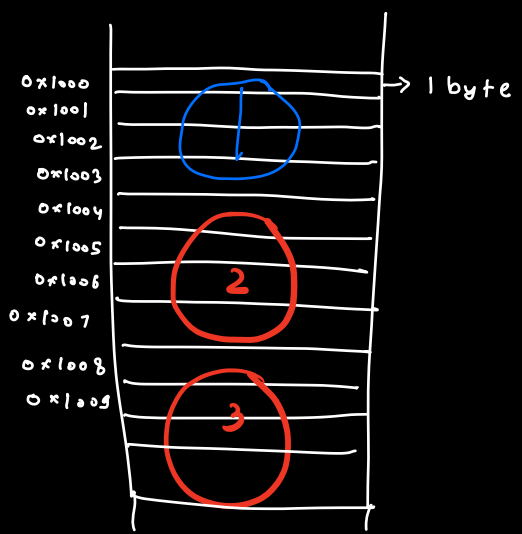
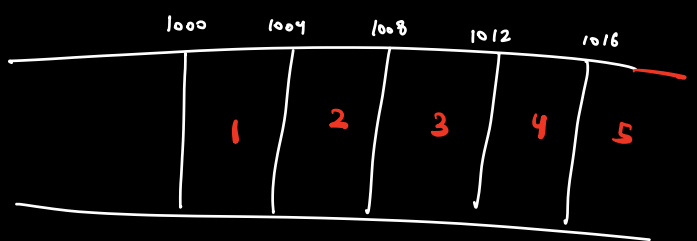
Arrays

→ contigous memory allocation

1 int = 4 bytes



1 byte = 8 bits



First element start at 1000

Start index

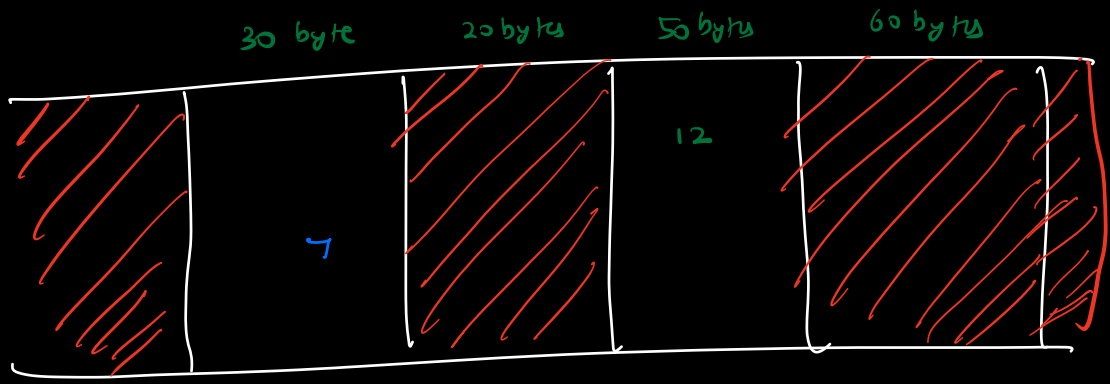
arr[N]

0th → 1000 + 0 4

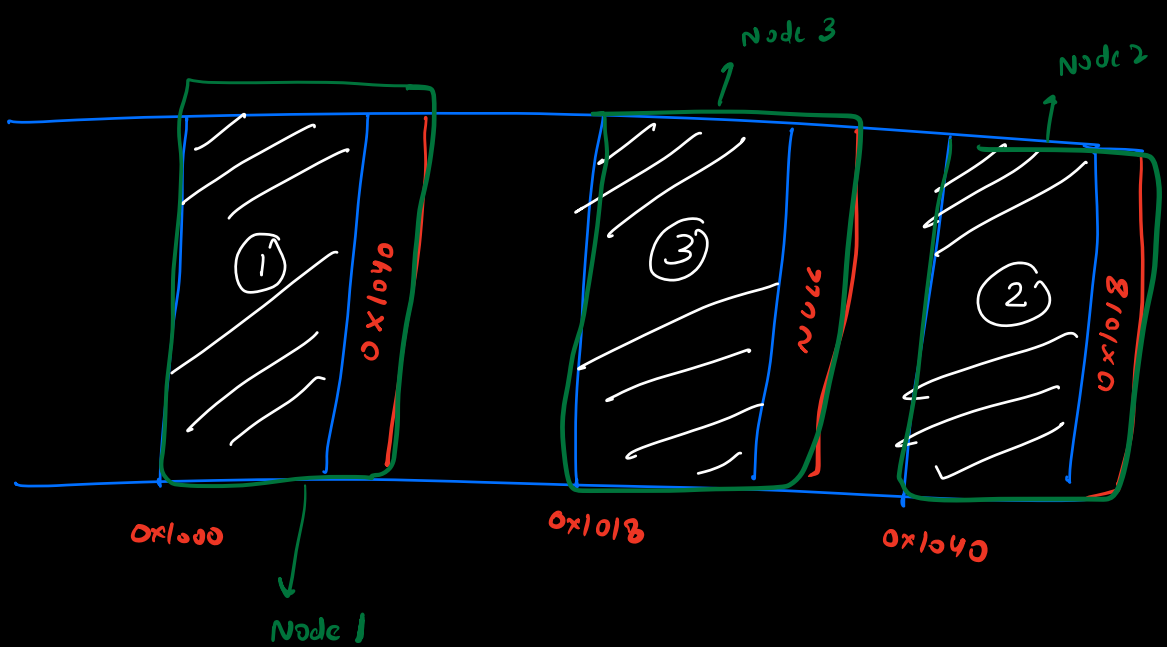
1st → 1000 + 1 . 4

2nd → 1000 + 2 x 4

ith → 1000 + i x 4

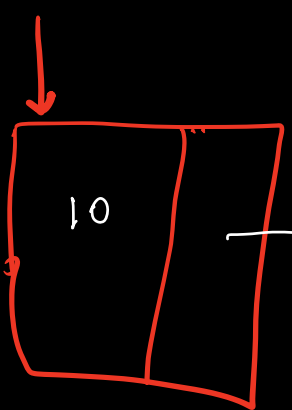


$$\frac{80 \text{ bytes.}}{4} = 20 \text{ integers.}$$

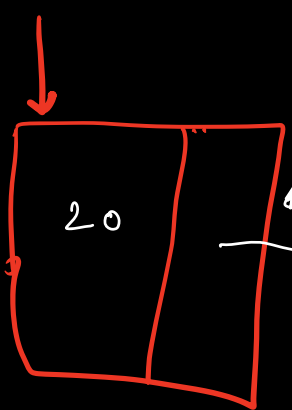


```
Node {

    int data
    Node next
    Node ( int x)
    {     this.data = x
          this.next = null }

}
```

```
Node h1 = new Node (10)
```




```
Node h2 = new Node (20)
```

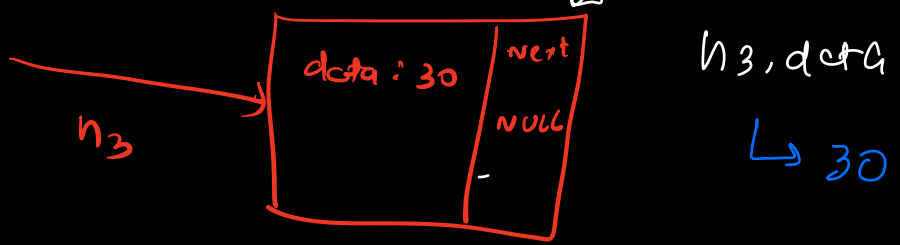



```
print(h2.data)
```

20

h1.next = h2

Node h3 = new Node (30)



h3.data
↳ 30

h2.next = h3

print (h2.next.data)
↳ 30

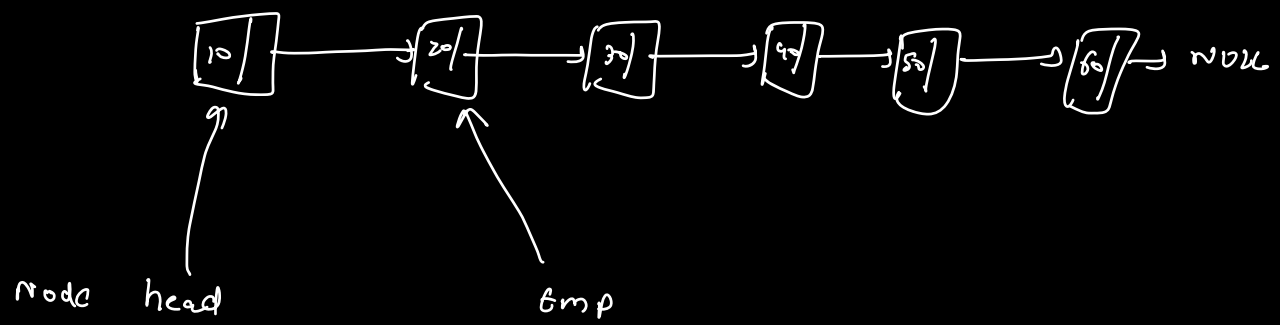
print (h2.next.next.data)

Null pointer exception

h2.next.next → equal to NULL

NULL . _ }→ NPE

NULL . _



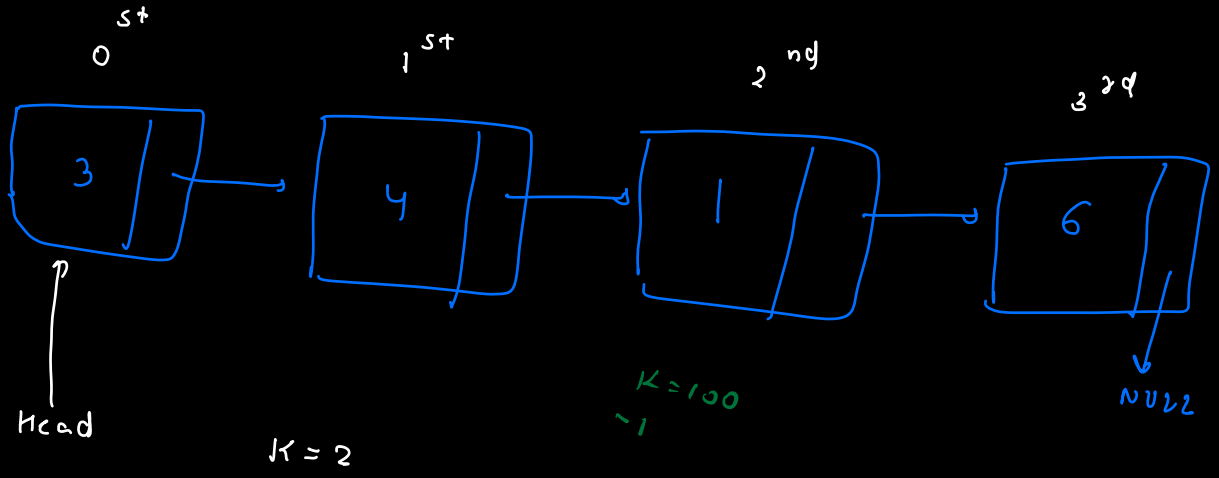
Node tmp = head

tmp = tmp.next

Q. Given a LL (given the head). find data stored at K^{th} index (from start). Return -1, if not exist

↓

(0 based)

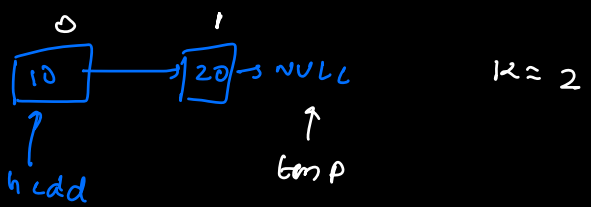



```
for( i = 1 ; i <= k ; i++)
{
    head = head.next
}
print (head.data)
```

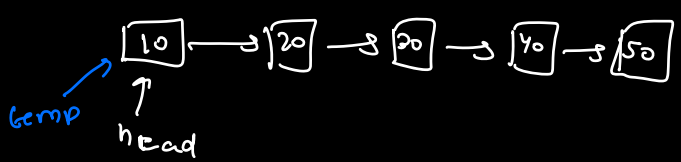
never move the head

```
int kthNode ( Node head, int k)
{
    Node tmp = head
    for( i = 1 ; i <= k ; i++)
    {
        if( tmp == NULL) return -1
        tmp = tmp.next
    }
    if( tmp == NULL) return -1
    return tmp.data
}
```

TC: O(k)
SC: O(1)



Q Print the LL

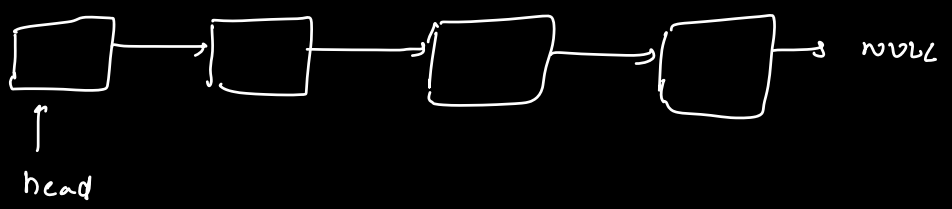


10
20
30
40
50

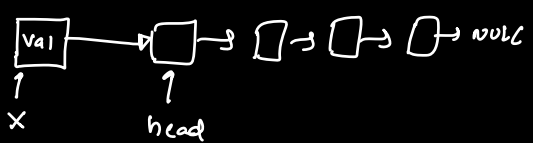
```
Node temp = head
While ( temp != NULL )
{
        Print( temp.data )
        temp = temp.next
```

TC: O(N)
SC: O(1)

Insertion in LL



1. Insertion at head

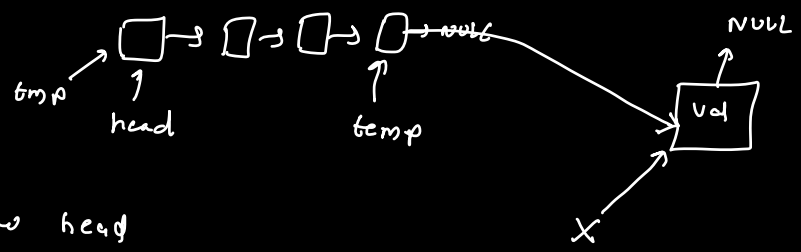


return new head

```
Node insert_at_head ( Node Head, int val )
{
        Node X = new Node (val)
        X.next = head
        return X
```

TC: O(1)
SC: O(1)

II Insertion at end

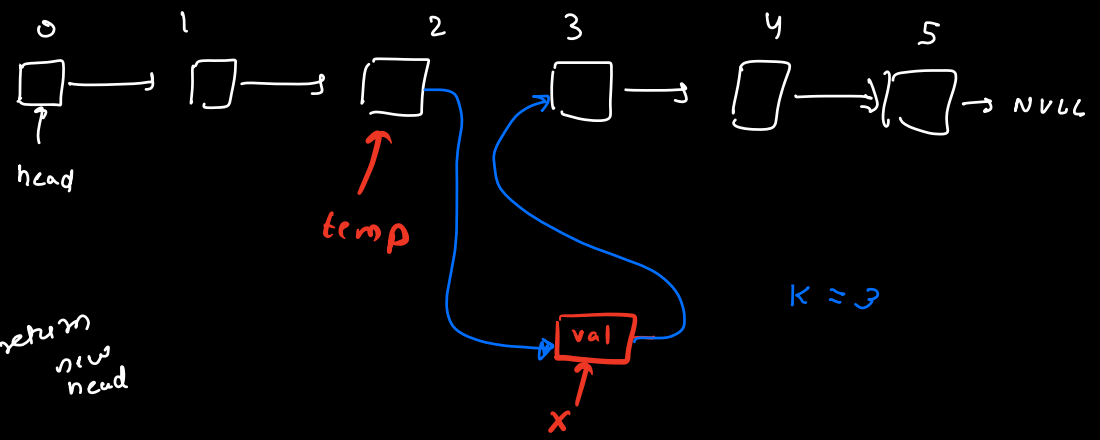


return new head

```
Node insert_at_end ( Node Head, int val)
{
    Node X = new Node (val)
    Node temp = head
    If ( temp == NULL) return x
    While ( temp.next != NULL)
    {
        temp = temp.next
    }
    temp.next = x
    return head
}
```

TC: O(N)
SC: O(1)

Insertion at k^{th} index (Not head not not tail)



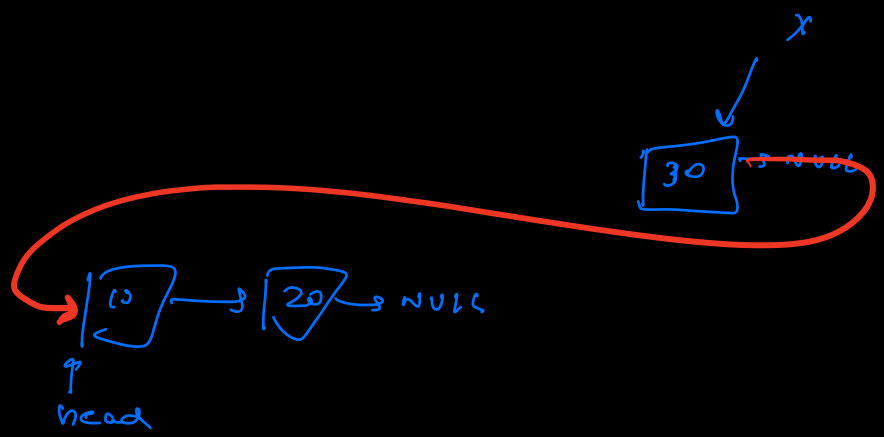
return new head

Node insert_at_k^{th}_Node (Node Head, int val, int k)

```
Node x = new Node (val)
Node tmp = find k^th node (k-1)
If (tmp == NULL) return NULL
x.next = tmp.next
tmp.next = x
return head
```

TC : O(k)

SC : O(1)


x
30
10
20
NULL
head