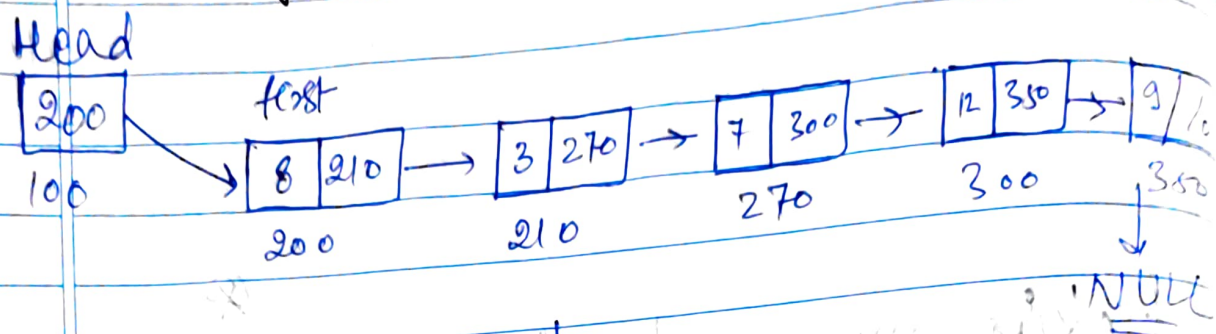
Date / /

LINK LIST

Display LinkList := (LinkList → LL)
Abbreviation.




```
Struct Node *p = Head
while (p != NULL){
cout << p -> data;
p = p -> next;
}
```

Keep incrementing Node *p. Until reach NULL.

Recursive Display LL :-

```
void Display (Struct Node
              *p) {
if (p != NULL){
cout << p -> data;
Display (p -> next);
}
}
```

→ condition to go ON

Display Trace Tree

[Trace Tree for Recursive Display]

Display(first)

Disp(200)

print 8 | Disp(210)

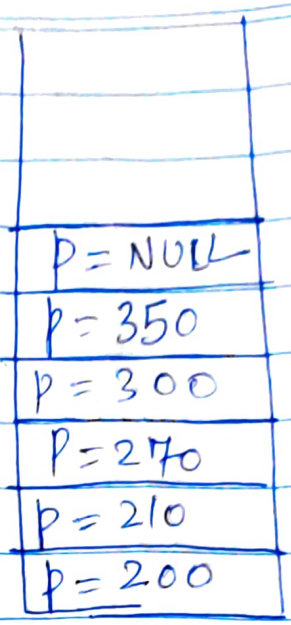
print 3 | Disp(270)

print 7 | Disp(300)

print 12 | Disp(350)

print 9 | Disp(NULL) ⊗ Terminate

Nothing left to execute



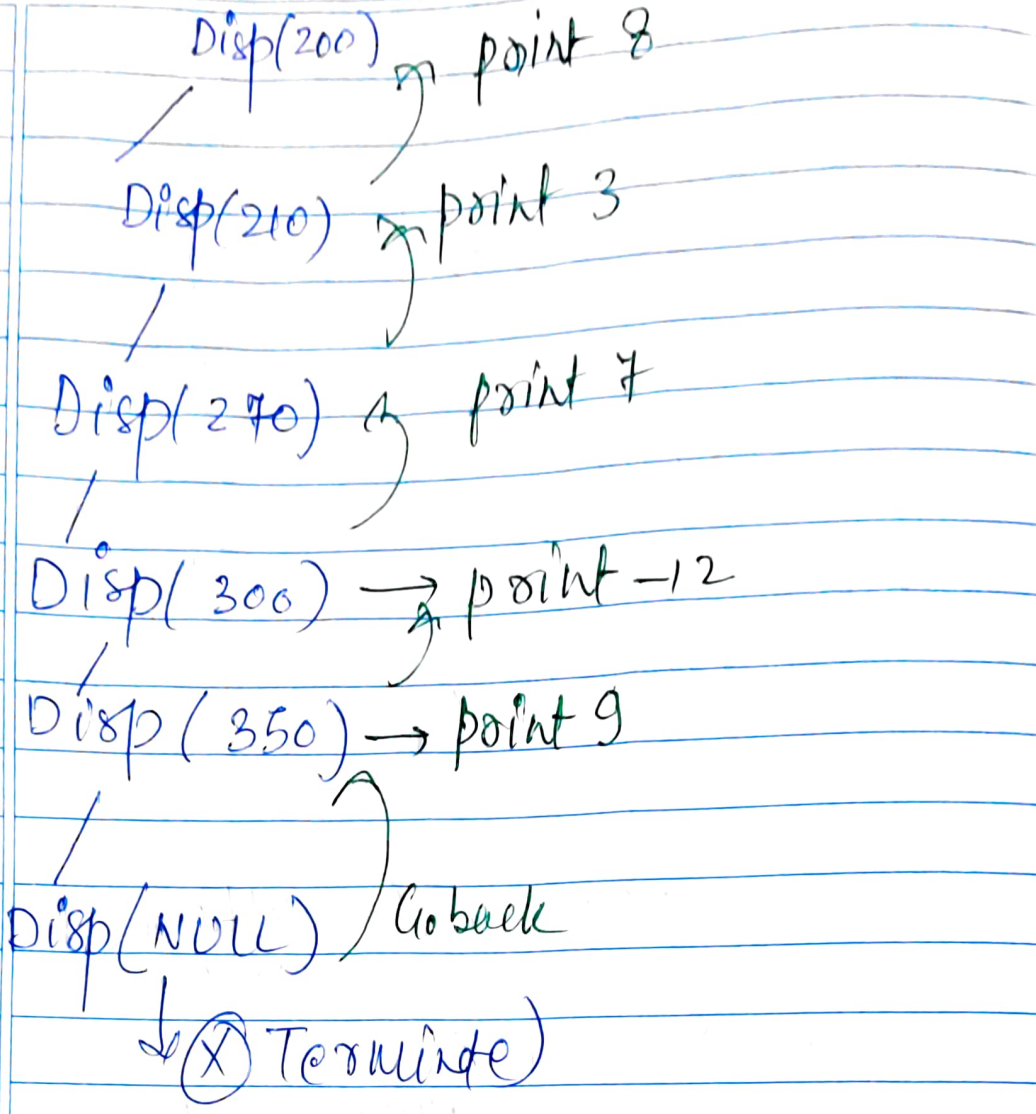
Stack

Output = 8 3 7 12 9]

Change the code

```
void Display(struct Node *p){
    if(p != NULL){
        ~~cout<<p->data;~~
        Display(p->next);
        cout<<p->data;
    }
}
```

Trace Tree.



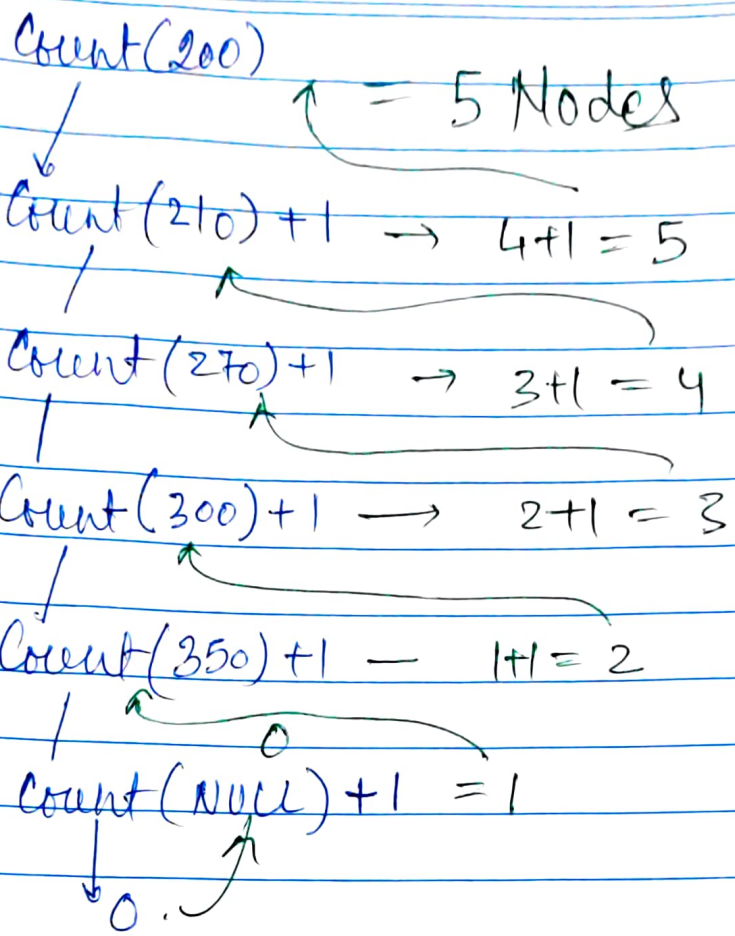
Output: 9 12 7 3 8 (Print Reverse)

Count Number of Nodes in LL Recursive :-

```
int count(struct Node *p){
if(p != NULL){
if(p == NULL){
  return 0;
else
  return count(p->next)+1;
}
```

Trace Tree

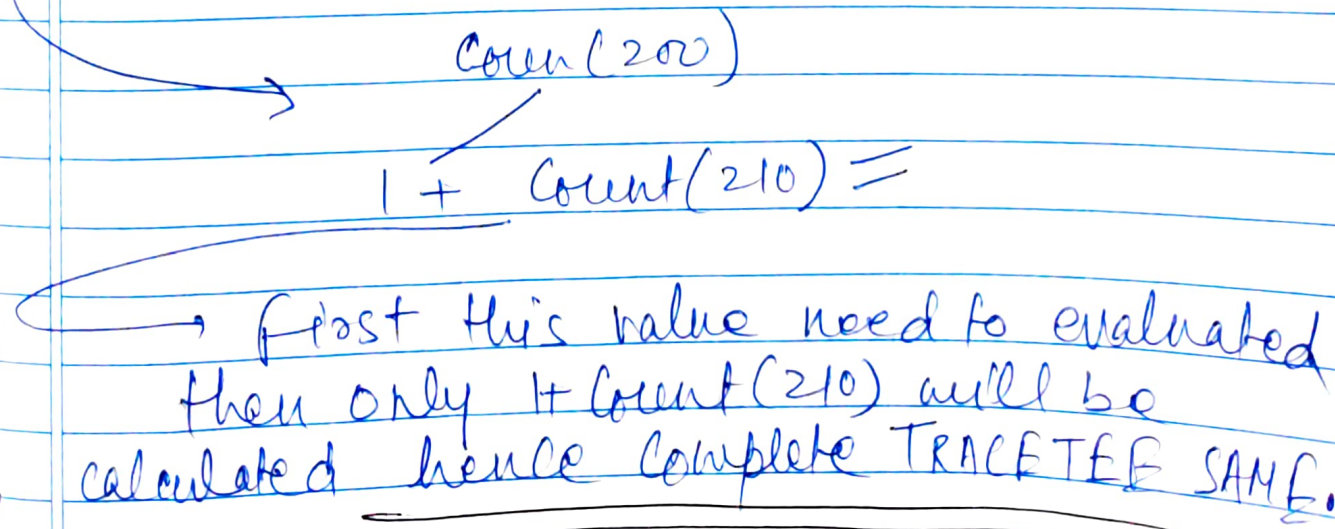
Date / / /



* Addition is DONE returning time Not calling time.

* Small change in code

↳ Return (1+ count(p→next) equal
= Return (Count(p→next)+1)



→ first this value need to evaluated then only 1+Count(210) will be calculated hence Complete TRACE TEE SAME.

Date / /

Sum of LL elements Recursive

```
int add(struct Node *p){
  if (p == NULL) {
    return 0;
  } else {
    return add(p->next)
           + (p->data);
  }
}
```

Recursive Search in LL :=

```
Node* Search
(Node *p, int key) {
```

Maximum Element in LL Recursive :-

```
int my_max(Str Nod *p){
  int x=0;
  if (p == NULL) {
    return INT_MIN
  } else {
    x = my_max(p->next)
  return
  (x > p->data ? x : p->data)
  ;
```

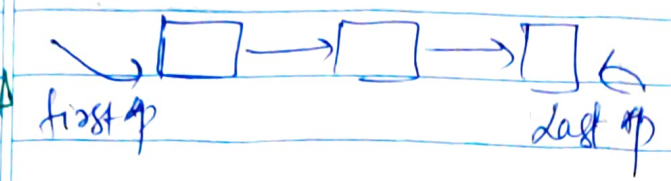
```
if (p == NULL) {*1
return NULL
}
else {
if (p->data == key) *2
  return p;
else *3
Search(p->next, key);
}
}
```

*1 = Boundary Condition

*2 = True Condition

*3 = Fail Condition.

Insert a Node in LL at Last Using Two Pointer :-



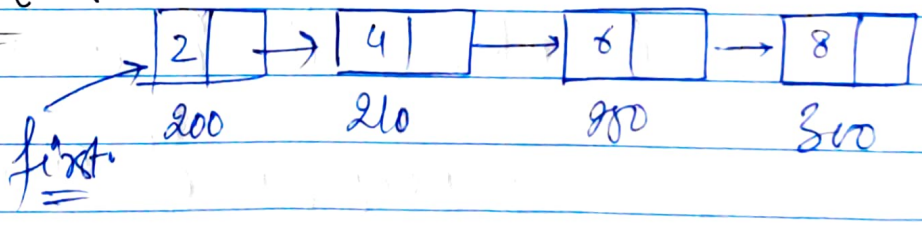

```
void Insert(int x) {
  Node *t = new Node
    t->data = x
    t->Next = NULL
  if (first == NULL) {
    first = last = t
  } else {
    last->next = t;
      last = t;
  }
}
```

DeleteNode *

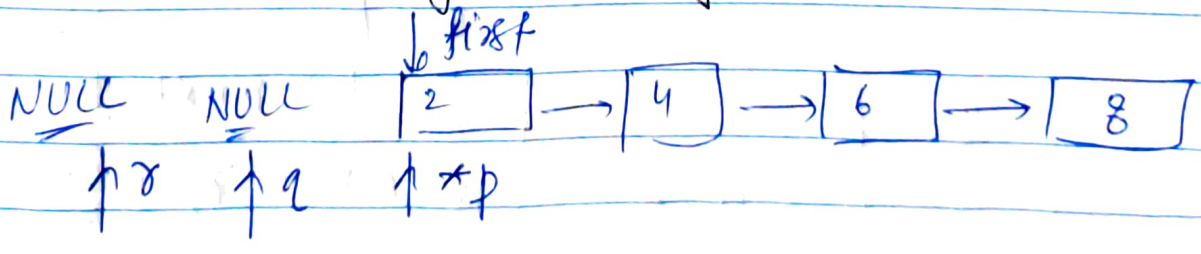
Say pos = 4, Not head

```
Node *p = first; Node *q = NULL
for(i=0; i=pos-1; i++) {
  q = p
  p = p->Next
}
q->next = p->next;
delete p;
```

Reversing LL :=



Reverse LL by Sliding Pointer :-



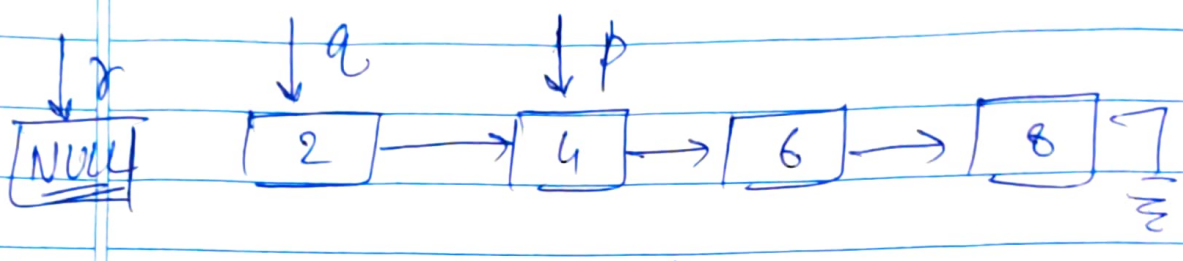
These three pointer SLIDE.

Date / /

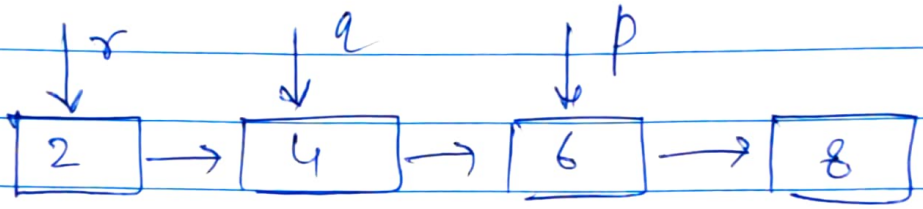
(SLIDE MECHANISM)

1st ⇒ r SLIDE q
2nd ⇒ q SLIDE p
3rd ⇒ p SLIDE to NEXTNODE

So LL look like below:



Again SLIDE Mechanism



That's how slide pointer & 3 Pointer work!.

So putting it in code

p = first, q = NULL, r = NULL

SLIDE Pointers:

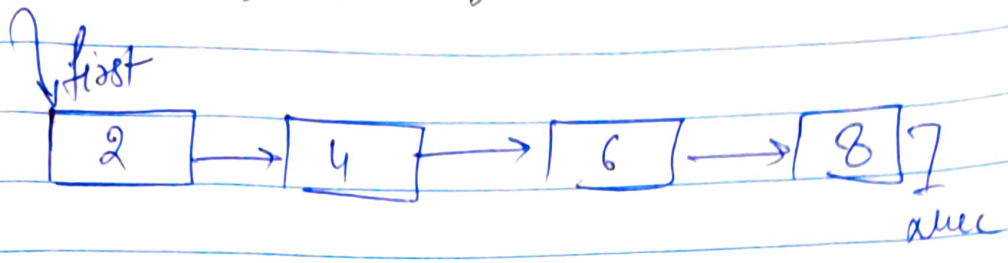
1st → r = q
2nd → q = p
3rd → p = p → next

[Slide + Reverse LL]

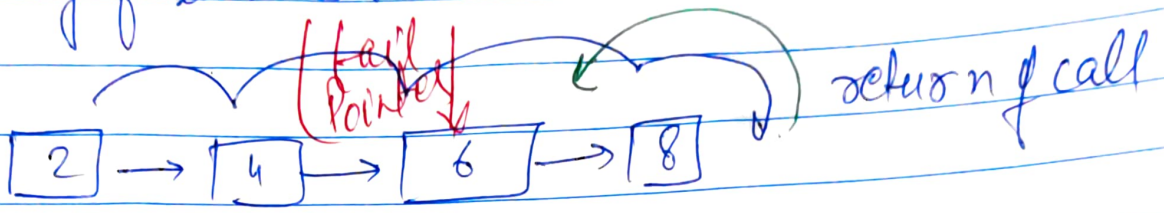
```
while(p != NULL){
    r = q
    q = p
    p = p->next      } SLIDE
    q->next = r;     } Reverse
}
```

Recursion Reverse LL :=

Date / / /



* Reversing of link should be done while return



While returning of RECURSIVE FUNCTION we should have Tail pointer also to reverse Link.

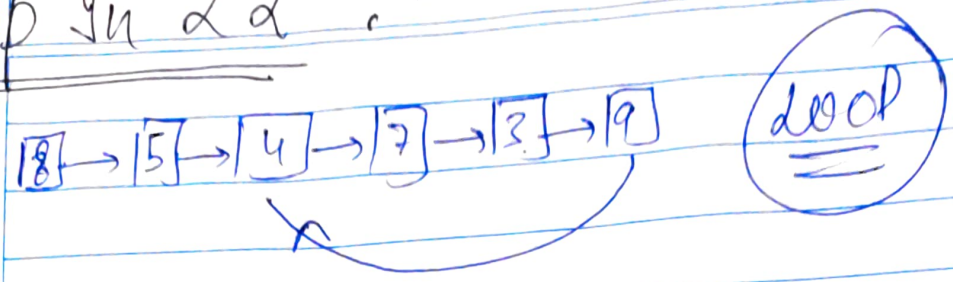
```
void Reverse(Node *P, Node *Q){
if(P != NULL){
Reverse(P, P->Next);
P->next = Q;          → reverse link while Return
} else {
first = Q             ← Return time if last node, assign to first.
}
```

Reverse(NULL, first);

call from main function;

Date: /

Loop In LL :-



Fast & Slow Pointer Method

```
int isloop(Node *f){
    Node *p, *q;
    p = q = f;
    do {
        p = p -> next;
        q = q -> next;
        q = (q != NULL ? q -> Next
                : NULL)
    } while(p && q)
    return p == q ? true : false
```

↑ Check if q has reach end in case of Linear of No Loop LL.