Data Structures and Algorithm

19/10/22

non-primitive data types are abstract types.

```
student_Info {
string  stdName;
string  stdEmail
int     stdCntNum
string  stdRegNo
int     stdAttendance [][]
}
```

Dynamic
Abstract data types
stack] linear D.S
queue]
tree
graph.] non-linear D.S

Linked list is a linear dynamic D.S.

Data Structure
Relationship b/w datas

Array → (Non Primitive / Static)

20/10/22

Insert

Stack → front (entry point or exit point)

Stack is a dynamic memory / temporary

Queue

front — rear.
↓ deletion ↓ addition

! OS use queue structure

code segment

stack
code
data.

Tree

directory hierarchy
root → entry point

Graph

no hierarchy
. source . destination . .

shortest path algorithm.

25/10

DSA

- relationship among data is known as structure.
- set of steps to achieve an objective.
- steps to access any data structure → algorithm.
  an algorithm should be optimized.
  optimization: using minimum resources, getting maximum benefits. (subjective term)

1. Processing time optmization, time for execution
2. Line of code optimization.

CR
LF

defragmentation

Implementing Data Structures:

(i) Array / List (interchangeable).

- array can store same type data
- homogeneous data types
- continuous memory,
- array is static.

(ii) Linked List / Node.

- node can store multiple data type. data
- heterogenous types.
- continigious memory.
- node is dynamic.

1000

M + (i − L) * W

1000 + (2 − 0) * 4

1000 + (8)

1008

26/10/22

Memory tomorrow

Operations on array:

(i) creation of array

N

a0 a1 a2 a3 a4

int a[5]; | 300 | 301 | 302 | 303 | 304 |

char

int

tag

tags → variables.
memory locations (memory addresses).

to access 300 (address).

a[0]
300 + 0 = 300

a[1]
300 + 1 = 301

a[2]
300 + 2 = 302.

a[4]
300 + 4 = 304

L U

Upper bound(U) Lower bound(L) size

(ii) Traversing: Visit / travel on all nodes.

Assignment: ① make functions of creation & traversal. and all operations on an array.

(iii) Insertion.

tweek
twirk

3a) insert_end.(array, value), isFull
3b) insert - start.(array, value), isFull
3c) insert - loc (array, value, loc), isFull.

U = upperbound
L = lowerbound

↓

```
for(int i=U ; i ≥ loc ; i--) {
        a[i] = a[i-1]
}
a[L] = value;
```

U++ ⇒ 5

{0,1,2,3,4}, 8, 3

i=5 5>3 ✓
a[5] = 4
i=4 4>3
a[4] = 3
i=3 3>3 X

DSA Lab

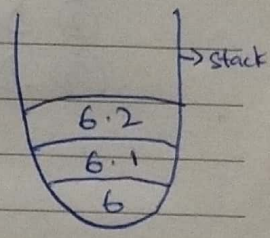
Linear Search
- Sentinel Linear Search
- Two way linear Search
- Probabilistic Search.
- Binary Search

Memory:

27/10/2022

Code Seg.
Data Seg.
Stack Seg.
Extra Seg.

Instructions
External Data.
Temporary memory.



code Seg:

```
input a and b.
function sum(a, b) {
        return a+b
}
SOUT(sum(a, b));
```

on function calling
:~~CALL or JUMP~~
① Push PC
② Call or JUMP
that address will be stored in stack.
on return memory update
③ Pop PC.
PC = PC+1

Instruction Execution Cycle

1. PC - Program counter holds the address of next instruction to be executed
2. IR - Instruction Register holds opt. codes or addresses of operands. (opt. → operational)
3. AC / ACC - Accumulator is the execution unit

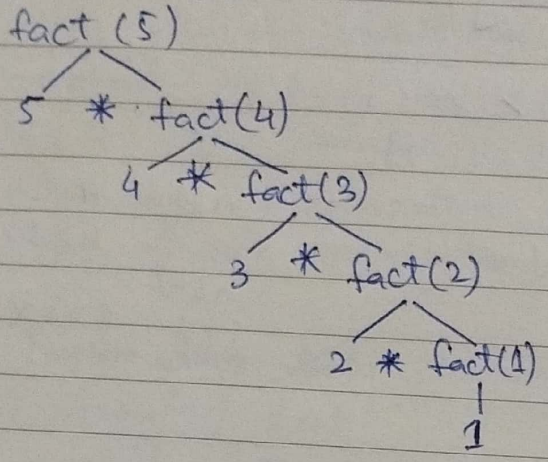
DSA

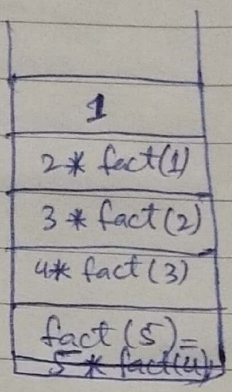
1/11/22

① Function Call
② Recursion.

```
fact(n){
  if(n==1)
    return 1
  else
    return n * fact(n-1)
}
```

• recursion tree





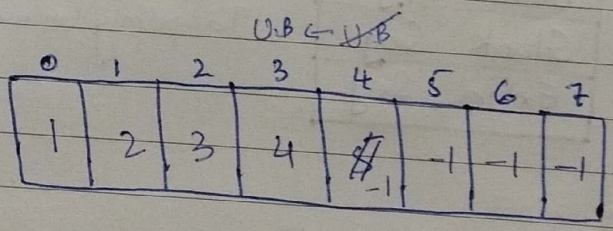
Assignment: Fibonacci Series (draw tree and stack).

Array

Deletion

L.B → where elements start
U.B → where elements end.

① Delete 1st element → loc = 0 ≠ L.B
② Delete last element → loc = U.B
③ Delete element at loc.



② Delete last element (array)

```
{ temp = a[U.B]
  a[U.B] = -1; // Empty value of compiler.
  U.B--;
  return temp;
}.
```

if U.B = L.B = -1 → array is empty.
whenever there is a single element in an array, so U.B & L.B are same.

3, 11, 22

① Array Empty (LB = UB = -1)
② Array contain only one element (UB = LB).
③ Array contain more than one element (LB < UB)

Searching ← Best, Average, Worst

Complexity → Time, Memory or all resources

Best case → complexity = 1.
Average case → complexity > 1 → n/2
worst case → complexity = length-1 or not found

① Linear Search

data may be sorted or unsorted.
returning location/index of any number to be searched through linear search, we returns -1, because -1 is the index which is not part of index of an array.

```
search(array, x) {
    for(int i=0; i < size; i++) {
        if(array[i] == x) {
            return i;
        }
    } return -1;
}
```

8/11/2022.

② BINARY SEARCH:

. sort (certain condition).

dynamic approach
modular approach
↓
ex. recursion

8/11/22

Sorting:

arrangement of data under a certain condition.

① Bubble Sort.

a a<b
b

0	1	2	3	4
2	45	0	11	9

i x j
m x n
iterations performed

```
for( i=0; i < size ;i++ ){
    for (j=0; j < (size-2); j++) {
```

② Selection Sort: compares 1st coming element with every other element.

0	1	2	3	4.
2	45	0	11	9

i	j					
0	1	2	45 (swap)	0	11	9
0	2	0	45	2	11	9.
0	3	0	45	2	11	9
0	4	0	45	2	11	9.

③ Insertion Sort 10/11/22.

Complexity

Time complexity.

```
int i, n;          → 1 sec
n = 10             → 1 sec
for (i=0; i<n; i++) { → 1 sec + (n+1) sec + n sec
    i = i+1;       → n sec + n sec
    print(i);      → n sec.
}
```

1
1
1
n+1
n
n
n
n

5n+4

5n + 4 → 4 negligible

5n → time complexity

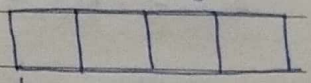
15/11/22

LINKED LIST

- different data types
- in a list, first element will have its data and address to the next element.
- each — Data.
  Link to next node(address).
- linked list is dynamic.
- Head node → 1st in linked list

contiguous
non-continuous
randomly.

array.

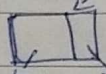


Base address.

For linked list.

First element.

when Head == null, (means no list).

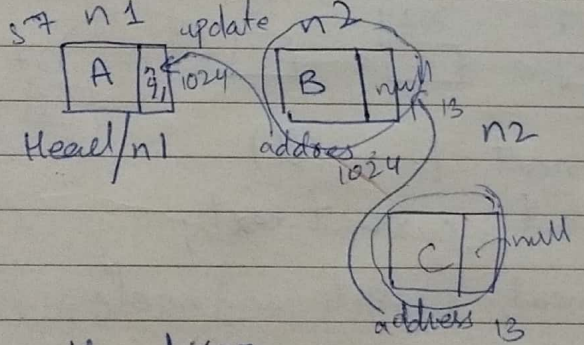


Data address

```
create node {
    DT Data
    Address next = null
}
```

Head will be updated
- inserting 1st element → if (Head == null).
- last element deletion.

```
→ Head = create n1.node
{ create node &
  update head }.
```




```
temp = create n.node {
    DT Data
    Add next
}
n1.next = temp
```

```
temp = Head//57
temp = temp.next //1024

if (temp != null)
    temp = temp → next.
```

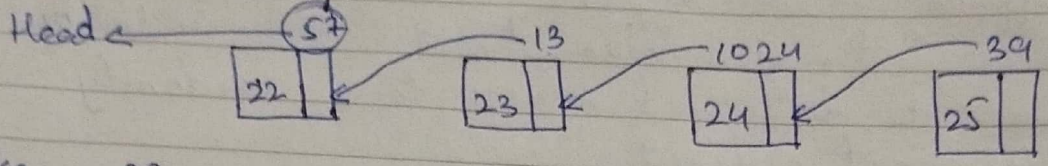
singly
doubly

PAPERWORK

17/11/22

Operations On Linked List

① Searching:

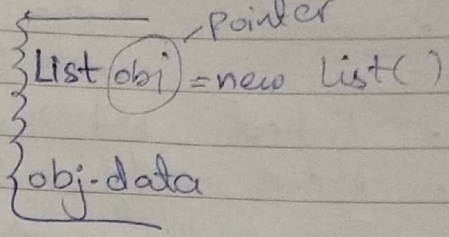


key = 22

Head = address of 1st node

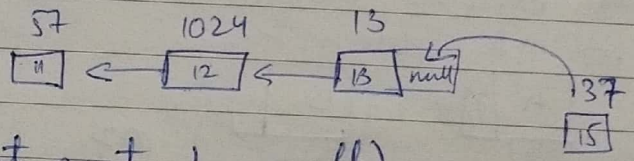
Head.data

Head.next.



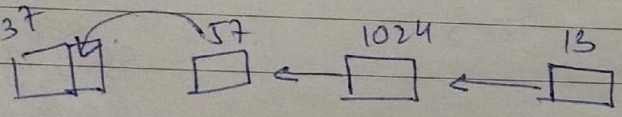

```
if(key == Head.data) found.
else {
    current = Head; while (current != null){
    if( current.data == key){
            return / break (if key found).
    }
}
    current = current.next;
```

② Insertion:




```
if (current.next != null)
        current = current.next;
}
current.next = create new node,
```

Head




```
temp = create new node.
new node.next = Head.
Head = temp
```

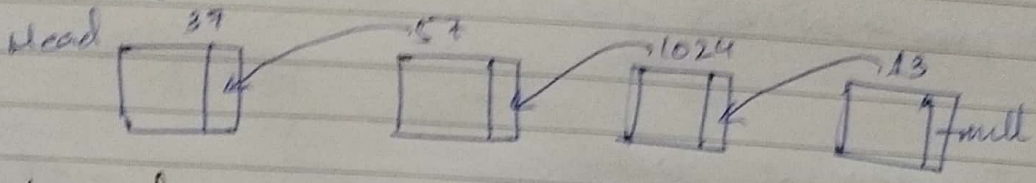
DSA

22/11/22

Insertion in Linked List

Deletion in Linked List

Delete
- Last node
- First node
- any Middle node

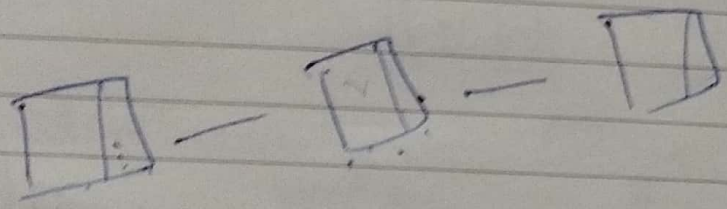


① Last node:

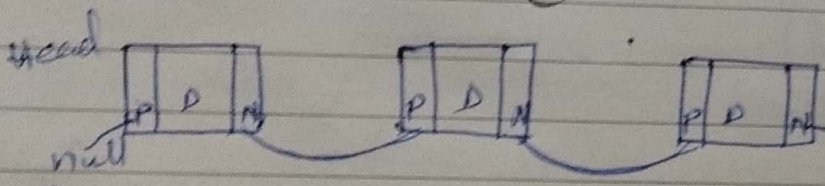
② Head node,
Head = Head.next ✓

You brought mine from your garden or wait for my visit to garden

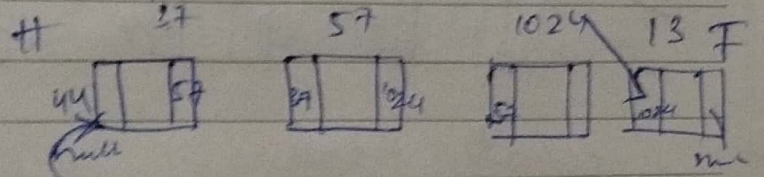
Ayavan

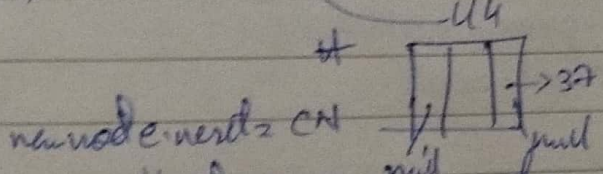


DOUBLY LINKED LIST:




```
node {
    DT Data;
    ref Prev;
    ref Next;
}
```

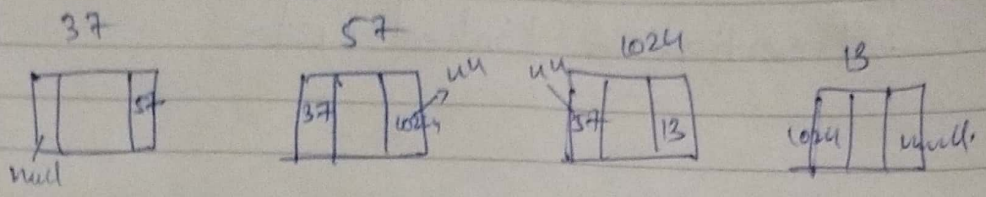


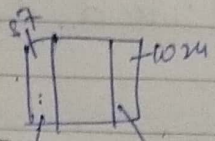


newnode.next = CN
Head.prev = newnode

CS-21229

PAPERWORK





44.prev → n.prev = (57) CN

44.next → n.next = CN.Next

57.next → CN.next = n

1024.prev → CN.next.prev = n

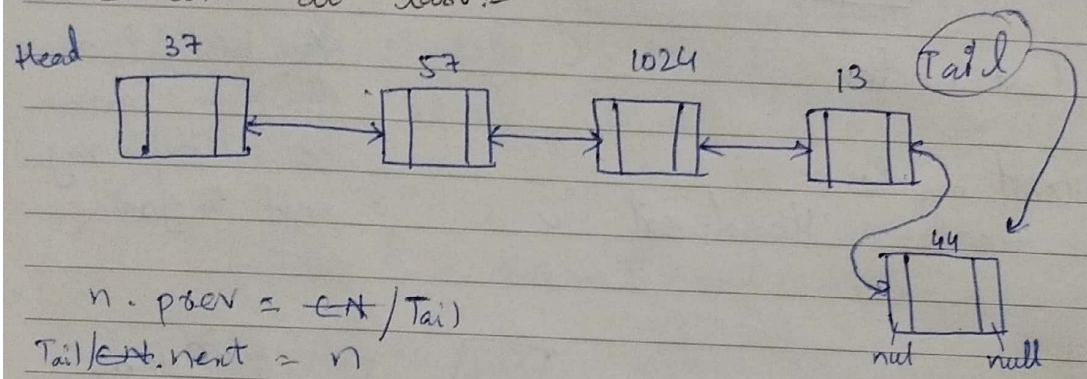
③ CN.next = newNode

① newNode.p = CN

② newNode.n = CN.next

24/11/22.

—: Insertion at last :—



n.prev = ~~CN~~ / Tail)

Tail/~~CN~~.next = n

Tail = n

29/11/2022

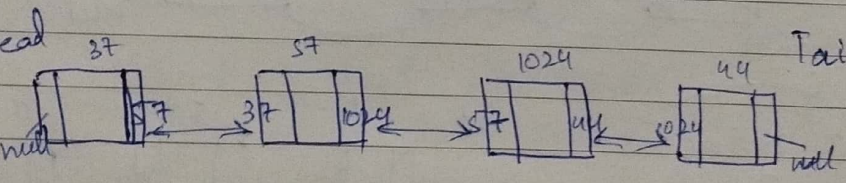
Delection :-

A → Middle

B → 1st (Head).

C → last element (tail)

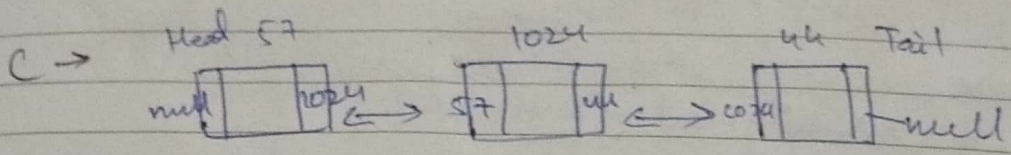
B



current = Head.

current.next.prev = null

Head = current.next

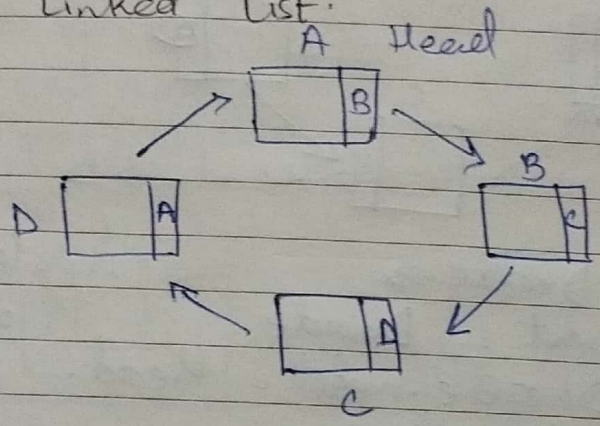


Tail.prev.next = Tail.next.

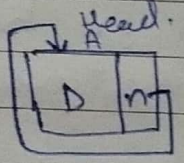
~~Tail.prev~~ = Tail = Tail.prev

Circular Linked List

By Singly Linked List.

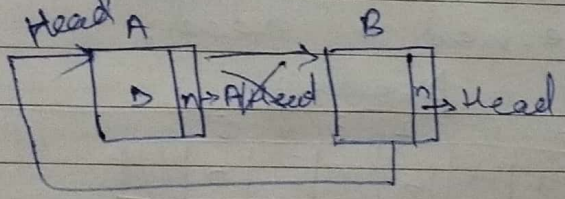


- Head = null → There is no linked list.
- Inserting first node ①

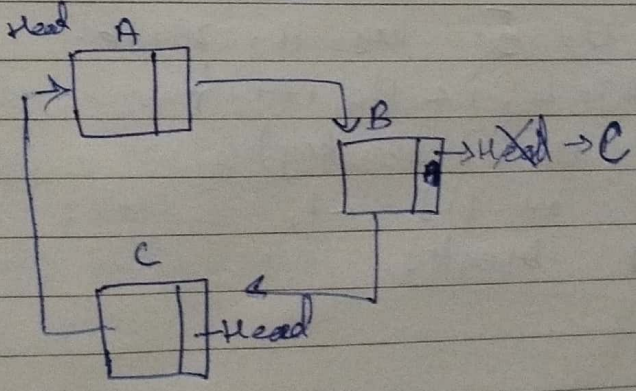


Head = new node.
Head.next = Head

- Inserting second node ②



- Inserting third node ③

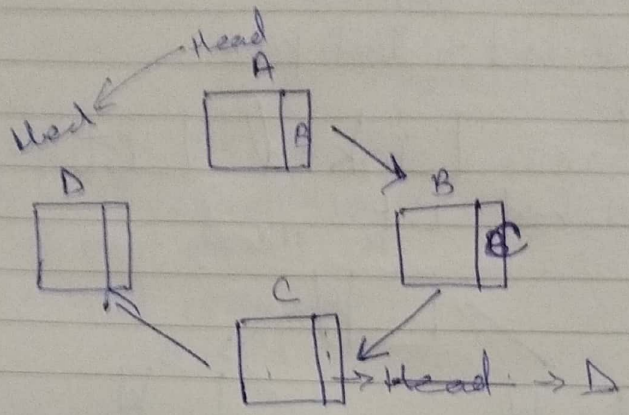


PAPERWORK

2%5 =

Traverse

if (current.next == Head).
to break;



~~Head~~ = Dnode

current.next = Dnode.

~~Head~~ = Dnode.next = Head.

Head = Dnode

while (current.next != head)
current = current.next

Insert new node in place of head.

Dnode = new node.

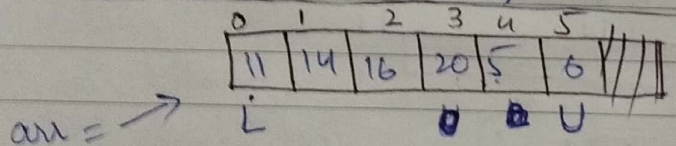
while (current.next != head) {
current = current.next;
}.

current.next = Dnode;

Dnode.next = Head;

Head = Dnode;

%2 (0-1)
%8 (0-7)
%16 (0-15)



L = 0, U = 5. key = 11, 20, 16.

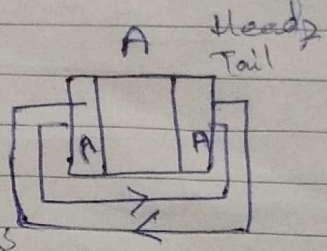
for (int i = L; i < U; i++) {
if (arr[i] == key) {
arr[i] = -1; L = i+1;
break;
}
}

i % length < length

Doubly Circular Linked List

1/12/22

Head = null
Tail = null

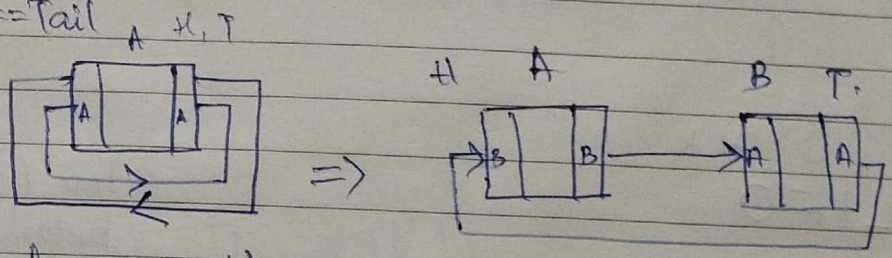


case I => Head == Tail == null

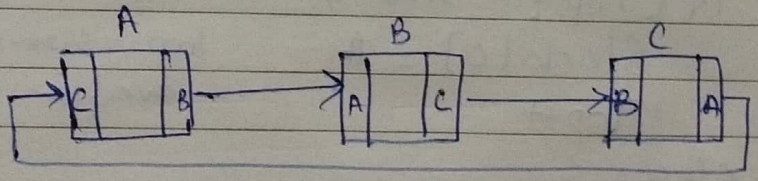
① if (Head == Tail == null) {
    Head = Tail = newNode
    Head.next = newNode
    Head.prev = newNode.
}

case II => Head == Tail

②



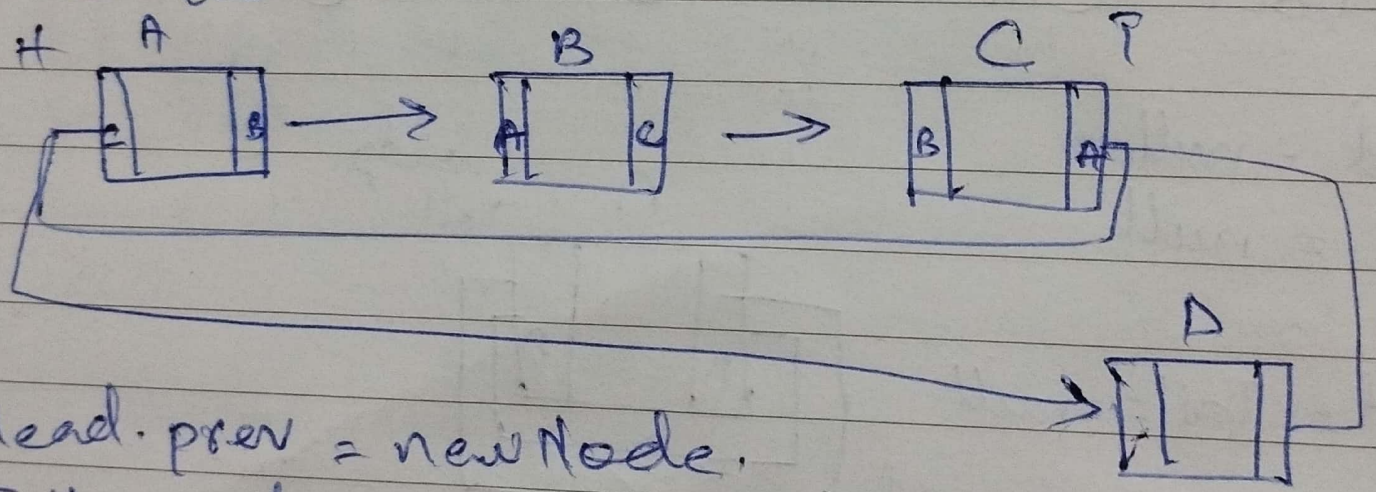
if (Head == Tail) {
    Head.next = newNode
    Head.prev = newNode.
    newNode.next = Head
    newNode.prev = Head.
    Tail = newNode
}



case => Insert at last

Head.prev = newNode
Tail.next = newNode
newNode.prev = Tail
newNode.next = Head
Tail = newNode.

case ⇒ Insert at Head.



Head.prev = newNode.

Tail.next = newNode.

newNode.prev = ~~Head~~ Tail

newNode.~~n~~next = Head.

Head = newNode.