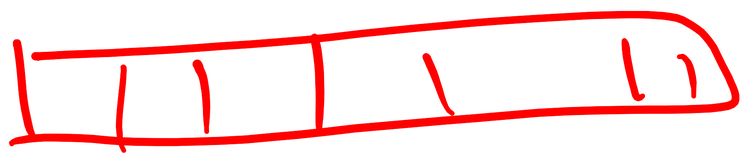
Dynamic Memory

Allocation

int a[100]



Static	Dynamic
→ Compile time	→ on the fly (runtime)
→ auto destroy (scope)	→ Programmer has to destroy / clear.
→ Small	→ Large.
→ 'Static' allocation/ memory	→ 'Heap' Memory

eg int a = 10 ;

int b[10] ;

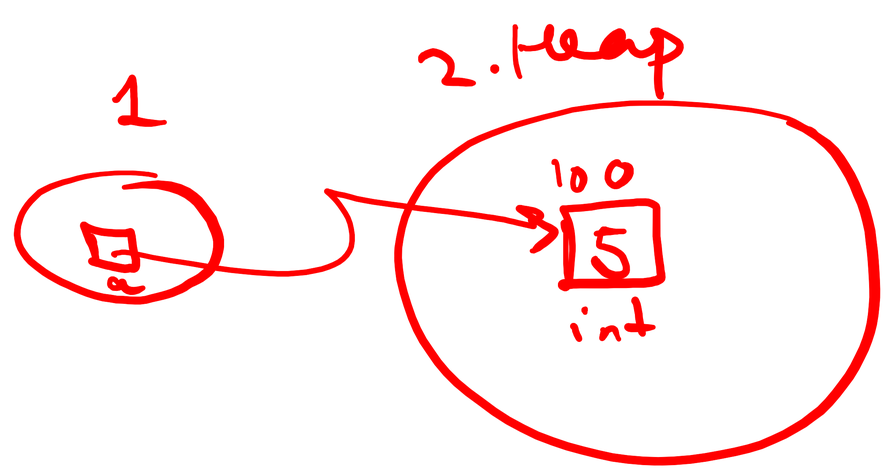
int *ptr

C

int * ptr

=(int *)malloc(40) ;

Size in Bytes for allocation



C++ new

C malloc

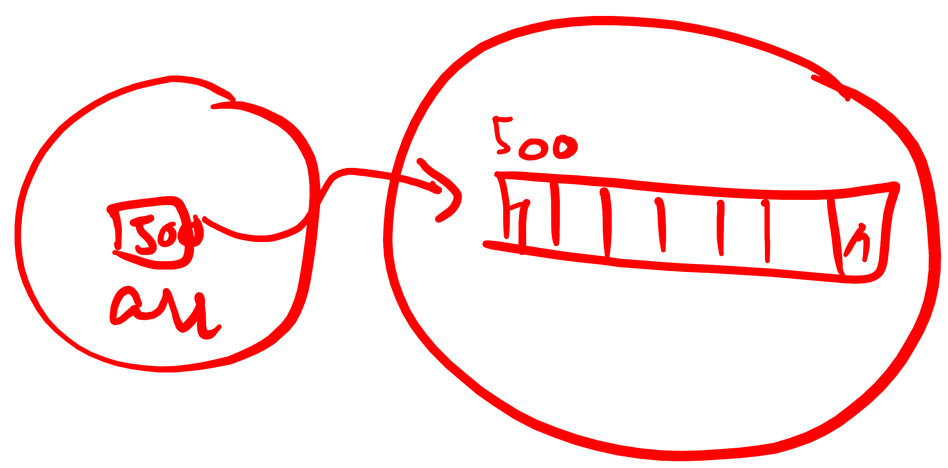
int *a ;

a = new int ;

;*a = 5.

ARRAY

int *arr = new int[10];



arr[0] = 1 ;

arr[9] = 11 ;

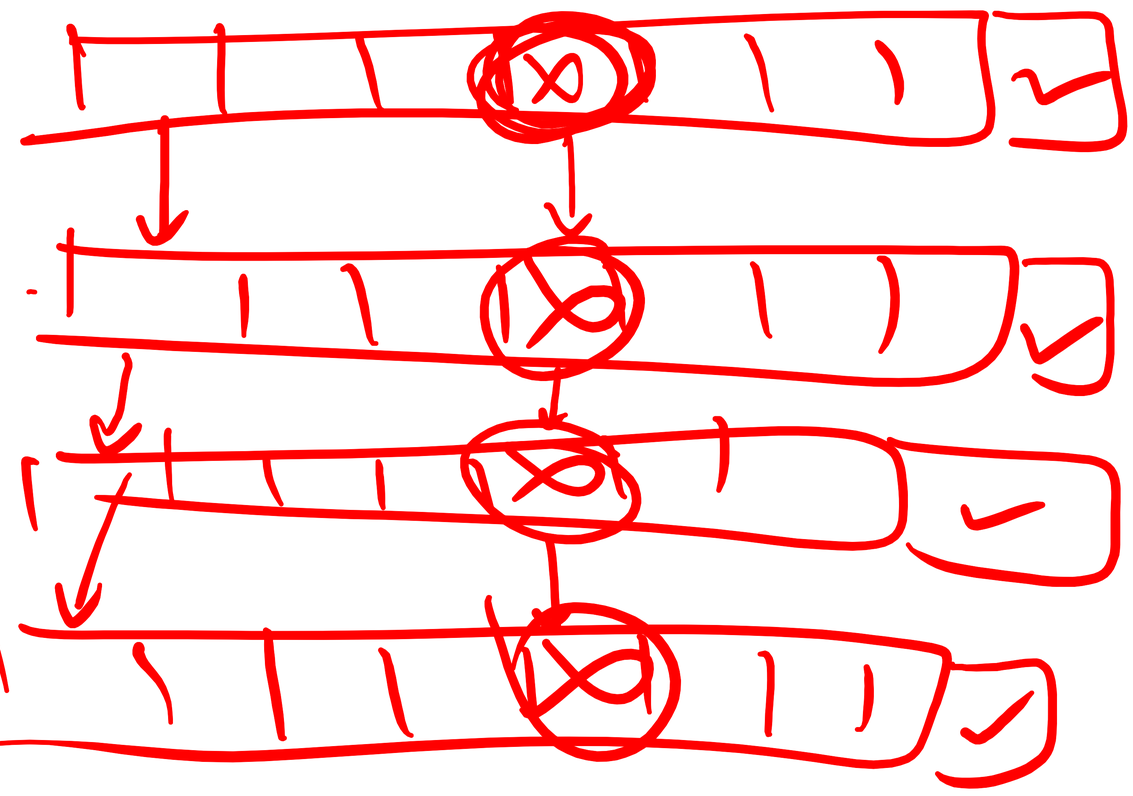
*(arr+0) = 9 ;

*(arr+5) = 6 ;

Classes & Objects

N students

Name
Marks
Roll No
Age



Student Class / Struct

mark
Roll
Age
Name

memory

Name
Roll
Age

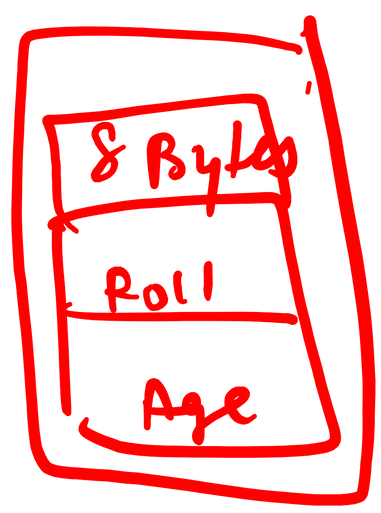
Class
(Blueprint)

```
Class Student{
  public:
     int roll_no;
     int age;
};
```

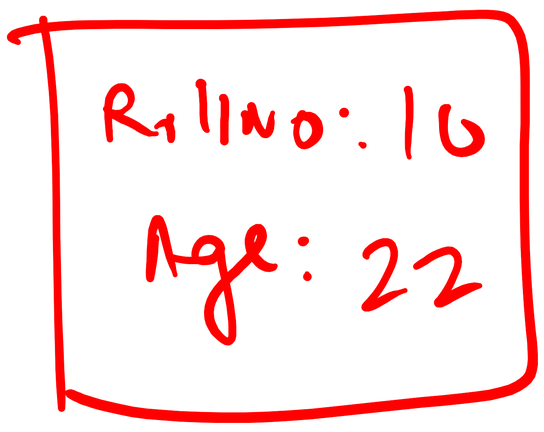
Name: Prateek
Roll : 101
Age : 22

Object
(Memory)

Student S;



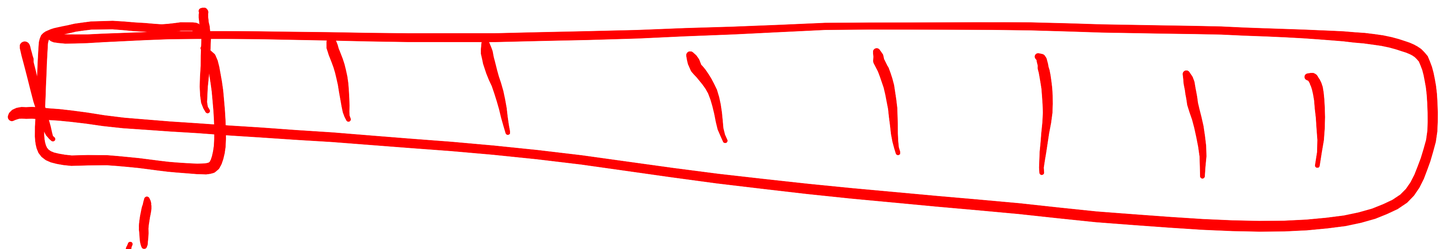
int x;



S.rollNO = 10

S.Age = 22

Student S[100]; // Array of objects



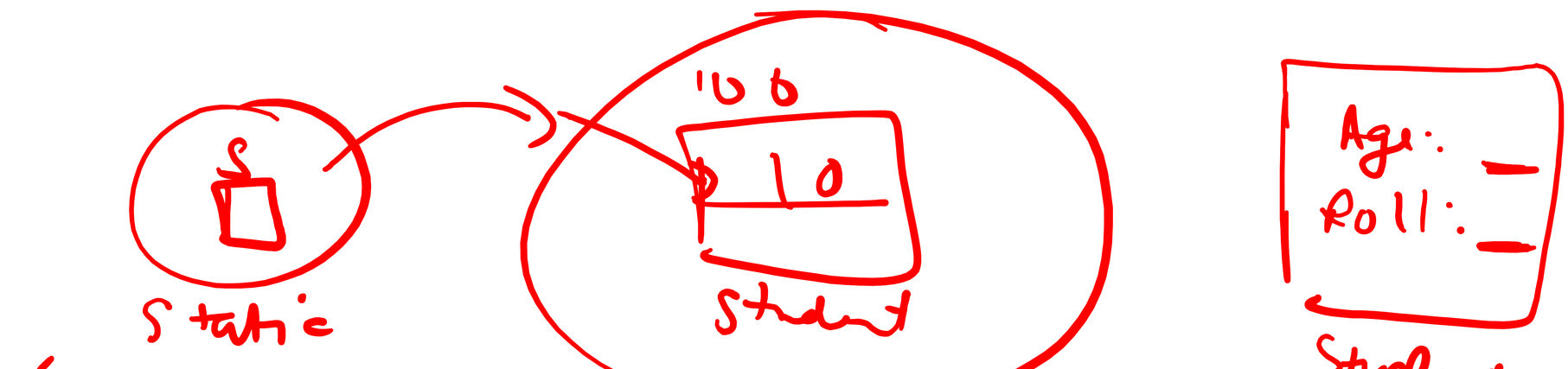
Each bucket is an object of Student class.

S[0].rollNo = 1.

S[0].age = 20;

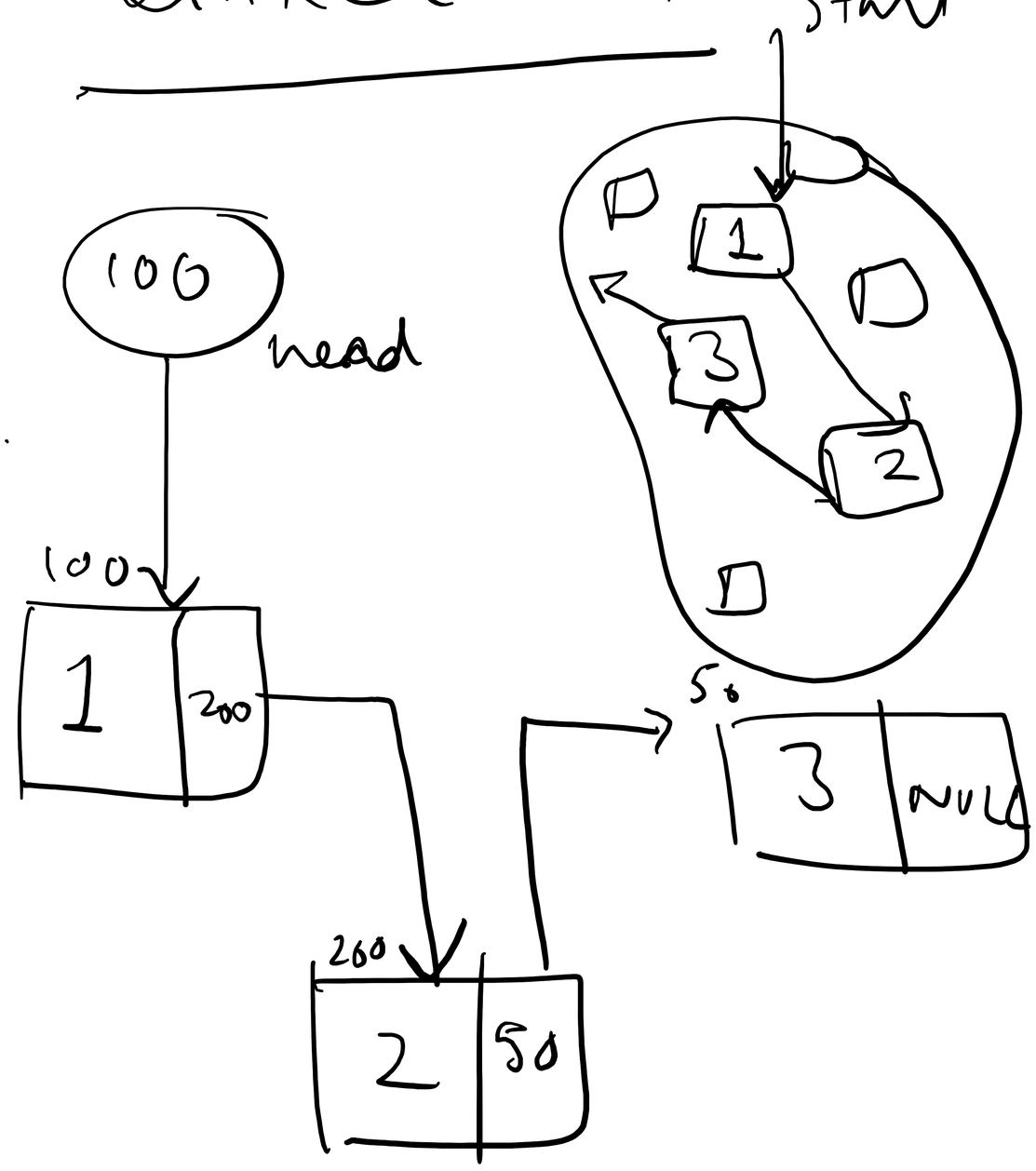
Dynamic Object

Student * S = new Student;



Same ← { (*S).rollNO = 10;
S → rollNO = 10;
S → age = 22;

Linked List

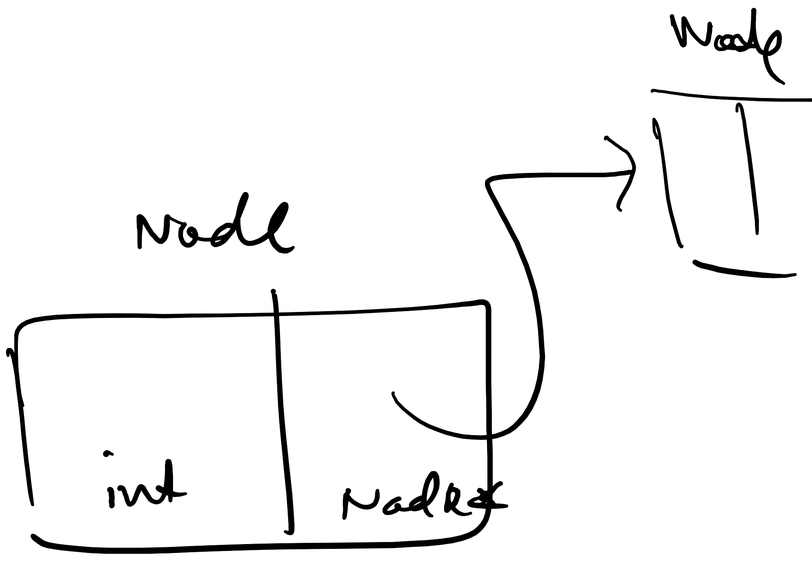


ARRAY

← 10 →
Boxes

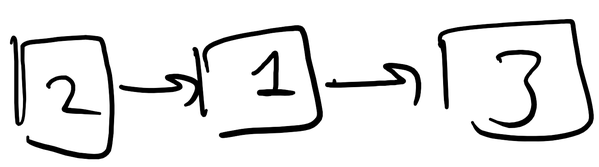
→ fixed ✓
Expand ✗
Contract ✗

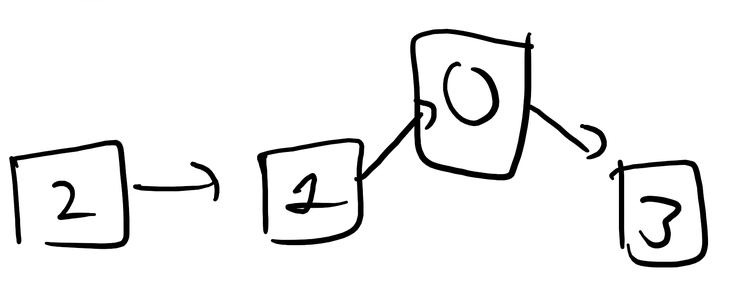
```
class Node {
    int data;
    Node next;
}
```



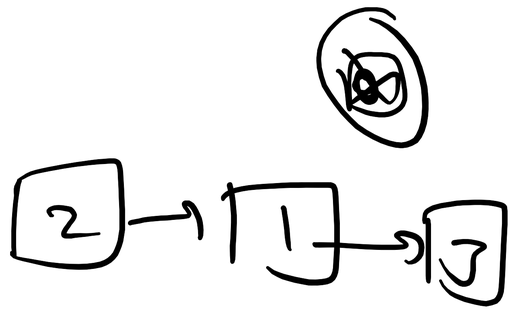
Linked List operations

1) Insert Nodes.
   - → head
   - → tail
   - → middle.



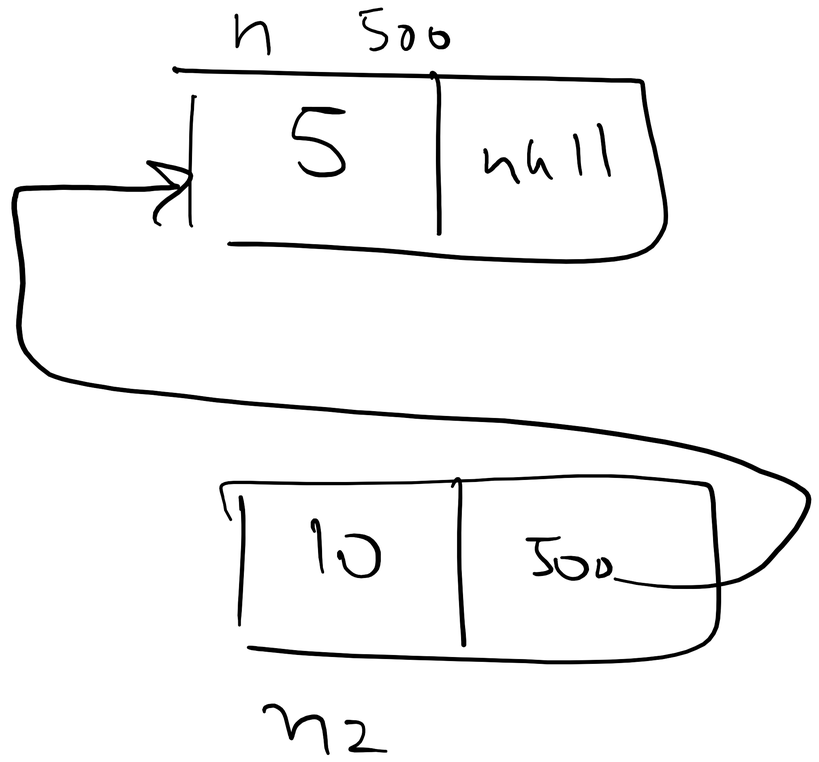


2) Delete Node
   - → head
   - → tail
   - → middle.



2) Searching

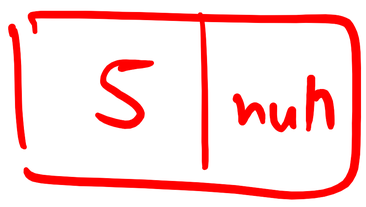
```
Node n;
n.data = 5;
n.next = NULL;
```




```
Node n2;
n2.data = 10;
n2.next = &n;
```

```
Node * makeNode(int data){
    Node n;
    N.data = data;
    N.next = NULL;
    return &n;
}
```

-5



Auto Destroy

Dynamic Allocation — Solution

```
Node * makeNode(int data){
    node * n = new Node;
    n->data = data;
    n->next = NULL;
```

Dynamic

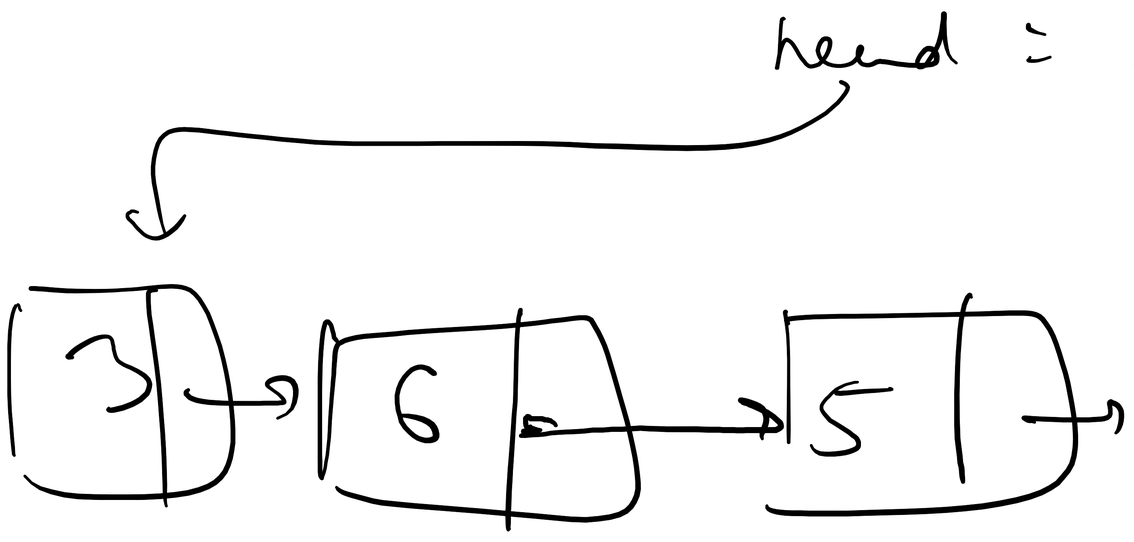
return n;

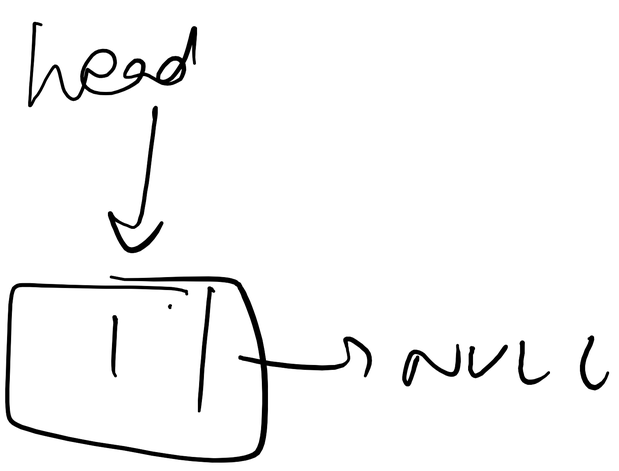
}

Memory (Heap)

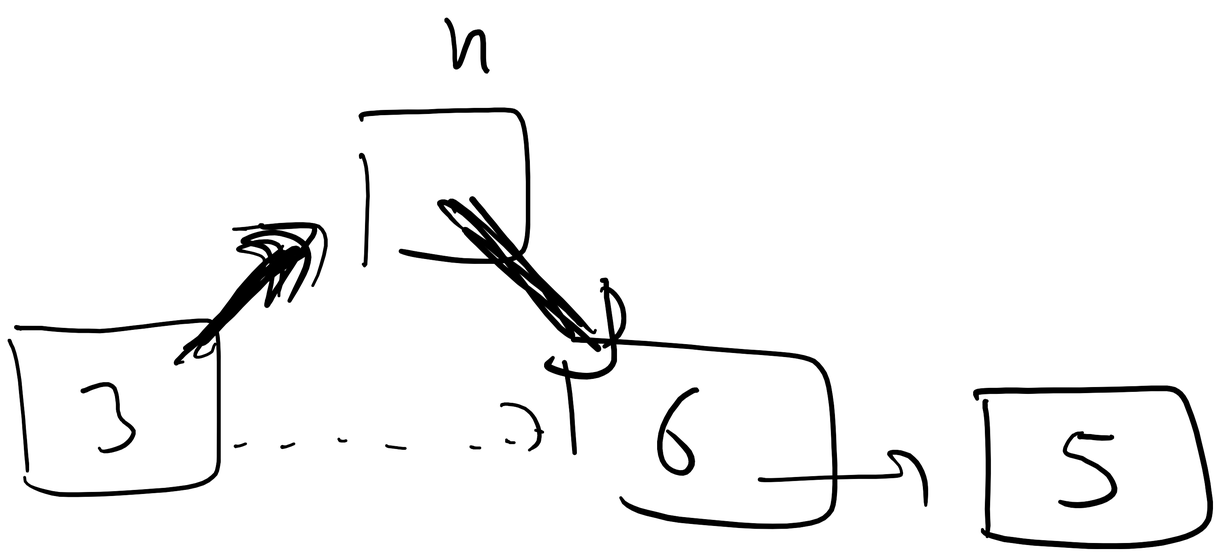
CODE

① Insert At front





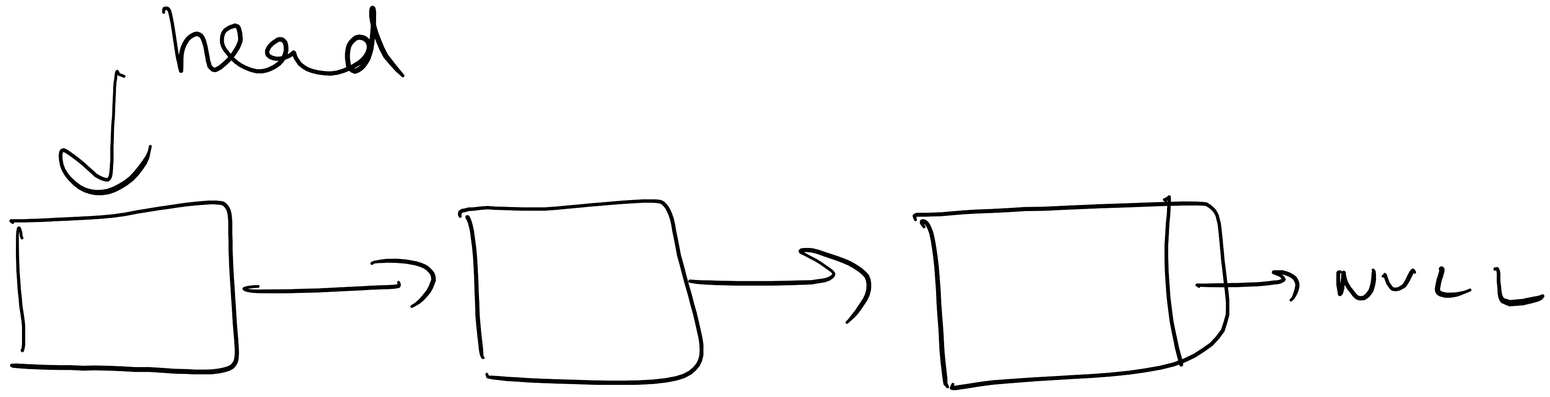
n → next = head;
head = n;



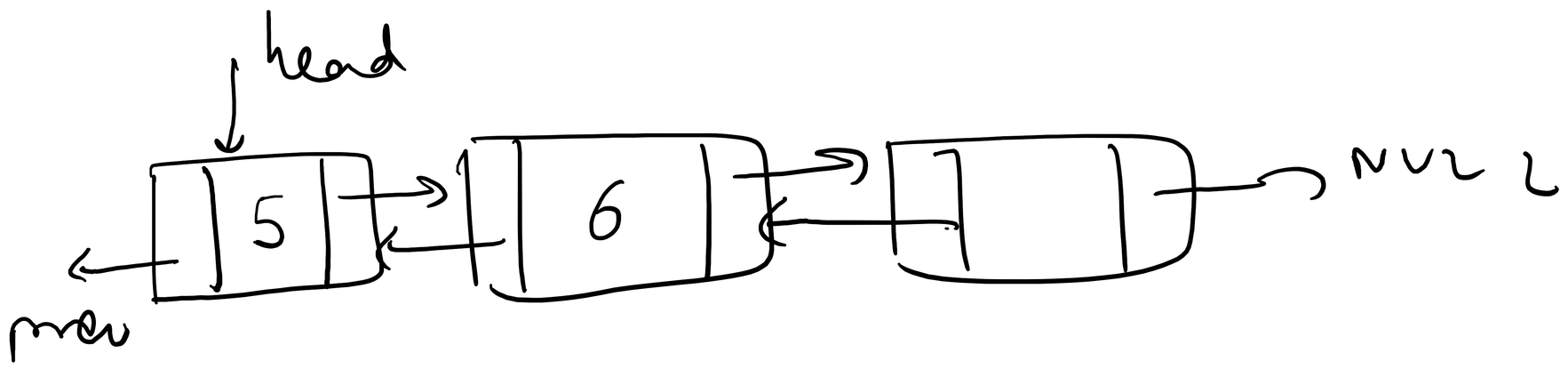
Insert at position.

n → next = temp → next
temp → next = n.

Singly LL

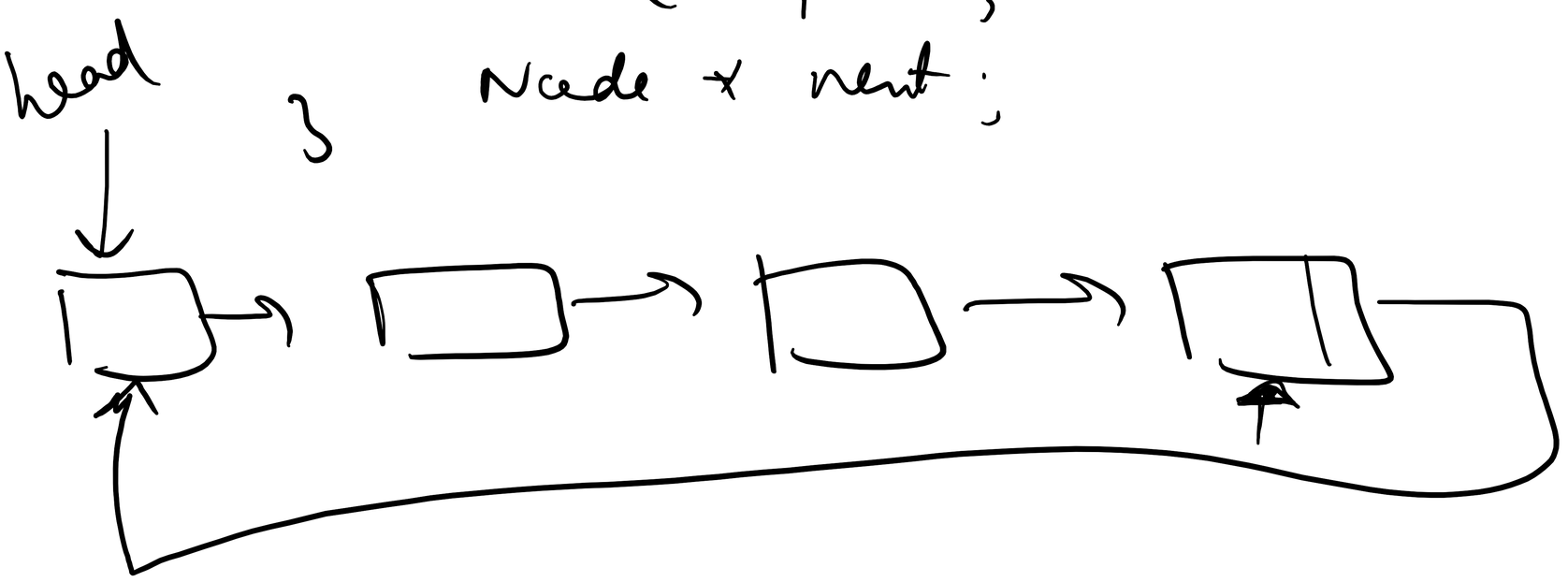


Doubly LL



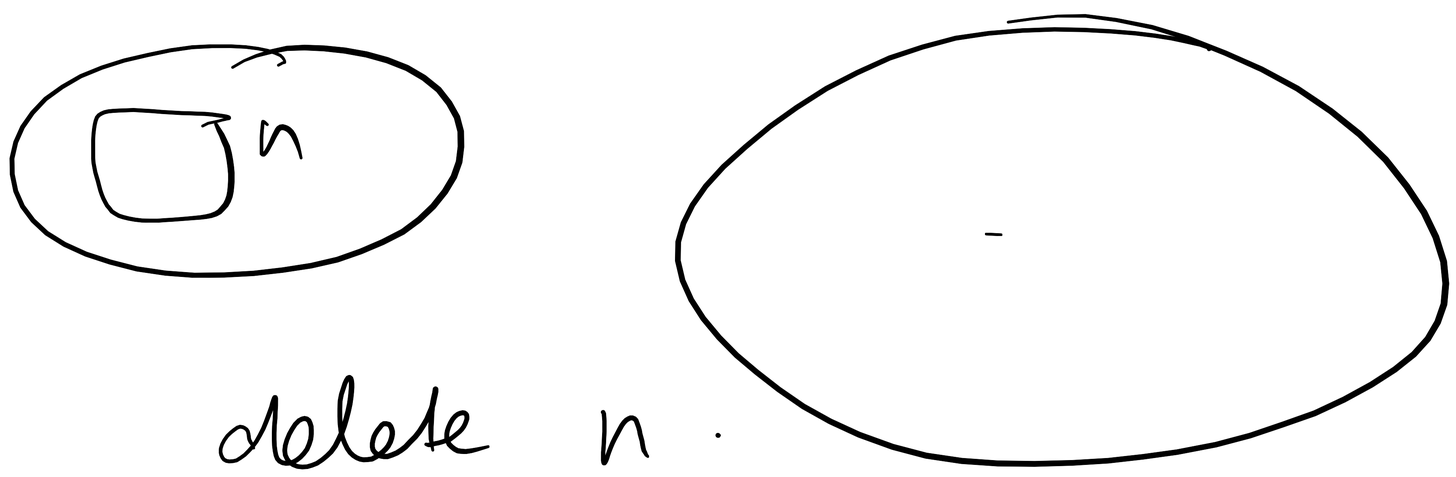

```
Struct Node {
    int data;
    Node * prev;
    Node * next;
}
```

Circular LL

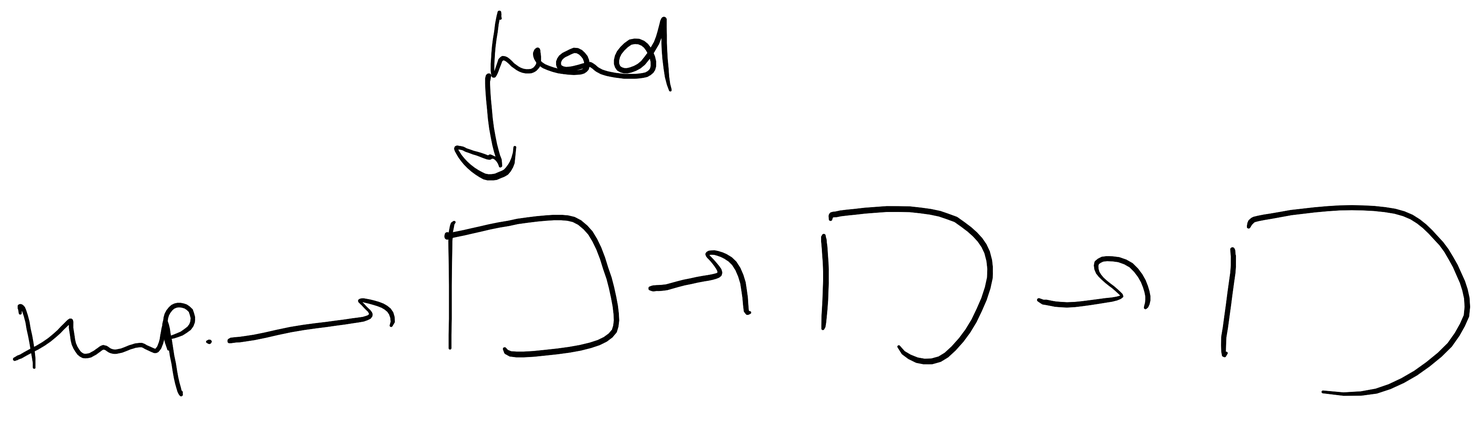



```
temp = head
while (temp -> next != head)
{
    cout<< temp->data;
    temp = temp->next;
}
cout<<temp->data;
```

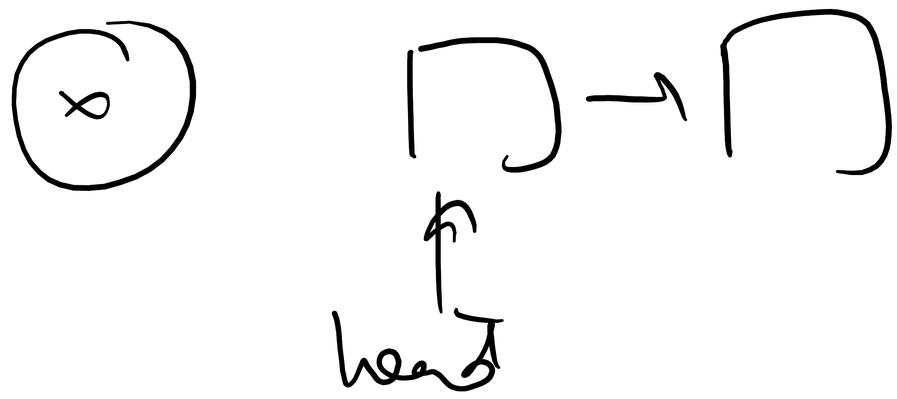
```
Node * n = new Node
```




```
delete n;
n = NULL;
```

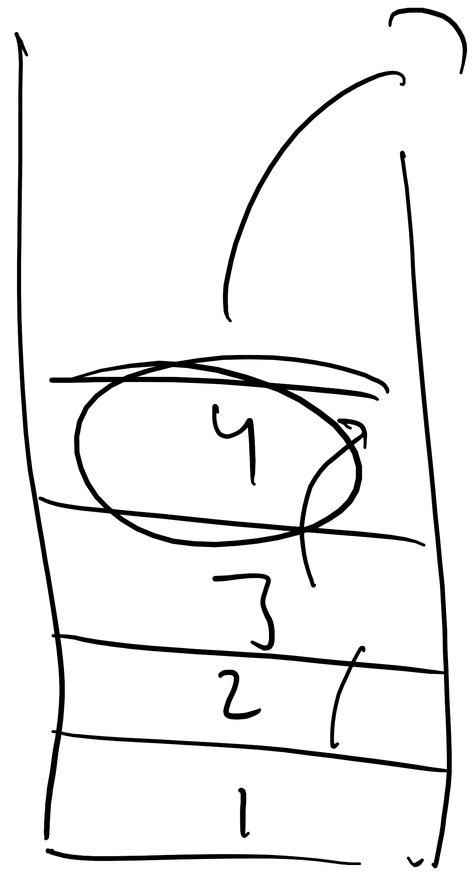



```
Node * temp = head;
head = head->next;
delete temp;
```



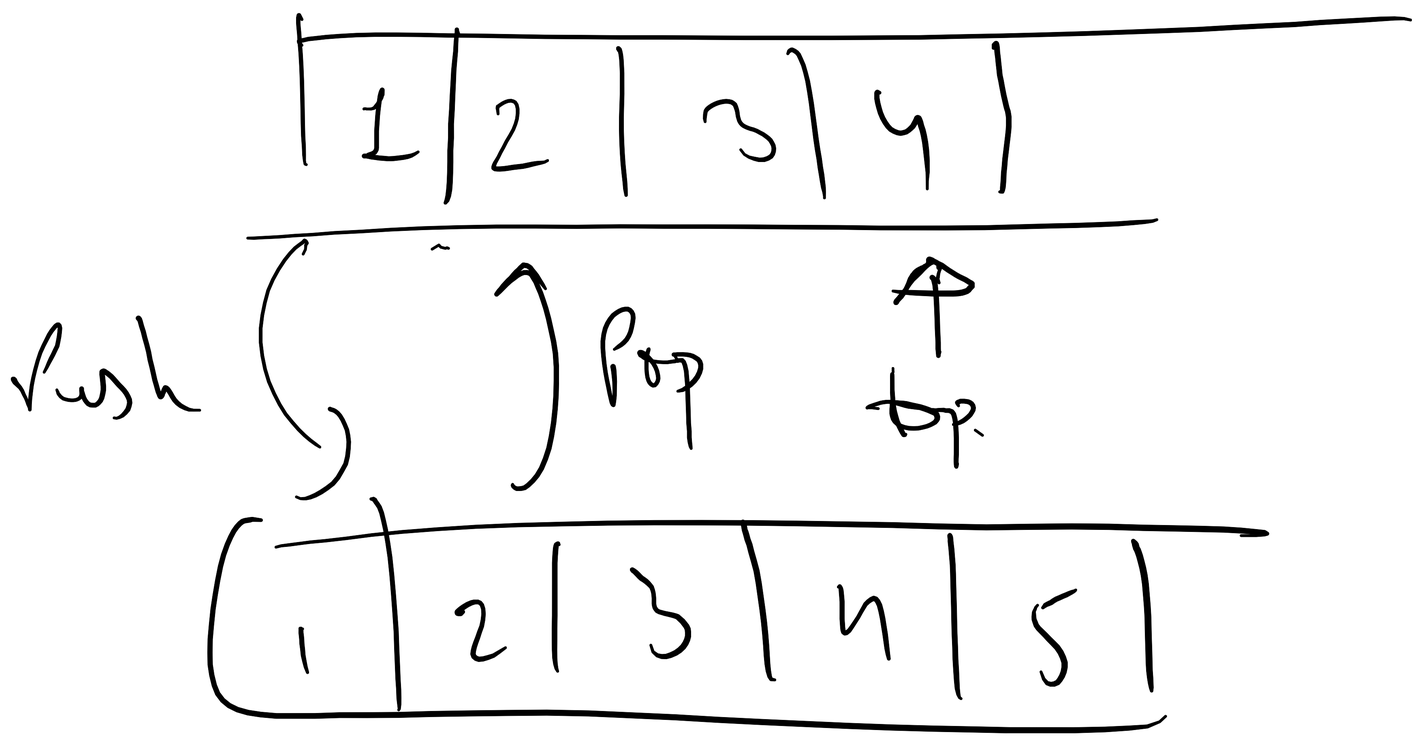
Stacks

- Push
- Pop
- Top.



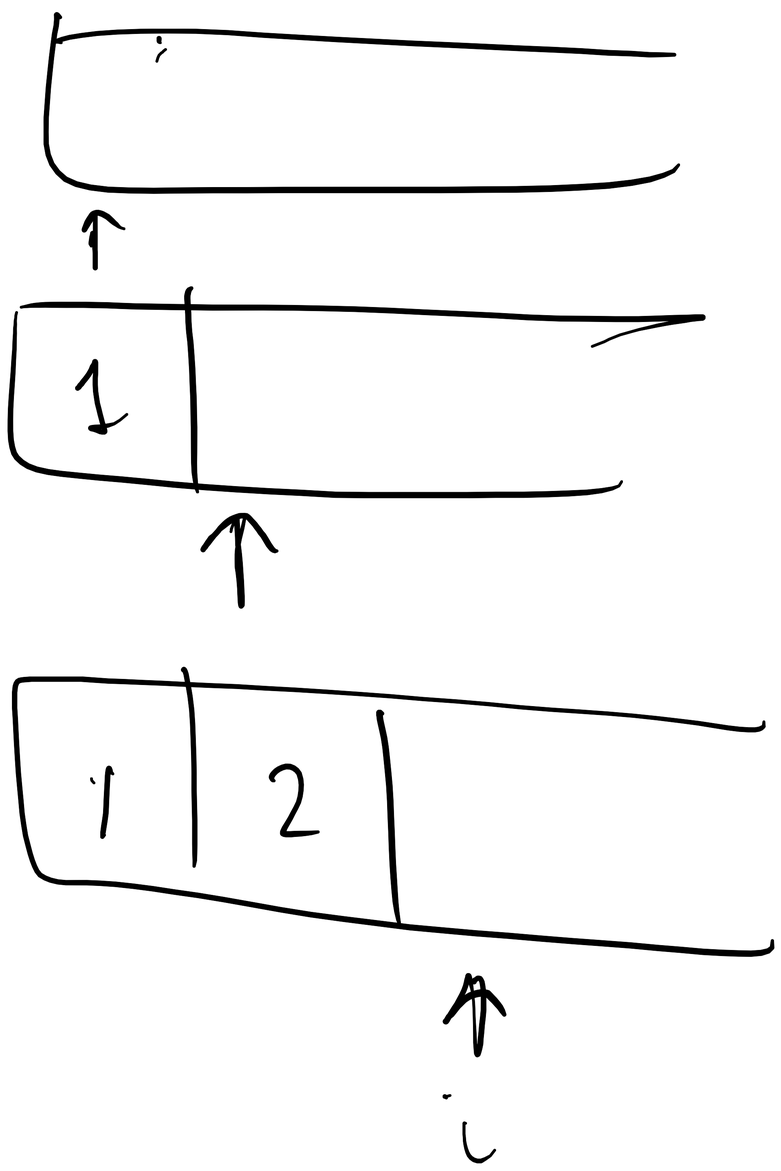
Implement

- Array
- LL



Push

top = -1 ;



push(data)

int pop()

int top(){

return a[i-1];

}

insert At Front