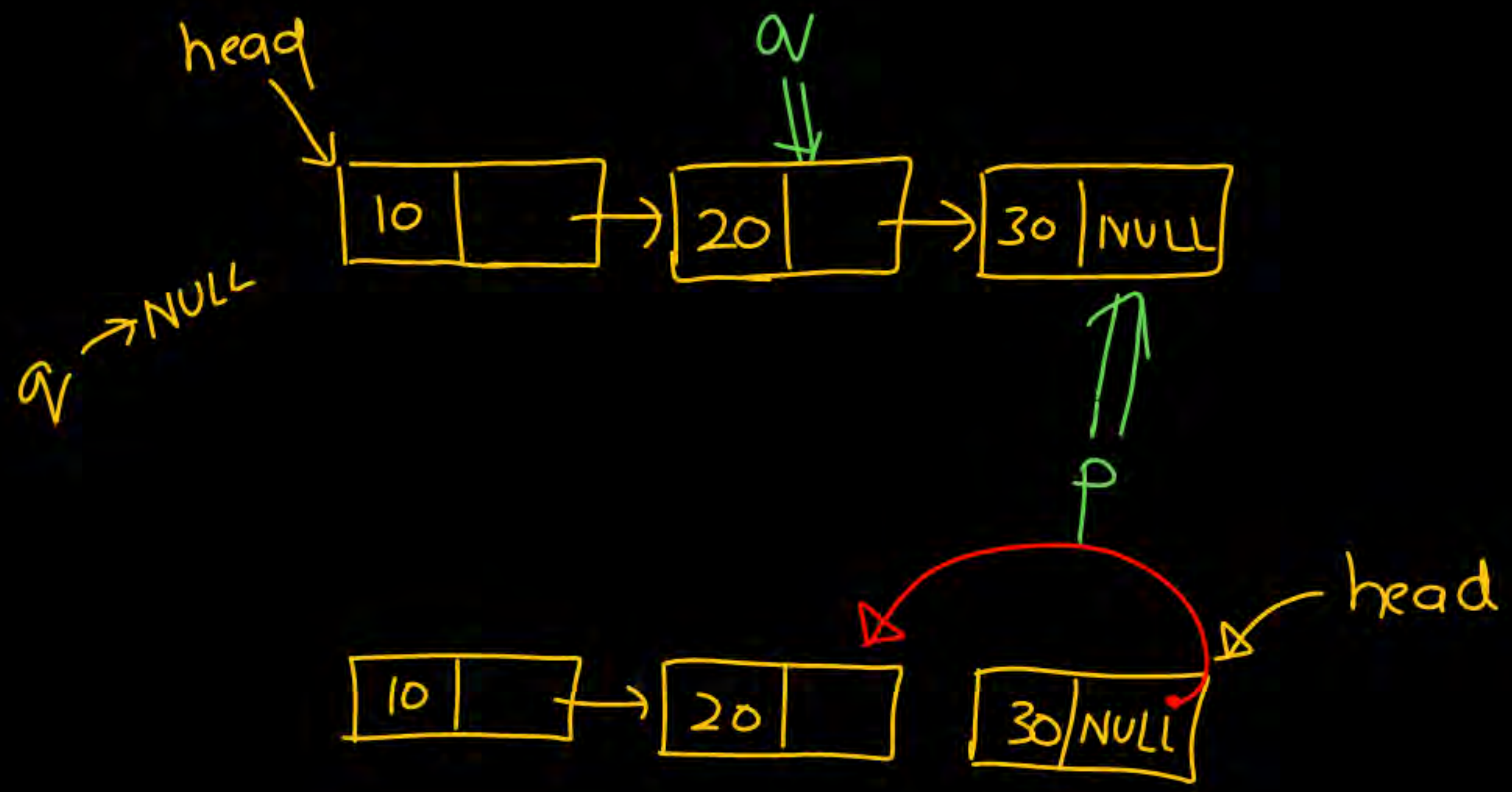
Choose the correct alternative to replace the blank line.

(A) q = NULL; p → next = head; head = p;

(B) q → next = NULL; head = p ; p → next = head;

(C) head = p ; p → next = q; q → next = NULL;

(D) q → next = NULL;p → next = head; head = p;


head
q
10
20
30
NULL
q → NULL
p
head
10
20
30
NULL

Consider the following ANSI C program:

```
# include <stdio.h>
#include<stdlib.h>
struct Node{
                int value;
                struct Node * next;};
int main ( ){
struct Node * boxE, *head, *boxN; int index = 0;
boxE = head = (struct Node *) malloc (sizoof (struct Node));
head → value = index;
for (index = 1; index < = 3. index ++) {
      boxN = (struct Node *) malloc (sizeof(struct Node)).
      boxE → next = boxN;
      boxN → value = index;
      boxE = boxN;}

for (index = 0; index < = 3; index ++) {
printf ("Value at index %d is %d\n" index, head → value);
head = head -> next;
printf("Value at index %d is %d\n", index +1, head → value); }}
```

H.W

Which one of the statements below is correct about the program?

(A) It dereferences an uninitialized pointer that may result in a run-time error

(B) It has a missing return which will be reported as an error by the compil

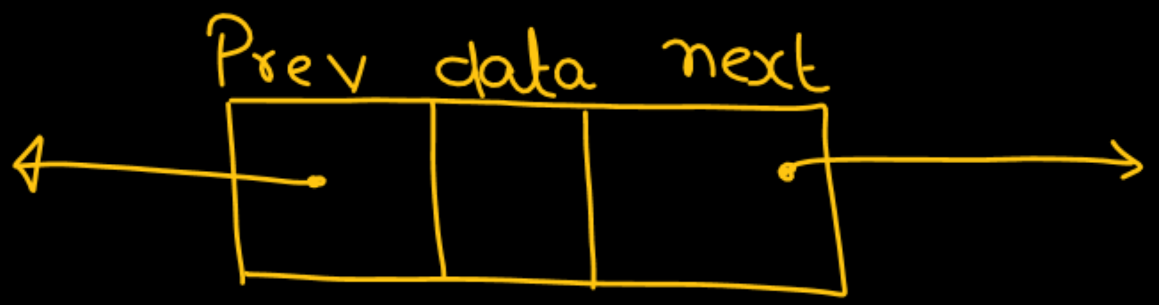
(C) Upon execution, the program creates a linked-list of five nodes.

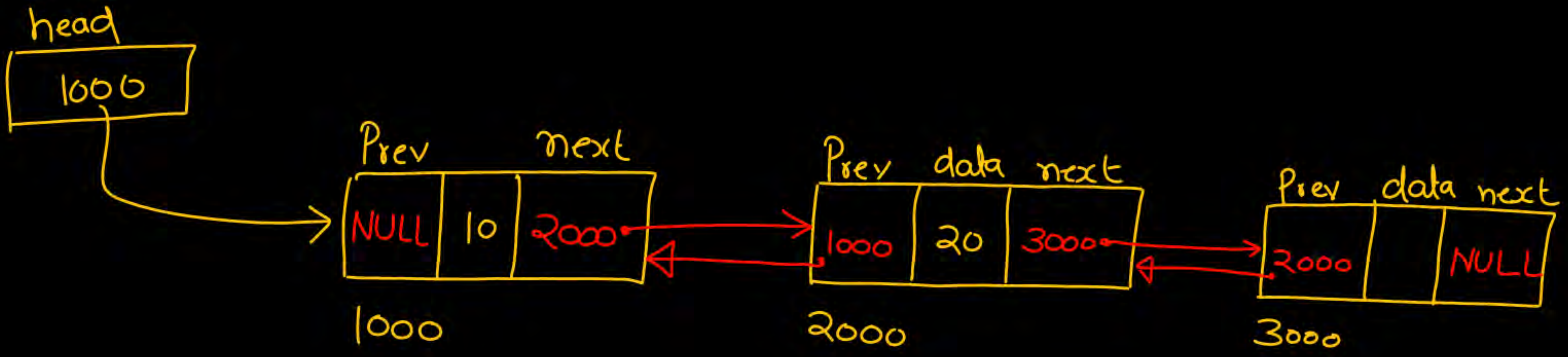
(D) Upon execution, the program goes into an infinite loop.

Types of Linked List

① Singly Linked list

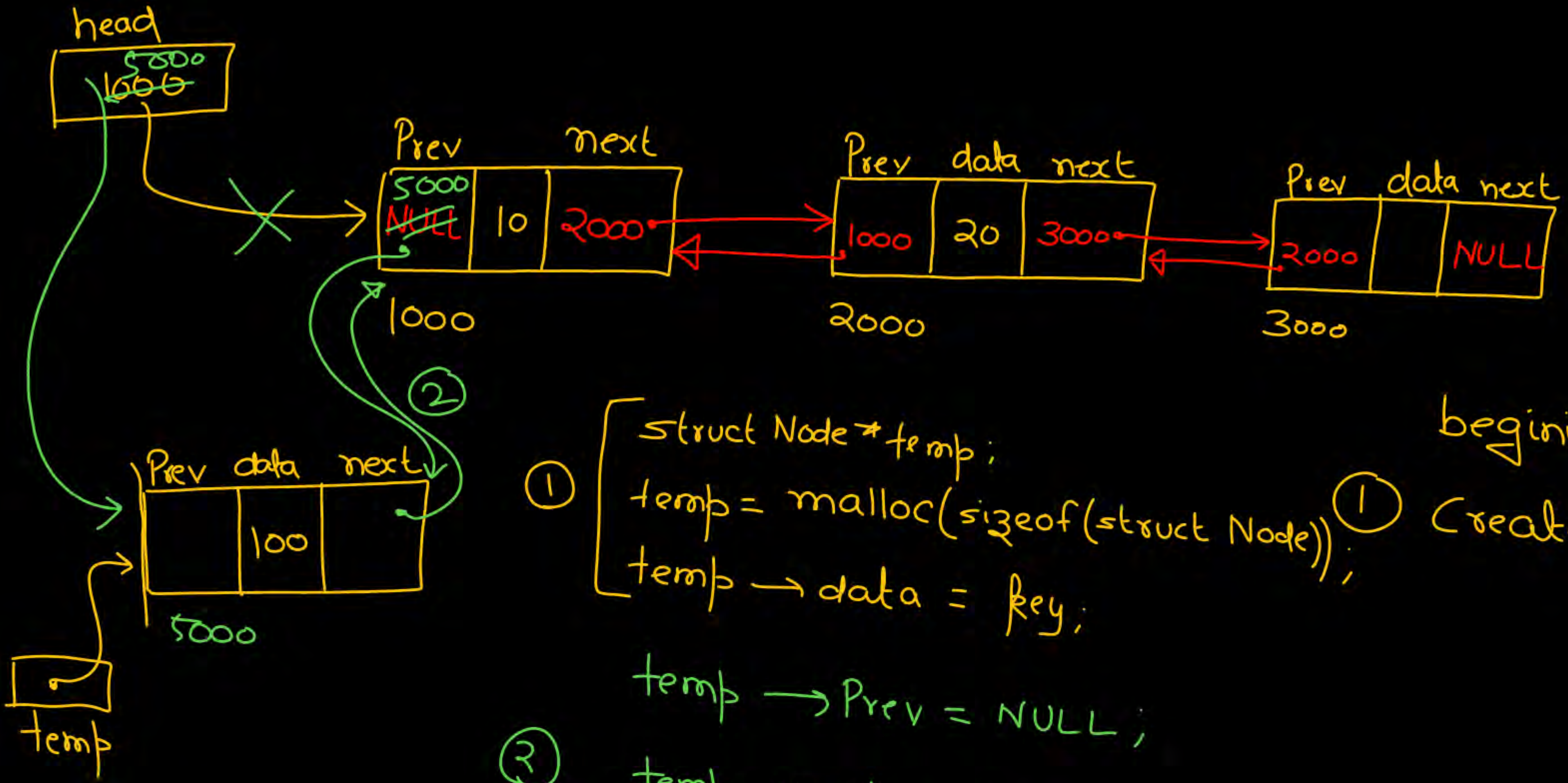
② Doubly linked :





```
struct Node {
    struct Node *Prev;
        int   data;
    struct Node * next
              };
```

template



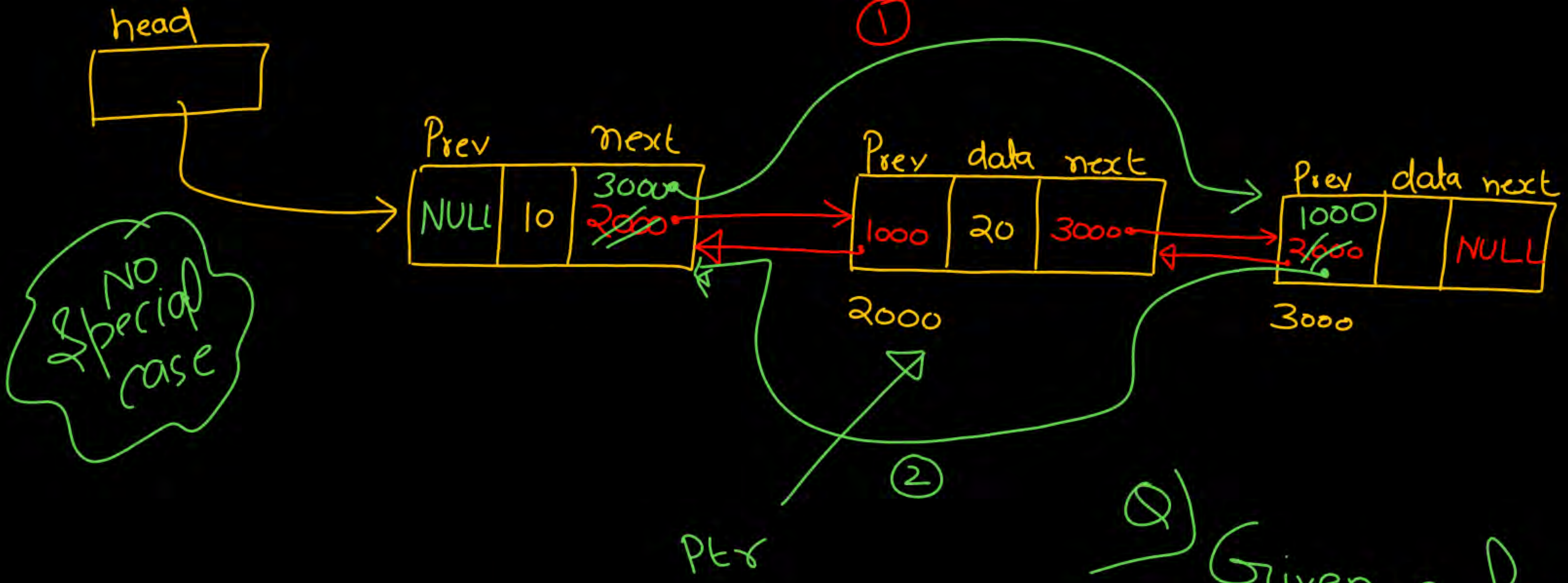
begining

① Create a node

```
① struct Node *temp;
  temp = malloc(sizeof(struct Node));
  temp → data = key;

  temp → Prev = NULL;

② temp → next = head;
③ head → Prev = temp;
④ head = temp;
```



Q) Given a pointer to a node, we need to delete that node.

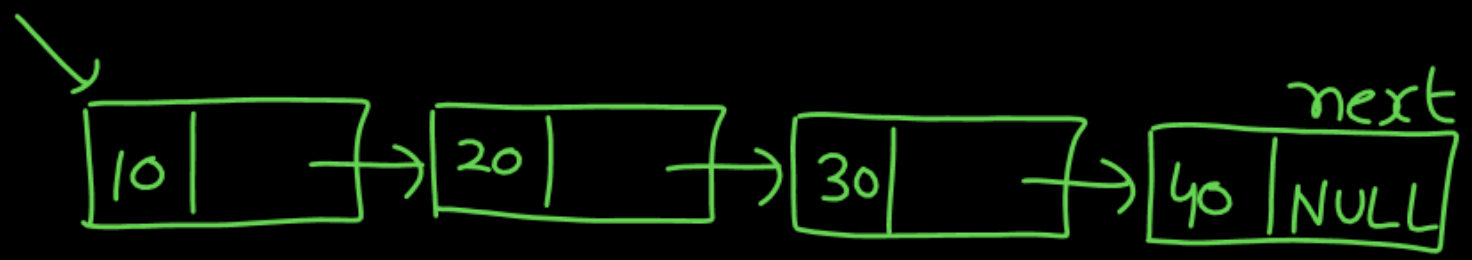
① Ptr→Pre→next = Ptr→next

② Ptr→next→Prev = Ptr→Prev;

③ free(Ptr)

3. Circular Linked List

S.L.L

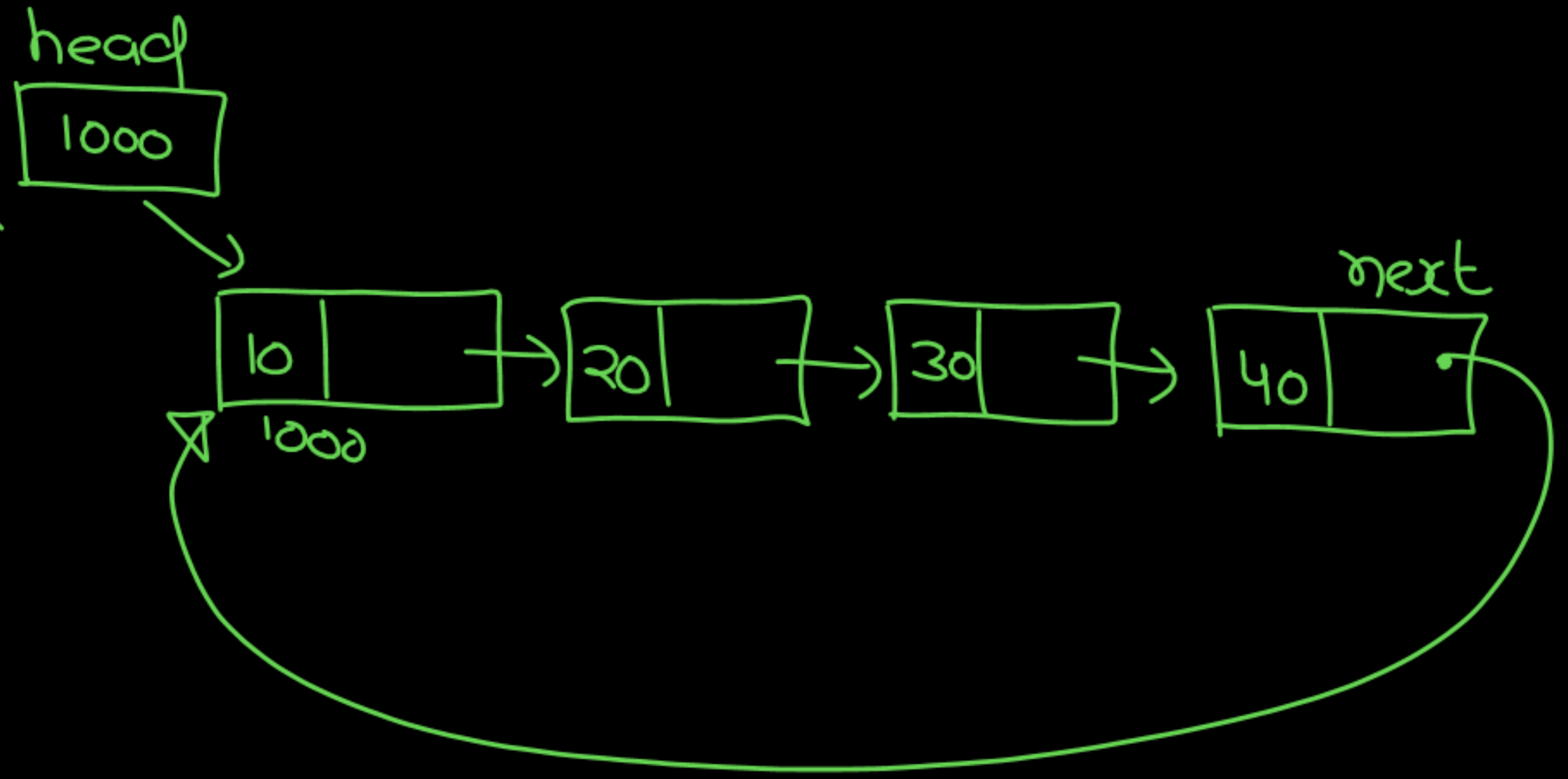


① →

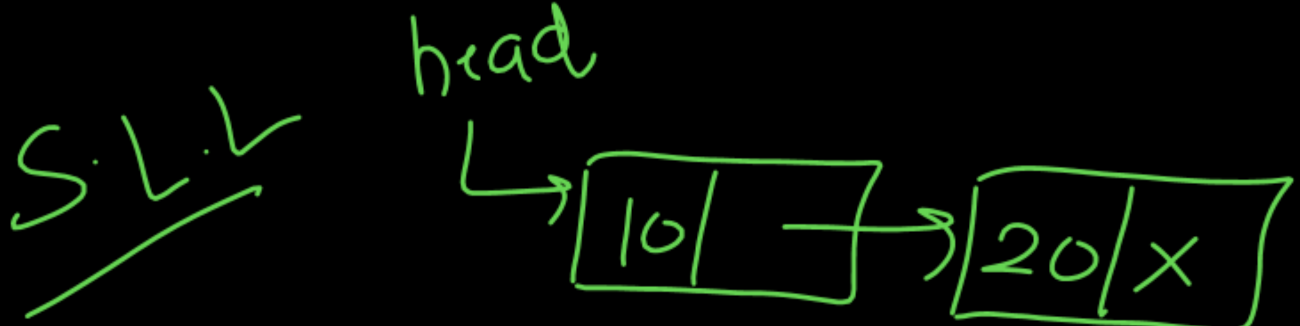
② →

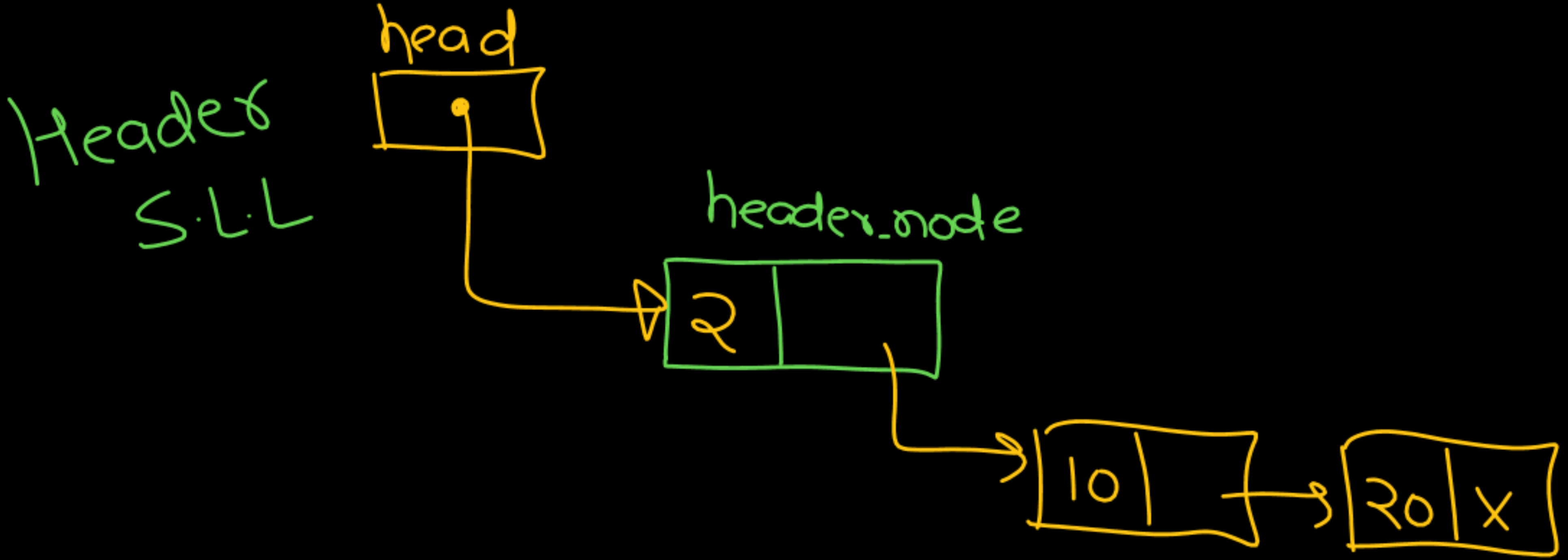
③

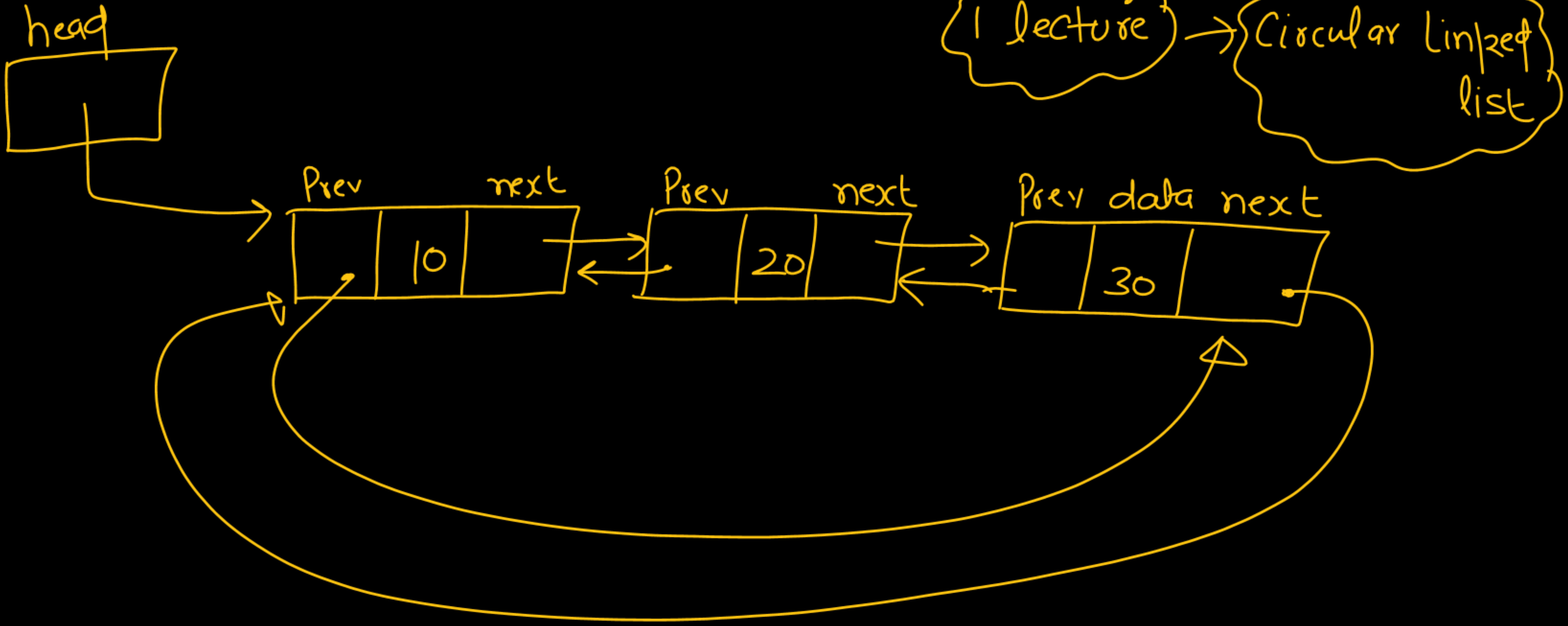
Circular linked list



4 Header Linked list






head
Prev
next
10
Prev
next
20
Prev data next
30
1 lecture
Circular linked list

THANK YOU GW SOLDIERS !