Contest 1 → Reattempt 2 ⇒ Aug 12 - 12:01 AM
Aug 20 - 11:59 PM

Reattempt 3 ⇒ Aug 21 - 12:01 AM
Oct 4 - 11:59 PM

Welcome 

Agenda : Classes / Objects
Constructors.
Linked List D.S
Problems 3

Classes & Objects

Classes → blueprint

Object → real instance of a class / blueprint

Components of classes

1. Attributes / Properties
2. Functionalities / Methods.

```
Class Car
{
    // attributes
    Seat
    Color
    Engine
    Model
    Airbags
    Types
    // functions / methods
    accelerate ( )
    break ( )
    music ( )
}
```

```
Sanjay's Car
{
  Merc
  2 seater
  Black
  // functions / methods
  accelerate ( )
  break ( )
  music ( )
}
```

```
Hemanth' Car
{
  Ferrari
  4 seater
  Red
  // functions / methods
  accelerate ( )
  break ( )
  music ( )
}
```

```
class Student
{
    string Name;
    int rollNo;
    int m1, m2, m3;

    // methods
    int totalmarks()
    {
        return m1 + m2 + m3;
    }
    int manMarks()
    {
        return man(m1, man(m2, m3))
    }
    void printName()
    {
        print(name);
    }
}
```

⇒ We define/initiate an object using 'new' keyword

```
new Student();
```

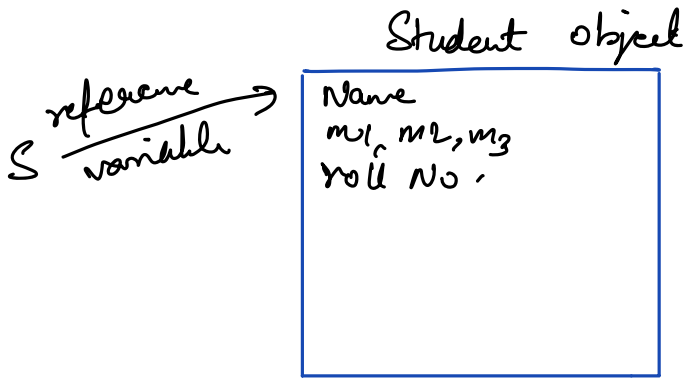
① creates a new object in memory.

But how to access that memory ?

```
Student s = new Student();
```

s → reference variable

Type of reference variable ⇒ same as class b/c it is holding the reference to object of that class.



How to assign values to the object ?

S.name = "Aman"
S. roll No = 152
S. m1 = 90
S. m2 = 91
S. m3 = 92

Student S2 ;

⇒ Error → does not point to anything right now.

∴ We need to initalize it.

① Student S2 = new Student ( ) ;

② Student S2 = NULL ;

③ Student S2 = S1 ;

S1 ← S2

S.name = "Aman"
S. roll No = 152
S. m1 = 90
S. m2 = 91
S. m3 = 92

⇒ S2.m1 = 100

print (S1.m1) ⇒ ?

100

Constructors

⇒ Can initialize attributes at the time of object creation.

```
class Pair
{
   int x, y

   // Constructor
   Pair ( int a, int b)
   {
      x = a ;
      y = b ;
   }
}
```

① Name of constructor is same as name of class

② It does not have any return type.

```
Pair p = new Pair ( 10, 20)
```

```
class Pair
{
   int x, y

   // Constructor
   Pair ( int x, int y)
   {
      this.x = x ;
      this.y = y ;
   }
}
```

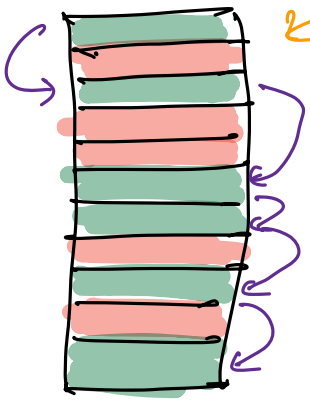
```
Pair p = new Pair ( 10, 20)
```

"THIS" Keyword holds reference to object being declared.

Linked List

Drawbacks of using arrays.

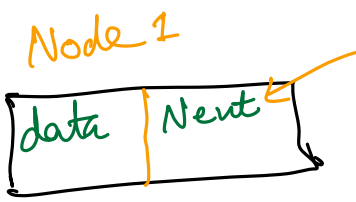
1. Fixed size
2. Contiguous memory.

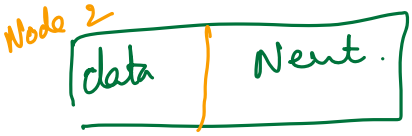


creating array is not possible
bc no contiguous memory available.

Linked list

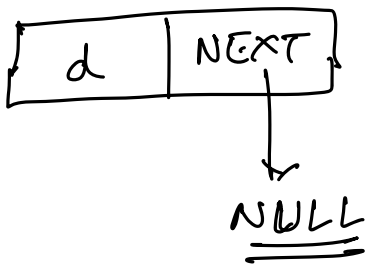
Dynamic data structure. consisting of sequence of nodes, each containing a data element



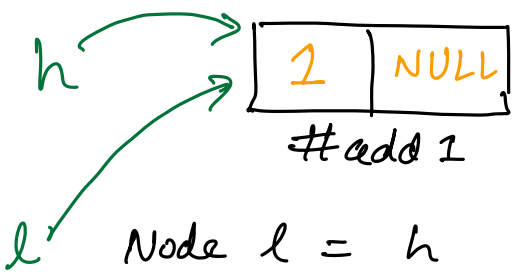


Structure of Node

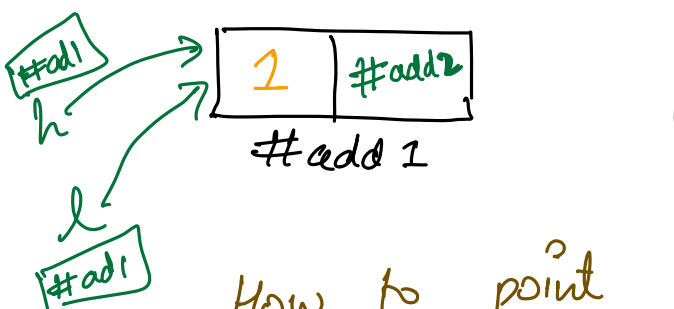
```
class Node
{
    datatype data
    Node Next

    Node ( datatype d)
    {
      this.data = d ;
      next   =  NULL;
    }
}
```



Eg: `Node h = new Node (1);`

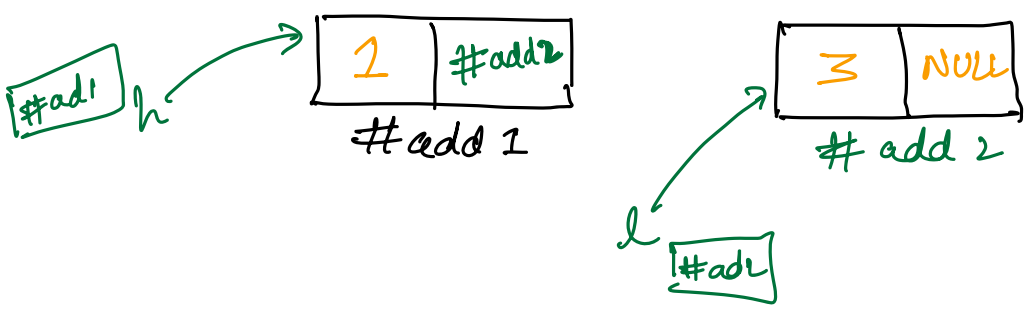


`l.next = new Node (3);`



How to point l to next node ?

⇒ `l = l.next`



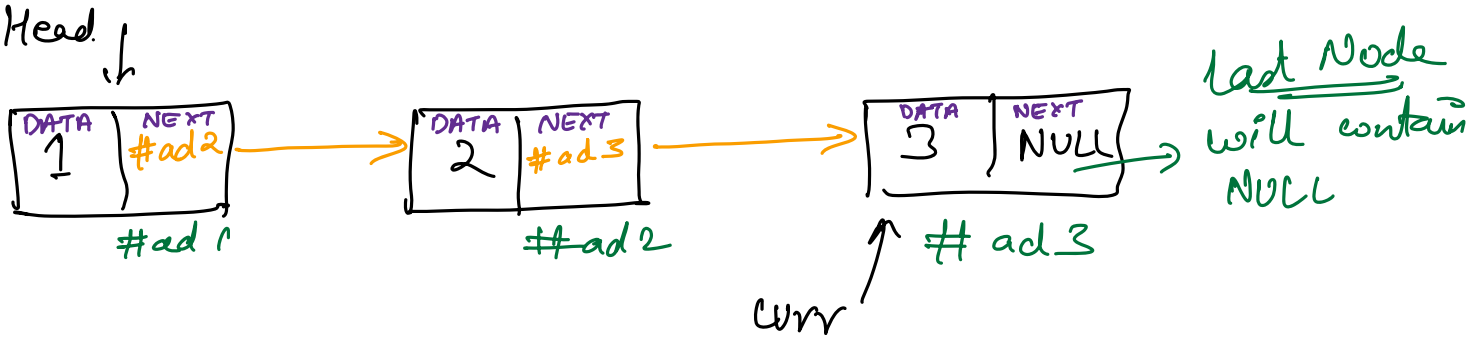
⇒ We call first node of linked list as Head.
We should always keep first node so that we don't loose start of linked lists..

---

① Given N, create linked list with N nodes having data 1, 2 ... N. Return head node address.

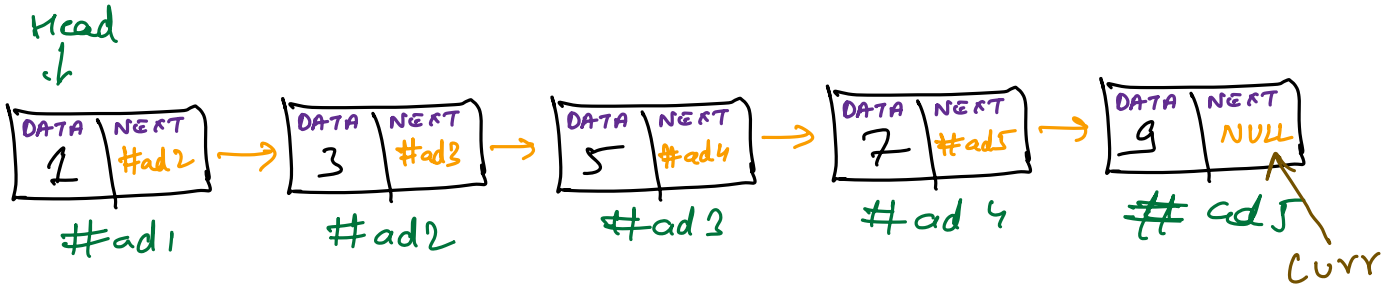
⇒
```
Node head = new Node(1);
Node curr = head;
for(i=2; i<=N; i++)
{
  curr = new Node(i);   // this line is wrong.
  curr.next = new Node(i);
  curr = curr.next;
}
return Head;
```

T.C ⇒ O(N)
S.C ⇒ O(N)



Q2 Given head of linked list, return size of linked list.

eg:

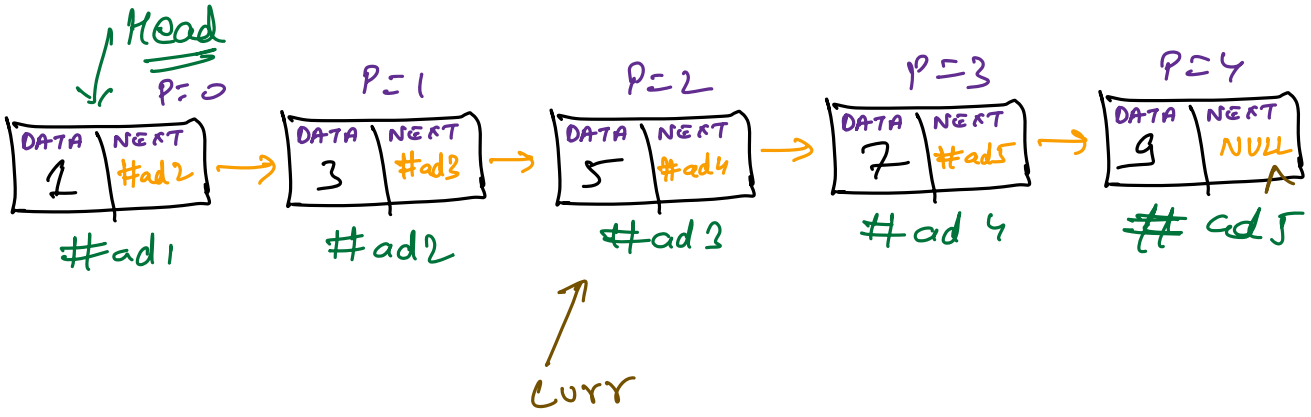



```
int size ( Node head )
{
    Node curr = head ;
    int count = 0;
    while (curr != NULL)
    {
        count ++ ;
        curr = curr.next;
    }
    return count;
}
```

count = 0 1 2 3 4 5

Q3 Insert a new node with data 'v' at index 'p' in a Linked list.

eg: v = 60 p = 3



Approach

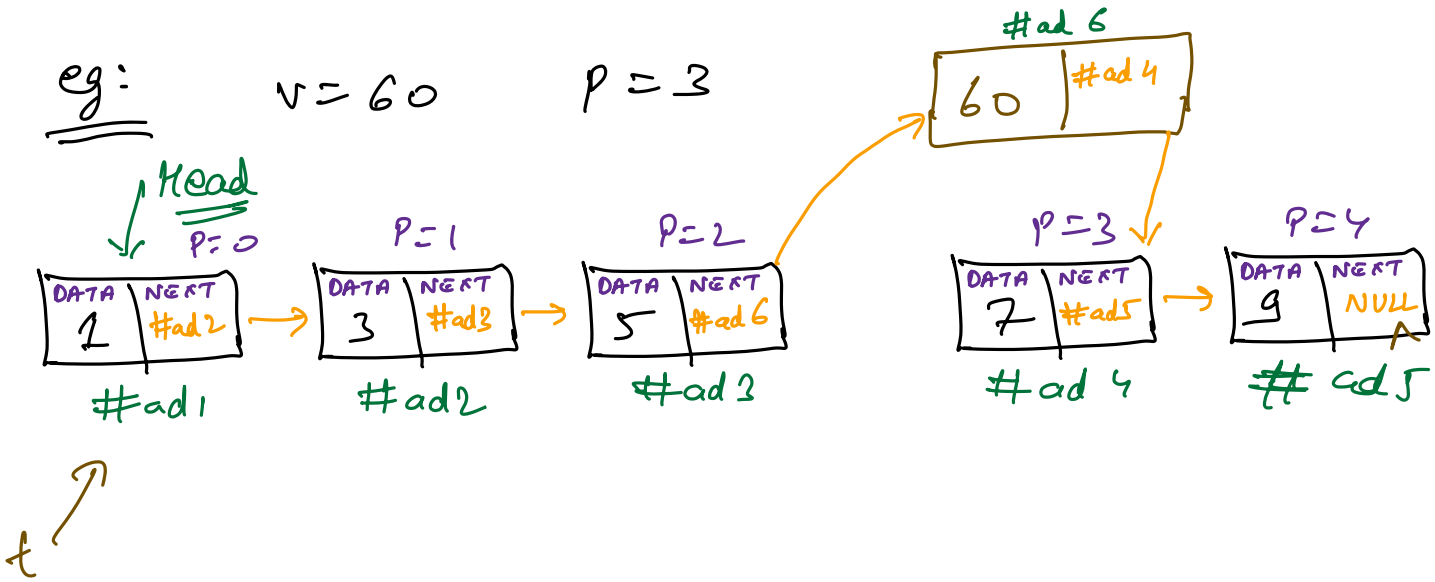
1. Traverse till $(p-1)^{th}$ node. Lets call it 't'
2. Create newNode 'v' with data 'v'
3. Set newNode.next equal to t.next
4. Set t.next to newNode.

Code

```
void insertAtIndex ( p, v, Node head )
{
    Node t = head;
    // Traverse.
    for (i=1; i < p; i++)
    {
        t = t.next;
    }
    // Create new node
    Node newNode = new Node(v);
```

```
// inserting node.
① newNode.next = t.next ;
② t.next = newNode;
}
```

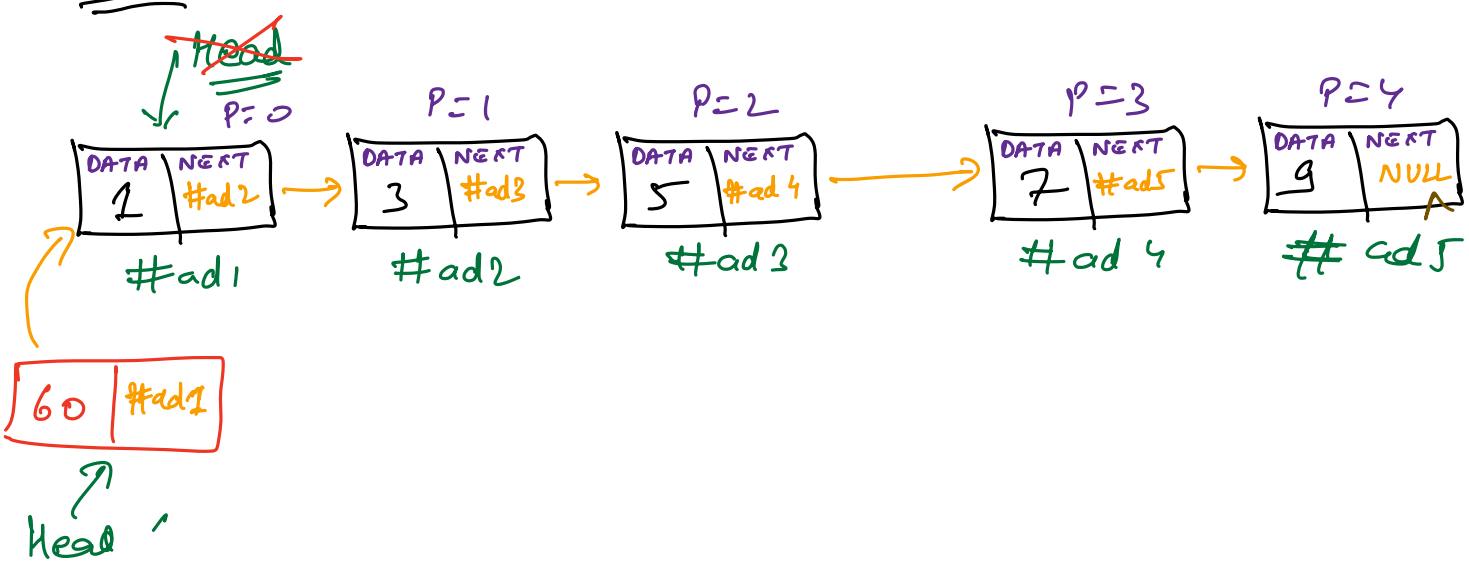
eg: v = 60 p = 3



★ Edge case

When p = 0 , insert at head of the list.

eg: v = 60 p = 0



```
void insertAtIndex( p, v, Node head )
{ // create new node
    Node newNode = new Node( v );
    // edge case -> when p=0
    if( p == 0)
    {
      newNode.next = head;
      head = newNode
      return;
    }
      Node t = head;
      // Traverse.
      for(i=1; i < p; i++)
      {
          t = t.next;
      }
      // inserting node.
      ① newNode.next = t.next;
      ② t.next = newNode
}
```