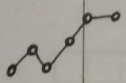
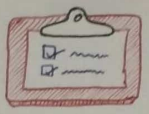
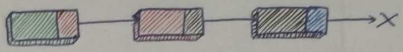
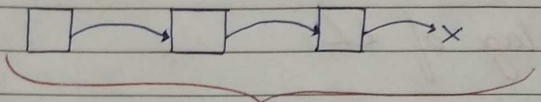


DATE

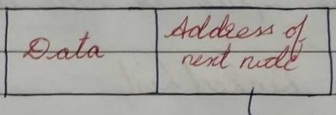
29. Linked List - I

→ Linear Data Structure that has a collection of nodes.



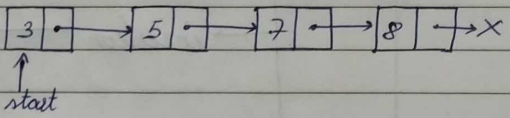
Linked List

Node



Class Structure

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



C++ code

```
class Node {
  public:
  int data;
  Node * next;

  Node (int d) {
      this -> data = d;
      this -> next = NULL;
  }
}
```

This is optional

```
main() {
  Node * first =
      new Node (3);
  cout << first -> data;
  //3
  cout << first -> next;
  //0x0
}
```

DATE

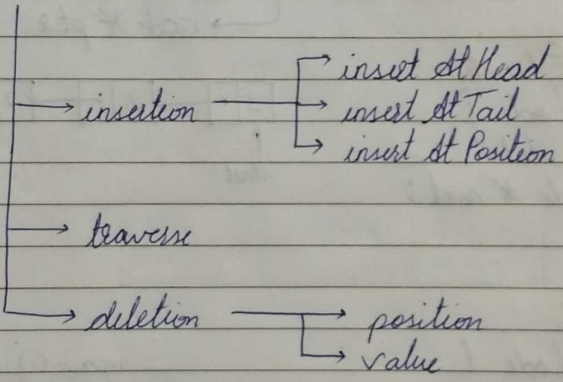
Advantages of LL

→ grow and shrink dynamically
→ non-contiguous
→ insertion & deletion in O(1)

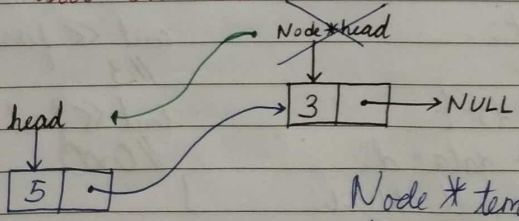
Disadvantage of LL

→ Random access not possible
  ↳ need to traverse whole list in O(n)
→ Need extra space to store address

Operations in Linked List



Insert At Head

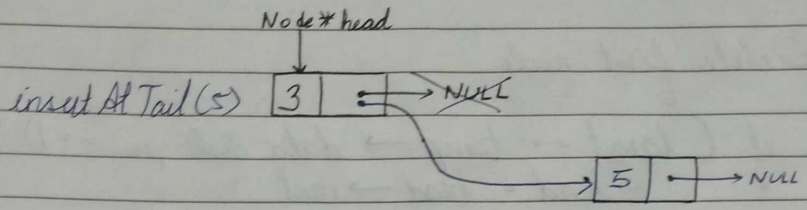


insertAtHead(5)

```
Node *temp = new Node(5)
temp -> next = head
head = temp
```

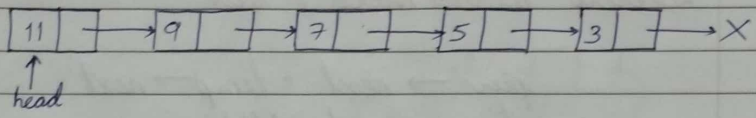
DATE

Insert At Tail



create
tail → next = temp

Traverse

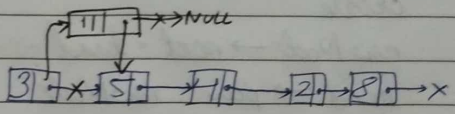


* Never change value of head unless necessary
  ↳ use pointer variables

```
traverse (head)

Node * temp = head;
while (temp != NULL) {
        cout << temp → data;
        temp = temp → next;
}
```

Insert At Position



insert At Position (2)

1. create a node — if (n == 1) insertAtHead(d)
2. traverse k-1 node — if (n == length) insertAtTail(d)
3. newNode → next = temp → next
   temp → next = newNode

} order is important

DATE

Deletion

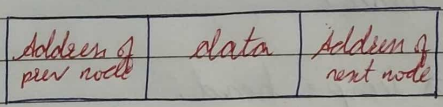
Delete first node

```
if ( target == temp → data && pos == 1) {
        head = head → next ;
        temp → next = NULL;
        delete temp;
}
```

Delete middle node / tail

```
prev → next = temp → next
temp → next = NULL
delete temp
```

Doubly Linked List

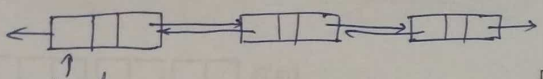


Insert At Head

```
create
newNode → next = head
head → prev = newNode
head = newNode
```

Insert At tail

```
create
newNode → prev = tail
tail → next → newNode
tail = newNode
```



DATE

Insert after prev (any)

create

newNode → next = prev → next

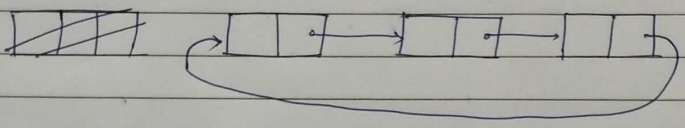
prev → next → prev = newNode

prev → next = newNode

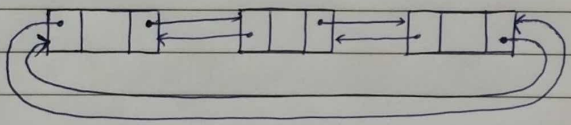
newNode → ~~next~~ prev = prev.

H/W Operations Home Work.

Circular LL



Circular Dabby LL



HomeWork

SLL
DLL
CLL
CDLL
} All operations