Linked List :-

Linked list is a linear type of dynamic data structure.

Linked list is a collection of Nodes.

Each Node divided into two parts.

1) Data or into Part

That hold actual data

2) Pointer or Link Part

That hold address of Next Node

Types of Linked List

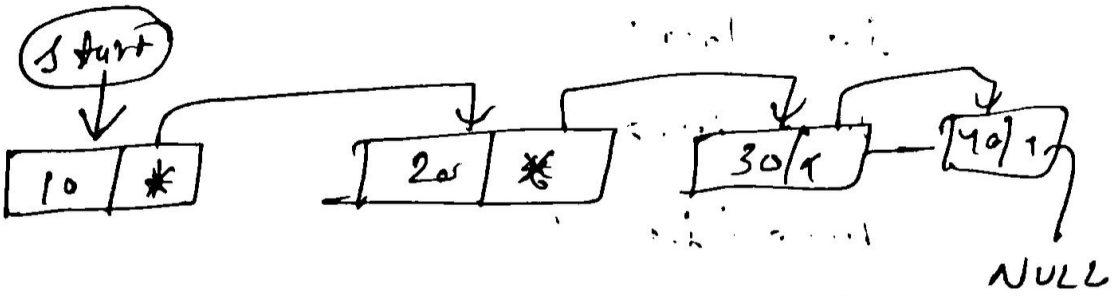
1) Singly Linked list
2) Circular " "
3) Doubly " "
4) Circular Doubly " "

Singly Linked List :-

Each Node has one Pointer that hold address of Next Node

Pointer field of last Node contain NULL.

We can traverse singly Linked only in forward direction



Linked list ADT (Abstract Data Type)

	Singly	doubly
addfirst()	✓	✓
addLast()	✓	✓
addAtIndex()	✓	✓
removefirst()	✓	✓
removeLast()	✓	✓
removeAtIndex()	✓	✓
getfirst()	✓	✓
getAtIndex()	✓	✓
getLast()	✓	✓
count()	✓	✓
search()	✓	✓
disp()	✓	✓
reverse()	✓	✓
sum()	✓	✓
big()	✓	✓
⋮		

Implementation of Singly Linked List

```
clam Node
{
   iw data;
   Node Next;
   Node (iw x)
   {
      data = x;
      Next = null;
   }
}

clam SLink
{
   Node start = null;

   void addLast (iw x)
   {
   Node ptr = new Node(x);
   if (start == null)
   {
      start = ptr;
   }
```

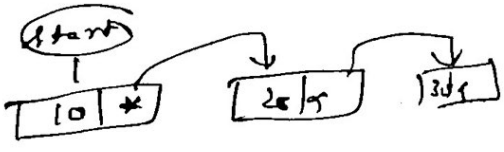
```
else
{
  Node t = start;
  while (t.Next != null)
  {
    t = t.Next;
  }
  t.Next = Ptr;
}
}

void addfirst (int x)
{
Node Ptr = new Node(x);
if (start == null)
{
  start = Ptr;
}
else
{
  Ptr.Next = start;
  start = Ptr;
}
}
```

```
void disp()
{
if (start == null)
SopL ("List is empty");
else
{
  Node t = start;
  While (t! =null)
  {
   System.out.print(t.data + " ");
   t = t.Next;
  }
}
}
```

getFirst()
getLast()
count() ✓
sum()
big()
removeFirst()

① start → null

②



```
void getfirst()
{
  if(start==null)
    sopl("List is empty");
  else
    sopl(start.data);
}
```

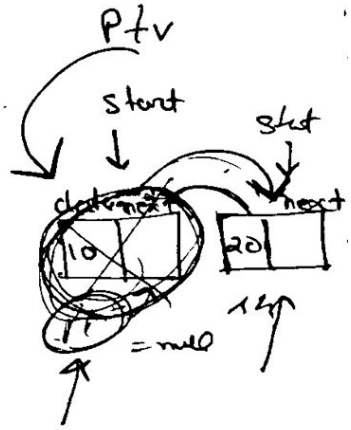
```
void getlast()
{
  if(start==null)
  sopl("List is empty");

else
{
  Node t = start;
  while(t.Next!=null)
  {  t = t.Next;
  }
  sopl(t.data);

}
}
```

```
int Count()
{
    int c=0;
    Node t = start;
    While (t != null)
    {
      c++;
      t = t.Next;
    }
    return c;
}
```

```
int Sum()
{   int Sum=0;
    int Sum=0
    if (start == null)
    { SOPL ("List is Empty"); return 0;
    }
    else
    { node 7 = start;
      while (7 != null)
      { Sum += 7.data;
        t = t.Next;
      }
    return Sum;
}
```

```
int big()
{
    int bignum = Integer.MIN_VALUE;
    if (start == null)
    {
        SOPL("Sum is empty"); return 0;
    }
    else
    {
        node t = start;
        while(t != null)
        {
            if (bignum < t.data)
                bignum = t.data;
            t = t.next;
        }
        return bignum.data;
    }
}
```

Or

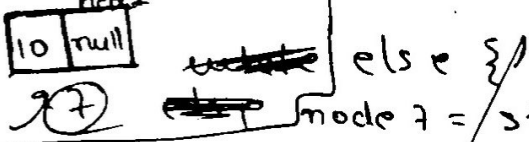



```
void remove_first()
{
    if (start == null)
    {
        SOPL("List is Empty");
    }
    else
    {
        node ptv = start.next;
        start.next = null;
        start = ptv;
    }
}
```

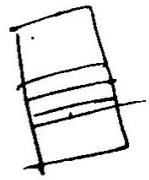
```
Node ptr = start;
start = start.next;
ptr = null;
```

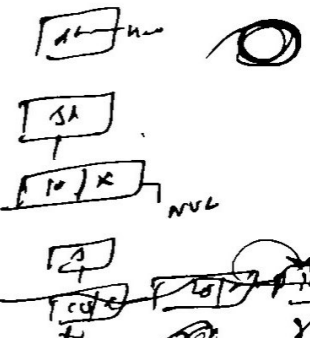
```
void remove_last()
{
    if (start == null)
    { SOPL ("List is Empty");
    }
    else
    {
        node t = start;
        ~~while (t.next != null)~~
        ~~{ t = t.next;~~
        ~~}~~
        while (t.next.next != null)
        { t = t.next;
        }
        t.next = null;
    }
}
```

1) 

2) 


```
else {
    node 7 = start;
    if(7.next == null)
        start = null;
    else
    {   while(7.next.next != null)
        {   7 = 7.next;
        }
        7.next = null;
    }
}
```



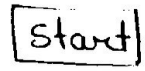


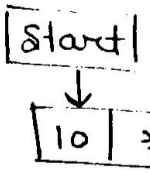
```
void removeLast()
{
    if (start == null)
        sopl("List is empty");

    else if (start.Next == null)
        start = null;

    else
    {   Node t = start;
        Node r = start.Next;
        while(r.Next != null)
        {   t = r;
            r = r.Next;
        }
        t.Next = null;
        r = null;
    }
}
```

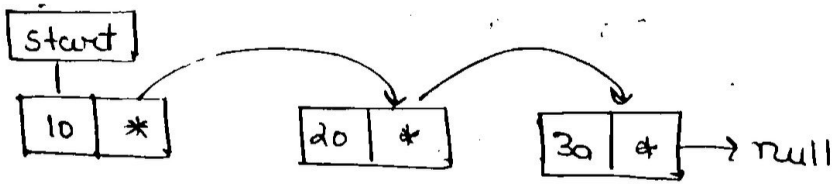
RAM

3 Cases

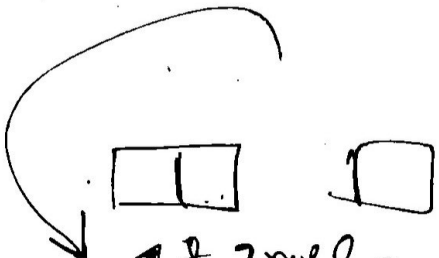
5/2 = 2
6/2 = 3

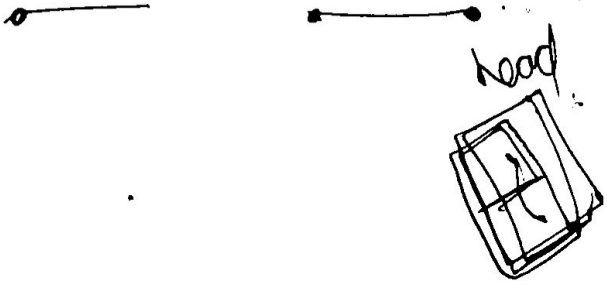


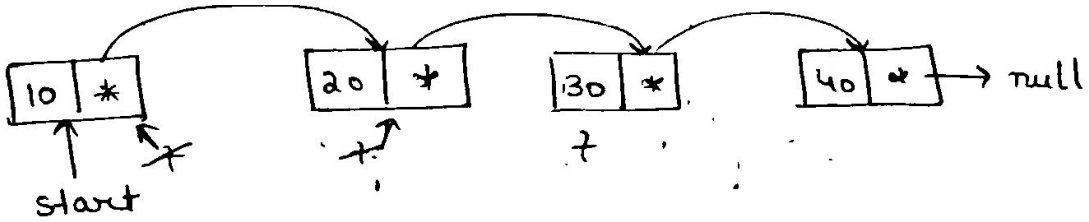





```
int Search (int item)
{
    int p = -1;
    int i = 0;
    node t = start;
    while (t != null)
    {
        if (t.data == item)
        {
            p = i;
            break;
        }
        i++;
        t = t.next;
    }
    return p;
}
```







Search value → 30

p	i
~~1~~	~~0~~
2	1
	2

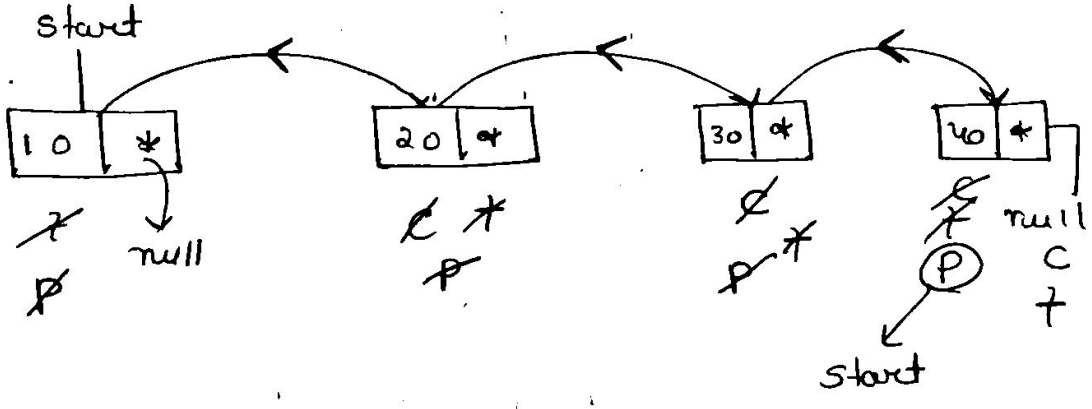
```
void reverse()
{
    node t = start;
    node Previous = null;
    node Current = null;
    while (t != null)
    {
        Current = t.next;
        t.next = Previous;
        Previous = t;
        t = Current;
    }
    start = Previous;
}
```

5/2 = 2
6/2 = 3
(3)

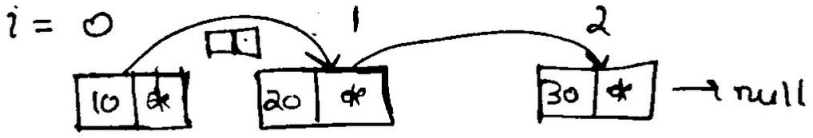
5/2 = 2H
6/2 = 3H (4)
(3)
= 2

5/2 6/2

10/2 = 5
9/2 = 4 (3)



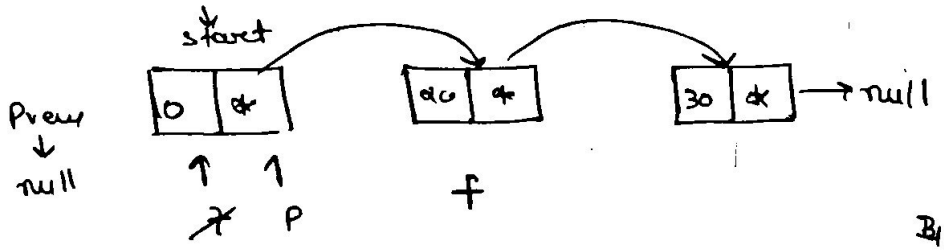
Add At Index



null Prev ~~Current~~ t Current

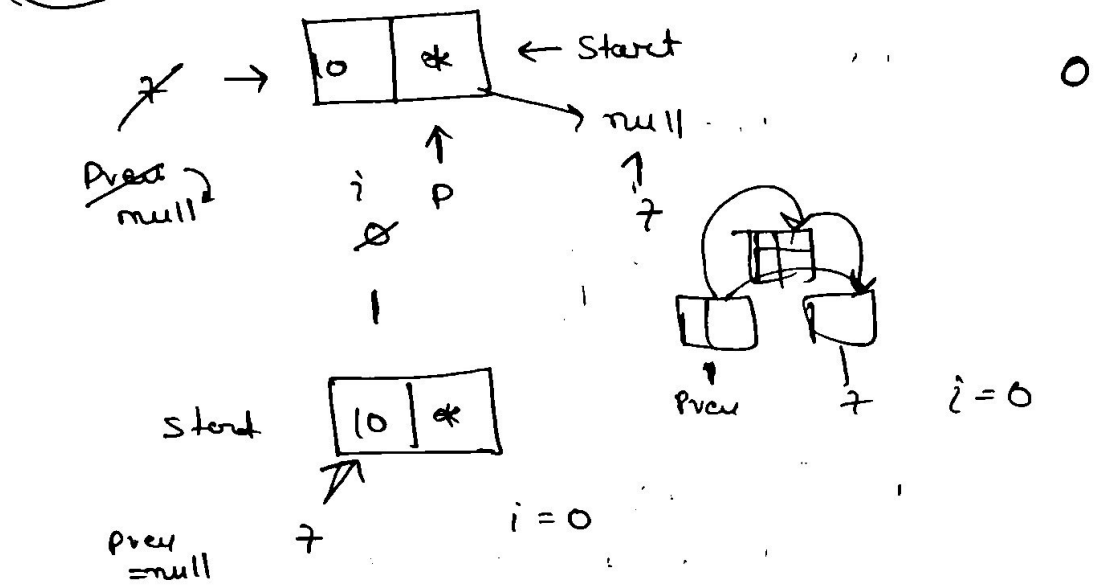
Index out of bound

```
void add at index (int value, int index)
{
    ~~int i = 0;~~
    ~~node t = start;~~
    ~~while (i != index)~~
    ~~{~~
}   node t = start;
    node Previous = null;
    int i = 0;
    while (~~i != index~~) (t != null)
    {
        ~~Previous = t;~~
        ~~t = t next;~~
        ~~i++;~~
        Previous = t
        t = t next
        i++;
        if (i == index)
            break;
    }
    if (~~t == null~~ t == null)
        SOPL ("Not found")
    else
    {
        node ptv = new (value);
        (Prev.next = ptv;)
        ptv.next = t;
    }
}
```

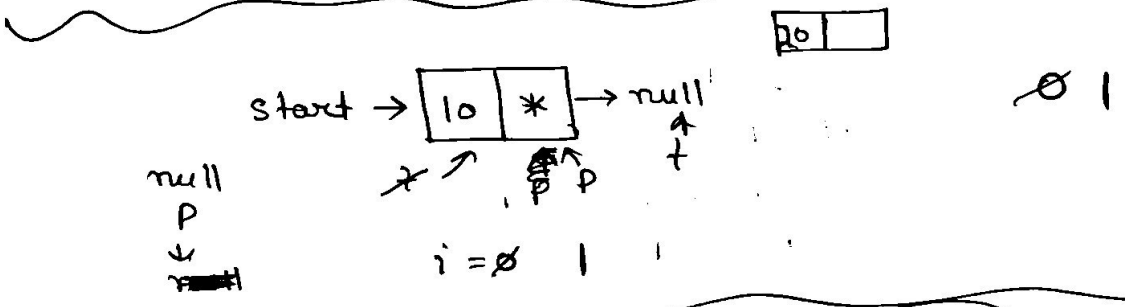

start
10
20
30
null
Prev
null
p
f

i = 0 + 4

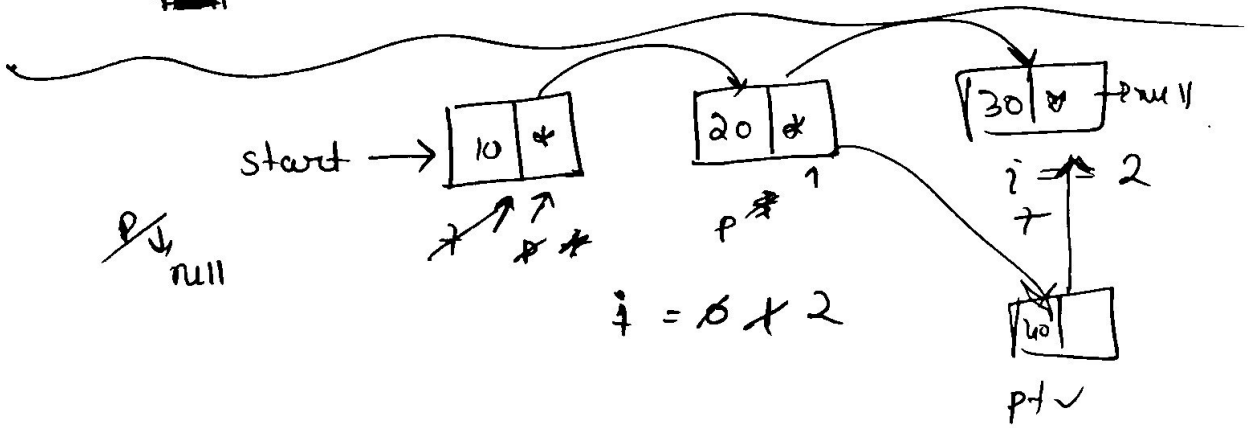
i
0
1


10
Start
null
Prev
null
i
p
1
7
Prev
7
i = 0
Start
10
Prev
=null
7
i = 0

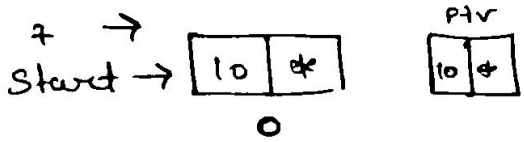
0


20
Start
10
null
null
P
i = 0 1
p
p
t

0 1


Start
10
20
30
null
null
P
i = 2
7
i = 0 + 2
40
p+✓
p
1

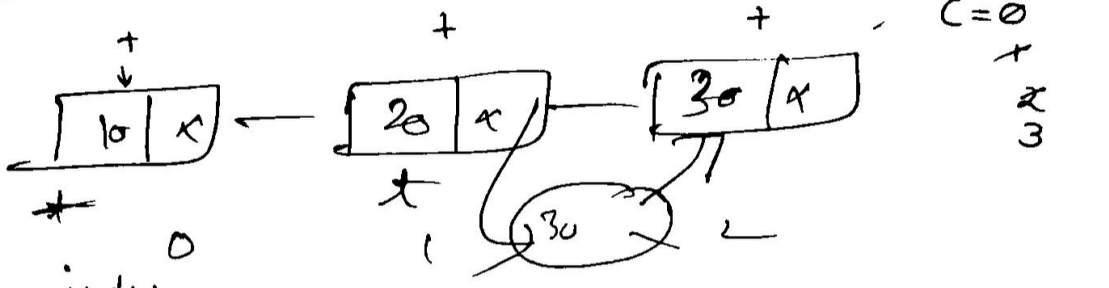
```
void addatIndex (int value, int index)
{   if (start == null)
        SOPL ("empty list");
    else
    {
        node t = start;
        node Previous = null;
        int i=0;
        while (t != null)
        {   if (i == index)
                break;
            Previous = t;
            t = t.next;
            i++;
        }
        if (t == null)
            SOPL ("Index Not found")
        else
        {   node ptr = new (value);
            if (Prev != null)
                Prev.next = ptr;
            ptr.next = t;
        }
    }
}
```



p → null

i

0



index

-1 X

4 X

0 ✓

3 ✓

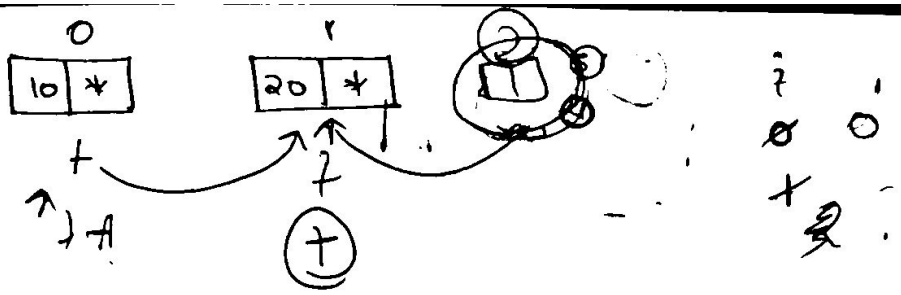
2 ✓

```
void addAtIndex(int pos, int n)
{
if (pos < 0 || pos > count())
    cout ("Can Not Add at given index");
else if (pos == 0)
    addfirst(n);
else if (pos == count())
    addlast(n);
}
```

```
else
{
    Node Ptr = new Node(n);
    Node t = start;
    for( int i=1; i<Pn; i++)
    {
        t = t.New;
    }
                        t.next;
    Ptr.New = ~~Next~~;
    t.New = ptr;
}
}
```

```
class Main
{
public static void main(String K[])
{
SLink obj = new SLink();
    obj.addLast(10);
    obj.addLast(20);
    obj.addLast(30);
System.out.println("List = ");
    obj.disp();

    obj.addfirst(100);
    obj.addfirst(90);
System.out.println("List = ");
    obj.disp();
    obj.addLast(300);
    obj.addLast(400);
    obj.addfirst(80);
    System.out.println("List = ");
    obj.disp();
}
}
```

```
int getAtIndex (int index)
{
    int value = -1
    if (index < 0 || index > (count()) - 1)
    {
        SOPL ("Can not [illegible]
            get [illegible] at given index");
        return value;
    }
    else
    {
        node t = start;
        for (int i = 0; i < (count(); i++)
        {
            if (i == index)
            {
                value = t.data;
                return value;
            }
            else
            {
                t = t.next;
            }
        }
    }
}
```




```
void removeAtIndex(int index)
{
    if (index < 0 || index > (count()-1))
    {   SOPL("Cannot Remove");
    }
    else if (index == 0)
        removeFirst();
    else if (index == (count()-1))
        removeLast();
    else
    {   node t = start;
        for (int i = 0; i < index; i++)
        {   t = t.next;
        }
        t.next = t.next.next;
    }
}
```

index
2