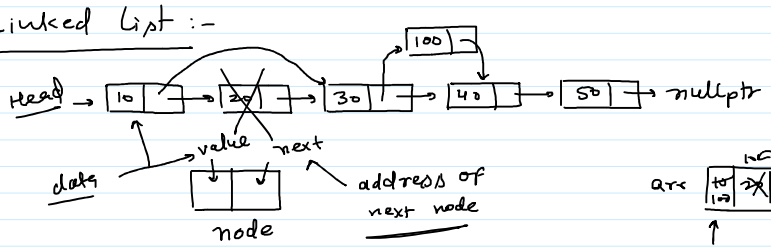
Day 15

03 April 2025 08:33 AM

Linked List :-



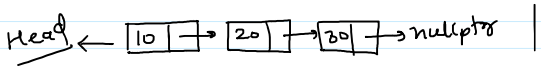
arr 10 20 30 40 50

Add delete → shifting time taking

Types of Linked List :-

1. Singly Linked List
2. Doubly Linked List
3. Circular → Singly
   → Doubly

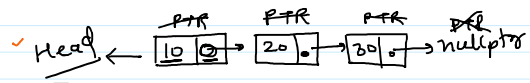
1. Singly Linked List :-



operations →

1. Add First
2. Add Last
3. Add before
4. Add After
5. Del first
6. Del Last
7. Del node

Traverse

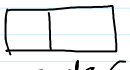


pointer

```
PTR = Head
while (PTR != nullptr)
{
    print(PTR -> val);
    PTR = PTR -> next;
}
```

output

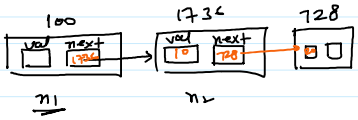
10 20 30



node (object)

```
class node
{ public:
    int val;
    node *next;
};
```

← Self Reference (pointer)




```
node n1;
node n2;
n1.next = &n2;
```

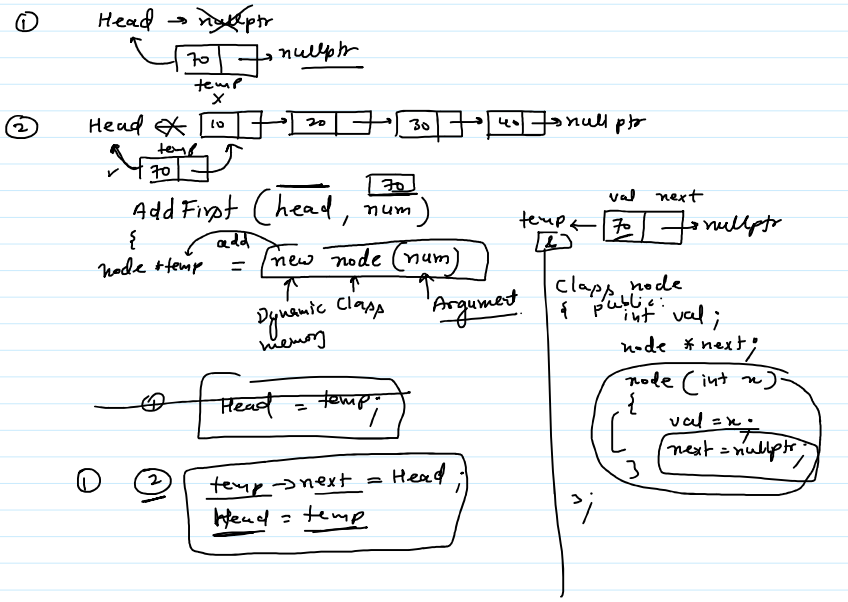
n1.next → val = 10;

n1.next → next → val = 20;

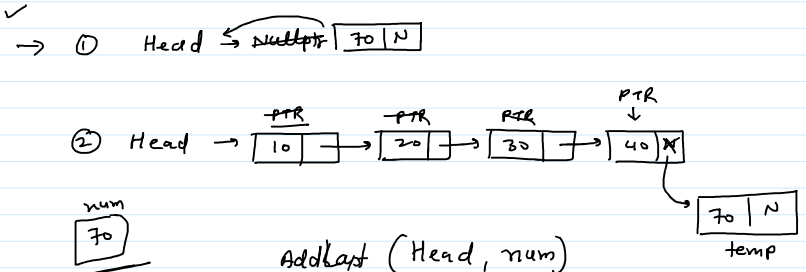
object

1. Index not available
2. Continious memory allocation

Add First :-

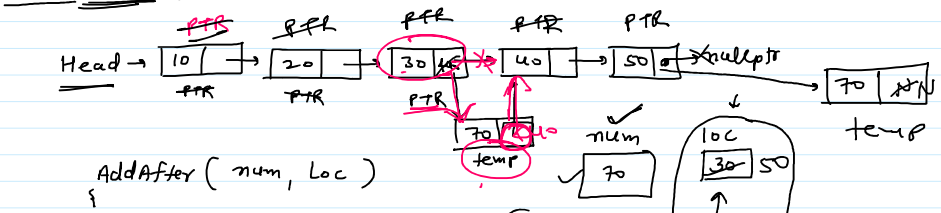


Add Last

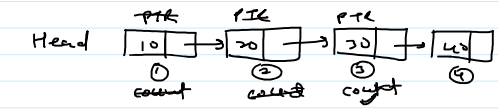



```
AddLast (Head, num)
{
    node *temp = new node (num);
①   Head = temp;
②   PTR = Head
    while (PTR->next != nullptr)
    {
        PTR = PTR->next;
    }
⇒   PTR->next = temp
```

Add After :→

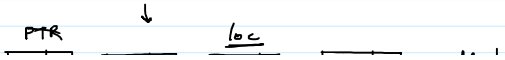



```
AddAfter ( num, Loc )
{
    PTR = head
    while (PTR != nullptr && PTR->value != loc)
    {
        PTR = PTR->next;
    }
    if (PTR == nullptr)
    {
        location not found, Return.
    }
    node *temp = new node (num)
    temp->next = PTR->next;
    PTR->next = temp;
```

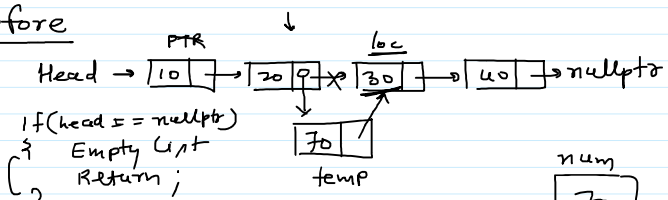



```
PTR = Head
count = 1
num 70
loc 5
while (PTR != null && count != loc)
{
    PTR = PTR->next
    count ++;
}
```

Add Before



Add Before




```
if(head == nullptr)
{ Empty List
  Return ;
}
```

num: 70 ; loc: 30

① `if (Head->value == loc)`
```
{   AddFirst (num) ;
    return ;
}
```

② //find

```
PTR = head
while ( PTR->next != nullptr && PTR->next->value != (loc)
{
    PTR = PTR->next ;
}
if (PTR->next == nullptr)
{       loc not found , Return;
}
temp = new node(num) ;
temp->next = PTR->next
PTR->next = temp
```

DelFirst :-

① Head → nullptr

② Head → 10 | N

③ Head → 10 → 20 → 30 → 40 | N

① `if (Head == nullptr)` ——— `return ;`

②③
```
PTR = Head
Head = Head->next
delete PTR
```
`cout << PTR->value << "deleted";`

Del Last :→

① Head → nullptr

② Head → 10 | N → nullptr

③ Head → 10 → 20 → 30 → 40 | N

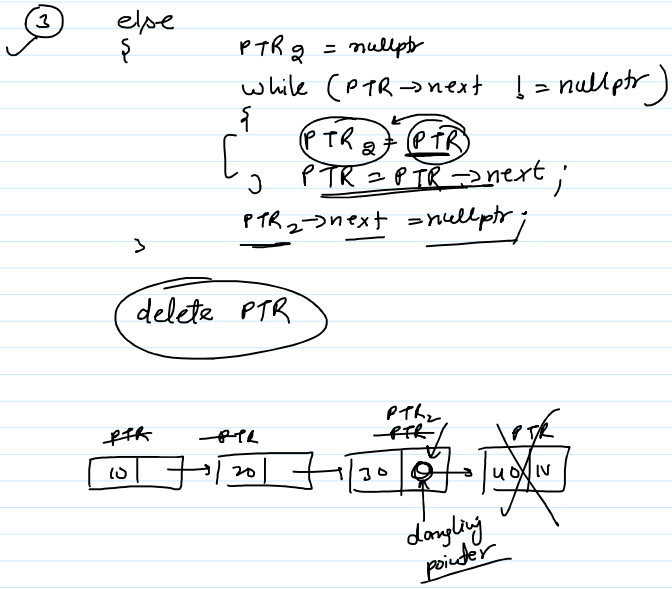
PTR1 = N

① `if (Head == nullptr)`
——— ; return;

PTR = Head

② `if ( Head->next == nullptr)`
```
{       Head = Head->next ;
}
```

③ else
```
{       PTR2 = nullptr
        while (PTR->next != nullptr)
```




```
#include<iostream>
using namespace std;
class node{
    public:
        int value;
        node *next;
        node(int x)
        {
            value = x;
            next = nullptr;
        }
};
class LinkedList{
    private:
        node *head;
    public:
        LinkedList()
        {
            head = nullptr;
        }
        void addFirst(int num)
        {
            node *temp = new node(num);     //new
dynamic node
            temp->next = head;
            head = temp;
        }
        void addLast(int num)
        {
            node *temp = new node(num);
            if(head == nullptr)
                head = temp;
            else
            {
                node *ptr = head;
                while(ptr->next != nullptr)
                {
                    ptr = ptr->next;
                }
                ptr->next = temp;
            }
        }
        void output()
        {
            node *ptr = head;
            while(ptr != nullptr)
            {
                cout<<ptr->value<<" ";
                ptr = ptr->next;
            }
            cout<<endl;
        }
        void addAfter(int num, int loc)
        {
            node*ptr = head;
            while(ptr != nullptr && ptr->value != loc)
                ptr = ptr->next;
            if(ptr == nullptr)
            {
                cout<<"Location not found\n";
                return;
            }
            node *temp = new node(num);
            temp->next = ptr->next;
            ptr->next = temp;
        }
        void addBefore(int num, int loc)
        {
            if(head == nullptr)
            {
                cout<<"Empty List\n";
```

```
                return;
            }
            if(head->value == loc)
            {
                addFirst(num);
                return;
            }
            node*ptr = head;
            while(ptr->next != nullptr && ptr->next->
value != loc)
                ptr = ptr->next;
            if(ptr->next == nullptr)
            {
                cout<<"Location not found\n";
                return;
            }
            node *temp = new node(num);
            temp->next = ptr->next;
            ptr->next = temp;
        }
        void delFirst()
        {
            if(head == nullptr)
            {
                cout<<"Empty List\n";
                return;
            }
            node *ptr = head;
            head=head->next;
            cout<<ptr->value<<" deleted\n";
            delete ptr;
        }
        void delLast()
        {
            if(head == nullptr)
            {
                cout<<"Empty List\n";
                return;
            }
            node *ptr = head;
            if(head->next == nullptr)
            {
                head = nullptr;
            }
            else{
                node *ptr2 = nullptr;
                while(ptr->next != nullptr)
                {
                    ptr2 = ptr;
                    ptr = ptr->next;
                }
                ptr2->next = nullptr;
            }
            cout<<ptr->value<<" deleted\n";
            delete ptr;
        }
};
int main()
{
    LinkedList list;
    list.addFirst(10);
    list.addFirst(20);
    list.addFirst(30);
    list.addFirst(40);
    list.output();
    list.addLast(70);
    list.addLast(60);
    list.output();
    list.addAfter(100,60);
    list.output();
    list.addBefore(200,40);
    list.output();
    list.delFirst();
    list.output();
    list.delLast();
    list.output();
}
```