__/__/____

Linked Lists

① Basic struct, class & pointer program

- we create a class (a user defined datatype) named node (now a datatype node is created)
- containing 2 items i.e - int data
  - node *next
  (next is a pointer of type node)
  } with public access specifier
- now we create a constructor to assign values for data & next
  data = d
  next = NULL Ptr;
  } Public

main function
arr = { 2, 5, 7, 9}

- here we create a node using new keyword for an element in the array it creates a node in a random place in heap memory
- we store this newly created nodes address in n.

__/__/____

* new Node (v[2])
- basically what happens here is v[2] is passed to the constructor
- stores v[2] as data in node & next as null

- now if we print n - it prints address of the new node

- n->data - this basically means it points & gets access to the class member data
  - simplified version of (*n).data

Class Definition

```
Struct Node {
Public:
  int Data;
  Node *next;
```

Struct all members are Public by default where as Class members are Private by Default unless specified

*next is a pointer Variable of type node i.e *next holds the adress of the next Node & each node has Data & next

```
Public:
 Node (int data1)
 {
   Data = data1;
   next = nullptr;
```

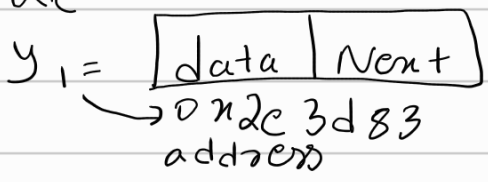
Constructor
For the Defined class youre basically assigning value

```
 int main()
{
 vector <int> arr = {2, 5, 8, 7};

Node * y1 = new Node (arr[0]);
Node * y2 = new Node (arr[1]);
Node * y3 = new Node (arr[2]);
Node * y4 = new node (arr[3]);
```

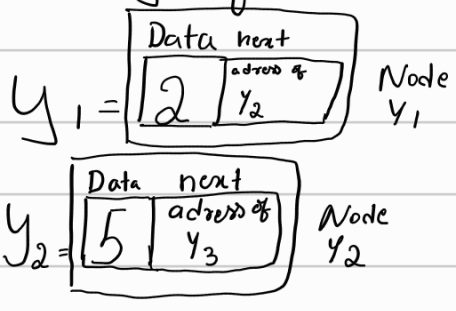
Assigning values
- y1 is a pointer variable of type node which means

y_1 has data & next
- we create new Node & pass the data which gets assigned in constructor
now y_1 has the address of the new Node



$y_1 \rightarrow next = y_2$
$y_2 \rightarrow next = y_3$
$y_3 \rightarrow next = y_4$

Linking of nodes




```
cout << y1->Data << " " << y1->next;
cout << y2->Data << " " << y2->next;
```

2 address of y_2
5 address of y_3

.
.
.

__/__/__

③ Array to linked list

St 1- create a class with data & node *Next
- & here *next itself is a node type so it will have data & & next.

St 2- function to convert array to LL
- returns the head node

- assign head node to the first ele of the array Node *head = new node &(arr[i]);

- keep a current pointer used to link the nodes initially set to head (so current now has headnode)

- iterate through the array from a[1] → N

- for each element you create a new node & store that in a temp pointer.

- now to store this temp (i.e adress of new node in previous nodes next)

- we access current nodes next & store temp init.

current -> next = temp

so after linking we move the current ahead repeat for all elements of arr.

__/__/____

St 3- traversal through the created linked list

- we get the head node.
- we keep a current pointer to initially point at head.
- now we create a while loop till the current points to null
- until then we can print the data & its next.
- then update current to move to next node
  ↳ create temp node get the current node's next & store that in temp
  ↳ now current = temp.

```
Node * temp
        temp = current -> next;
        current = temp;
```

OR

↳ directly update current

```
current = current -> next;
```

__/__/____

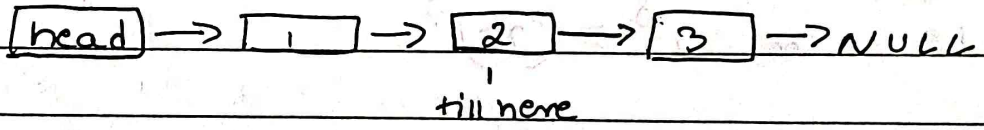
Linked Lists

④ Deletion of head;

- Store the current head node in a temp node
- move the head to the next node
- free up the space by deleting temp.

⑤ Deletion of tail

- Start traversing from the head node
- do the traversal till last but 2nd element

i.e 

so when you get current → next → next = null then come out of the loop

- just delete current's next node (i.e last node)
- & set current → next = Null.

__/__/____

⑥ Delete k^{th} node

Edge cases

- if list is empty return null
- if k = 1 then delete head
- basically you have 2 trackers one is current & another for prev
- keep a counter
- traversal till current points to null
  - if count == k then link previous node to current's next node & del current
  - & break
- if not just keep traversing by storing current in prev move current ahead & count ++.

⑦ Delete node with value x.

- Same as previous one but just compare data with x

__/__/__

(8) Insertion at head

- ~~edge case~~
- just create a newnode
- just point the newnodes next to head
- return new node.

(9) Insertion at tail

Edge case

- if list is empty the just create new node & return

Else

- create new node
- set last nodes next to newnode
- & newnodes next to null.

(10) Insertion at kth position

Edge case

- if given posi is <1 then return null.
- if given pos = 1 then insert at head.

- just start counting the nodes while tracking current node & previous node

- once the count matches the given position then

- set new nodes next to <u>current</u> or <u>prev → next</u>

- then prev node → next to new node.

- Edge case
  - if at last ~~ca~~ posi is last place i.e insertion at tail

(11) insert value K before X value

- Same as previous one
  Edge case
  - if X is first node value or heads value then insert node before head.

- & in while loop just check if value = current data.

(12) Array to DLL

- create head node
- traverse through the array from 1 to n creating new node for each element.

- & initially current is initialized to head node as & when we create new nodes we initialize current → next to newnode & new nodes prev as current & then move the current ahead.
  & finally return the head.