Sunday Live → OOPS → Pratical OOPs
&
PW Skills

Linked List

Part - 1

User defined data type ✓

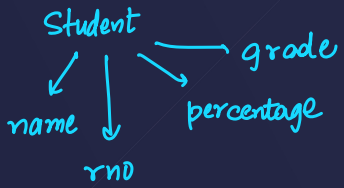
Raghav Garg

Revision of OOPS

Defining a class

Declaring and Initialising an object

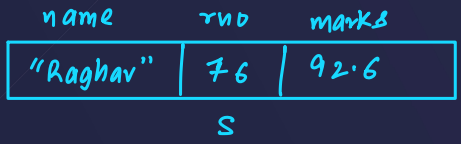
```
class Student {
public:
    int rno;
    float percent;
    string name;
};
```



raghav → name
→ rno
→ percen


```
string name = "Raghav"
int rno = 76
float per = 92.6
```

```cpp
void change(Student s){
    s.name = "Harsh";
}
int main(){
    Student s("Raghav",76,92.6);
    cout<<s.name<<endl;
    change(s);
    cout<<s.name<<endl;
}
```



Output:

- Raghav
-

int x = 4;

change(x)

Revision of OOPS

Constructors & making parameterised constructors

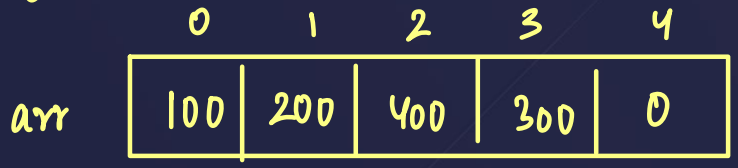
Arrays → Linear D.S. → multiple dabbe bana sakte hai

→ list

Limitations

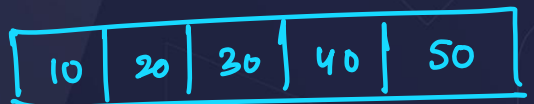
1) Fixed Size → vector

↓ problem

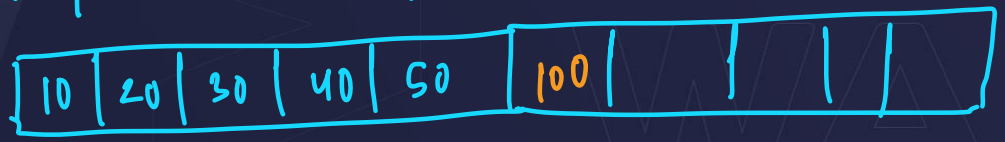


arr[2] = 500; // O(1) T.C

vector<int> arr(5);



v. push_back(100)



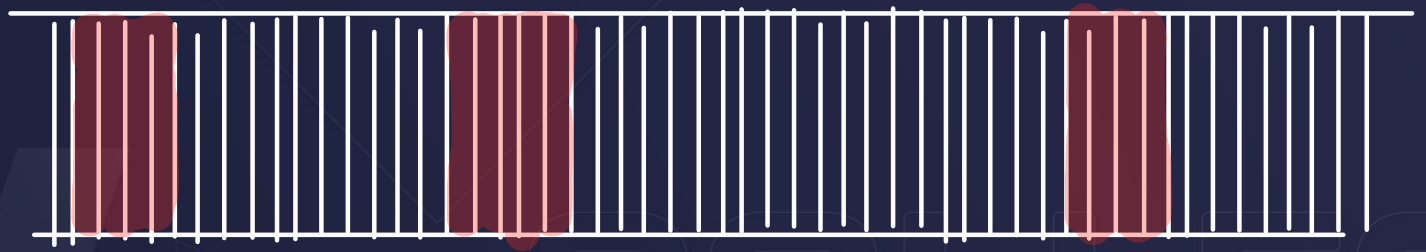
Arrays

Limitations

2) Contiguous memory allocation :

int arr[4];
↓
16byte

int arr[6]; → 6 × 4 = 24 bytes



11 free bytes

20 bytes

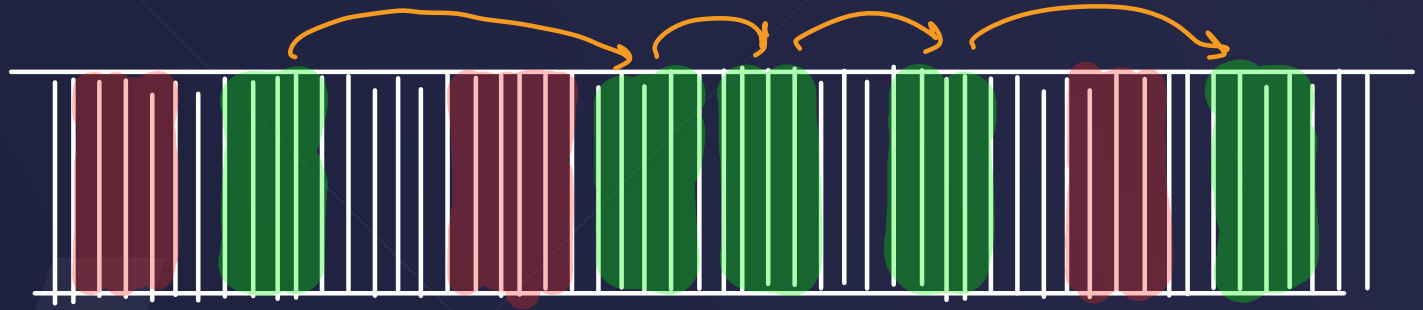
8 bytes

Arrays

Need for a new linear data structure

Introduction to Linked List

Idea of linking two non-contiguous memory locations (nodes)

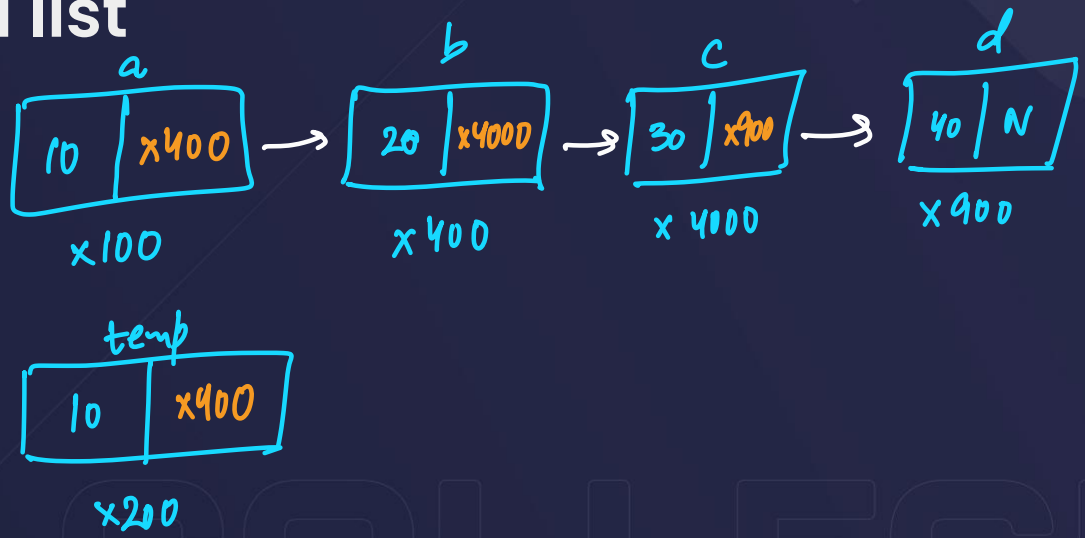


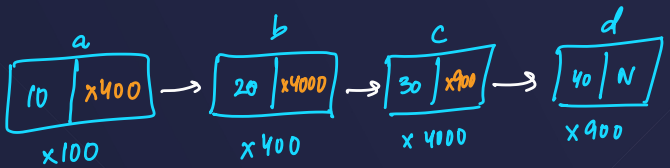
5 size ki lk list

Introduction to Linked List

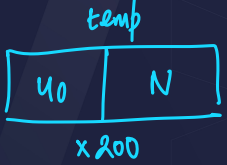
```
class Node{ // Linked List Node
public:
    int val;
    Node* next;
    Node(int val){
        this->val = val;
        this->next = NULL;
    }
};
int main(){
    // 10 20 30 40
    Node a(10);
    Node b(20);
    Node c(30);
    Node d(40);
    // forming ll
    a.next = &b;
    b.next = &c;
    c.next = &d;
}
```

ed list



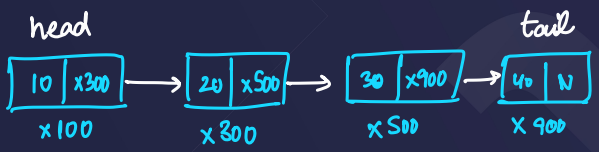



```
Node temp = a;
while(temp.next!=NULL){
    cout<<temp.val<<" ";
    temp = *(temp.next);
}
```



- 10 20 30

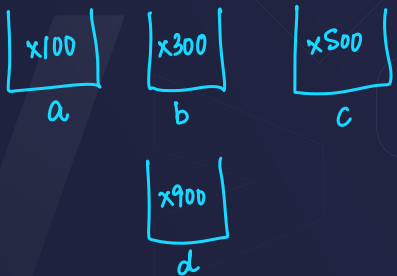
```
Node* a = new Node(10);
Node* b = new Node(20);
Node* c = new Node(30);
Node* d = new Node(40);
a->next = b;
b->next = c;
c->next = d;
```

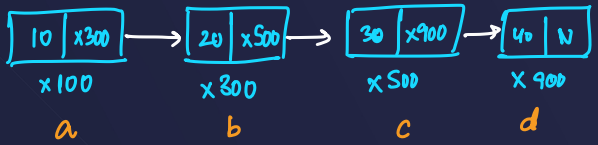


a → next

↓

(*a). next



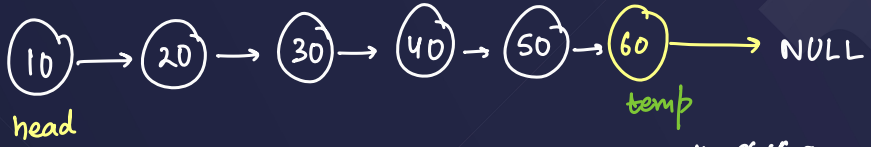



```
Node* temp = a;
while(temp!=NULL){
    cout<<temp->val<<" ";
    temp = temp->next;
}
```

Output

10 20 30 40

Representation :




```
n = 0;
Node* temp = head
while (temp != null) {
    temp = temp -> next;
    n++;
}
```

Introduction to Linked List

Does linked list overcomes the limitations of arrays?

- Unlimited Size
- Continuous memory allocation

Implementation

Node class

```
class Node{
    int val;
    Node* next;
};
```

With Parameterised constructor

```
class Node{
    int val;
    Node* next;
    Node(int val){
        this->val = val;
        this->next = NULL;
    }
};
```

Implementation

Linking nodes to form a linked list

Displaying a Linked List ✓

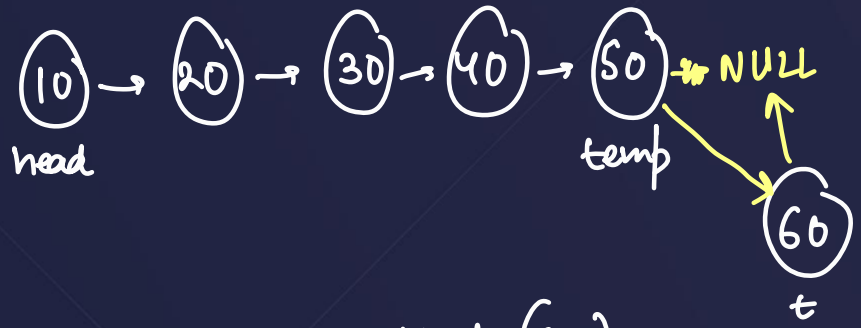
& size

```
Node* temp = head;
while (temp != NULL){
    cout << temp->val;
    temp = temp->next;
}
```

T.C. = O(n)

S.C. = O(1)

Displaying a Linked List




```
while (temp→next != NULL)
{
    temp = temp→next;
}
temp→next = t;
```

```
Node *t = new Node(60);
```

T.C. = O(n)

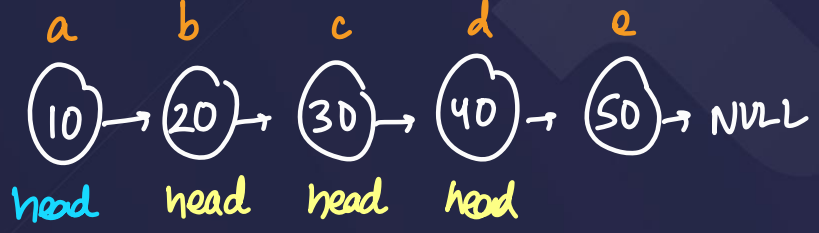
Displaying a Linked List

Can we do it recursively?

```
void display (Node* head){
    if(head == NULL) return;
    cout << head -> val;
    display (head -> next);
}
```

Output

10 20 30 40 50



~~display(N)~~	~~head = NULL~~
~~display(50)~~	head = 50
~~display(40)~~	head = 40
~~display(30)~~	head = 30
~~display(20)~~	head = 20
~~display(10)~~	head = 10

T.C. = O(n)

S.C. = O(n)

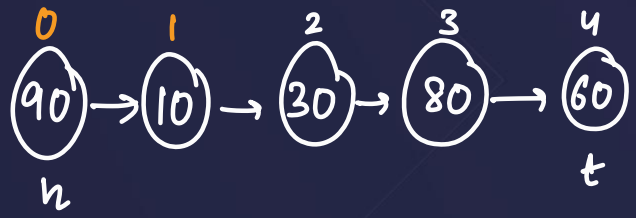
What will this function do?

```
void display(Node* head) {
    if(head == NULL) return;
    display(head->next);
    cout<<head->val<< " ";
}
```

```
void display (Node* head){
    if(head == NULL) return;
    cout << head -> val ;
    display (head -> next);
}
```

(a) Print all the elements of the linked list.
(b) Print all the elements except last one.
(c) Print alternate nodes of linked list
(d) Print all the nodes in reverse order

Implementation



Linked List class

```
class LinkedList {
    Node* head;
    Node* tail;
    int size = 0;
}
```

```
LinkedList ll;
ll.add(60);
ll.addAtHead(90);
ll.deleteAt(2);
ll.insertAt(3,80);
```

size = 0

Display method

Implement display method to print all the elements

Length method

Implement a method to find out the length of a Linked List (Iterative and Recursive)

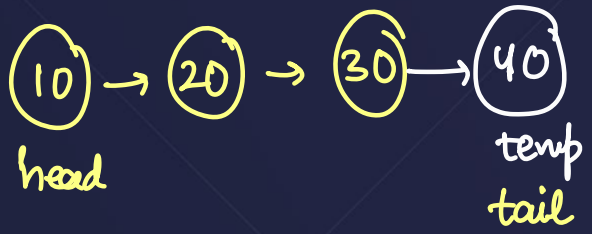
InsertAtEnd ~~method~~ function

T.C. = O(1)

Implement a method to insert a node at the end of a linked list.

Case-1 : Size > 0

ll.insertAtEnd(40);




```
Node* temp = new Node(40);
tail->next = temp
tail = temp;
size++;
```

Case-2 : Size = 0

ll.insertAtEnd(10)


```
Node* temp = new Node(10);
head = tail = temp
size++;
```

InsertAtBeginning ~~method~~ function

Implement a method to insert a node at the start of a linked list.

Case - 1 : Size > 0

```
ll. insertAtHead(50);
Node* temp = new Node(50);
temp->next = head;
head = temp;
size++;
```

Case - 2 : Size = 0

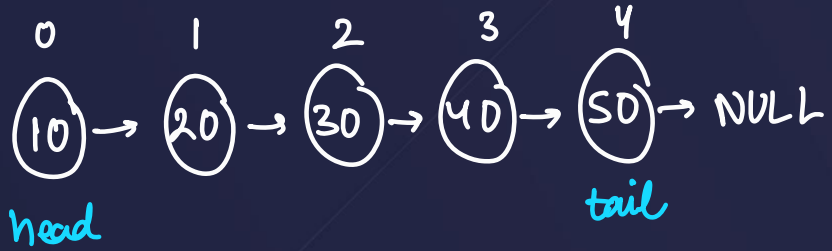
```
NULL
head
tail
Node* temp = new Node(50);
head = tail = temp;
size++;
```

Insert method

insertAt(int idx, int val);

T.C. = O(n)

Implement a method to insert a node at any given index.



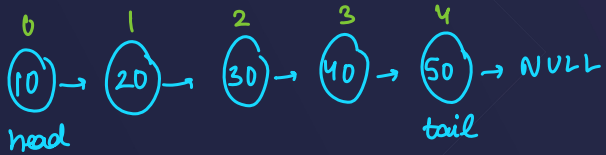
size = 5

if (idx == 0) → ll.insertAtHead

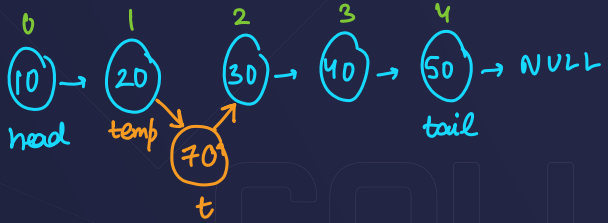
if (idx == size) → ll.insertAtTail

if (idx < 0 || idx > size) → invalid

else



ll.insertAt(2, 70);



1) Traverse temp to idx-1

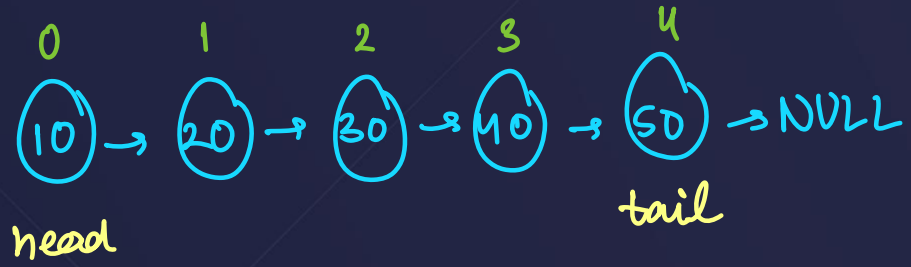
2) t → next = temp → next

3) temp → next = t

getElement method

T.C. = O(n)

Implement a method to return the element at any given index of the linked list.

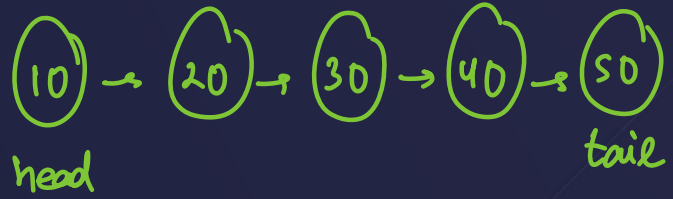


if(idx<0 || idx>=size) invalid else

if(idx==0) return head->val

if(idx==size-1) return tail->val

An evident limitation of Linked List




```
cout << arr[4];

arr[4] = 8;
```

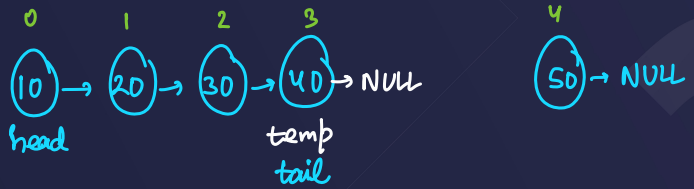
Delete At Head :

if (size == 0) → List is Empty

if (size == 1) head = tail = NULL; NOT REQD.
size--

if (size >= 1)
head = head.next;
size--;

delete At Tail :

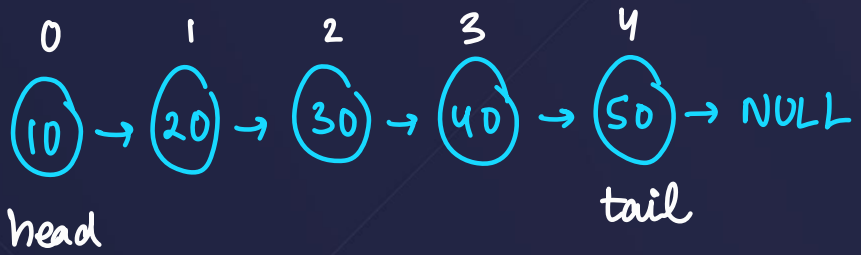


Steps :

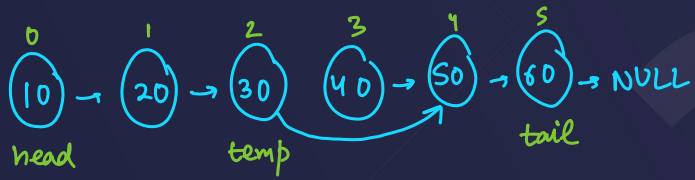
1) traverse temp such that
   temp → next = tail
2) temp → next = NULL
3) tail = temp

deleteAtIndex method

Implement a function to delete a node at a given index




```
if (idx < 0 || idx >= size) → invalid index
else if (idx == 0) deleteAtHead();
else if (idx == size-1) deleteAtTail();
else {
    ——————
    }
```



deleteAtIdx(3);

Steps:

1) traverse temp at idx-1

2) temp → next = temp → next → next

Next Lecture

More interesting things about Linked List

Homework : Implement LL class but , with only head & size

Next Lecture

More interesting things about Linked List