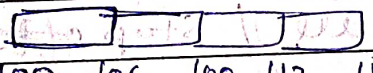
Lecture 8.

What is Data Structure

Data Structure is the way to store data.
~~For~~ for every problem we have to decide the best data structure to store data on the basis of problem we have.
→ Choosing a appropriate data means a lot.

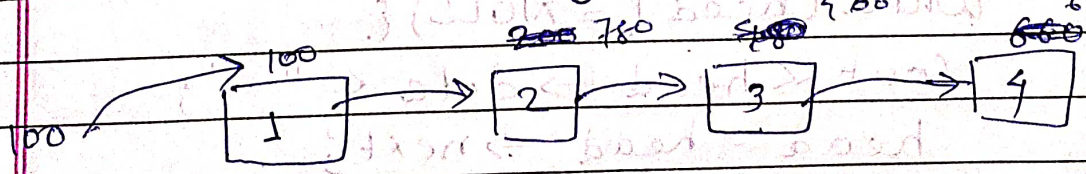
Linked list → (A data structure)

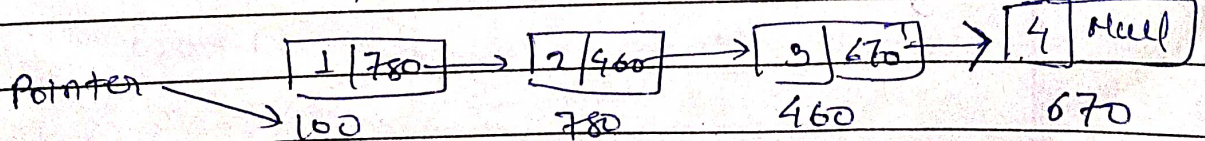
Problems with Array
↳ size is fixed, which is decided before going ahead to solve problem at declaration of array.
(Problem bcz array store data in continious manner)
So at a stage if we decide array of size 4.



now ~~not~~ if you want to incr. size then we can't bcz it is not confermed ~~that~~ as memory location of 116 to 120 are empty. or not.

~~So~~ ~~use~~ linked list, where data are stored not in continious manner and ~~can~~ size can be increased any time.



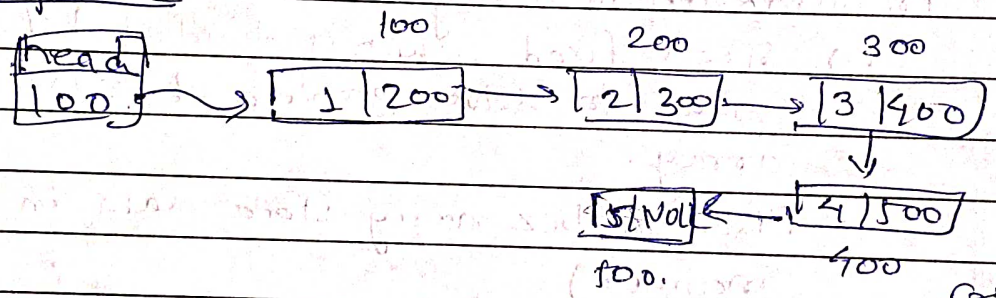


linked list

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

// Note :- 1st node of linked list is head and we have to store it some where, for whole Programme (so we can access the linked list), and last node is called tale.

Code objective



Note head → next != NULL // Stop at last node.
head != Null // Stop at last node.

Code

```
#include <bits/stdc++.h>
using namespace std;

void print (Node *head) {
  // Node * temp = head;  (( head की value change नहीं करनी है ))
  While (head != NULL) {
    cout << head -> data << " ";
    head = head -> next;
  }
```

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

Class Node {
  Public:
  int data;
  Node *next;
  Node (int data) {
    this -> data = data;
    next = NULL;
  }
};

int main () {
  Node n1(1); //Statically
  Node *head = &n1;
  Node n2(2);
  Node n3(3);
  Node n4(4);
  Node n5(5);
  n1.next = &n2;
  n2.next = &n3;
  n3.next = &n4;
  n4.next = &n5;
  print(head);
  // print(head);
```

idle method is to create temp variable.

this is pass by value

Yes this will print linked list twice as main ची head value change होती नाही!

```
/*
Dynamically → unlike java we have to use here pointers (bcz it store address)
Node * n3 = new Node (10);
Node * head = n3;

Node *n4 = new Node (20);
n3 → next = n4;
*/
}
```

// Taking data from user and returning head of linked list.

code

```
Node *takeInput(){
    int data;
    cin >> data;
    Node *head = NULL;
    while (data != -1){
        Node *newNode = new Node (data);
        if (head == NULL){
            head = newNode;
        }
        else{
            Node *temp = head;
            while (temp → next != NULL){
                temp = temp → next;
            }
            temp → next = newNode;
        }
        cin >> data;
    }
    return head;
}
```

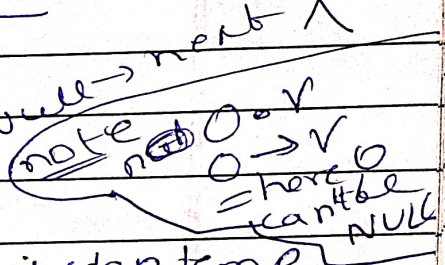
Note:- if we create node statically eg.
Node n(data); // then after each iteration of while loop this node get released from memory as it is created statically.

- In while loop, ~~data~~ variable like newNode, temp are statically variable, so it is created and destroyed after each iteration of a loop.

Complexity of takeInput() function.

Note we can't not do this bcz

```
while (temp != NULL) {
    temp = temp->next;
}
temp = newnode;
```



by this step temp always become zero. and pointing to memory allocation of zero.

where at moment (temp == NULL that mean temp == NULL don't point data) than at that time sequence of list cannot be accessed now).

time complexity is $O(n^2)$.

1 → K
2 → 1
3 → 2
4 → 3
n → n-1

→ How to improve this..?
→ Solution:- use tail as well.

modified code is below ↓

```
Node * takeInput_Better(){
    int data;
    cin >> data;
    Node *head = NULL;
    Node * tail = NULL;
    while (data != -1){
        Node * newNode = new Node (data);
        if (head == NULL){
            head = newNode;
            tail = newNode;
        } else {
            tail -> next = newNode;
            tail = tail -> next;
            //OR
            // tail = newNode;
        }
        cin >> data;
    }
    return head;
}
```

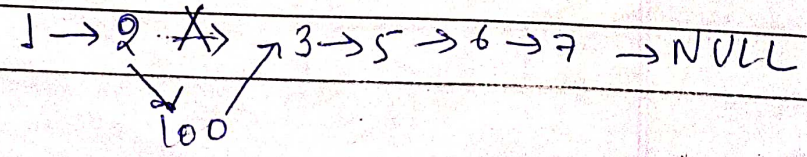
//now the time complexity is reduced to O(N).

- operation-2

  insert node at i^{th} position.

ex:- initially $1 \to 2 \to 3 \to 5 \to 6 \to 7 \to NULL$

insert 100 at i=2

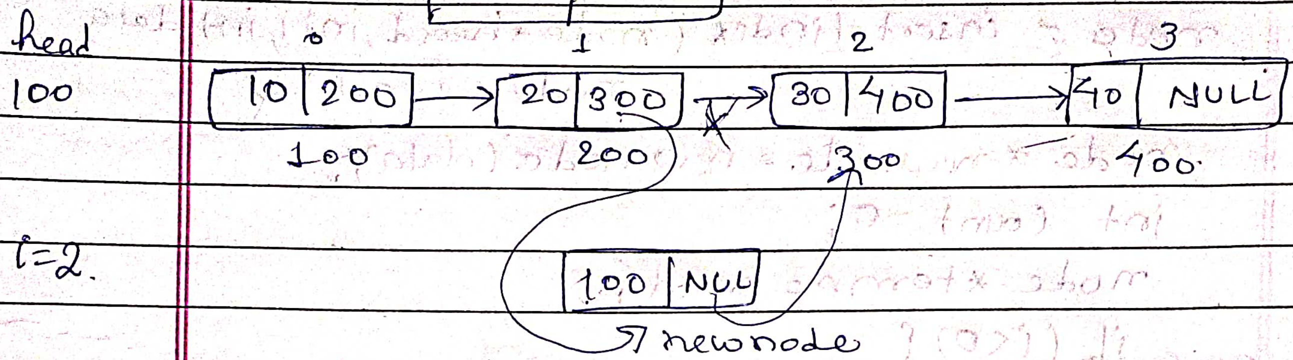


head 100 200 300 400 — cout 0 1 2 3 — i = 4 — 2 < 3, 3 < 3

Date ____
Page ____

1st Step:- let create a node of 100

| 100 | NULL |



~~so next~~ ~~(200)~~

so newnode → next = addr(200) → next
add(200) → next = newnode.

↳ must be in given order.

<u>Side cases</u>

1) index = negative
2) index = 0
3) index = length of LL
4) index > length of LL

↓ sol^n
temp = NULL

sol^n
```
if (i<0) {
    cout<<"Pls give suitable i value"<<endl
    return
}
```

now case (2) if index=0

sol^n
```
if (index==0) {
    newNode->next=head
        head=newnode
    return head;
}.
```

case (3) index = length of LL

actually if we insert at last index, then it insert perfectly but we need to update tail as well so we have to pass tail as reference not value

as main function में भी head update करना होगा

~~finale code~~

finale code

```
node * insertatindex (node *head, int i, int data,
                      node *&tail) {
node *newnode = new node (data);
int count =0;
node *temp = head;
if (i<0) {
cout << "please give suitable index" <<endl;
return head;
}
if (i==0) {
newnode -> addr = head;
head = newnode;
return head;
}
while (temp! =NULL && count < i-1) {
temp = temp -> addr;
count++;
if (count == i-1 && temp -> addr == NULL)
{
tail = newnode;
}
}
if (temp != NULL) {
node * A = temp -> addr;
temp -> addr = newnode;
newnode -> addr = a;
}
return head;
}
```

case 1

case 2

case 4

use NULL

case 3

move this

case 5

Call: head = insertatindex (head, i, data, tail)

(*) Note:- Case 3 वाले code को shift करना होगा क्योंकि अगर linklist + 1 size का होगा and हम insert at 1 index में करेंगे तो fir tail update नहीं होगी ~~क्योंकि cout < i-1=0 (false आएगा)~~

Delete Node at ith position.

ex:- head → 1 → 2 → 3 → 4 → 5 → NULL
(0 1 2 3 4)

Provide index → i → 2 (x got deleted)

head → 1 → 2 → 4 → 5 → NULL

2 → 3 → 4
temp a b

a = temp → next
b = a → next
temp → next = b
delet a; // deallocate ath address node

Similary cases as in insert.

Code

```
node * deletenode (node *head, int i, node*&tail)
{
  int count=0;
  node *temp= head;
  if (i<0){
      cout << "please give suitable index" << endl
      return head;
  }
  if (i==0){
    head= head → addr;
     delete temp;
     return head;
}
while (temp!= NULL && count<(i-1)){
    temp= temp → addr;
     count++;
}
```

```
if(temp != NULL) {
    if (count == i-1 && (temp->addr)->addr==NULL)
    {
        tail = temp;
    }
    node *a = temp->addr;
    temp->addr = a->addr;
    delete a;
}
return head;
}
```

Calling :- head = deleteatindex (head, node, tail);

~~Insert the~~

Get the length of linked list recursively

```
int lengthofLLrecursive (node *head, int count)
{
    if (head != NULL)
    {
        count = lengthofLLrecursive(head->addr, count);
        count++;
    }
    return count;
}
```

```
Cout << lengthofLLrecursive(head, 0) << endl;
```

Inserting a node → Recursive

> base case
> do for 0th node and then simply call for n+1th node recursively

~~18 19~~ (insert and delete recursively) ⇒ see github

20 find in LL

```
int findLL (node *head, int value, int count)
{
    if (head == NULL)
    {
        return -1;
    }
    if (head->data == value)
    {
        cout << count << endl;
        return 1;
    }
int res2 = findinLL (head->addr, value, count+1);
return (res2 == -1) ? -1 : 1;
}
```

Append Last N To first.

eg:-

Linked list 1→2→3→4→5→null

if n = 3

Ans 3→4→5→1→2→null

if n = 2

Ans 4→5→1→2→3→null

Date
Page

Code

```
node * appendlast(node *head, node *&tail,
                  int i)
{
  if(i==0)
  { node *a=head->addr;
    tail=head;
    head->addr=NULL;
    return a;
  }
head = appendLast(head->addr, tail, i-1);
return head;
}
```

Calling ~~method~~ method

```
cout<<"give n"<<endl
int N;
cin>>N;
int l= length of LLreccursive (head, 0);
if(N==0||N==l||N>l||N<0)
    {
      cout<<"wrong value of N"<<endl;
                give suitable N
    }
else{
    tail->addr=head;
    head = appendLast(head, tail, l-N-1);
    }
PrintLL(head);
```

calling method from main