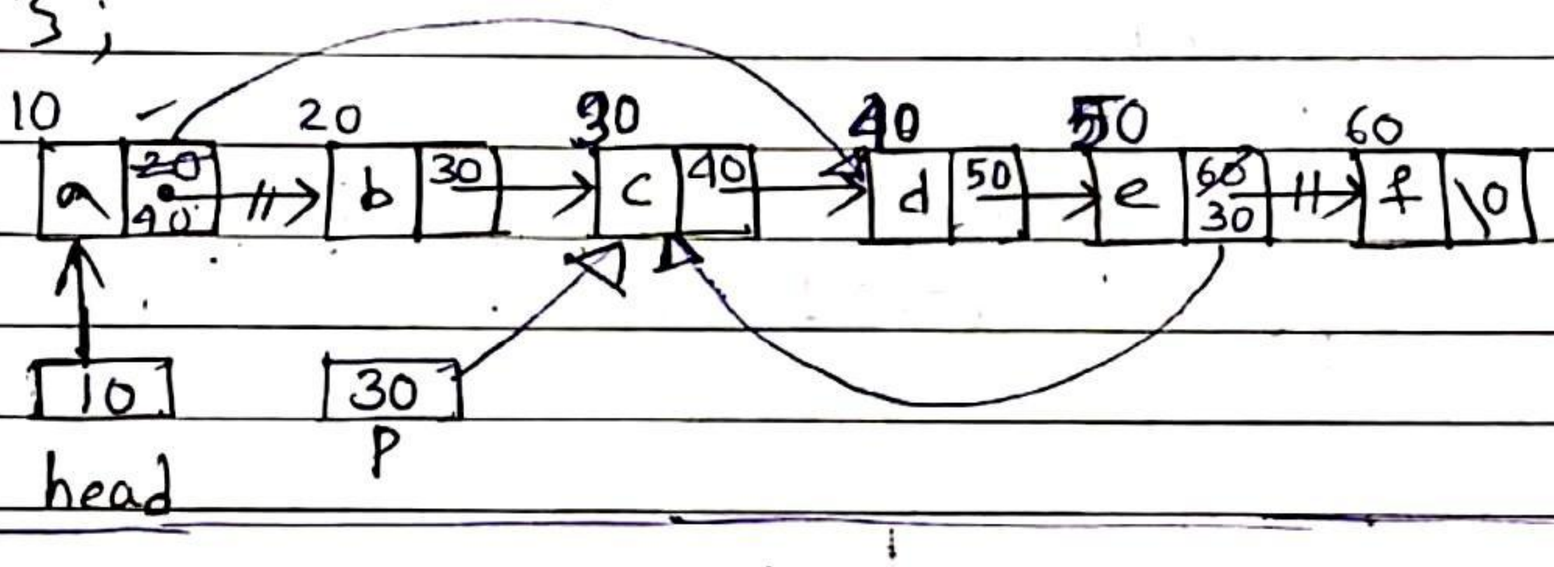
Linked-list

- Introduction of single linked list :

→ single linked list contain nodes, and this node generaly structured ~~st~~ and which are self ~~referrancial~~ referrencial.

```
struct node
{
  char data;
  struct node *link;
};
```



→ Every node has only one link (pointer) so, that this is called single linklist.

struct node (*head);

→ linklist are sequential access.

→ going to any particular number structer (link list) take O(n) time.

```
Struct node *p;
p = head → link → link ;
p → link → link → link = p;
head → link = p → link ;
pf ("%c", head → link → link → link → link → data);
```

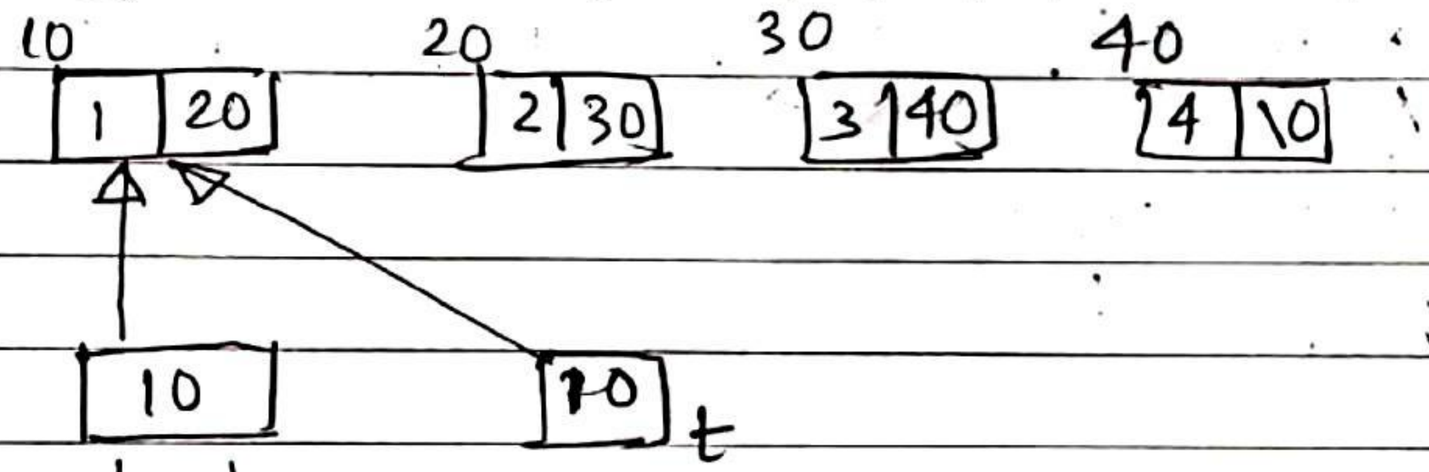
Output : "d"

~~Traversing of~~

In link list generally perform three operations —

- Traversing. (traverse the Entire list).
- inserting. (creat new node and insert it).
- delete. (detete a node).

```
struct node
{
  int i;
  struct node *link;
};
```

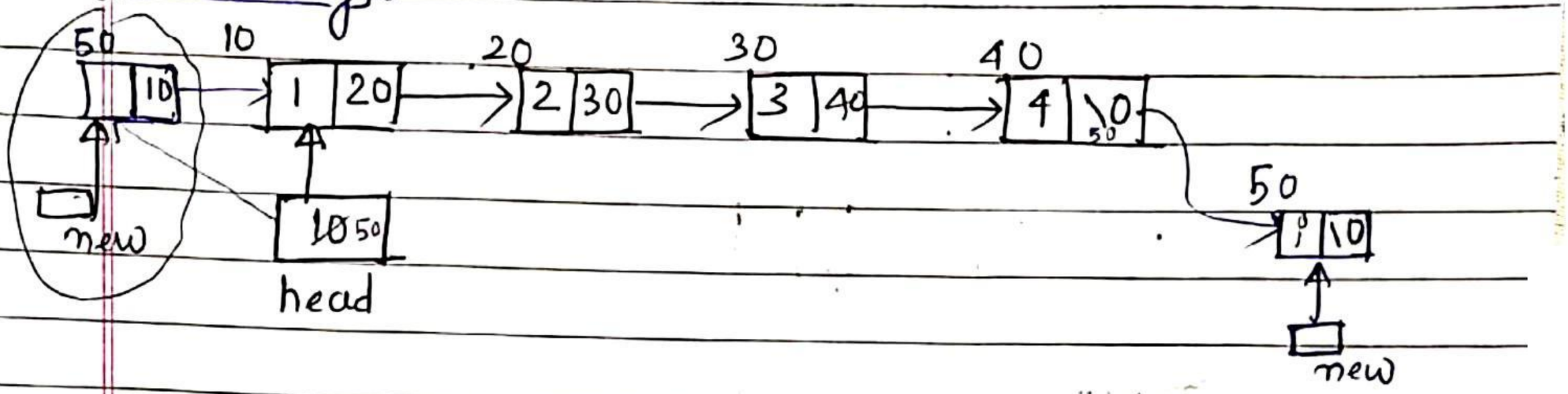


① Traversing:

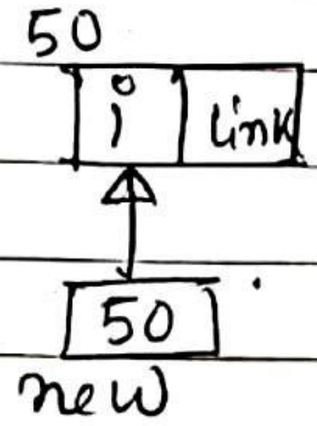
```
struct node *t;
  t = head;

while (t != NULL)   ≡ while (t)
{printf ("%d", t → i)
  t = t → link. }
```

output: 1 2 3 4

② Inserting :



struct node *new = (struct node *) malloc (sizeof (struct node))



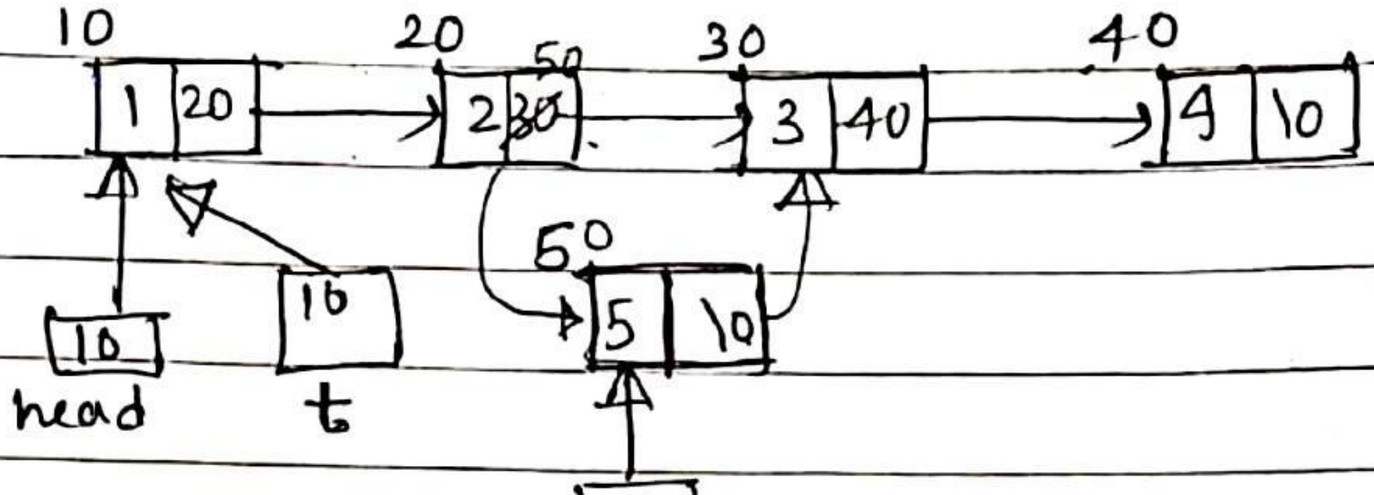
Case-1 Insert at the ~~beg~~ beginning =

```
new → link = head
head = new
```

Case-2 insert at the end =

```
struct node *t = head;

While (t → link != NULL)  ≡  While (t → link)
{ t = t → link; }

t → link = new;
New → link = NULL;
```

Case-3 Insert at the middle.



Struct node *t; = head;

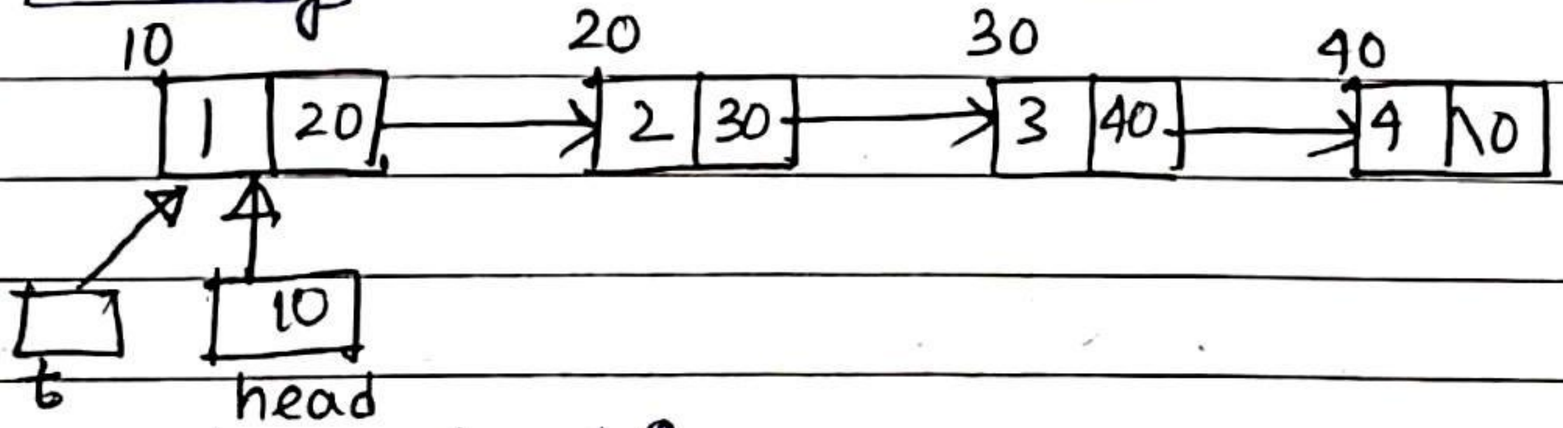
while (t → info != 2)

{ t = t → next; (link) }

t → link = new → link

t → link = new;

③ **Deleting** :-



- delete from head :-

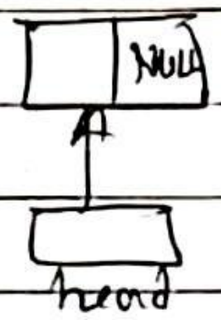
struct node *t = head;

head = head → link; (to delete first one)

free(t);

Recursive Condition Checking =

```
if (head == NULL)
    Return
if (head → next == NULL)
    free (head)
```



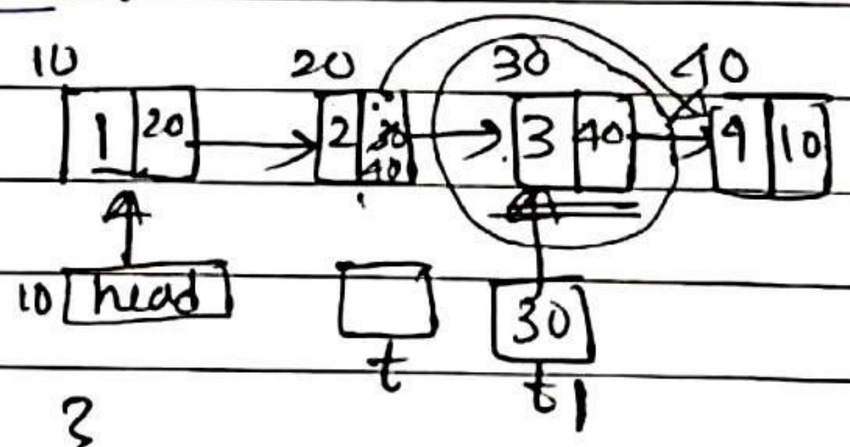
before deletion check this one. (head are null or not)
(head → link is null or not)

- delete from tail :

```
struct node *t = head;
While (t → next → next != NULL)
While (t → link → link != NULL)
    { t = t → next/link list; }

free (t → next/link list)
t → list == null
```

- delete node which contain i = 3 :

```
Struct node *t = head;

While (t → list → i != 3)
  { t = t → list; }

struct node *t1 = t → next list;
t → next = t → next → next;
t → list = t → list → list;
Free (t1);
```



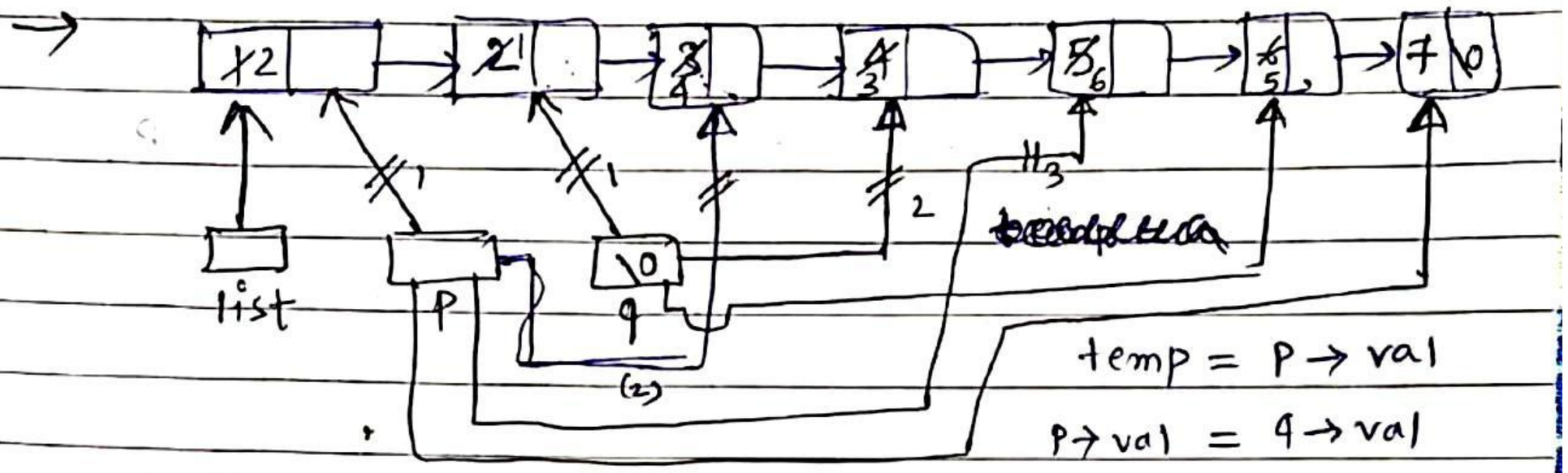
Question - 1

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

void rearrange (struct node *list)
  {
    struct node *p, *q;
    int temp;
                (or)
    if (!list || !(list -> next)) return;
    p = list; q = list -> next;

    while (q)   ≡ (while (q != NULL))
    {
      temp = p -> val; p -> val = q -> val;
      q -> val = temp; p = q -> next;
      q = p ? p -> next : 0;
    }
  }
```

list == NULL
≡ !list

1 2 3 4 5 6 7

(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.

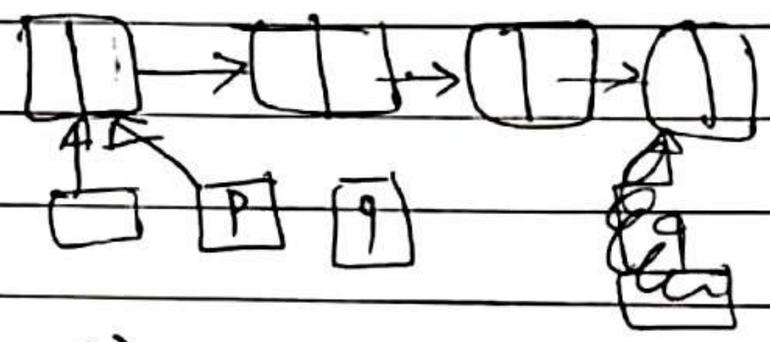


2 1 4 3 6 5 7

Gate-2010

Question-2

```
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 != NVLL)
    {
        q = p;
        p = p → next;
    }
    ____________ (blank)
    return head;
}
```

Choose the correct alternative to replace the blank

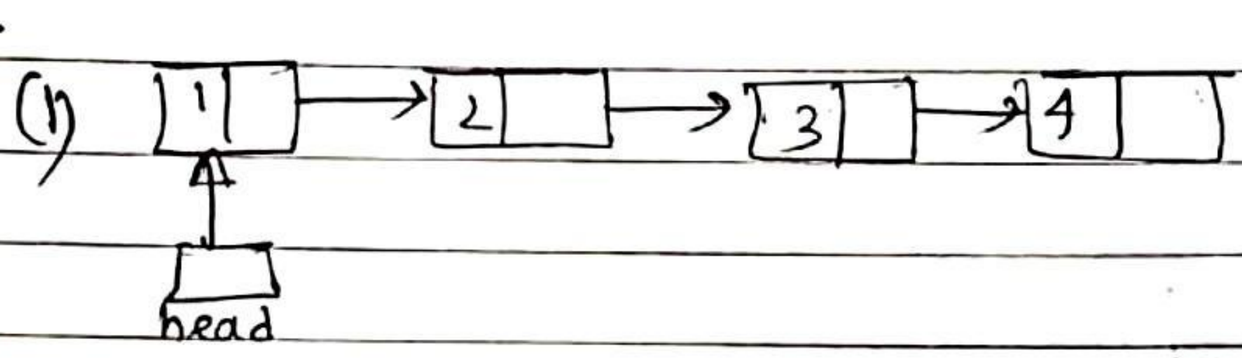
a) q = NULL ; p → next = head ; head = p ;

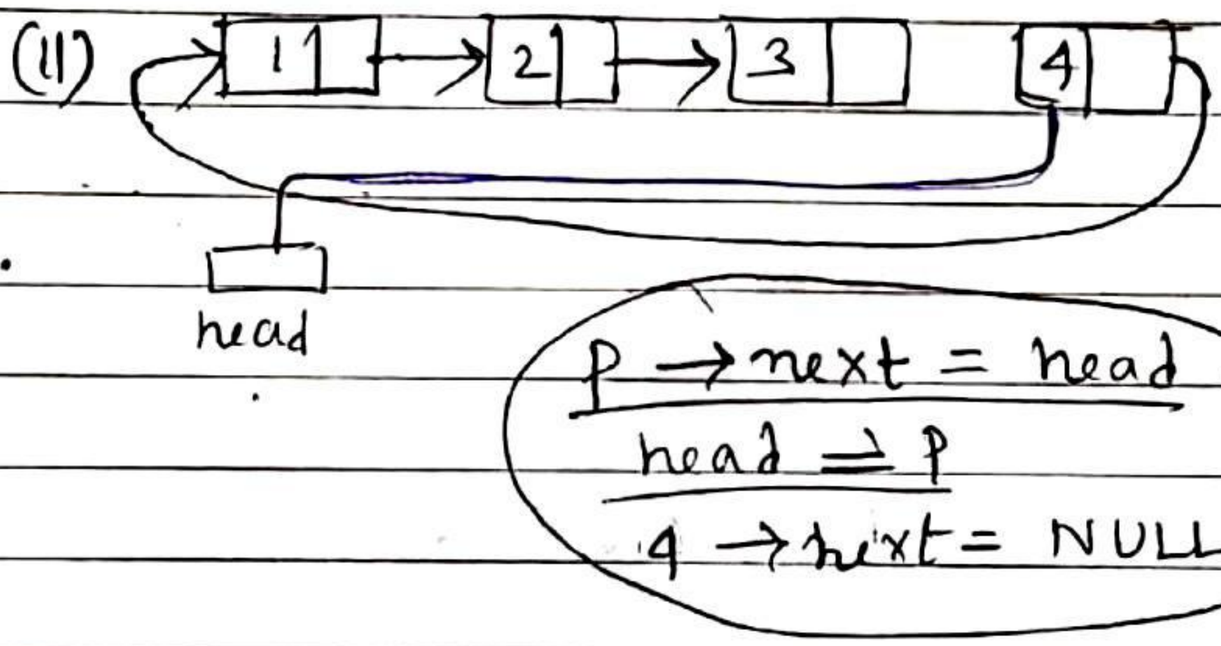
b) q → next = NULL ; head = p ; p → next = head ;

c) head = p ; p → next = q ; q → next = = NULL ;

d) q → next = NULL ; p → next = head ; head = p ;

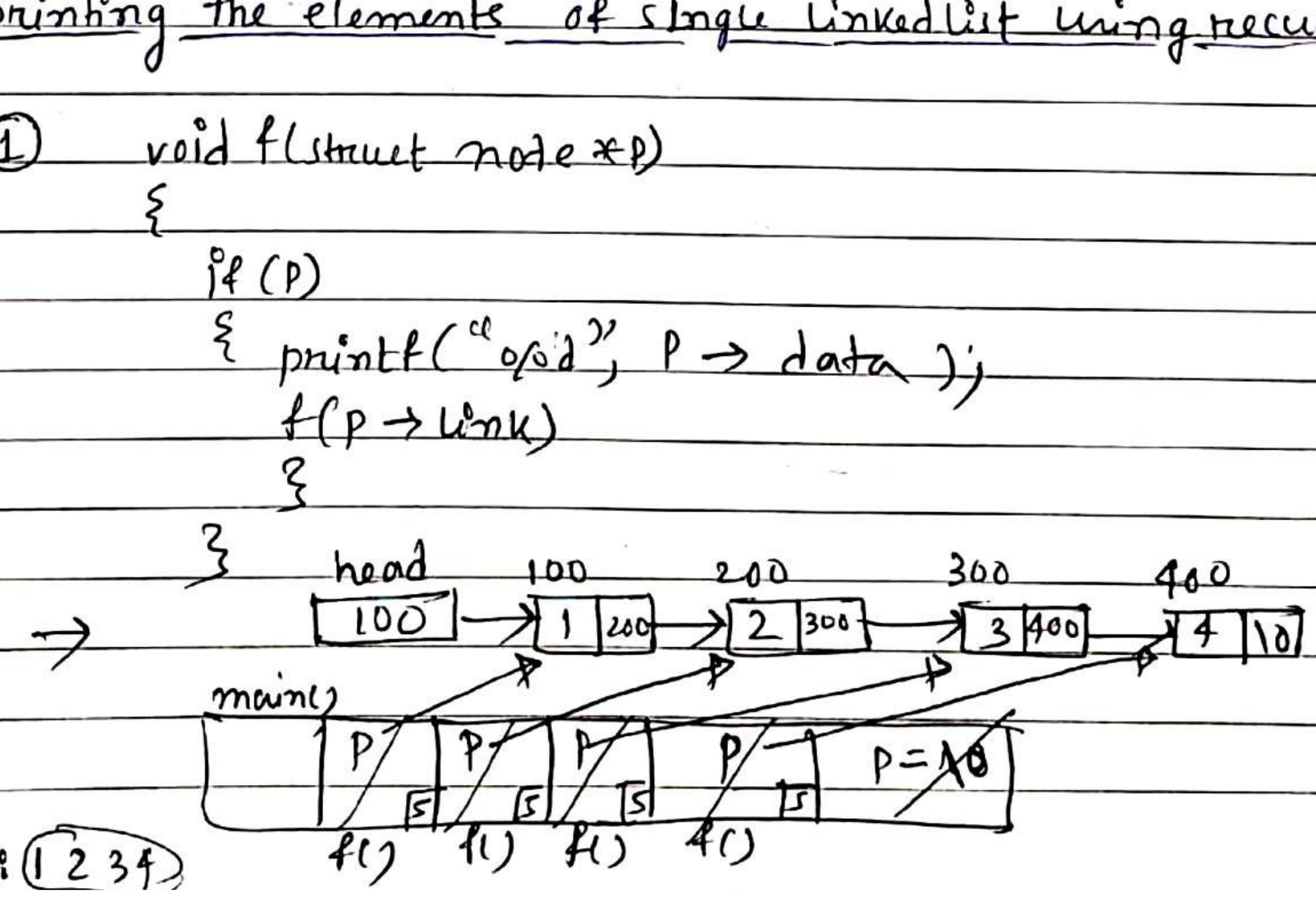
→





- printing the elements of single linked list using recursion.

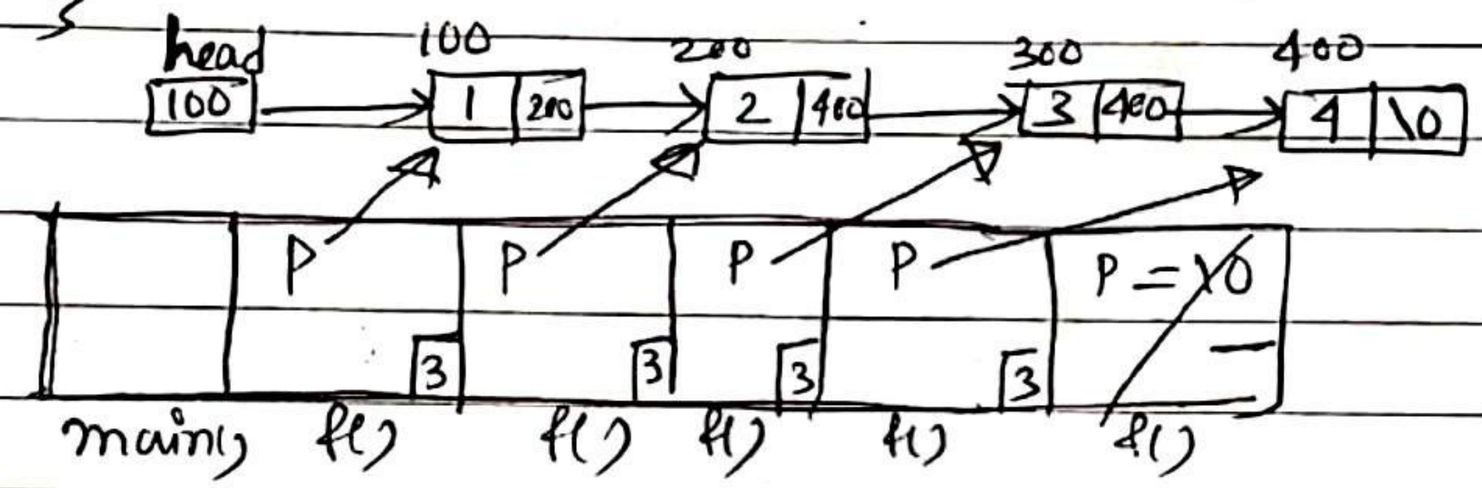
```
(1)   void f(struct node *p)
      {
      if (p)
      { printf("%d", p → data);
        f(p → link)
        {
      }
```



output: (1 2 3 4)

(2)

```
void f(struct node *P)
1  {
   if (P)
   {
2    f(P→link);
3    printf("%od", P→data);
   }
  }
```

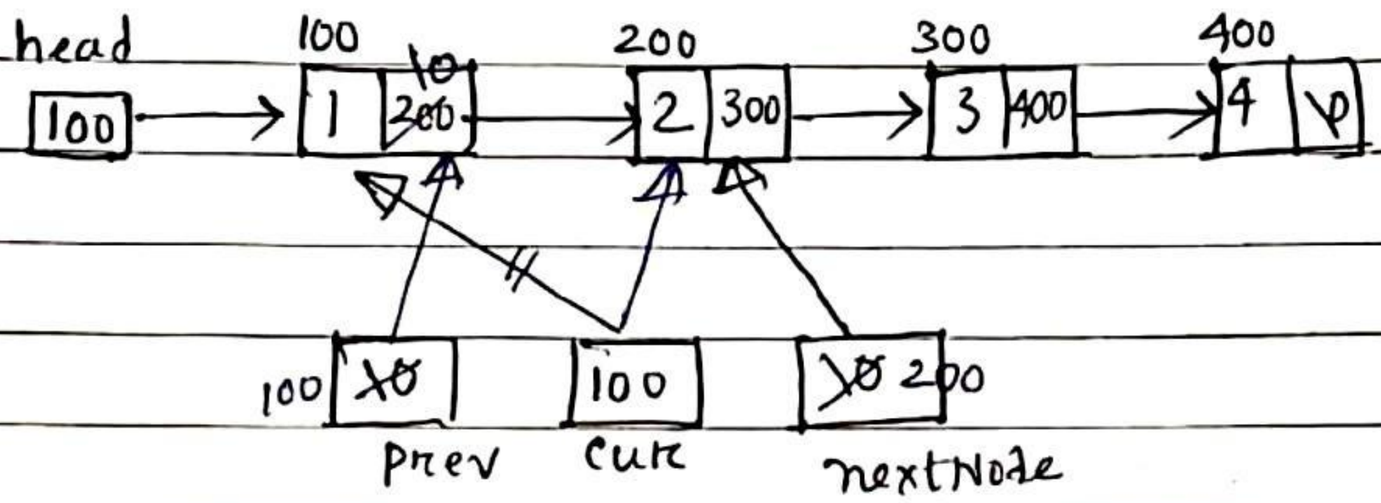


o/p: 4 3 2 1

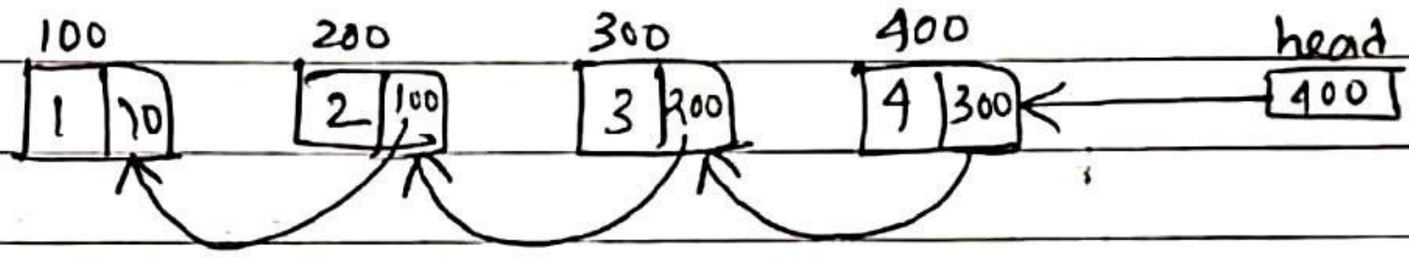
- Reversing an single linkedlist using iteration program

①
```
Struct node
{
  int i;
  struct node *next;
};

struct node *reverse (struct node *cur)
{
  struct node *prev = NULL, *nextnode = NULL;

while (cur) {
  nextNode = cur→Next;
  Cur→next = prev;
  prev → cur;
  cur → nextNode;
}
  return prev; }
```

```
main(){
head = reverse(head);
}
```

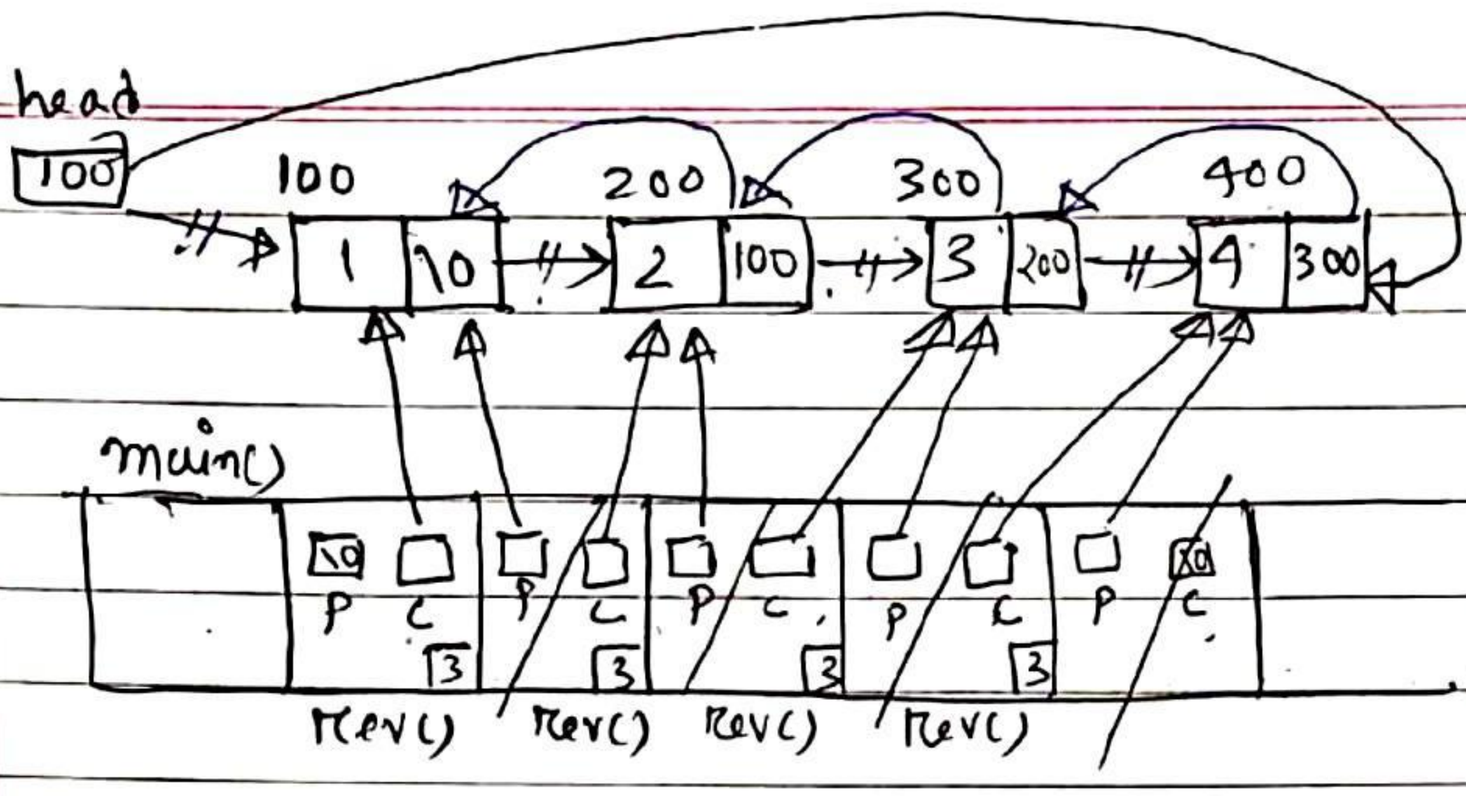


after operation



- Recursive program for reversing an single linked list =

```
struct node *head;

void reverse(struct node *prev, struct node *cur)
{
  ① if (cur)
    {
    ② reverse(cur, cur->link);
    ③ cur->link = prev;
    }
  else
    head = prev;
}

void main()
{
  :
  reverse(NULL, head);
  :
}
```



GATE-2003

Question-3

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

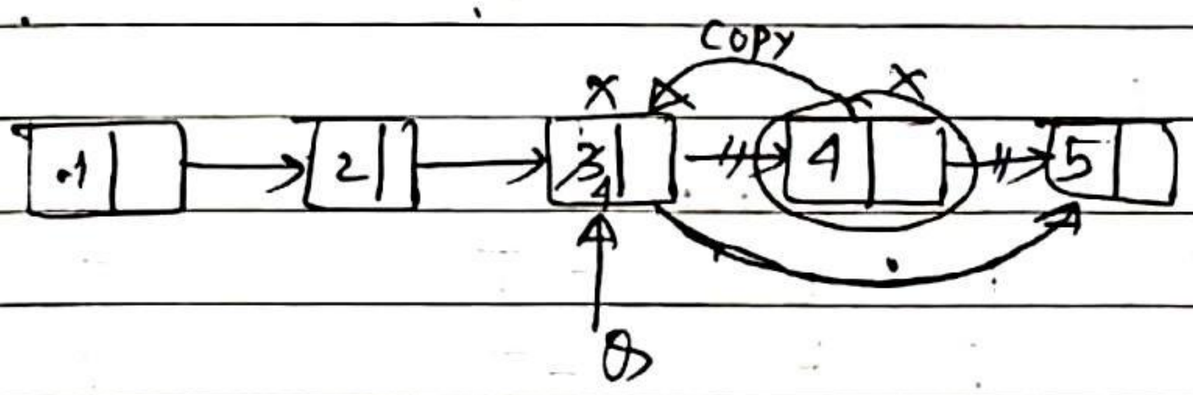
(a) $\log_2 n$ (b) $n/2$ (c) $\log_2 n - 1$ ✓(d) n.

GATE-2004

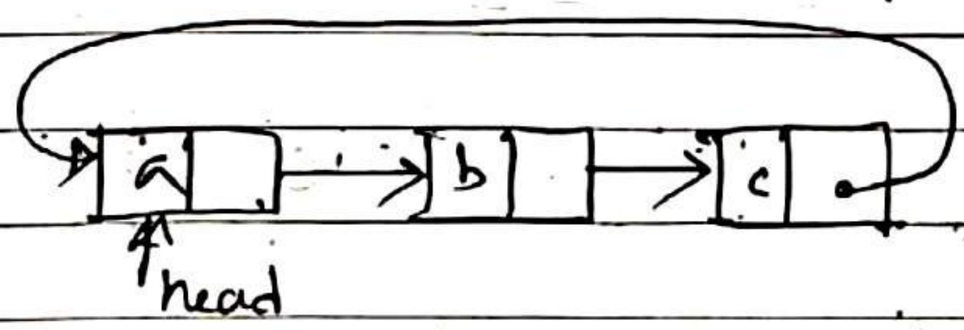
Question-4

Let 'p' be a single linked list, let 'q' be a pointer to an intermediate node 'x' in the list. What is the worst case time complexity of the best known algorithm to delete the node 'x' from the list?

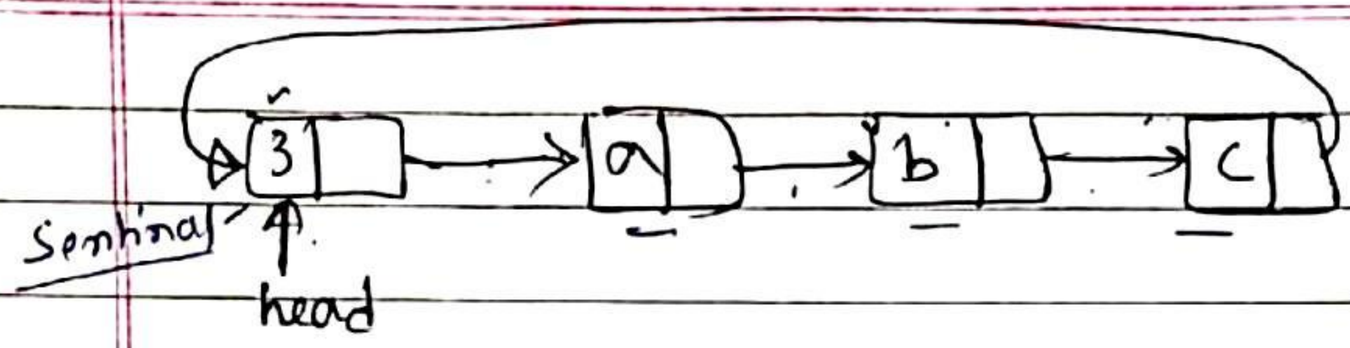
→ O(1).



- Circular linked list =




```
p = head;
while (p->next != head)   → to find the end of the list.
{ : }
```



adv:→ given any point we can search entire list.
→ last pointer are not wasted.

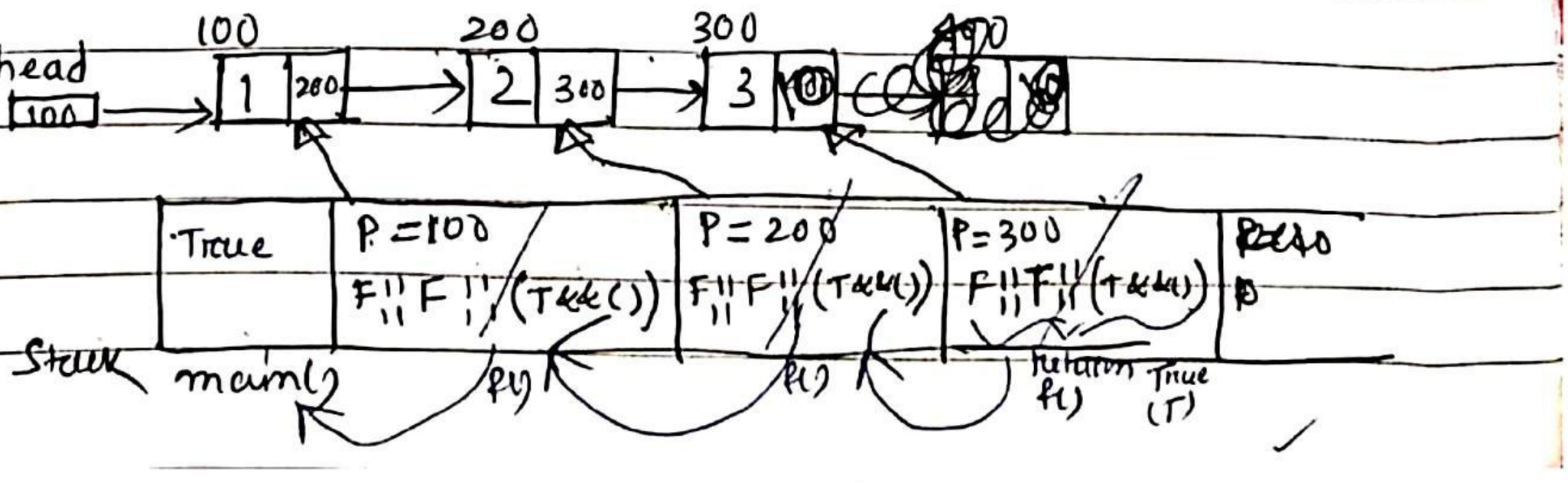
Question-5 (Checking if the list is in non decreasing)

```
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)));}
```

→ (a || b || (c && d)) (OR) (OR) (AND)

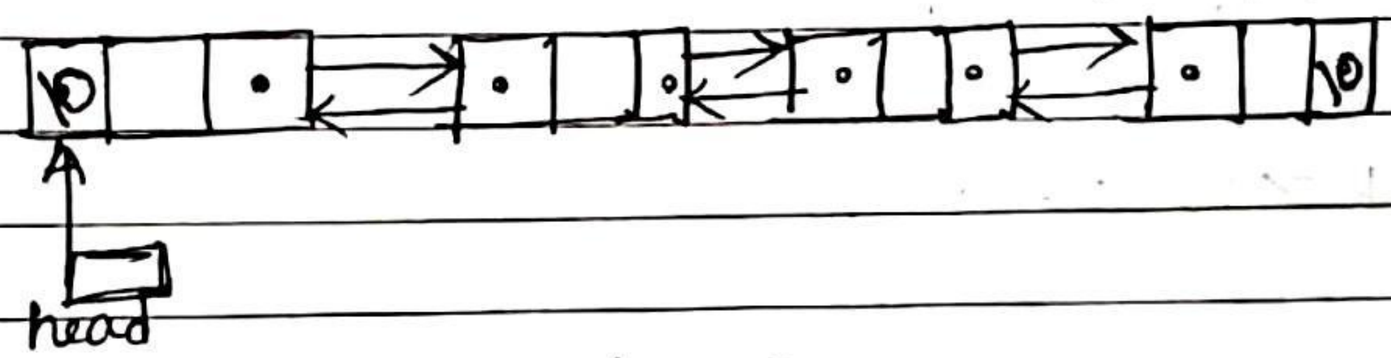
a	or	b	or	(c and d)
T		F	F	T
T		T	F	T
T		T	T	T
F		F	F	F

c	and	d
T	T	T
T	F	F
F	T	F
F	F	F

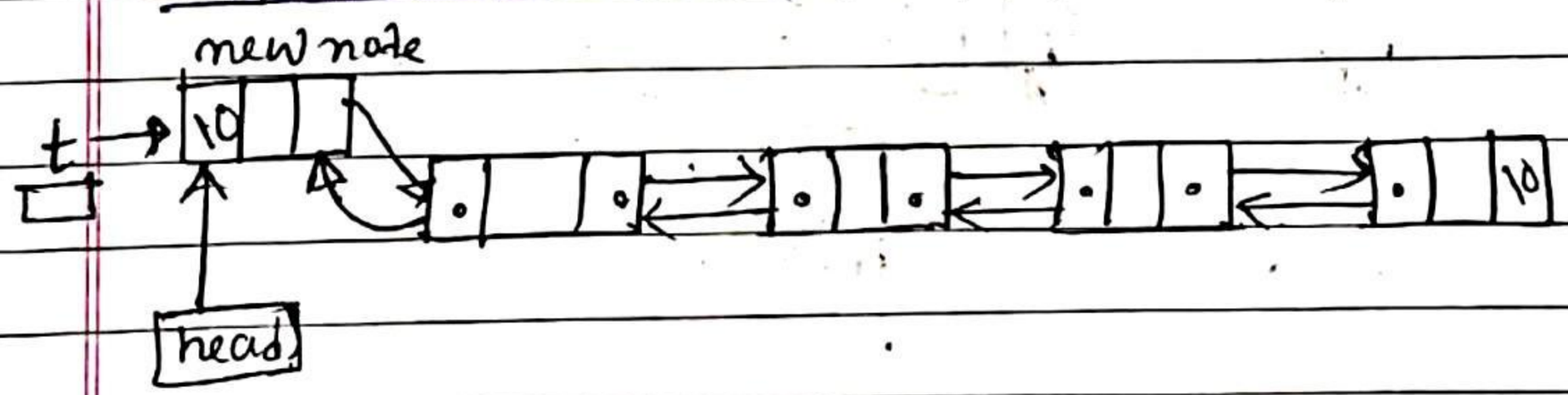


- Insertion into doubly linked list :

```
Struct node
{
  int i;
  Struct node *prev;
  Struct node *next;
};
```

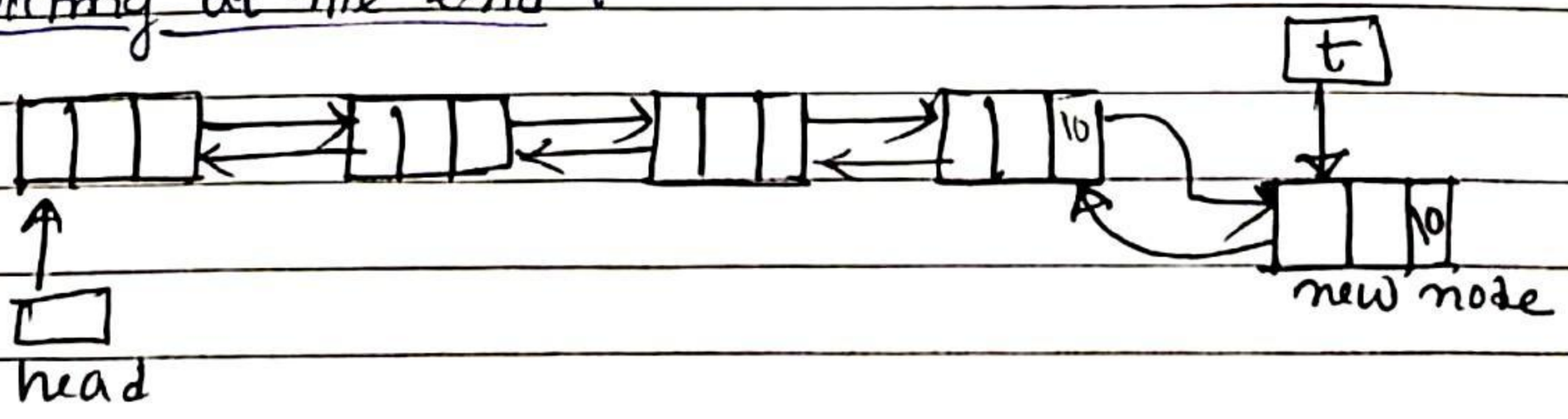


- Insert a node at ~~front~~ beginning :-




```
struct node *t;
t -> next = head;
head = t;
t -> prev = NULL;
head -> next -> prev = head;
```

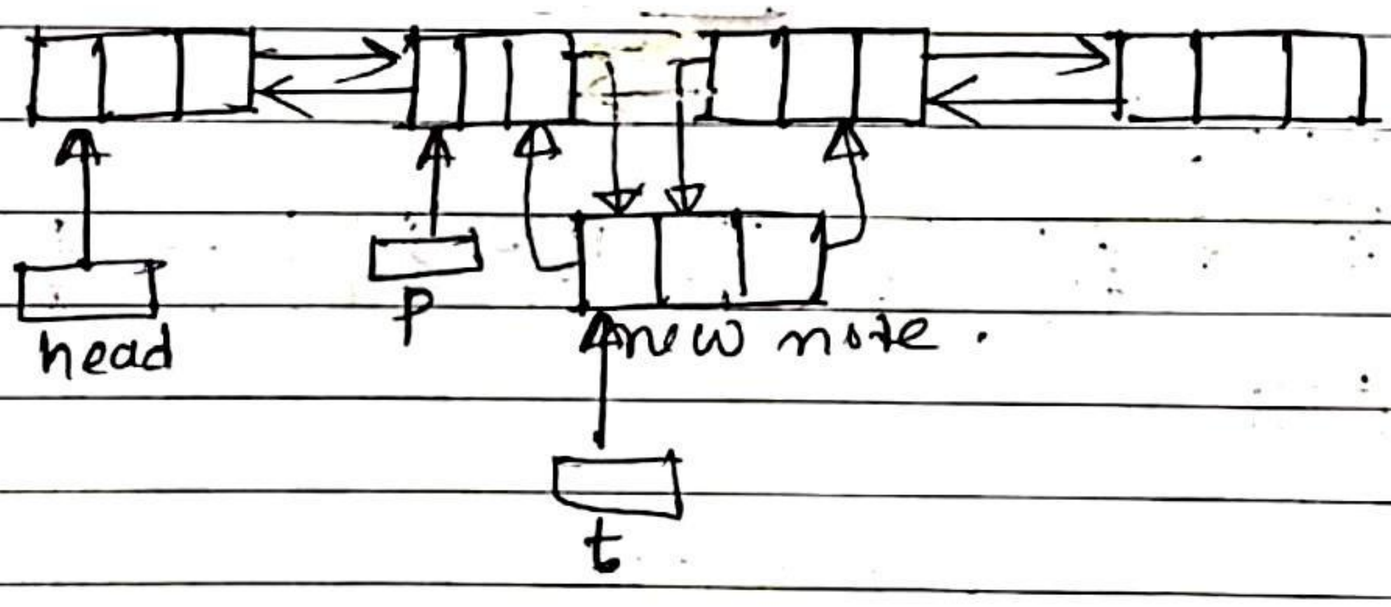
- Inserting at the end :-



~~while {~~

```
Struct node *p;
  p = head;

While (p → next)
  { P = p → next; }

p → next = t;
t → prev = p;
t → next = 10;
```

- Inserting the node at intermediate :-




```
struct nod *t;                    (hold the link
                                   and then modify)
{ t → prev = p;
  t → next = p → next;
  p → next = t;
  p → next → prev = t;
```