

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 } with public access specifier
(next is a pointer of type node)
- 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 arr.

* 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

- new 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

(data = (*n).data)

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
- ~~initially~~ assign head node to the first ele of the array $\text{Node *head} = \text{new node}(\text{or } [i]);$
- keep a current pointer used to link the nodes initially set to head (so current now has head node)
- iterate through the array from $a[1] \rightarrow N$
- for each element you create a new node & store that in a temp pointer.
- how to store this temp (i.e address of new node in previous nodes next)
- we ~~access~~ access current nodes next & store temp init.

$\boxed{\text{current} \rightarrow \text{next} = \text{temp}}$

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

S+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;
```

— / — / —

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
ie head → [1] → [2] → [3] → NULL

|
till here

So when you get $current \rightarrow next \rightarrow next = null$
then come out of the loop

- just delete current's next node (i.e. last node)
- $\&$ set $\text{current} \rightarrow \text{next} = \text{Null}$.

//_

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 new node
- just point the new node's next to head
- return new node.

⑨ Insertion at tail

Edge case

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

Else

- create new node
- set last node's next to new node
- & new node's next to null.

⑩ Insertion at kth position

Edge case

- if given pos 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 node's next to current or prev → next

- then $\text{prev node} \rightarrow \text{next}$ to new node.

- Edge case

- if at last ~~pos~~ is last place i.e. insertion at tail

(11) insert value ~~at~~ before X value

- Same as previous one

Edge case

- if X is first node value or head's value then insert node before head.

- & in while loop just check if $\text{value} = \text{current data}$.

(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 $\text{current} \rightarrow \text{next}$ to new node & new node's prev as current & then move the current ahead.
& finally return the head.