Linked List Basics

Content

- Classes & Objects
- Multiple obj references
- Constructor
- Array & Linked list
- Size of Linked list
- Insertion/ Deletion in LL

Class → It is a blueprint (for objects)

eg floor plan of a house/ building

Object → Real instance of a class

eg Physical house/ building

(one class can have multiple objects)

Class →
- Attributes (used to define data)
- Methods (to define funtionality)

```
Class Car {
    brand
    color
    category
    mileage
    ⋮
    drive()
    AC()
    music()
    ⋮
}
```

attributes / property

methods / functionality

objects

Car1 :
brand → mercedes
color → blue
category → sedan
mileage → 8
⋮

Car 2
brand → ferrari
color → red
category → f1
mileage → 4
⋮

drive() drive()
AC() AC()
music() music()
⋮ ⋮

Same functionalities in all objects of same class.

```
Class Student {
    String name
    int id
    ⋮
    study()
}
```

attributes

Student S1 = new Student()

[#2368]

object reference of Student class

name : Gaurav
id = 32

[#2368] memory address

S1.name = "Gaurav"

S1.id = 32

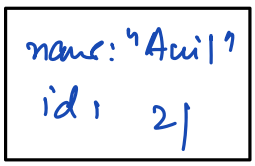
↳ dot is used to access attributes & methods

S1.study()

Student s2 = new Student()

s2.name = "Anil"

s2.id = 21



[#1234]

Student s3; | → NULL)

print(s3.name)

↳ null pointer exception error

Student s4 = s2; [#1234]

shallow copy

print(s4.name) → "Anil"

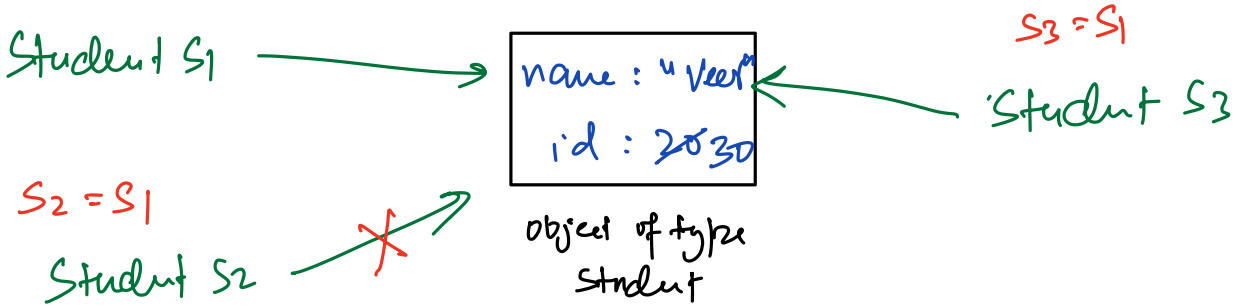
s4.name = "Rahul"

print(s2.name) → "Rahul"

same object referenced by multiple obj. references

Object vs Object reference

s1 = new Student()



s2 = s1

Student s2

s3.id = 30

s1.id → 30

s2.id → 30

s3.id → 30

s2 = null

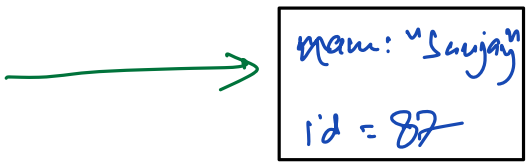
s1.id → 30

s2.id → ERROR

s3.id → 30

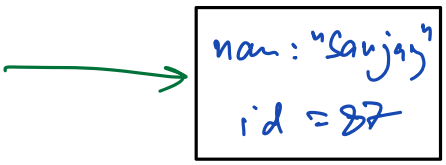
Deep copy : diff. objects for diff. obj. references

```
Student s1 = new Student()
s1.nam = "Sanjay"
s1.id  = 87
```



[# 6741]

```
Student s2 = new Student()
s2.nam = s1.nam
s2.id  = s1.id
```



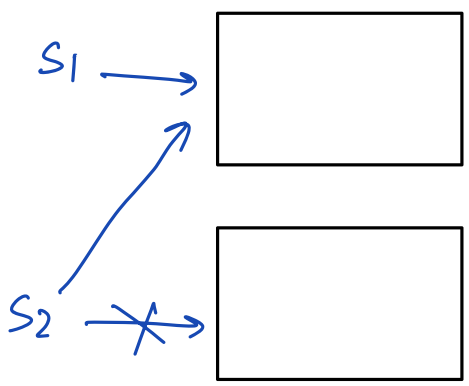
[# 6911]

```
s1.nam = "Kiran"
print(s2.nam) → "Sanjay"
```

Deep copy

```
Student s3 = new Student(s1)
```

```
s1 = new Student()
s2 = new Student()
s2 = s1
```



Question

Create a class Rectangle that support following methods:

1. find area of rectangle
2. Check if rectangle is square or not ?

```
class Rectangle {
    int length, breadth
    Rectangle (int x, int y) {
        length = x, breadth = y
    }
    int area () {
        return length * breadth
    }
    bool isSquare () {
        return (length == breadth)
    }
}
```

```
Rectangle r = new Rectangle()
r.length = 4
r.breadth = 6
print(r.area()) → 24
```

l = 4
b = 6

Constructor

Special method used to initialize the attributes of a class at a time of object creation.

→ 1. name is same as class name

2. no return type

```
Rectangle r2 = new Rectangle(4,6)
print(r2.area()) → 24
```

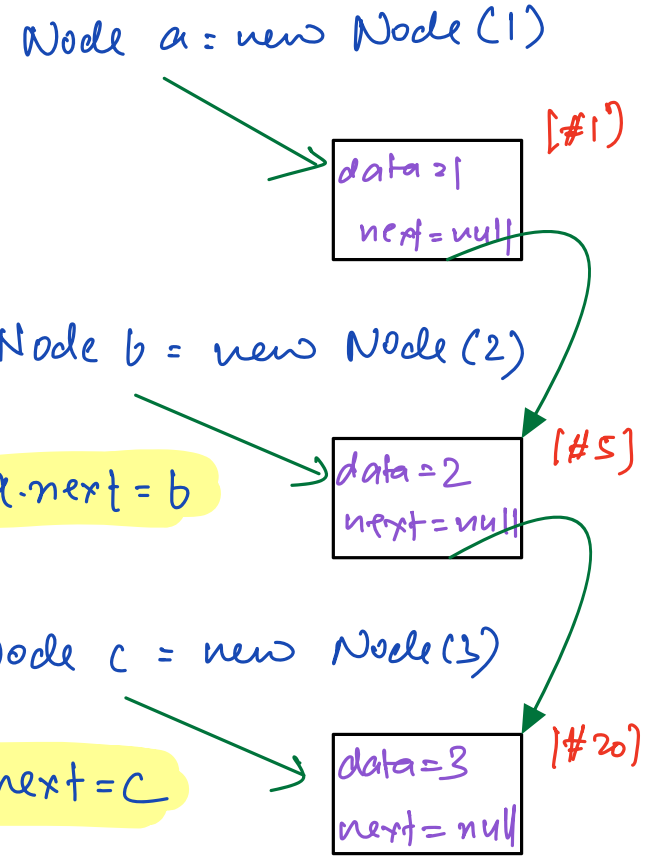
```
class Rectangle {
    int l, b;
    Rectangle(x, y) {
        l = x, b = y
    }
    Rectangle(Rectangle r) {
        l = r.l , b = r.b
    }
}
```

```
Rectangle r1 = new Rectangle(4,6)
Rectangle r2 = new Rectangle(r1)
```

Object reference inside a Class

```
class Node {
    int data;
    Node next; // obj. reference
    Node(int x) {
        data = x
        next = null
    }
}
```

```
Node a = new Node(1)
Node b = new Node(2)
a.next = b
Node c = new Node(3)
b.next = c
```




```
a.data → 1
a.next.data → 2
a.next.next.data → 3
```

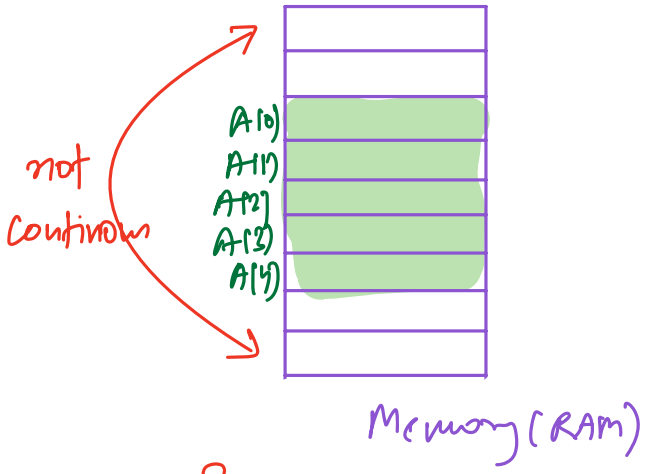
Linked list

Linked List vs Array

Array : O(1) random access

int A[5]

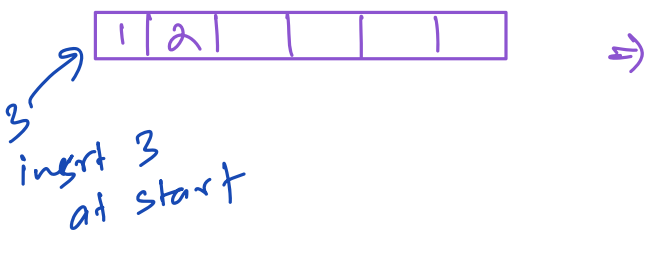
A[i] → O(1) TC



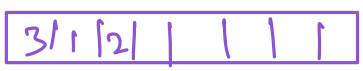
Can you store B[4] in the memory ?

↳ Can't store since we don't have 4 contigous locations [Memory fragmentation]

	At Start	At end	At K^{th} position (random)
Insertion	O(N)	O(1)	O(N)
Deletion	O(N)	O(1)	O(N)

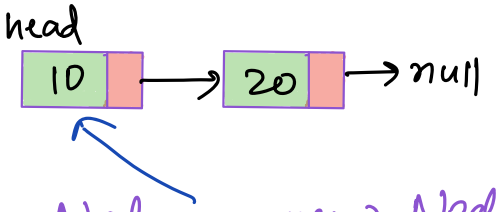


⇒



Shift existing elements to right & then insert

Linked list




```
class Node {
    int data;
    Node next;
    Node (int val) {
        data = val
        next = null
    }
}
```

```
Node a = new Node (10)
Node b = new Node (20)
a.next = b
```

Create a list of n values → A[n]

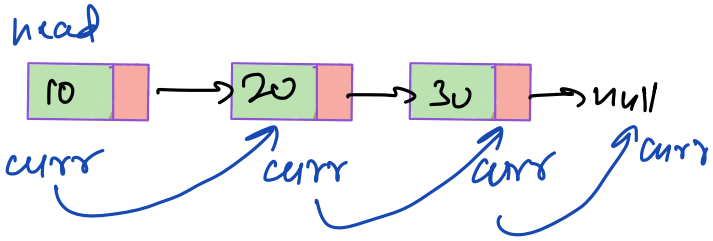
```
Node head = new Node (A[0])
head.next = new Node (A[1])
head.next.next = new Node (A[2])
head.next.next.next = new Node (A[3])
```

Print

```
head.data
head.next.data
.
.
.
!
```

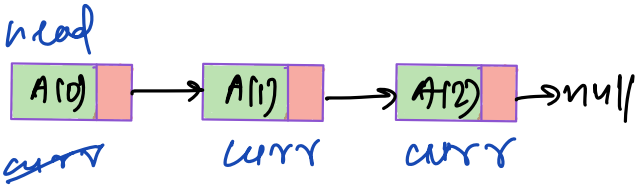
Iterate over LL

```
Node curr = head
while (curr != null) {
    print(curr.data)
    curr = curr.next
}
```



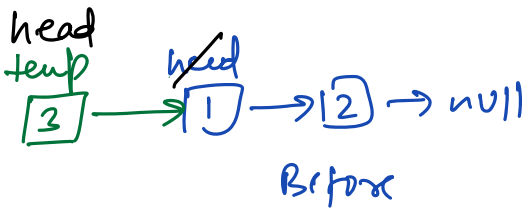
⇒ 10 20 30

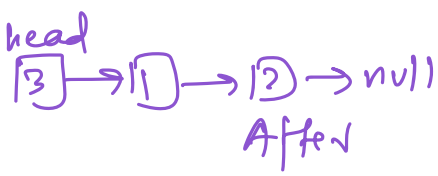
Create LL using loop




```
Node head = new Node(A[0])
Node curr = head
for(i=1; i<n; ++i){
    curr.next = new Node(A[i])
    curr = curr.next
}
```

Insert At Start



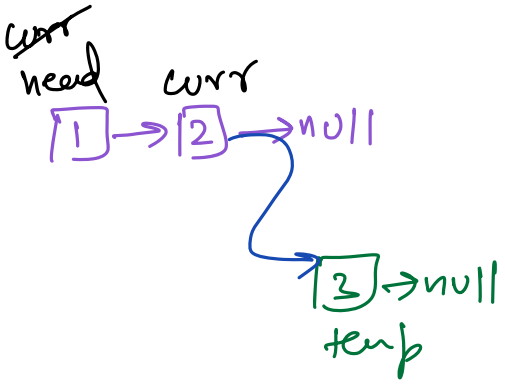



```
Node insertAtStart(head, val=3){
    Node temp = new Node(val)
    temp.next = head
    head = temp
    return head
}
```

TC : O(1)

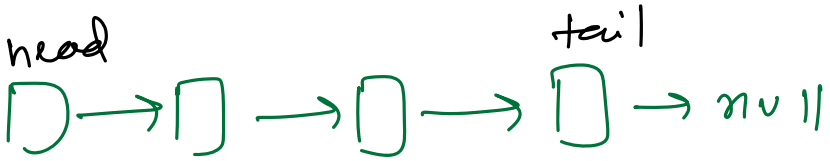
Insert At end

```
Node insertAtEnd(head, val){
    Node temp = new Node(val)
    if(head == null)
        return temp
    Node curr = head
    while(curr.next != null){
        curr = curr.next
    }
    curr.next = temp
    return head
}
```



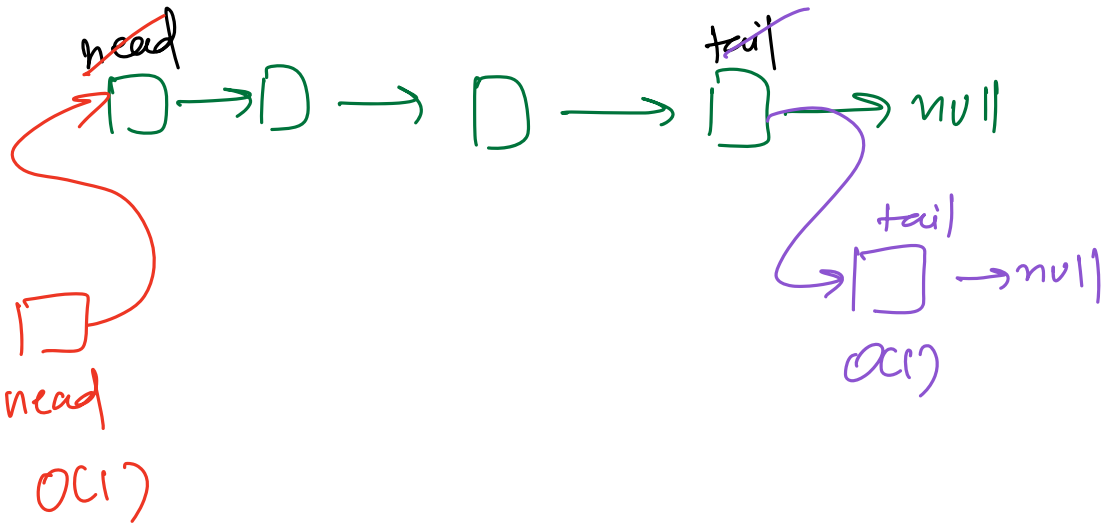
null exception if head=null

TC: O(N)



Using tail reference we can insert at end in O(1)

```
tail.next = temp
tail = temp
```

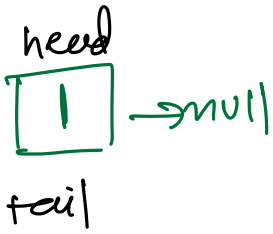


head = tail = null

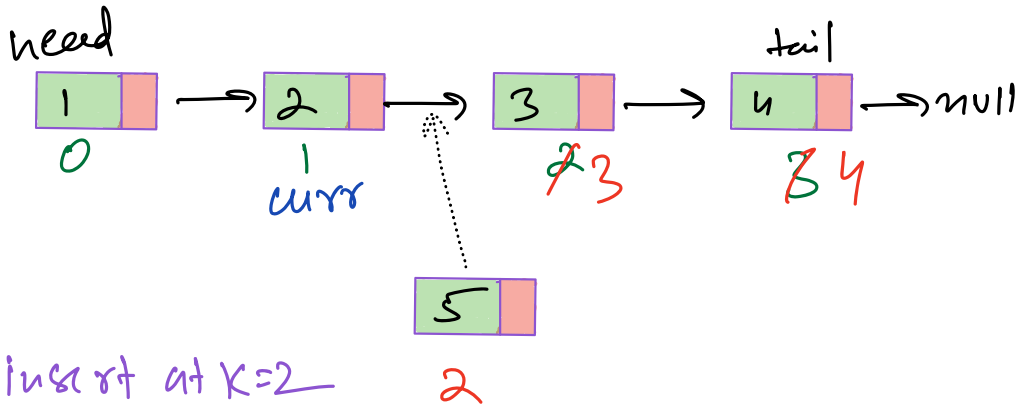
temp = new Node (1)

tail = temp

head = temp



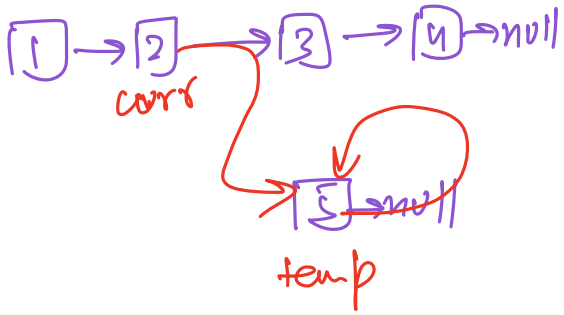
Insert at k^{th} position (random)

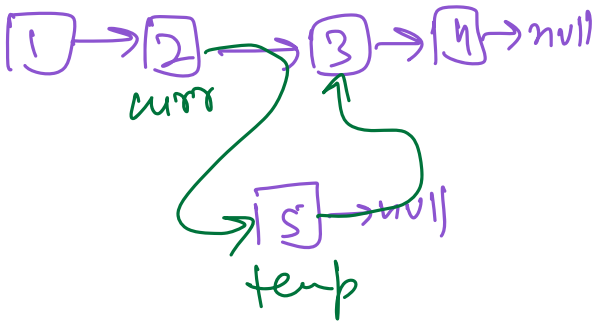


```
Node insertAtK(head, val, K) {
    temp = new Node(val)
    curr = head

    for(i=0; i<K-1; ++i) {
        curr = curr.next
    }

    curr.next = temp
    temp.next = curr.next

    return head
}
```

(val = 5, K = 2)

```
if(K==0)
    return insertAtStart(
        head, val);
```

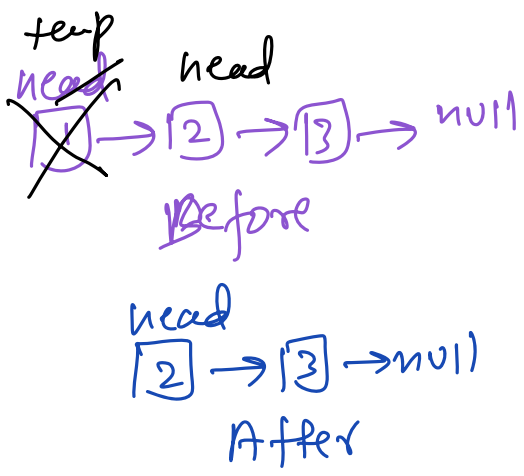




TC: O(K) : O(N)

Delete At Start

```
temp = head
head = head.next
delete(temp)
```



Delete At End
Delete At K
→ HW

Recommendations

1. Write code on Paper
2. Dry run
3. Never use "hit & trial"
4. Edge cases → LL is null ? (head = null)
   Size of LL = 1 ?
   Size of LL = 2 / 3 ?
   Problem specific