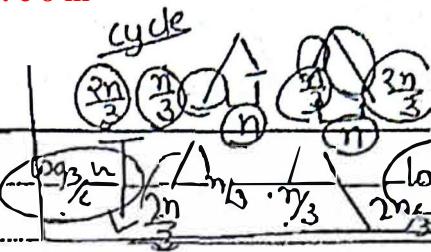




## Data Structure

### Hashing



1. It is a searching technique.

2. Worst Case,  $T_C = O(1)$

Array means continuous m/m

3. Direct Address Table (DAT) -

0	0
1	
2	2
3	
4	
5	5
6	
7	7
8	
9	9

size of table  $m = 10$

4. It can be accessed directly

$HT[x]$ .

↓ to search  $x \Rightarrow O(1)$

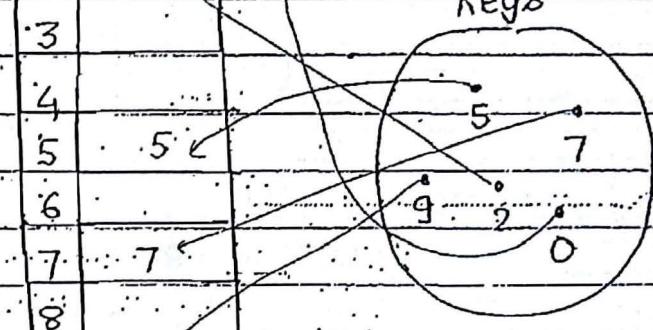
time is required

(No need to search at any

place just retrieve the

value)

Hash Table (HT)



1. In direct address table (DAT), key itself is the address with any calculation.

2. Worst case searching time is  $O(1)$  [BC, AC, WC].

3. the largest key no of slot is required ie if you want to store 100000 you must have (100,000+) slots so greatest disadvantage is for storage of 6 keys, 10000 slots are required.

4. the disadvantage with DAT is, even though no of keys are very less but one of the key may be very large ( $2^{1000}$ ) then the required hash table is of size  $2^{1000}$ .

To eliminate above drawback we are moving

to hash functions.

- If we have duplicate data then no need to store again & again for searching.

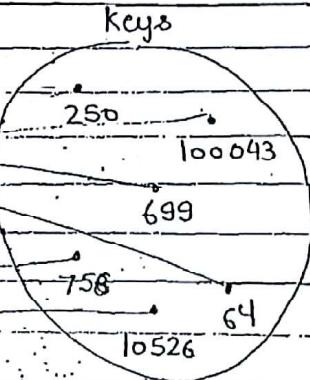
(If 11 keys were there)?

Hash Function -

HT

M=10

Hash address	0	1	2	3	4	5	6	7	8	9
	250			100043	64	H A S H E U N G				



it will check for one's place of the given key i.e. ( $\% 10$ ) we get the (0-9) key just store it

to retrieve 758

as  $699 \downarrow$   $699 \% 10$

again  $\% 10 = 8$  (758)

is stored at

8.

∴ Again time complexity = O(1)

Hash function take more time compared to Hash table due to function calculation.

Hash function is better as it also save the space.

in if  $n < 10 \rightarrow$  slot = 0 to 9

$n > 10 \rightarrow$  slot = 0 to 9

Q. Collision - If two keys mapped to same hash address by hash function then it is called collision.

i.e. Ex- 699 759

both get collided at 9 pos^n

12345

YII CIS DEPARTMENT

Hash function is better if no. of collision is as min. as pos.

Types of Hash function-

1. Division Modulo Method-

2. Digit Extraction Method

3. Mid-Square Method

4. Folding Method -

(i) Fold Boundary Method.

(ii) Fold Shifting Method.

1. Division Modulo Method.

ex1- size of table m = 1000 (0 to 999)

key = 1 2 3 4 5 6 7 8 9

Acc to Division Modulo Method

key Hash-function  $f(key) = \text{key Modulo } m$

$$= 123456789 \% 1000$$

$$= 789$$

0	
..	..
789	123456789
..	..

ex2- size of table m = 8 ( $2^3$ )

(take LSB 3 bit

key = 1010010101100101 here)

$H_f(key) = 1010010101100101 \bmod 2^3$

$$= 101$$

if  $m = 2^k$

then  $H_f(key)$

this method do not bother about the whole

data to how to store just take last  $k^{th}$  bits-

( $k$  bits)  
of LSB

Ex 1010101111111111 mod 2<sup>4</sup> = 1111

Ex 3  $m = 2^k$

key = 101010010101111011101 mod  $2^k$  = LSB k bits

if  $k = 8$

$H_f \notin \text{key} = 11011101$

Hash function of

- if M is in power of 2, key will be LSB only that lead to more collision.
- So do not choose M as a power of 2.

Ex 101010010101111011101  $k=7$

1010101111111011101 LSB

$H_f(\text{key}_1) = 1011101$

$H_f(\text{key}_2) = 1011101$

1. So, do not pick M as exact power of 2 because if  $M = 2^k$  then Hash function of key = LSB k bits always.  
but if M is not power of 2 then less chances of collision

2. Pick the M value which is a prime no. but it cannot be close to power of 2.

Ques - Which M value can be chosen for better result:

(a) 61 (c) 729

(b) 523 (d) 1024

729 is chosen as it is not power of 2 & not close to powers

Steps - (1) Assume all are prime no

(2) choose one which is not close to power of 2.

encoding - decoding  
store - retrieve

## 2. Digit Extraction Method - M = 1000

key = 789123456

$$\text{hash function(key)} = (7 \ 8 \ 9 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6) \\ (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9)$$

	0
	1
	715 789123456

extract some digits

$$= 715 \quad \text{extract digit} - (1, 4, 8)$$

the digits to be extracted are defined in question. You cannot extract 4 digits as size of stack is from 0-999

: Here no consideration is done to storage time.

- When you are extracting 3 digit, may lead to one digit or two digit address (if one of them become zero)

- Collision may also occur here as if

7 7 7 1 2 3 4 5 9  
1 2 3 4 5 6 7 8 9

but it is not producing that much less collision i.e no of collision is more here.

- Digit Extraction Method produces more collision as compared to Division Modulo Method (as for division modulo, you require  $M = 2^k$ ) but here only some bits have to be changed.

- We extract 3 digit address as 3 digit addresses >> 2 digit add

### 3. Mid-Square Method-

$$M = 1000$$

$$\text{key} = 84925$$

Step 1-

1. Square the given key

2. check middle of obtained key (take 3 bit from mid as it will have majority of 3-bit addresses)

Making counter counter here will be bit difficult, so ill lead to less counter & less collision. Because If key a larger no produces a more larging no.

$$\text{Hash function} = (84925)^2$$

$$= 721225625$$

225    255... as no exact mid possible

225	72184925

store at any one address

To Retrieve - (i) Square the key

(ii) take the mid, go to mid & extract the key.

Counter is not easy because of large each & every no (digit) participate in the calculation of square.

Folding Method -  $M = 1000(0-999)$   
 key = 123456789

(i) Fold Boundary Method

- fold the boundary  
 (take k bit from the key from boundary)
- $k \geq$  no of digit which have more address

for 3 bits - more address

123 456 789  
 123  
 789  
 913

If it lead to

1912	191	folding boundary again
913	193	9 again
key stored	912	123456789

123 45  
 123  
 45  
 168

Counter example = 123 89 789  
 913  
 ↓ again

counter is bit tougher than collision  
 bit extraction as 6 digit participate

(ii) Fold Shifting Method

- Fold it according to fold boundary but fold every bit

123 456 789

123

456

789

13 5 8

136

8

144 - 12345678

144

∴ fold shift is better than fold boundary as it require participation of 9 bit i.e all the bits.

• Counter is also possible:

Ex 789 456 123

↓ since addition satisfies commutative prop

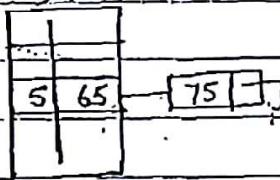
## Collision Resolution Technique

1. chaining

2. open addressing

chaining

- outside (add linked list)



(If more than m keys are to be kept then chaining is used)

Open Addressing

- when only m keys are to be kept
- inside (go to other slots)

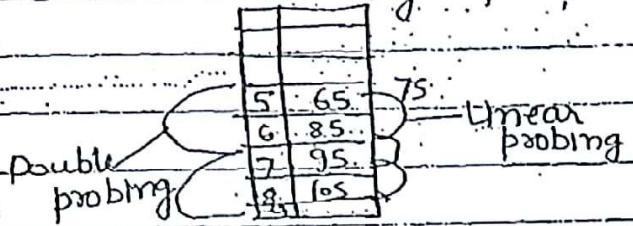
which are empty

• Linear probing if 75: check

for 5 fill 6 → fill and

continuous checking is done

when no empty space



- Quadratic probing, where random jumps are made known as quadratic probing
- Double hashing, when two values are directly jumped

Chaining Process -  $m=10$  (0-9)

key = (55, 98, 63, 75, 99, 73, 60, 95, 44, 29, 35, 73, 25, 108)

$$hf(key) = key \bmod m$$

Collision Resolution Technique = chaining

0	60			
1				
2				
3	63	73		Slot = 10
4	44			key = 14
5	55	75	35	25
6				
7				7
8	98	108		(Not present)
9	99	29	39	

0	6000	60 Null	1 length
1	Null	0	
2	Null	0 5000	500 1
3	5000	63 5001 73 5002 173 Null	3
4	Null	0 4000	
5	4000	44 Null	1
6	1000	55 1001 75 1002 95 1003 25 Null	
7	Null	1000 1001 2000 2001	Length of Longest chain = 4
8	2000	98 2001 108 Null	→ 2
9	3000	99 3001 29 3002 39 Null	Min. chain length = 1

Here searching time increases.

$$Avg = \frac{14}{10}$$

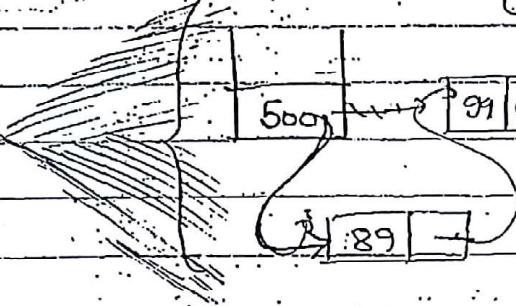
∴ in Worst case,  $TG = O(n)$  as same if all element occur in 1 slot.

but if you be using better hashing function then it can be reduced as  $O(1)$ .

It is also

## drawbacks

1. In chaining, space is wasted in the form of linked list even the space available inside.
2. The length of the longest chain possible is  $n$ . that lead to Worst case searching time =  $O(n)$ .  
Best Case =  $O(1)$   
Avg. Case =  $O(n/2) \approx O(n)$
3. the greatest advantage with the chaining is infinite no. of collision can be handled by the infinite no. of keys can be stored because of linked list.
4. Insertion of one key will take  $O(1)$  time (BC, WC, AC)  
(as insertion is done in linked list only)



insertion will take  $O(1)$  time.

5. Deletion time will take - Best case =  $O(1)$   
Worst case =  $O(n)$  (as it may be at last so you have to visit each node)

If address of node which has to be deleted is given then

time =  $O(1)$  (BC: ~~W.C.~~)

$O(n)$  - (WC)

b [since you cannot go back so you have traversed by start]

## Solve More (K)

is to be deleted & its address is provided with do  
+ then time to delete = O(1)  
= O(1)

exists in existence as in 1 slot you can store 100  
cell.

Addressing - &

probing - M = 10 (0-9)

Hashing) Keys = {55, 99, 60, 75, 89, 42, 58, 69, 25}

hash function (key) = k mod m

C.R.T = Linear probing

simg (key, i) = (h<sub>f</sub>(key) + i) Mod m       $\rightarrow$  attempt  
 $\forall i = \{0, 1, 2, \dots, m-1\}$        $\rightarrow$  no

0	60	$LP(55, 0) = (h_f(55) + 0) \text{ mod } 10$ here sign $= 5 \text{ mod } 10$	0	here sign 1st attempt
1	89			
2	42			
3	69	$LP(99, 0) = 9$	— (0)	N <sub>a</sub>
4		$LP(60, 0) = 0$	— (0)	inc
5	55	$LP(75, 0) = 5$ (fail collision)	— (0)	cc
6	75	$LP(75, 1) = 6$ (pass)	→ (2)	
7	25	$LP(89, 0) = 9$ (fail)		
8	58	$LP(89, 1) = 0$ (fail)	→ 2	
9	99	$LP(89, 2) = 8$ (pass)		
		$LP(42, 0) = 2$	— (0)	
		$LP(58, 0) = 8$	— (0)	
		$LP(69, 0) = 9$ (fail)		
		$LP(69, 1) = 0$ (fail)	$LP(69, 4) = 3$ (cc)	
		$LP(69, 2) = 1$ (fail)		
		$LP(69, 3) = 2$ (fail)	(4)	

Total collision = 9

5

You can store only  $M$  elements in  $M$ -slots.

To Retrieve same for 75 → goto 5 if there Yes  
else  $+1$

Quadratic

Double Hashing - two hash functions -  $h_1$  and  $h_2$

Quadratic probing (probing)

Quadratic Probing :  $M = 10$

Key { 25, 39, 46, 55, 89, 23, 68, 70, 94 } (56)

$H_f(\text{key}) = \text{key mod } m$        $c_1 = c_2 = 1$  (78)

CRT = QP

$QP(\text{key}, i) = (H_f(\text{key}) + c_1 \cdot i + c_2 \cdot i^2) \bmod m$

quadratic  
pattern

0	70	$\forall i = \{0, \dots, M-1\}$
1	89	$QP(25, 0) = 5$ (0) collision
2	23	$QP(39, 0) = 9$ (0)      Total collision = 2
3	23	$QP(46, 0) = 6$ (0)
4	94	$QP(55, 0) = 5$ fail
5	25	$QP(55, 1) = 7$ (pass) (0)
6	46	$QP(89, 0) = 9$ fail (i)
7	55	$QP(89, 1) = 1$ Pass
8	68	$QP(23, 0) = 3$ (0) $QP(56, 0) = 6$ (0)
9	39	$QP(68, 0) = 8$ (0) $QP(56, 1) = 6+2=8$
		$QP(70, 0) = 0$ (0) $QP(56, 2) = 6+5=11$
		$QP(94, 0) = 4$ (0) $QP(56, 3) = 6+1=7$

$$\begin{array}{ll}
 QP(78, 0) = 8 \text{ (fail)} & QP(78, 3) = \text{fail} \\
 QP(78, 1) = 0 \text{ (fail)} & QP(78, 4) = 8 \text{ fail} \\
 QP(78, 2) = 14 \text{ (fail)} & QP(78, 5) = 8 \text{ fail} \\
 \hline
 QP(78, 6) = 0 \text{ (fail)} & QP(78, 7) = 4 \text{ (fail)} \\
 QP(78, 8) = \cancel{2} \cancel{10} \text{ fail} & \cancel{2} \cancel{10} QP(78, 9) = \text{fail}
 \end{array}$$

} to collide

$$\therefore \text{Total collision} = 12$$

In Worst case, m-1 to 0 is covered  $\therefore O(m)$  you have to check.

$$\boxed{\text{Worst case} = O(m)}$$

Quadratic probing is faster than linear probing. But it will generate inaccurate result (as gap may be there but you are not able to keep the element).

If Linear Probing is used, if an empty slot is present, element will surely get stored.

In Linear Probing, Worst case searching time = O(m)

Best case = O(1)

Average case = O(m)

## Double Hashing

$$M = 10$$

keys = (25, 39, 46, 55, 89, 23, 68, 70, 94, 78)

$$h_{j_1}(\text{key}) = \text{key mod } m$$

$$h_{j_2}(\text{key}) = 1 + (\text{key mod } (m-1))$$

CRT = Double hashing

$$DH(\text{key}, i) = (h_{j_1}(\text{key}) + i \cdot h_{j_2}(\text{key}))$$

$$\forall i = (0 \text{ to } M-1)$$

0	70	$DH(25, 0) = 5$
1	89	$DH(39, 0) = 9 + 0 \cdot 8 = 9$
2	78	$DH(46, 0) = 6 + 0 \cdot 8 = 6$
3	55	$DH(55, 0) = 5 + 0 = 5 \text{ fail}$
4	94	(4) $DH(55, 1) = 5 + 1 \cdot h_{j_2}(55) = 5 + (1 + 55 \bmod 8) = 5 + 8 = 13 \bmod 10 = 3$
5	25	
6	46	$D(89, 0) = 9 \text{ (fail)}$
7	23	(4) $D(89, 1) = 9 + (1 + 89 \bmod 8) = 11 \bmod 10 = 1$
8	68	
9	39	$D(23, 0) = 0 \text{ (fail)}$
		$D(23, 1) = 3 + (1 + 23 \bmod 8) = 11 \bmod 10 = 1 \text{ (full)}$

$$D(23, 3) = 7$$

$$D(23, 2) = 3 + 2(8) = 19 = 9$$

$$D(68, 0) = 8 \text{ (pass)} \quad (0)$$

$$D(70, 0) = 0 \text{ (pass)} \quad (0)$$

$$D(94, 0) = 4 \text{ (pass)} \quad (0)$$

$$D(78, 0) = 8 \text{ (fail)}$$

$$(2) D(78, 1) = 8 + 1(1 + 6) = 15 \bmod 10 =$$

$$D(78, 2) = 8 + 2(7) = 22 \bmod 10 =$$

$$= 2 \text{ (pass)}$$

Total collision = 1

Problem 1 Apply all 3 open addressing strategies-

$$m=10$$

key = 25, 15, 90, 75, 69, 59, 44, 58, 85

- if you want to find any element. And if gap occur then element is not present in table.

- 1. We are not wasting space outside in the form of linked list
  - 2. Searching time =  $O(1)$  for Best case  
 $O(m)$  for worst case
  - 3. Insertion time =  $O(1)$  for Best case  
 $O(m)$  for worst case
  - 4. Deletion time =  $O(1)$  for Best case  
 $O(m)$  for Worst case
  - 5. Whenever delete a particular element, place \$ otherwise at a time of searching gap occurs & it may lead to miss searching.

~~if your table~~

- ⑤ If you delete one element, it will create trouble to other elements but we can manage with help of \$ symbol if more dollar symbol than rehash again.

### Quadratic Probing -

1. No space is wasted outside in form of linked list.

2. Search time  $\Rightarrow O(1) = BC$   
 $O(m) = WC$

3. Insertion time  $\Rightarrow O(1) = BC$   
 $O(m) = WC$

3) deletion time  $\Rightarrow O(1) = BC$   
 $O(m) = WC$

4. deletion of one element will trouble to other element but can manage with \$, if more deletion, then rehash again.

### Double Hashing -

0	90	$DH(25, 0) = 5$
1	69	$DH(15, 0) = 5$ fail
2	15	$DH(15, 1) = 5 + 8 = 13 = 3$
3	15	$DH(90, 0) = 90$
4	44	$DH(75, 0) = 5$ fail
5	25	$DH(75, 1) = 5 + 11$
6		
7	59	
8	55	
9	75	

Ex3 - m=10

keys = 53, 60, 62, 82, 91, 80, 90

$$h_f(k) = k \bmod m$$

CRT = linear probing

0	60	$LP(80, 0) = 0+0 = 0$
1	91	$LP(91, 1) = 1+1 = 1$
2	82	$LP(82, 2) = 2+2 = 2$
3	53	$LP(53, 3) = 3+3 = 3$
4	80	$LP(80, 4) = 4+4 = 4$
5	90	$LP(90, 5) = 5+5 = 5$
6	.....	.....
7	.....	.....
8	.....	.....
9	.....	.....

↓  
1 person      mth person

Avg -  $\frac{1+2+3+\dots+m}{m} = \frac{m(m+1)/2}{m} = \frac{m+1}{2}$

Primary clustering: if two keys same initial hash address they both will follow same path unnecessarily. In linear fashion because of this avg. time increases, this problem is known as primary clustering.

\* Primary clustering: if two keys same initial hash address they both will follow same path unnecessarily. In linear fashion because of this avg. time increases, this problem is known as primary clustering.

\* Linear probing is suffering with primary clustering.

$$\text{Avg Case} = 1 + 2 + 3 + \dots + m$$

$$= \frac{m(m+1)}{2} = \frac{m+1}{2} \text{ as it suffers with the problem of primary clustering.}$$

as 2 will suffer with problem of 1.

y using Quadratic Probing

$$O \Sigma = 11$$

$$2 =$$

0	60	$QP(80, 0) = 0$ (fail)
1	91	$QP(80, 1) = 0 + 1 + 1$ (fail)
2	82	$QP(80, 2) = 0 + 2 + 2^2 = 6$ (true)
3	53	
4		
5		$QP(90, 0) = 0$ (fail)
6	86	$QP(90, 1) = 0 + 1 + 1$ (fail)
7		$QP(90, 2) = 0 + 2 + 2^2 = 6$ (fail)
8	90	$QP(90, 3) = 0 + 3 + 3^2 = 12 = 2$ (fail)
9		$QP(90, 4) = 0 + 20 = 0$ (fail)
		and so $QP(90, 5) = 0$ (fail)
		on $QP(90, 6) = 2$ (fail)
		$QP(90, 7) = 8$ (true)

Here primary clustering occurs as what path is followed by.

1 element will definitely followed by 2 element

but here Avg. case deg decrease due to large jump

Here clustering is known as secondary clustering

Secondary Clustering: if two key contain same starting hash address, they both will follow same path unnecessarily in quadratic manner, because of this reason avg. searching time increases (it is less than linear search probing time), this problem is known as secondary clustering

Quadratic probing is suffering with secondary clustering

Avg case =  $O(m)$  but actually it is less than  $(m+1)$

(Asymptotically same but mathematically less)

## Double Hashing:

		80	
0	60		$DH(80, 0) = 0$
1	91		$DH(80, 1) = 1$
2	82		$DH(80, 2) = 2$
3	53		$DH(80, 3) = 3$
4	80		$DH(80, 4) = 4$
5			
6	90		$DH(90, 0) = 0$
7			$DH(90, 1) = 3$
8	80		$DH(90, 2) = 0 + 6$
9	80		

Here path is diff with initially address (same).

Here diff path is carried out with same initial address. No path will be same.

$$DH(100, 0) = 0$$

$$DH(100, 0) = 0 + 5 = 5 \text{ (True)}$$

$$\text{Avg} = \frac{3 + 5 + 7 + \dots + 2}{m} = \frac{m - O(1)}{m}$$

DH

(as a particular slot is covered by single element only)

(It might be possible that there must be some cases which lead to redundancy)

there in avg case, we have to consider for all the given case) so on an avg it is said that it is repeating one time).

In Double Hashing, Even though two key contain same starting hash address, they both will follow diff path because second level hash fn will give diff value.

because of this reason 99% of redundancy is eliminated,  
 slot covered by I element

$$\text{Avg case} = 3 + 5 + 7 + \dots + 2$$

↓

slot covered

by II element

∴ on an avg

$$= \frac{m}{m} \Rightarrow cm = O(1)$$

$$\text{Avg. case} = O(1)$$



(as only few key will be repeating only)

let 4 key repeated

$$\text{total} = \frac{4m}{m}$$

$$= O(4)$$

∴ In double hashing, secondary clustering problem is present littlebit (1/.)

Note - Using Perfect Hashing, you will get Worst case Searching time =  $O(1)$ .

in hashing if collision occur to a, b, c at slot 5, they will go to 5. which have another  $h_5$  which is best  $f^n$  and a hash table to store those three element then collision is removed perfectly.

0	1	2	3	4	5	6	7	8	9

a	b	c
0	1	2

$$\text{space} = O(m^2)$$

You cannot decide of size so use m size else use linked list

so space increases but time decreases

$$\text{Total table} = m+1$$

$$\text{Total } f^n = M+1$$

$$\therefore \text{Worst case} = O(1)$$

m-slots n-keys

m slots - will store n keys

1-slot will store  $\frac{n}{m}$  keys



Load factor - no of keys getting stored in 1 slot

$$\text{Load factor } (\alpha) = \frac{n}{m} \text{ keys per slot}$$

Note 1 - the expected no of probes (attempt) in an unsuccessful search of an open addressing technique is

$$\left[ \frac{1}{1-\alpha} \right]$$

Note 2 - the expected no of probes in successful search of an open addressing technique is  $= \left[ \frac{1}{\alpha} \log_{\frac{1}{\alpha}} \frac{1}{1-\alpha} \right]$

(Cormen)

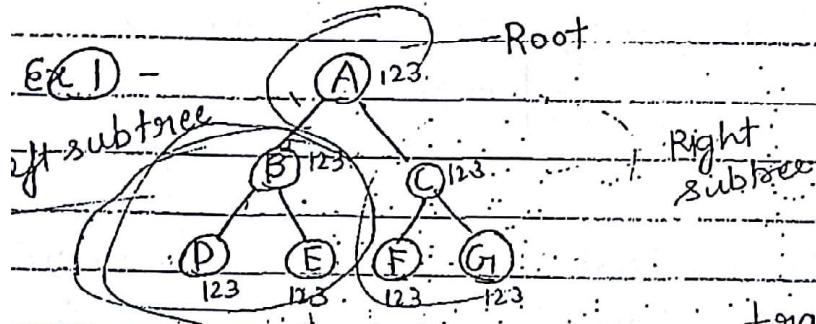
## TREE AND GRAPH TRAVERSALS (2 Marks)

### B Tree Traversals -

1. Preorder (Root LST. RST.)

2. Postorder (LSTRST Root)

3. Inorder (LST Root RST)

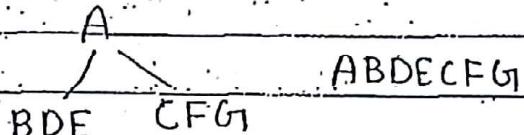


Tree is best solved by recursion as of having almost sym quality.

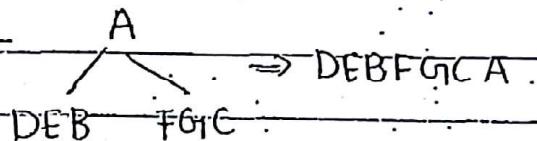
Traversal - visiting each node one by one in given tree.

Preorder Left & Right tree cannot be changed i.e. left will always come before Right subtree  
if changed  $\rightarrow$  3!

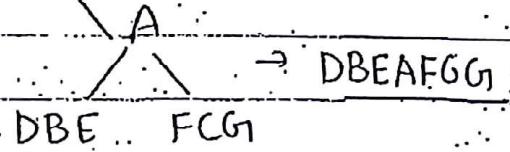
Preorder - ABDEC~~F~~G

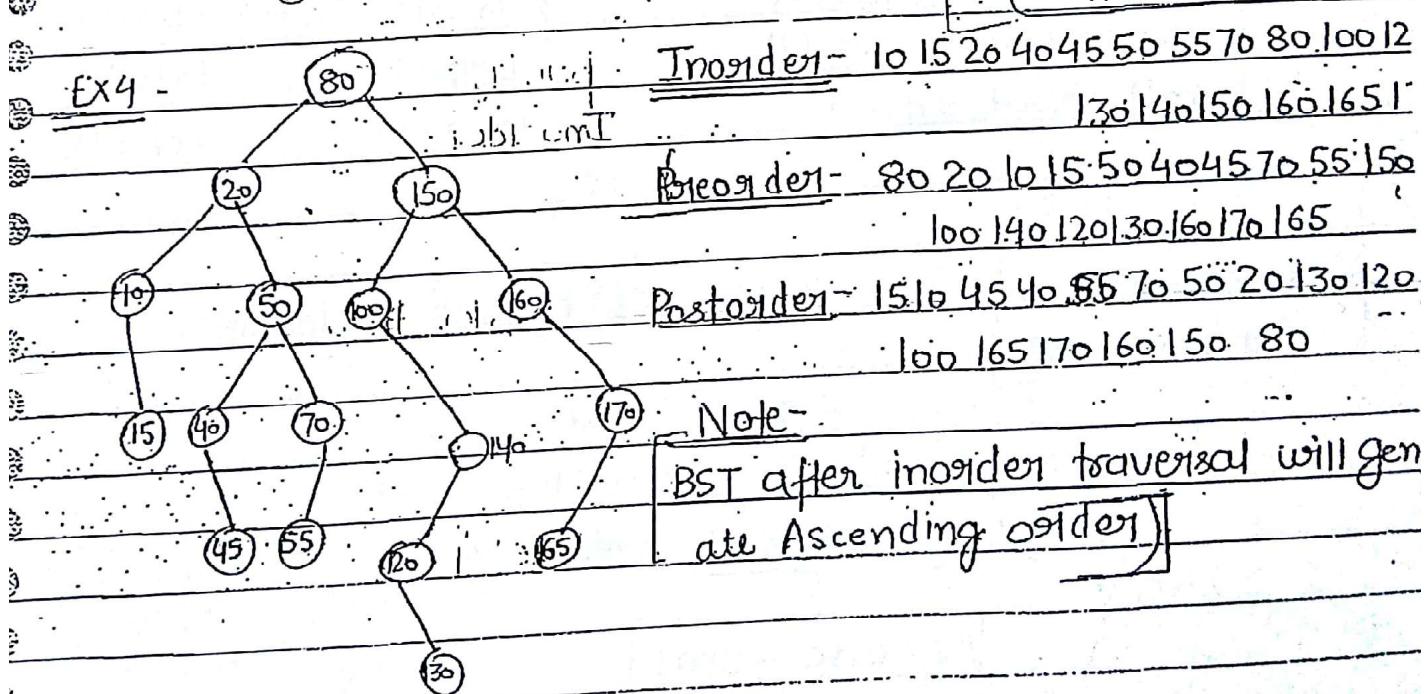
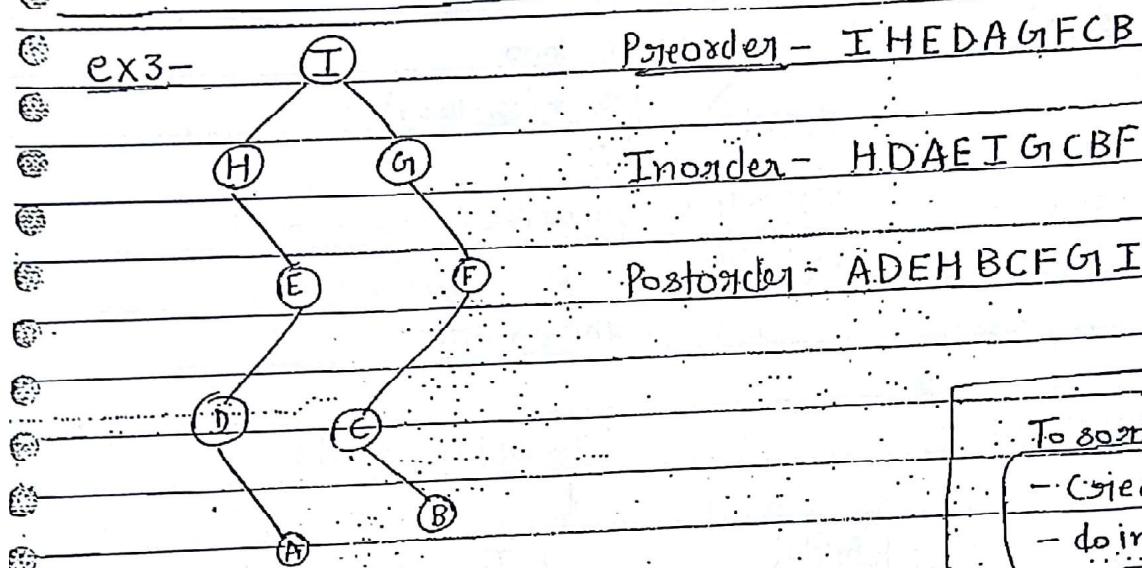
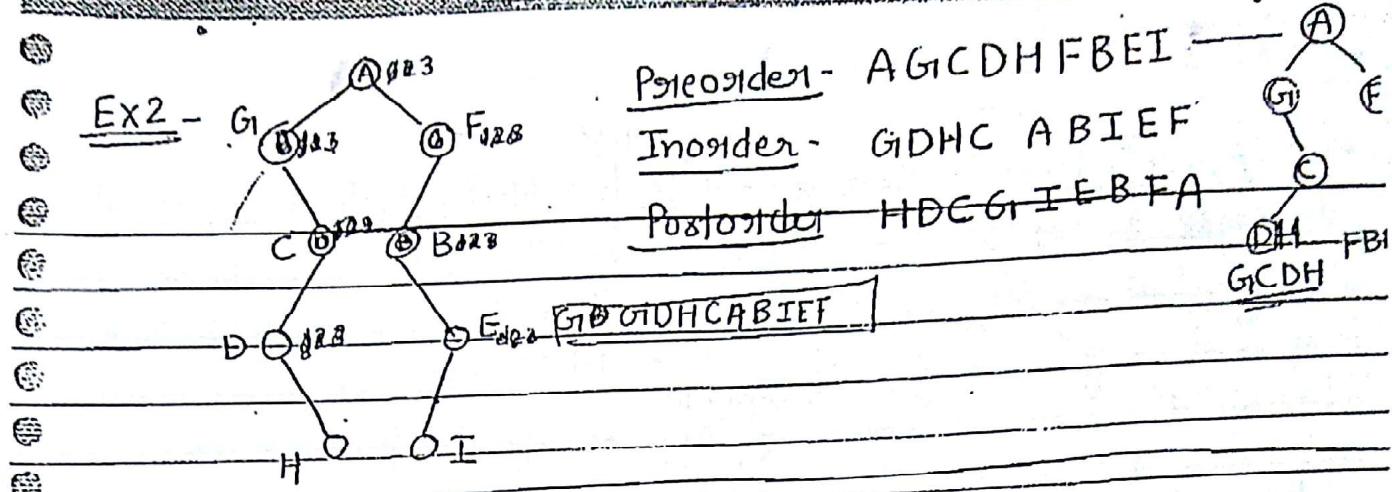


Postorder - DEBF~~G~~CA



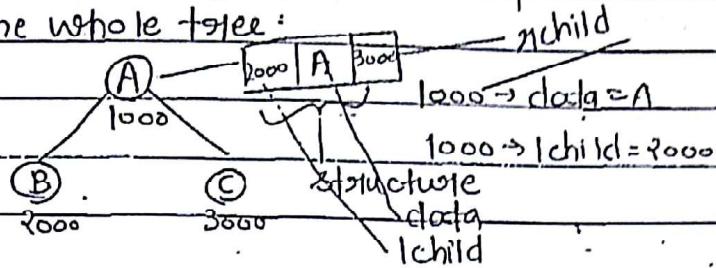
Inorder - DBEAFC~~G~~





BST after inorder traversal will generate Ascending order

Whenever a tree is given, a root address is provided which can be used to traverse the whole tree:



Preorder(root)

```

    {
        printf("root->data");
        O(1)
    }
    preorder(root->lchild); T(n/2)
    preorder(root->rchild); T(n/2)
}
and so on.
    
```

$\text{root} = 1000$

preorder(1000)

{ A

preorder(2000)

postorder(root)

Inorder(root)

```

    {
        postorder(root->lchild);
        postorder(root->rchild);
        printf("root->data");
    }
}
    
```

```

    {
        Inorder(root->lchild);
        printf("root->data");
        Inorder(root->rchild);
    }
}
    
```

for preorder  
of a binary tree

$$T(n) = 2T(n/2) + O(1) \text{ (by master theorem)}$$

= O(n) (Best case)

Worst partition -

$$T(n) = T(n-1) + T(0) + O(1)$$

= O(n) (Worst case)

for preorder,  
postorder,  
Inorder

Avg case = O(n)

WC, BC, AC same as each node has to visit once

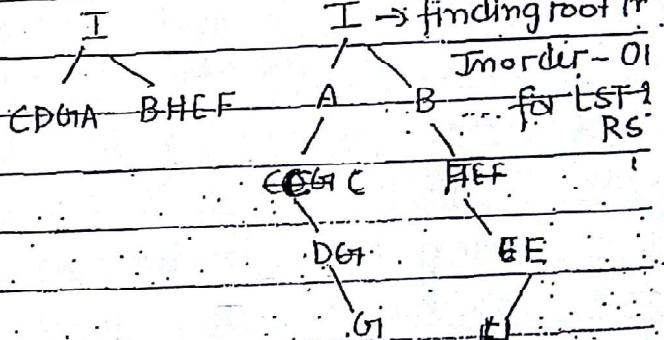
Recurrence relation can also tell about stack space  
 If  $T(n/2) \rightarrow \log n$   
 $T(n-1) = n$

Note -

- Preorder Inorder Postorder will take  $O(n)$  time on  $n$ -node binary tree ( $WC, AC, BC$ ), because each node has to be visited once.
- Stack space.  
 Space complexity =  $O(\log n)$  for balanced tree (Best case)  
 $O(n)$  for unbalanced tree (Worst case)
- Ques - if a BST is given, time to get sorted order -  $O(n)$

- Ques - if a BST is given time to sum element 20 to 50  
 $\rightarrow O(n)$  - Worst case  
 $O(1)$  - Best case (start from 20 when 20 came & add 50)

- Ques - Consider the following binary tree data -  
 $\rightarrow$  finding root  
 Preorder: I A C D G I B F E H  
 Inorder: C D G I A I B H E F  
 Postorder: G I D C A H E F B T



Inorder traversal cannot judge for pre-root because it cannot be balanced always  
 Preorder can give root as well as postorder  
 Inorder is used to give Left subtree & right subtree

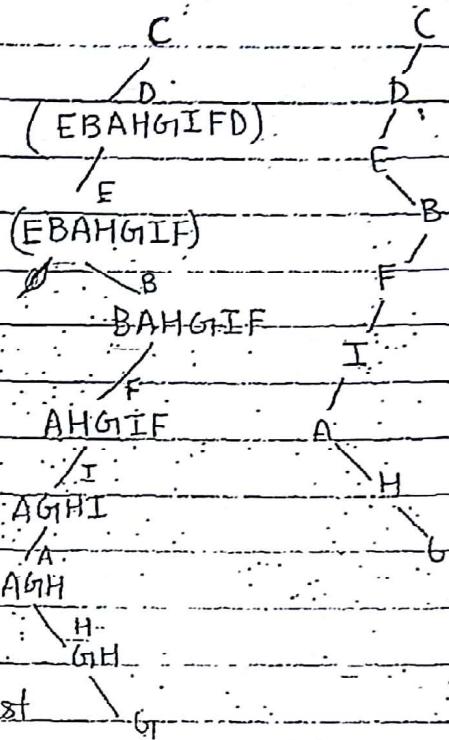
$$\therefore \text{Total complexity} = n \times O(n) \left[ \begin{array}{l} \text{at 1 level} \\ \text{(for LST & RST)} \end{array} \right] \\ = O(n^2)$$

Ques - Consider the following binary Tree data -

Postorder - G H A I F B E D C

Inorder E B A H G I F D C

Preorder - C D E B F I A H G I



X = sometimes possible)

To find out directly any order

find out root

find its pos<sup>n</sup> in inorder case

the digit or no. of element in inorder

try to find them in postorder just

reverse the order and vice versa

a shortcut Not applicable everywhere

Ex - Here postorder & Inorder given, C its position in inorder  
will most so C has empty right ST.

then go to postorder get the element in LST of C in Inorder  
take them & reverse them to obtain Post order.

Note - To construct Binary tree for the given Preorder Inorder  
or Postorder Inorder will take  $O(n^2)$  time (n times linear  
search to find LST & RST in given Inorder)

$$\begin{array}{c} (n^2 + n + \text{?}) \\ \downarrow \\ \text{to find LST} \quad \text{to find root} \\ \text{RST} \quad n \text{ time} \end{array}$$

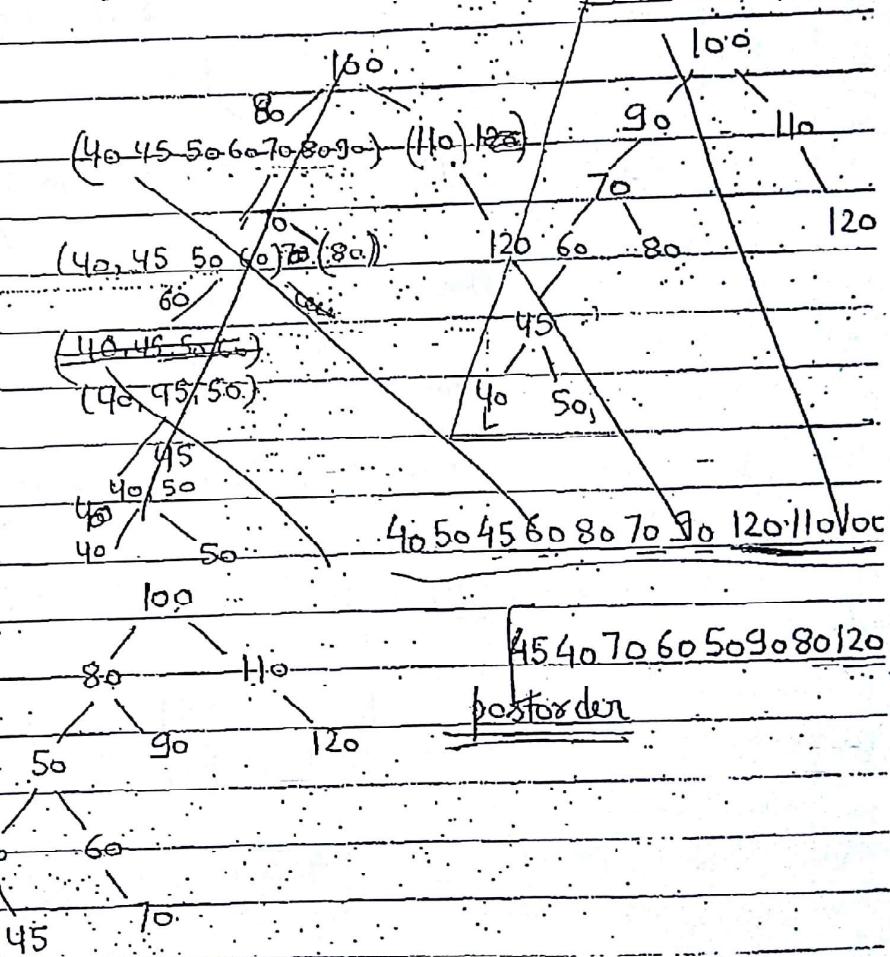
We cannot apply binary search as given sequence is not sorted but if BST it will take  $n \log n$  time.

Ques - Consider the following BST data-

Page 100 (80 50 40 45 60 70 90 110 120)

Post: 90 70 60 45 40 50 80 120 110 100

Im- (40 45 50. 60 70 80 90 100 110 120 (A.O))



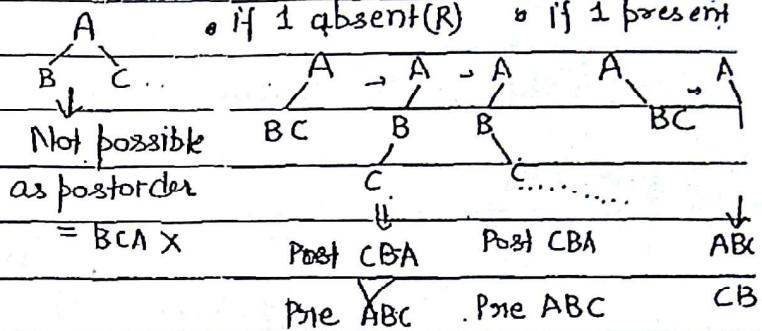
Note → for the given preorder inorder for postorder inorder to construct equivalent BT require  $(n \log n)$  time if inorder is already sorted.

preorder: ABC  
NLR

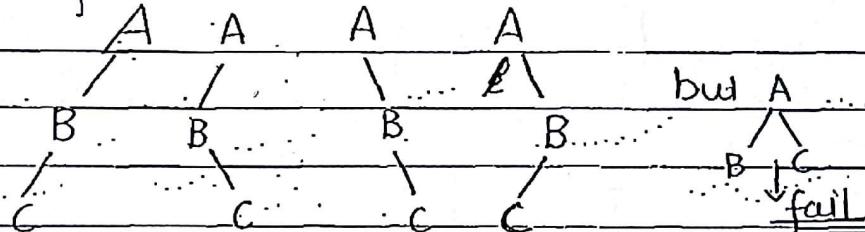
root = A

• Let both RST & LST present

postorder: CBA  
RLN



So possible trees are



Note

If Preorder Inorder given - unique binary tree  $\xrightarrow{\text{Binary tree}}$   
postorder Inorder given - Unique Binary tree  $\xrightarrow{\text{Unique BT}}$   
if preorder postorder given - more than one binary tree  $\xrightarrow{\text{Binary tree posh but not}}$

To construct unique binary tree, Inorder is required (compulsory)

When pre, post is given, manual checking is done To each node two possibilities either going to left or right so total possibility for n nodes =  $O(2^n)$

Upper bound as root will have only one possibility.

To find out cycle initially  
 flag = 0 then if visited  
 flag = 1 if again count<sup>th</sup>  
 flag = 1 then it is a cycle.

## Graph Traversal

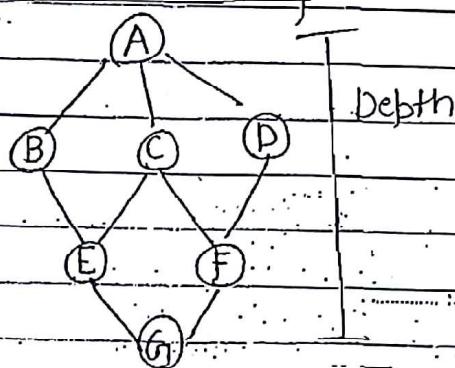
1. BFT (Breadth First Traversal)

2. DFT (Depth First Traversal)

Total Graph Traversal time =  $O(V+E)$

Breadth

By visiting each vertex & each edge



We do not consider edge in tree & edge is less than 2 vertices

$$\text{Tree} = (V+E) \times$$

$$V + V - 1$$

$$= 2V - 1 \approx O(V)$$

Breadth first traversal - ABCDEFG

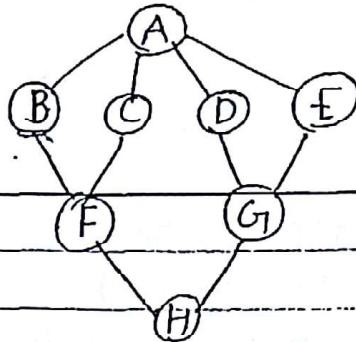
Depth " " ABEGFCFD

Breadth first Traversal - Tree traversal are unique but graph traversal not.

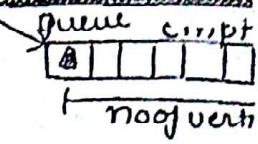
- Visited is required as graph contains cycle and there is no mean of visiting a vertex again & again.

- Queue is used in BFT as we need FIFO structure.

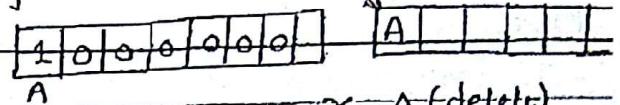
while loop is covering vertex & for loop is covering for edges.



Initially  
Visited = 0 0 0 0 0 0 0 0  
Starting with A



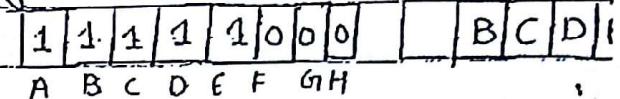
Ex



x = A (delete)

w = B C D E

Visited



B deleted

w = A F

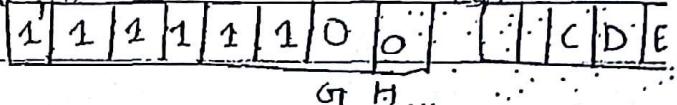
↓ add

Visited[V] = 1

add(V, Q);

while (Q is not empty)

Visited



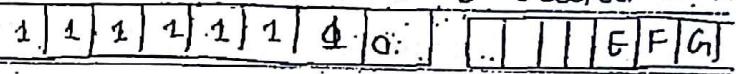
x = delete(Q)

print(Q)

for all w adjacent to x



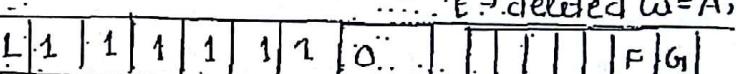
C deleted w = A



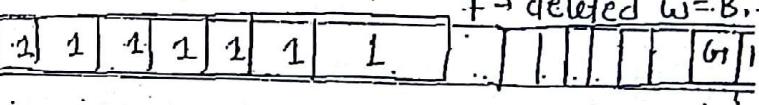
D deleted w = A

if (visited[w] = 0)

visited[w] = 1



add(w, Q)

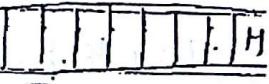


E deleted w = A

F deleted w = B

O/P → A B C D E F G H

(Order of deletion of  
element)



empty

loop end

Note -

- To implement BFT, we use queue data structure.

- Total complexity =  $O(V+E)$  (Time) (BC, WC, AC)

[If adj matrix is used  $TC = O(V^2)$ ]

Space  $\rightarrow$  I/P + extra

$\downarrow$        $\downarrow$       if graph is adjacency list  
 $(V+E)$      $(V+E)$   
queue      visited array

$$= (V+E+2V)$$

$$= 3V+E$$

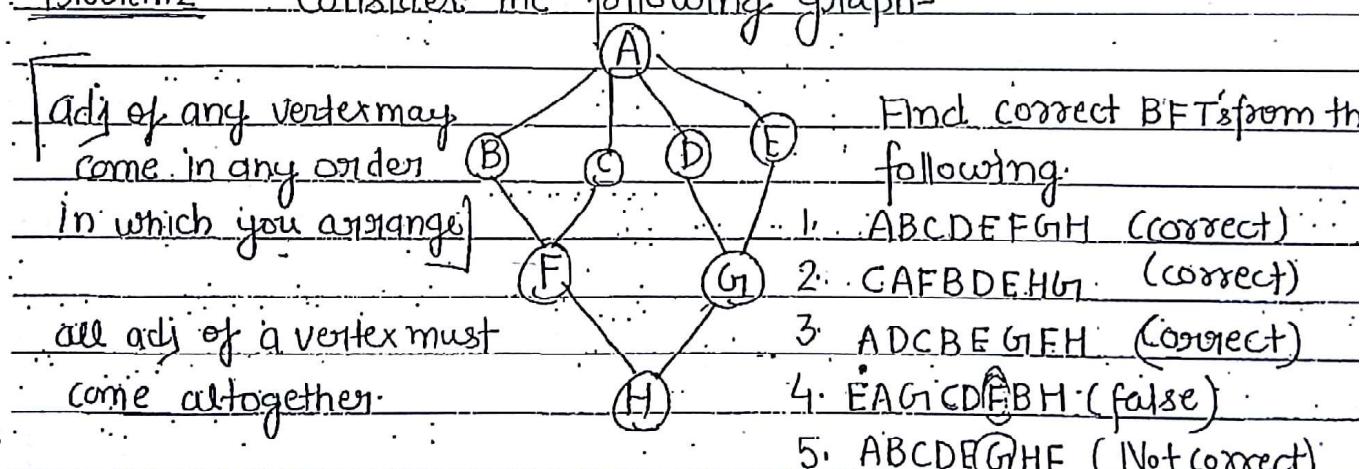
$$= O(V+E)$$

$= O(V^2)$       if adj. matrix is used ]

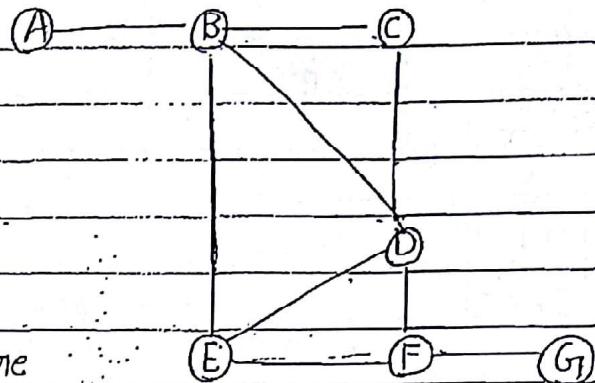
- BFT is also known as Level order traversal i.e level by level printing of nodes. (means adjacency lists)

A  $\rightarrow$  next level (adj of A)

Problem 2 - Consider the following graph-



Ques - Consider the following graph.



Whether these are

correct BFTs or not

(a) EBD~~E~~CA~~G~~I (Correct)

(b) G~~F~~CEDBA (incorrect)

(c) CBDAEFGI (Correct)

(d) DBCEFA~~G~~I (Correct)

### Application of BFT -

Ques - If a graph is given, how to check whether it is connected or not?

• Draw. Apply BFT

• Check visited array

• if any 0 in visited array

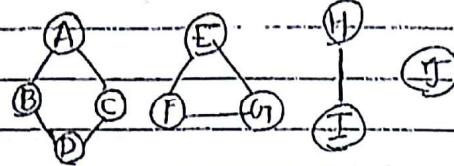
↓ to check for 0's

• then graph disconnected.  $O(V+E) + V$

If a linked list is given, to check whether it is connected?

A linked list is also a graph but directed so apply the same procedure ~

We can verify given graph is connected or not.



1	1	1	1	0	0	0	0	0
A	B	C	D	E	F	G	H	I

starting from A

no of disconnected component  
Count = 1 (initially zero)

When you get 1 zero, just stop & again apply the D.F.T again and so on, you can find disconnected components can be find out

again

1	1	1	1	1	1	1	1	1
A	B	C	D	E	F	G	H	I



again

~~1~~  
~~E~~  
~~D+F~~  
~~A+B+C~~

1	1	1	1	1	1	1	1	1
A	B	C	D	E	F	G	H	I



Count=4

No of connected components = No of time BFT applied

T.C to find connected comp = O(V+E)

Using BFT we can find out connected components in the given graph.

If a null graph  $G(V, E)$  - no of connected component =  $|V|$

3. to check for cycle in a given graph

- Apply BFT

→ If  $\text{visited}[i] = 1$  & it encounters again then there is a cycle.

Time complexity =  $O(V+E)$  (as BFT work)

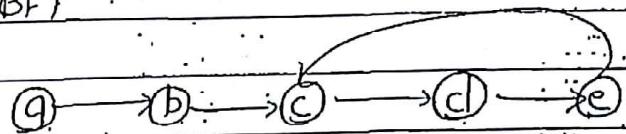
4. Whenever Kruskal is going on edges are added to MST we have to check for occurrence of cycle too as no cycle must occur, then by side BFT also working to check for  $\text{visited}[i]$  → it takes constant time (for 1 edge)

$$T(n) = E \log E + (V+E)$$

↓ for BFT cycle check

3. Using BFT, we can check given graph contain cycle or not

if a linked list contain cycle, it can be checked easily by applying BFT

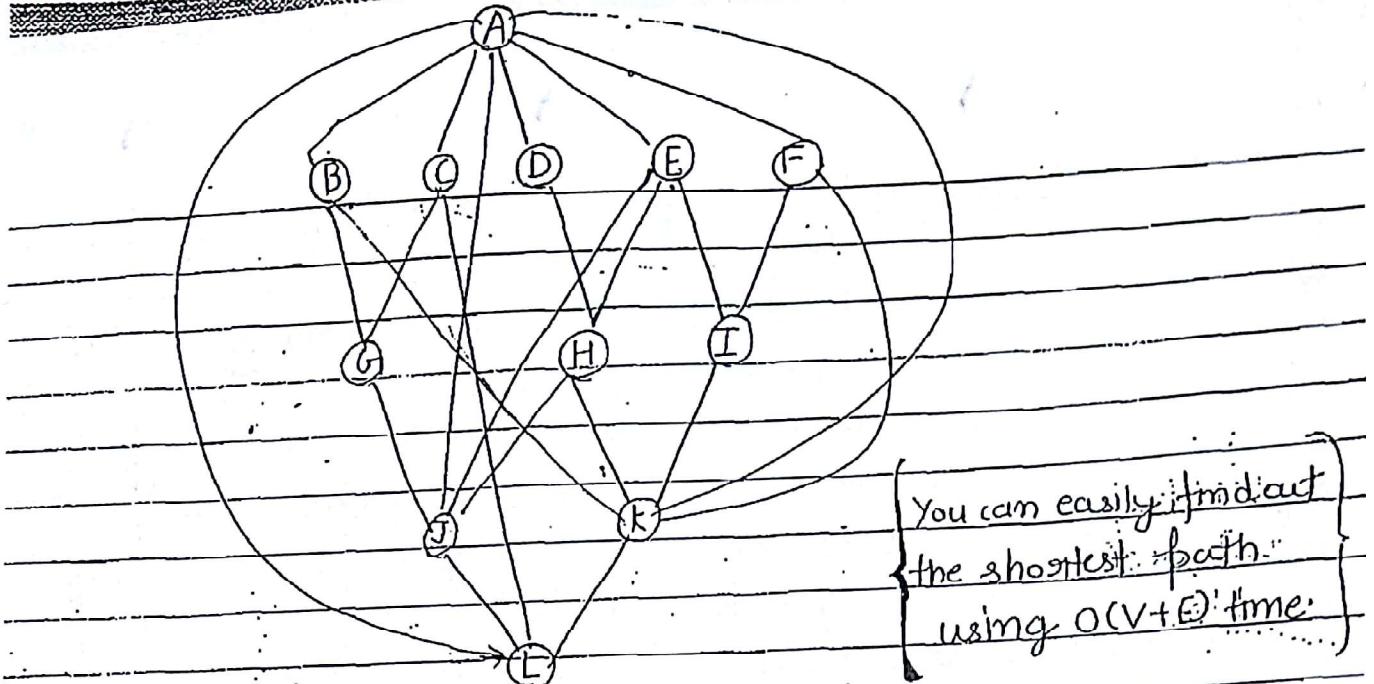


A B C D E

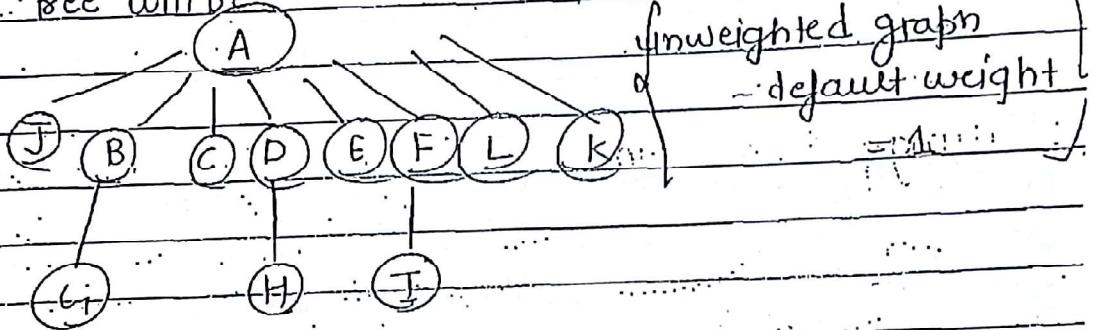
↑ diagram occurs ∴ cycle exists

4. If an unweighted graph is given & single source shortest path is not to be found, then BFT is the efficient one algorithm as it will find the shortest path.

∴ Using BFT, we can find out shortest path from given source to every vertex in given unweighted graph or the graph having equal edge weight (true)



the BFT tree will be



Ques- if a unweighted graph is given; the shortest path to be calculated from given source. which data structure is used?

Queue.

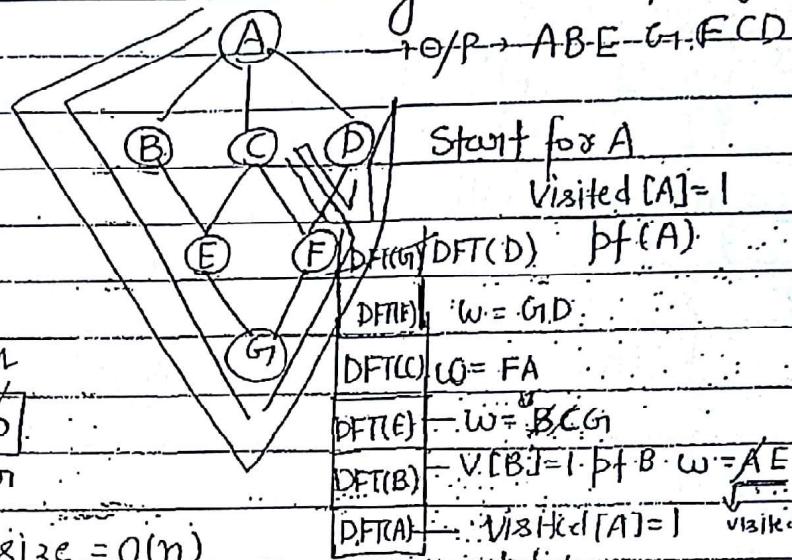
5. Using BFT we can verify a given graph is Bipartite graph?

## Depth First Traversal

- It uses stack as recursion occur here.
- it will also maintain a visited array to check for cycle.

size V

$\rightarrow$  O/P  $\rightarrow$  AB-E-G-FCD



and so on  
stack size =  $O(n)$

1. To implement DFT we are using stack.

2. Time Complexity = no of functional calls \* work done at each func call

$$= V + E \text{ (as if work is summed up)}$$

$$= O(V+E)$$

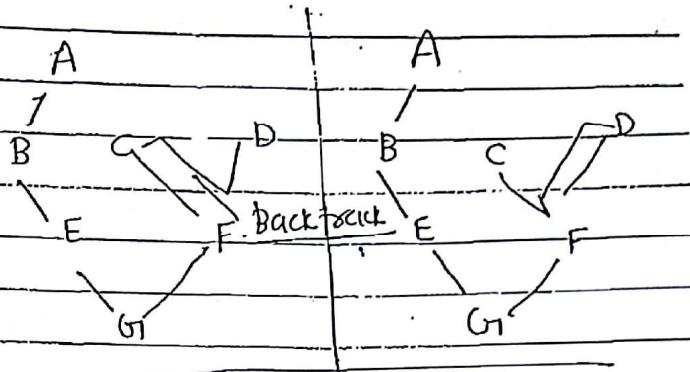
3. Space Complexity = i/p + extra

$$= V+2E \quad \downarrow \quad \text{stack visit} = O(V+E)$$

$V+2E \quad V \quad V \quad \downarrow \text{for adjacency list}$

DFT(V)

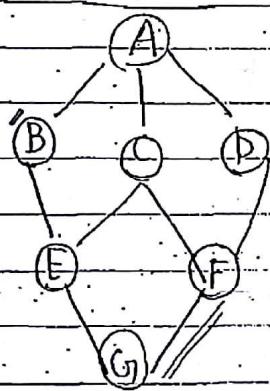
1. Visited[V] = 1
2. DFT(V);
3. for all w adj to V
  1. if (w is not visited)
    2. DFT(w);



In case of Adjacency Matrix =  $O(V^2)$

(space as well as time complexity)

Problem:- Consider the following graph-



Find Correct DFT's

(a) ABEGFCD

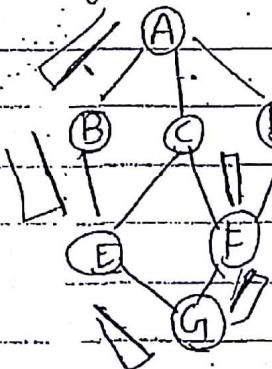
You can take  
any of the  
adjacent of  
a particular  
element.

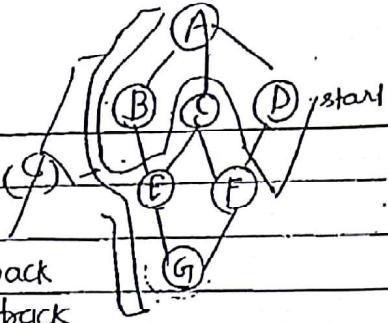
No of backtrack

(b) LABECFDG

backtrack  
occurs  
here

Total Backtrack = 6





DFCEBAG (correct)

• ? back  
• back

(d)

CFDABEG (No Backtrack)

(correct)

(Stack Size = n)

[max stack size = O(n)]

↓ When no back track occurs

⑥ (e) GIEGFABD: (Incorrect) correct - GIECEDAB

(f) ACFGEBD (correct)

(g) BACFDEG → (fail) Incorrect) correct - BACFDGIE

because at D when possibilities ended we back

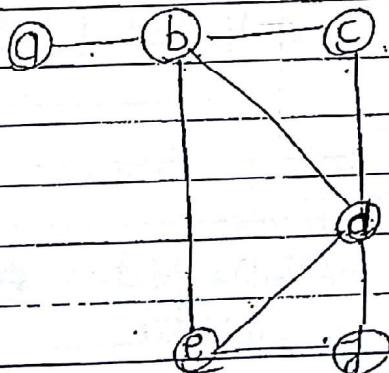
track to F & F has G to be done but it is  
first completing G.

B) [Backtracking is done when all possibilities ended]

Backtracking is nothing but popping

In DFT, if I print D after B, what is the relation b/w them  
No relation because of backtracking anything  
possible in DFT but BFT have adjacency.

Ques - Consider the following graph -



check whether they are DFT  
or ~~not~~ or not.

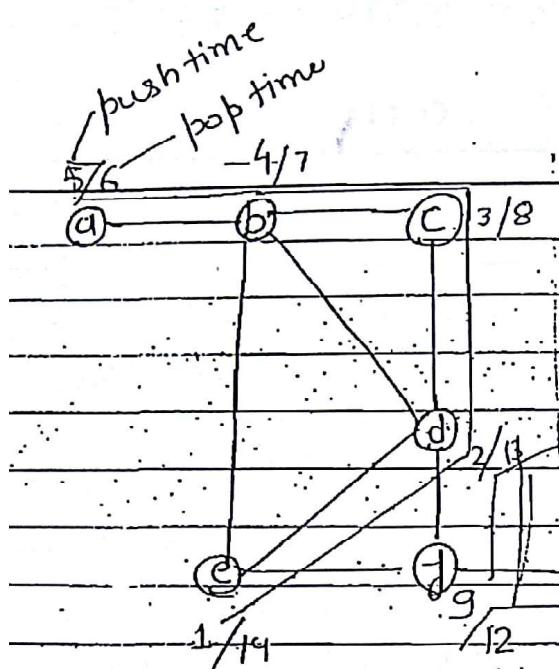
1) edcbafg (DFT)

2) fdebagc (incorrect)

3) abdfgce (incorrect)

4) dcbaefga (correct)

5) befgdca (DFT)



e d c b a f g

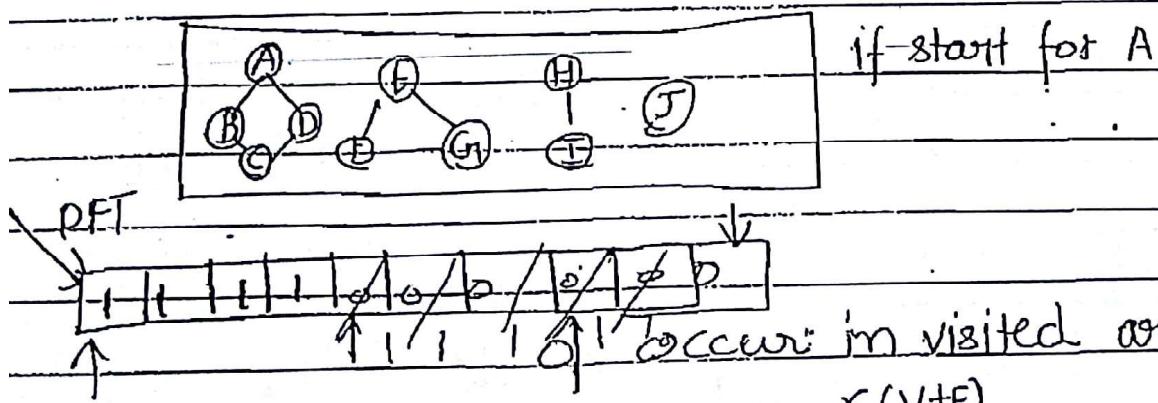
how many elements where  
the diff b/w push & pop  
time is 1 (immediate  
back track)

Total element = 2 (q,g)

1/14 2/13 3/12 4/9 5/8 6/7 10/11  
d c b e f g a  
1/14 2/11 3/10 4/5 6/9 7/8 12/13  
b e f g d c a

## Applications of DFT-

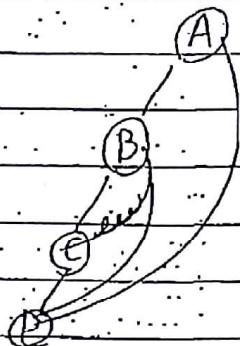
will check where graph is connected or not



3. We can verify given graph contain cycle or not  
 $O(V+E)$

3. We can find out no of connected components  
count = no of time DFT applied.

4



Here shortest path = ~~(A-B)~~ (1)

but if we take

A

B

C

D

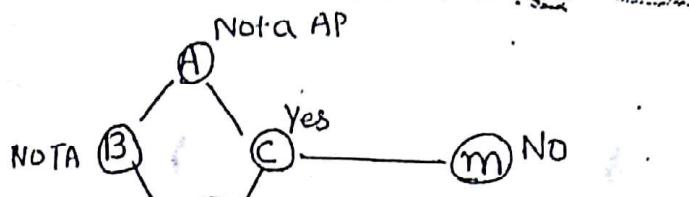
= 3 length path

So, DFT cannot (may or may not) work finding single source shortest path for given unweighted graph.

5. Using DFT, we can check given directed graph is strongly connected or not?

6. Using DFT we can verify a vertex is articulation point or not.

Articulation point - if the deletion of an ~~edge~~ ~~leaf~~ vertex ~~as~~ along with its edge lead to disconnection of graph known as articulation point.



To known whether a graph have vertices any articulation point

1) Apply DFT on given graph

2) Delete that vertex & edge.

3) Again apply DFT

4) If Visited contain 1 for all except that delete vertex

then that is an articulation point

$$\text{Total Complexity} = O(V+E) + O(V+E)$$

$$= O(V+E)$$

## Bipartite Graph using BFT

first time visited in one set

second time another set

(level by level storing of elements)

strongly connected using DFT

finding strongly connected component

You can use the concept of strongly connected graph articulation point.

No of articulation point = No of strongly connected compn  
cnt + 1

$$\text{Total Complexity} = E(V+E+V+E)$$

for articulation point

$$= EV + E^2$$

$$= O(E^2)$$

## Programming - I

Array - 1-D Array

Ex - 1

`int a[10] = { 10, 20, 30, 40, ..., 100 }` a will store base address.

↳ declaration (memory creation)

`a 1000 1002 1004 1006 1008 1010 1012 1014 1016 1018`

10	20	30	40	50	60	70	80	90	100	1000	1002	1004	1006	1008	1010	1012	1014	1016	1018
0	1	2	3	4	5	6	7	8	9	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009

continuous

(0-19)

1 to 7 → how many elements =  $7 - 1 + 1 = 7$

Ex 2

-5 -4 -3 -2 -1 0 1 2 3 4       $4 - (-5) + 1 = 10$  elements

$2 - (-2) + 1 = 5$  elements

print `a = 1000` (base address)

5 element occupy

1010 1011

$$\text{Loc}[a[5]] = \text{base address} + \cancel{\text{size of data type}} * (5 - 0) \\ = 1010$$

\* 1010 → to access that particular M/M location

$$\text{Loc}[a[9]] = \text{base address} + \text{size of int} * (9 - 0) \\ = 1000 + 2 \times 9 = 1018$$

(only 9-0 bco  
you dont use  
for 10th  
element)

Ex 2 -

$$a[55 \dots 550] \quad \text{Total element} = 550 - 55 + 1 = 496$$

Base addres = 990

Size of an element = 10 byte (C)

$$\text{Loc}[a[450]] = \text{Base address} + C * [450 - 55]$$

$$= 990 + 10 * 395$$

$$= 3950 = \underline{\underline{4940}}$$

before 450 from  
55

- array must be continuous no matter whether indices are (+)ve or (-)ve.

Ques  $a[-55 \text{ to } 55]$  BA = 1000 C = 5

$$\begin{aligned}\text{Loc}[a[5]] &= \text{BA} + [5 - (-55)] \times 5 \\ &= 1000 + 60 \times 5 = 1000 + 300 \\ &= 1300\end{aligned}$$

ex4  $a[lb \dots ub]$  BA = Base address

lower bound      upper bound      size = c  
total element = ub - lb + 1

$$\boxed{\text{Loc}[a[i]] = \text{BA} + c * [i - lb]}$$

array index when started from 0, no need to do subtract at time to calculate location but when it start with 1, a subtract has to be done. So less time is required to calculate the value when starting from 0.

~~offset calculation~~

$$\begin{aligned}&\text{Loc}[a[1 \dots 10]] \quad \text{Loc}[a[5]] = \text{BA} + (5-1) * c \\ &\text{Loc}[a[0 \dots 10]] \quad \text{Loc}[a[5]] = \text{BA} + (5-0) * c\end{aligned}$$

• Array index is, by default, will start from 0 only because no need to calculate offset value

row  
↓  
(0=3)      (0=4)

column  
→ (5-1)+1 = 5 column

	1	2	3	4	5
int a[4][5]	a <sub>11</sub>	a <sub>12</sub>	a <sub>13</sub>	a <sub>14</sub>	a <sub>15</sub>
	a <sub>21</sub>	a <sub>22</sub>	a <sub>23</sub>	a <sub>24</sub>	a <sub>25</sub>
total row ✓	a <sub>31</sub>	a <sub>32</sub>	a <sub>33</sub>	a <sub>34</sub>	a <sub>35</sub>
(4-1)+1	a <sub>41</sub>	a <sub>42</sub>	a <sub>43</sub>	a <sub>44</sub>	a <sub>45</sub>
= 5 Row	a <sub>51</sub>	a <sub>52</sub>	a <sub>53</sub>	a <sub>54</sub>	a <sub>55</sub>
Total element = 20 element					

In m/m 40 bytes for continuous m/m (no separate row & colur is present)

1000	02	04	06	08	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	1000
a <sub>11</sub>	a <sub>12</sub>	a <sub>13</sub>	a <sub>14</sub>	a <sub>15</sub>	a <sub>21</sub>	a <sub>22</sub>	a <sub>23</sub>	a <sub>24</sub>	a <sub>25</sub>	a <sub>31</sub>	a <sub>32</sub>	a <sub>33</sub>	a <sub>34</sub>	a <sub>35</sub>	a <sub>41</sub>	a <sub>42</sub>	a <sub>43</sub>	a <sub>44</sub>	a <sub>45</sub>	1000
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	1000

Row Major Order - When elements are getting stored row by row

Ex -

a. C follows Row Major Order

$$\text{Loc}[a[4][3]] = 1000 + [4-1] \times 2 \quad 1000 \quad 4 \times 2 + 2 \quad 2(5 * (4-1) + (3-1)) \\ 17 + 2 \quad 21$$

$$= BA + C [ \text{Column} \times (x - lb_1) + (y - lb_2) ]$$

$$= BA + C [(ub_2 - lb_2 + 1)(x - lb_1) + (y - lb_2)]$$

$$\text{Loc } a[2][5] = BA + C [ 5 * [2-1] + (5-1) ]$$

$$= 1000 + 2 [ 5 + 4 ]$$

$$= 1018$$

$$\text{Loc } a[i][j] = BA + C [ (ub_2 - lb_2 + 1)(i - lb_1) + (j - lb_2) ]$$

lb<sub>1</sub> ub<sub>1</sub> lb<sub>2</sub> ub<sub>2</sub>

visit correct row.

visiting column

Ex-2

$$a[25 \dots 750][80 \dots 150] \quad c = 10 \text{ byte}$$

Raw Major Order BA = 1000

$$\text{Loc}(a[550][140]) = \text{BA} + c[(150-80+1) * (750-25)(550-25) \\ + (140-80)]$$

$$= [1000 + 10[71(525) + 60]]$$

$$= 1000 + 10[37335]$$

$$= 373350 + 1000$$

$$= 374350$$

$$51 \quad 101$$

Ex 3 :  $a[-25 \dots +25][-50 \dots 50]$ , BA = 0, c = 1 Byte : RMO

$$\text{Loc } a[20][30] = \text{BA} + c[(50+50+1)(20+25) + (30+50)]$$

$$= 0 + 1(101 * 45 + 80)$$

$$= 4545 + 80$$

$$= 4625$$

Ex 4 -  $a[lb_1 \dots ub_1, lb_2 \dots ub_2]$

$$\downarrow \quad \downarrow \\ ub_1 - lb_1 + 1 \quad ub_2 - lb_2 + 1$$

$19[47][3]$

BA, C

$\cup n_c \text{ (no of column)}$

Row Major order  $a[i][j] = \text{BA} + c[(ub_2 - lb_2 + 1)(j - lb_1) + (i - lb_1)]$

Column Major order  $a[i][j] = \text{BA} + c[(ub_1 - lb_1 + 1)(j - lb_2) + (i - lb_1)]$   
(no of rows)

26-Mar

Column Major Order  $a[1..4][1..5]$

a) 1 2 3 4 5

1 |  $a_{11} \ a_{12} \ a_{13} \ a_{14} \ a_{15}$

2 |  $a_{21} \ a_{22} \ a_{23} \ a_{24} \ a_{25}$

3 |  $a_{31} \ a_{32} \ a_{33} \ a_{34} \ a_{35}$

4 |  $a_{41} \ a_{42} \ a_{43} \ a_{44} \ a_{45}$

$$BA = 1000 \quad C = 10$$

$a_{100}$	$a_{02}$	$a_{04}$	$a_{06}$	$a_{08}$	$a_{0D}$	$a_{12}$	$a_{14}$	$a_{16}$	$a_{18}$	$a_{20}$	$a_{22}$	$a_{24}$	$a_{26}$	$a_{28}$	$a_{30}$	$a_{32}$	$a_{34}$	$a_{36}$	$a_{38}$
$a_{11}$	$a_{21}$	$a_{31}$	$a_{41}$	$a_{12}$	$a_{22}$	$a_{32}$	$a_{42}$	$a_{13}$	$a_{23}$	$a_{33}$	$a_{43}$	$a_{14}$	$a_{24}$	$a_{34}$	$a_{44}$	$a_{15}$	$a_{25}$	$a_{35}$	$a_{45}$
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Columns 1      Columns 2      Columns 3      Columns 4      Columns 5

$$\begin{aligned} \text{Loc}[a[4][3]] &= BA + C[(3-1)*(\text{no.of Rows} + (4-1))] \\ &= 1000 + 10^2(2 \times 4 + 3) \\ &= 11022 \end{aligned}$$

$$\begin{aligned} \text{Loc}[a[2][5]] &= 1000 + 10^2[(5-1)*\text{no of Row.} + (2-1)] \\ &= 1000 + 2[4 \times 4 + 1] \\ &= 1034 \end{aligned}$$

both are same (Row Major, Column Major)

if no order is mentioned, go for Row Major (by default)

Ex2 -  $a[-75..125][80..150]$  BA=0 C=10 byte

CMO

$$\begin{aligned} \text{Loc}[a[15][130]] &= 0 + 10[(130-80)*(125+75+1) + (15+19)] \\ &= 10(50 * 201 + 90) \\ &= 10(10050 + 90) = 100640 \\ &= 101400 \end{aligned}$$

$$nr = lb_1 - lb_1 + 1$$

Ex3 -   $a[lb_1 \dots ub_1][lb_2 \dots ub_2]$  BA, C, CMo

$$\downarrow$$

$$nc = ub_2 - lb_2 + 1$$

$$\left\{ \text{Loc}[a[i][j]] = BA + c[(j - lb_2) * nr + (i - lb_1)] \right\}$$

3-D Arrays - collection of 2-D arrays

$a[3 \dots 9][15 \dots 35][8 \dots 50]$

$$\begin{array}{ccccccc} & \downarrow & & \downarrow & & & \\ 9-3+1 & 35-15+1 & & & 50-8+1 & \Rightarrow & 7 \text{ table of size } 21 \times 43 \\ = 7 & & & & \downarrow & = 7 [2-\text{D array of size } 21 \times 43] \\ & & & & 43 & & \\ & & & & \downarrow & & \\ & & & & \text{no of rows} & & \text{no of column} \end{array}$$

BA, Base Address = 1000 C = 10, RMo

$$\begin{aligned} \text{Loc}[a[7][30][40]] &= BA + c[(7-3)(nc * nr) + (30-15) * nc + (40-8)] \\ &= 1000 + 10(4 * 21 * 43 + 15 * 43 * 32) \\ &= 43890 \end{aligned}$$

(Mo)

$$\begin{aligned} \text{Loc}[a[7][30][40]] &= BA + 10[(7-3) * nc * nr * (40-8) * nr + (30-15)] \\ &= BA + 10 \end{aligned}$$

$$\begin{aligned} &= 1000 + 10(4 * 21 * 43 + 8 * 21 * 15) \\ &= 1000 + 10(3612 + 168 + 15) \\ &= 1000 + 10(3795) \\ &= 38950 \end{aligned}$$

Loc

Ex  $a[lb_1, ub_2] \quad a[lb_1, ub_1, lb_2, ub_2, lb_3, ub_3]$   
 BA, C

$$n_1 = ub_2 - lb_2 + 1$$

$$n_c = ub_3 - lb_3 + 1$$

$$\text{no of } 2D = lb_1 - lb_1 + 1$$

(R, M, O)

$$\text{loc}[a[i][j][k]] = BA + C [(i-lb_1) * nc * n_1 + (j-lb_2) * nc * (k-lb_3)]$$

CMo

$$\text{loc}[a[i][j][k]] = BA + C [(i-lb_1) * nc * n_1 + (k-lb_3) * n_1 + (j-lb_2)]$$

Lower Triangular Matrix -

	1	2	3	4
1	$a_{11}$	$a_{12}$	$a_{13}$	$a_{14}$
2	$a_{21}$	$a_{22}$	$a_{23}$	$a_{24}$
3	$a_{31}$	$a_{32}$	$a_{33}$	$a_{34}$
4	$a_{41}$	$a_{42}$	$a_{43}$	$a_{44}$

$a_{11}$	0	0	0
$a_{21}$	$a_{22}$	0	0
$a_{31}$	$a_{32}$	$a_{33}$	0
$a_{41}$	$a_{42}$	$a_{43}$	$a_{44}$

↓ 16. element

$$G = \text{zero}$$

$a_{ij}$  belong to upper part if

$j > i$

$(n_2)$  approx - zero

↓ to save spa

) return 0 at that

why we should store

time

zero

Here store in row Major order -

row - 1 will have 1 element -

row - 2 will have 2 element -

## on Major Order-

$a$	1000	100 <sup>2</sup>	04	06	02	40	12	14	16	.18
	911	921	912	931	932	933	941	942	943	944
	0	1	2	3	4	5	6	7	8	9

↓  
1001    1002    1003    1004    1005    1006    1007    1008    1009    10010

$$\begin{aligned}
 \text{Loc}(a[4][3]) &= BA + C \left[ (4+3+2+1) (4-1) (3+2+1) + (3-1) \right] \\
 &\quad \left( \text{sum of 3 natural no.} \atop \text{atmno.} \right) \\
 &\quad \left( 4-1 \right) \text{nat.no.} \\
 &= BA + C [ \text{sum of 3 natural no.} + (3-1) ] \\
 &= 1000 + 2 [ 6+2 ] = 1016
 \end{aligned}$$

[All triangular Matrix are square Matrix.]

$A[25 \cdot 85, 25 \cdot 85] \quad BA = 1000$  (starting row no & column no is also same)  
 $C = 10 \text{ Byte } RM6 \text{ LT.M (Lower Triangular Matrix)}$

$$\begin{aligned}
 \text{Loc}[a[60][55]] &= BA + c \left[ \text{sum of } (60-25) \text{ natural no} + (55-25) \right] \\
 &= 1000 + \frac{f(60-25+1)}{2} \\
 &\stackrel{!}{=} BA + c \left[ \text{sum of } (60-25) \text{ natural no} + (55-25) \right] \\
 &= 1000 + c \left[ \frac{(60-25)(60-25+1)}{2} + 30 \right] \\
 &= 7600
 \end{aligned}$$

## formula

$$\text{Loc}[a[ub_1 \dots ub_j; ub_2 \dots ub_k]] = BA \dashv C \quad \text{RMO, LTM}$$

$$\text{Loc}[a[i][j]] = BA + c \left[ \frac{\sigma(i-lb_1)(i-lb_1+1)}{2} + (j-lb_2) \right]$$

$$a[4][3] = BA + c [ \text{sum of 3 natural no} + (\text{total rows} - (3-1)) ]$$
$$\dots = 1000 + 10 [ 6 + 2 ]$$
$$= \underline{1016}$$

~~o  $a[25 \dots 85, 25, 85]$  BA = 1000 C = 10 Byte~~

~~CMO LTM  
Rows = 61 Column =~~

~~Loc  $a[60, 55]$  = BA + c [ sum of (55-25) natural no~~

~~+ no of rows - (55-1)~~

$$35 \dots = 1000 + 10 [ 35 \times 18 + 61 - (55-25-1) ]$$
$$18 \dots = 1000 + 10 [ 63 @ \dots + 27 ]$$
$$8 \dots = \underline{7570}$$

CMO  $(a[i][j] = BA + c [ \text{sum of } (j-1)b_1 \text{ no} + (\text{no of rows} - (j-1) \text{ natural} ) ] )$

~~extra~~

### Fibonacci heap

BA

$$\text{Loc}[a[4][3]] = 1000 + \left[ \frac{4 \times 5}{2} - 2 \times 3 + (4-3) \right] * 2$$

↓      ↓      ↓

crossing all column    crossing unnecessary column    Visiting 4th row and 3 column

$$= 1000 + 16 = 1016$$

$$\text{Loc}(a[4][1]) = \text{BA} + c \left[ \text{sum of } 4 \text{ nat. no} - \text{sum of } (4-1) \text{ no} \right]$$

it is used to cross all column    Visiting 1st row

$$= 1000 + 2 \left[ 10 - 10 + 3 \right] = 1006$$

crossing unnecessary crossed column

ex 2 - A[25...100, 25, 100] BA=1000 C=10B LTM CM0

$$\text{Loc}(A[80][45]) = \text{BA} + c \left[ \text{sum of } 76 \text{ natural no} - \text{sum of } (100-45+1) \text{ no} \right]$$

$$= 1000 + 10 \left[ 76 \times 77 + \frac{31 \times 32}{2} + 10 \right] + (80 - 45)$$

$$= 1000 + 10 \left[ 76 \times 77 + \frac{31 \times 32}{2} + 10 \right] + (80 - 45)$$

Visiting 80th row

5

38

77

31

266

12

266

36

266

36

$$= 1000 + 10 [ 38 \times 77 + 31 \times 32 + 10 ]$$

$$= 1000 + 10 [ 2926 + ]$$

$$= 1000 + 10 \left[ \frac{76 \times 77}{2} + \frac{56 \times 57}{2} + 35 \right]$$

$$= 151469$$

when you are at 15 row col  
you are at 15 to row

Loc EAF5

$A[-50 \dots 50, -50 \dots +50]$  BA = 0 C = 1 Byte LTM, CM

$$\text{Loc}(A[25][15]) = 0 + 1 \left[ \frac{\text{sum of } 101 \text{ nat - sum of } [50-15+}}{\text{no. nat. no.}} \right.$$

$$+ (25-15) \left. \right]$$

$$= 101 \times 101 + 10 - 36 \times 37 \quad \begin{matrix} 31 \\ 18 \end{matrix} \uparrow \text{since you at 15} \\ \text{row}$$

$$= 5161 - 666 \quad \begin{matrix} 2 \\ 2 \end{matrix} \quad \begin{matrix} \text{as 15 column} \\ \text{will start from} \\ 15 rows as all} \end{matrix}$$

$$= 4495 \quad \begin{matrix} \text{element before} \\ 15 \text{ are 2010} \end{matrix}$$

Ex 9  $A[lb_1 \dots ub_1][lb_2 \dots ub_2]$  BA, C, LTM, CMD

Loc E

$$\text{Loc}(A[i][j]) = BA + C \left[ \frac{\text{sum of } (ub_2 - lb_2 + 1) \text{ nat - sum of } (ub_2}}{\text{no. nat. no.}} \right.$$

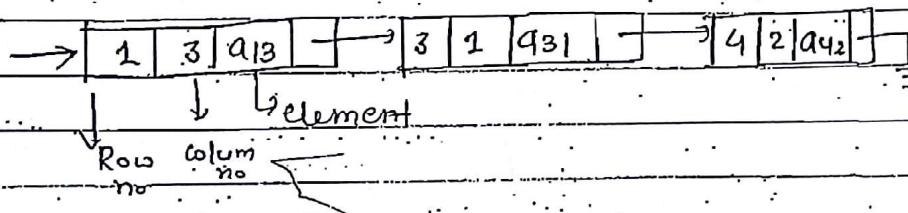
$$+ (j-i)] \quad \begin{matrix} \downarrow \\ \text{sum of all} \\ \text{column} \end{matrix}$$

Sparse Matrix - ~~very less element in array are 1, and other are zero & there is no relation in them~~

0	0	a <sub>13</sub>	0
0	0	0	0
a <sub>31</sub>	0	0	0
0	a <sub>42</sub>	0	0

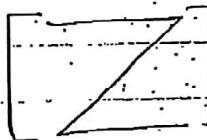
⇒ so need to store in array.

So we used linked list for its storage because if you use array lot of space is required.



Upper triangular Matrix

Z Matrix - all matrix except Z are zero



## Programming

to ex-<sup>er</sup>-<sup>cise</sup> | ^ lines  
to

## Scope of a variable

- i) Static Scoping
  - ii) Dynamic Scoping

(data type followed by name of variable)

```

<1 - int a=10;
      (C)
      main()
      {
          int a=20;
          BC();
          D();
      }
      BC()
      {
          int a=30;
          D();
      }
      D();
      int a=40;
      DL();
      }
      for all fn call, a
      single stack is used.
      (both static & dynamic as no
      scope problem.)
      Whenever a function uses
      memory to local varriable
  
```

The diagram illustrates three memory allocation regions:

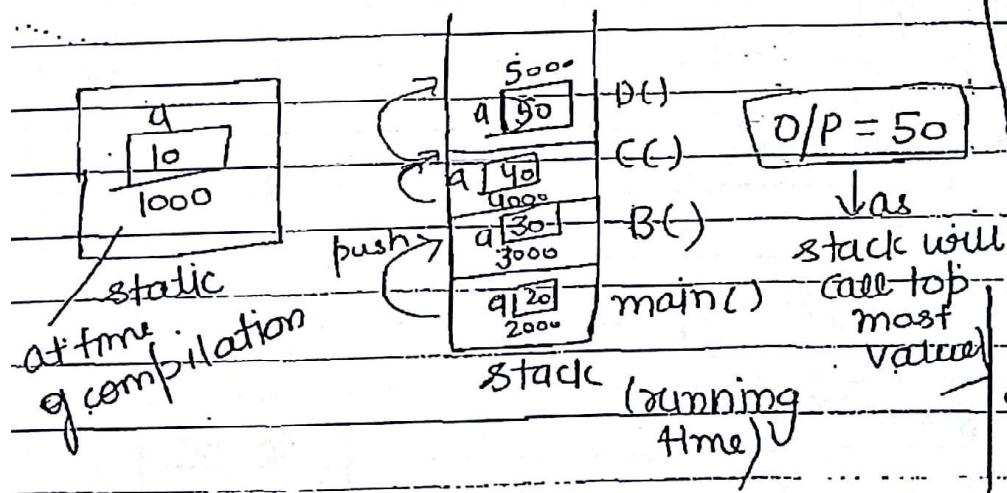
- stack** (for local f" variable)
- static** (at time of compilation)
- heap** (dynamic m/m)

A bracket on the left groups "stack" and "static" under the label "M/M".

Whenever a function is  
memory to local var1  
is created & its runn  
completed; m/m is  
deactivated but  
global variable; m

Created at time of compilation. the m/m is created in static area.)

the space taken by  
any  $f^n$  in stack  
is equal to local  
variable



static by default - C

if both local & global decls are not  
there, no mean ( $a = b + c$ )

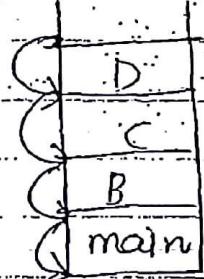
~~free~~ free variable - if a fn is using a variable which is not declared in it.

as if D()	no a is declared but is not initialized	A variable which can not be a does not exist
then	p(a)	[no a is declared but used]
O/P = f0(static)	f0(dynamic)	

whenever there is a free variable, compiler will identify it at time of compilation so, compiler will initialize it by global variable & this problem is known as scope problem. & this problem if resolved at time of compilation, known as static scoping because at that time only global variable was initialized by compiler can't solve problem at run time.

but if compiler has restricted to solve scope problem at the time of compilation that it is not initialized by global data it will be resolved at run time. If 'a' is not in D then go to its calling function & use from there and so on. if no calling function has 'a' then use global variable 'a'.

Dynamic Scope  
will generate closer scopes  
as selecting the values from the closer fn.



Dynamic is difficult to implement

static scoping - problem solved at compile time

dynamic scoping at run time

because In dynamic scoping worst case - all fn's to be passed to the

static scoping - by default

C, C++

static variable get m/m first & last  
deallocated at l  
re allocated b/t  
compilat

If a f<sup>n</sup> as D()

{ pf(a) and so on then static scoping - lo  
dynamic - go }

change done at compile time - static change

change done at run time - dynamic change (in dynamic  
change are m  
to previous)

Ex2 int a=10, b=20;

main()  
{

int a=5, b=6;

pf(a,b);

C();

pf(a,b);

C();

{ int b=3;

pf(a,b);

D(b);

pf(a,b);

a=3, b=4;

}

D(int a)

{

pf(a,b);

a=2;

b=3;

E();

pf(a,b);

a=1;

b=2;

E();

Static-

1000	2000
10, 3 5, 6	20, 8

a b

↓ static 3

D

CC

mai

5, 6 M

10, 3 C

3, 20 D

10, 3 E

2, 71 D

61, 3 C

5, 6 M

5, 6

5, 3

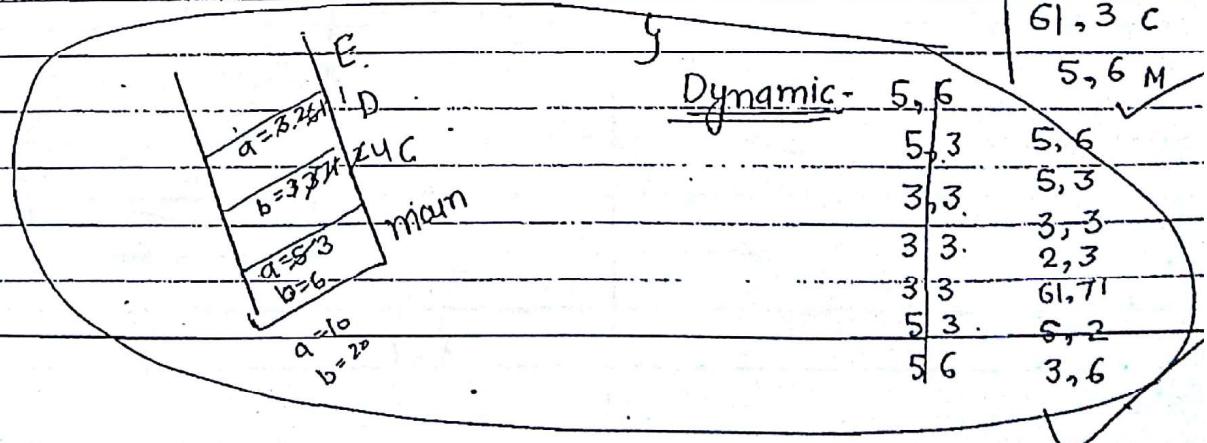
3, 3

3, 3

3, 3

5, 3

5, 6



5-3 int a=1, b=2;

main()

D (int a, int b)

bf(a, b);

c(b);

bf(a, b);

c(int a)

bf(a, b)

a=1, b=2

E();

bf(a, b);

a=3, b=4

a=3, b=4;

E()

bf(a, b);

D(b, a)

bf(a, b);

bf(a, b);

a=6;

a=3, b=7

b=7;

bf(a, b);

Static Scope - 1, 2

3, 4

4, 3

1, 4

3, 7

1, 2

3, 7

3, 7

4, 3

1, 2

3, 7

3, 7

4, 7

a=1, 3

b=2, 4, 7

a=4, 3

b=3, 2, 4

a=2, 3, 6

E

D

C

main

Dynamic

1, 2

3, 4

4, 3

1, 2

3, 7

3, 7

4, 7

a=4, 1, 3, 6, 3, 7

b=3, 2, 4, 7

a=2, 3, 6

b=2, 4, 7

C

m

a=1

b=2, 4, 7

static

m/m

## Static Variable

local var  
dynamic m/m  $\rightarrow$  heap area  
in char - '10'  
int - '0'  
float '0.0'  
pointer 'Null'

ex: main()

static int var = 5;

printf ("%d", var);

if (var--) 5

} d

if ("%d", var)

main();

} b

Var = 5

Without static with sta

5 4 infinite

5 4 loop

5 4 stack

5 4 overflow

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

5 4

Just static

static by default = 0

Note 1 - i. Static Variable M/M will create only once because m/m created at compile time.

2. for static variable, initialization will be done only once.

3. by default static variable will be initialized to zero.

Ex2 - main()

static int var=5;

if (var)

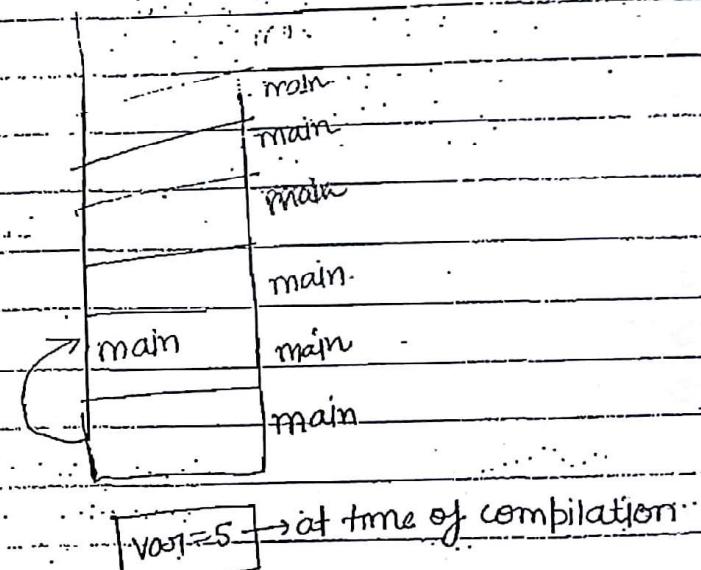
{

main();

bf(var--);

}

}



→ infinite loop (stack overflow)

main()

{

static int var=5

if (--var)

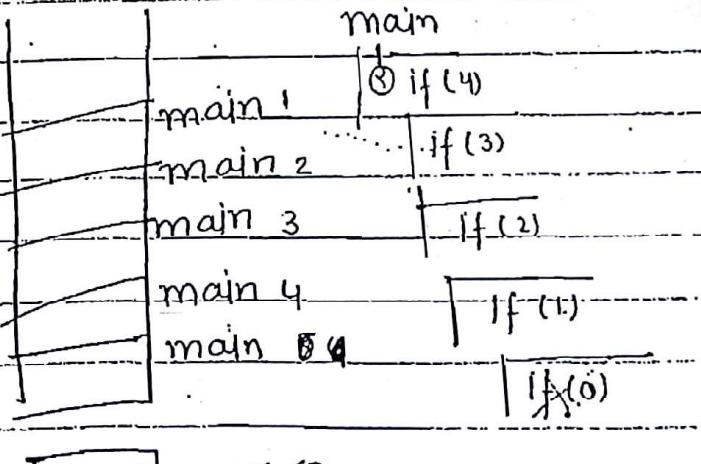
{

main();

bf(var)

}

}



var=5 143210

0000

O/P - 0.000

```
main()
{
    static int var=5;
    if(var--)
        main();
    pf(var);
}
```

O/P - -1,-1,-1,-1,-1

```
main()
{
    static int var;
    if(--var)
        main();
    pf(var);
}
```

as var=0

-0=(-1)

if(-1) =>

```
main()
{
    static int var;
    if(var--)
        main();
    pf(var);
}
```

O/P - No O/P

```
main()
{
    static int var=5;
    if(var)
        main();
    pf(var);
}
```

O/P - 4,3,2,1

if(var--)

O/P → 4,3,2,1,0

~~Ex-4~~

= (right-to-left)

static

main()

f (int x)

int x = 5, y = 10; int i, j;  
for (i = 1, j = 2; i <= 2; i++) {

int y

y = g(x);

return (x + y);

y = y + f(x) + g(x);

bf(y);

g (int x)

static int y = 20;

y++;

return (x + y);

x = 5

x = 5

g(5)

x = 5

x = 5

f(5)

y = 28

y = 2628

main

x = 5

y = 1668

static

two thing

m/m ft-comb

no m/m at sc

y = 28

21

22

23

24

68

y = 10 + 31 + 27

68

130

y = 68 + 28 + 5 + 29

O/P - 68

y = 68 + 33 + 29

130

= 130

When main() executes completely, then static variable are deallocated. its life span is to whole program.

- with same name variable in diff function  
you can create static m/m.

auto - M/M is created at run time in stack. fn are by default 'auto'.

Global variable is by default static then no need of putting it.

Extern - int a=10; global static variable.

main()

{

static int a=20;

bf(a);

}

local static variable a

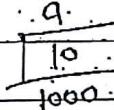
only main can use it

Extern - local variable with extern will not take @ M/M but point to the M/M outside. but that variable is local variable but not free variable.

int a=10

main()

{



extern int a; (no m/m point to global m/m)

bf(a);

O/P = 10

if no extern = O/P = Garbage Value

if static O/P = 0

if auto O/P - Garbage value

if extern is there & no global M/M with name a, then it will give error.

ex2 - main()

{

extern int a;      extern get executed at run time

pf(a);      } no memory due to extern

}      but no m/m outside

then error occurs. 'Undefined Symbol 'a'

ex3 - main()

{

extern int a; // no m/m created      No error as value of a is never checked  
pf(hi);      O/P = hi

1 local variable

ex4 - int a = 10;

main()

{

extern int a;      O/P = 10, 20

pf(a);

a=20;

pf(a);

}

q  
10 20

extern int a;

main()

{

extern int a;

pf(a);

a=20;

pf(a);

}

O/P → error

ex5 - extern int a=10;

main()

{ extern int a;

pf(a);

}

q  
10

O/P = 10

variable  
extern m/m globally.

with initialization (def)  
m/m

but extern variable gives

local with initialization give  
error.

ex5 int a;

main()

{

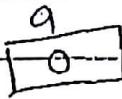
int a=5; O/P - error at compile time (Semantic error)  
int a=5; as no two variable in same fn with  
bf(a); name

}

int a;

main()

{



No. of variable - 3

(1, 2)

extern int a;

O/P = 0

global

local

extern int q; as both will point to single m/m so

bf(a); need to go for error

y

extern allow us to declare same variable many time because M/M is not creating

register int q - same as auto but M/M is created in register which make execution faster.

get  
If you want to get  
out of the program from  
stack level 5, use exit

```

int a=1, b=2;    C()      DC()
main()          {
    {
        static int b=7      extern int b
        int a=5;           bf(a,b);   bf(a,b)
        bf(a,b);          D();       a=4
        C();               bf(a,b);   b=97
        bf(a,b);          { a=8; b=9 } E()
        D();               bf(a,b);
        'bf(a,b);          E()       }
    }
}

```

auto int b=25; static

bf(a,b); a=11 b=25

a=11 b=25

b=12 b=12

bf(a,b) b=7

Static D b=7

5,2 5,2

E 1,7 1,7

D 1,7 1,2

4,25 4,25

C 11,12 11,12

11,97 11,97

main 11,97 11,97

5,2 8,97

8,97

4,25

11,12

11,97

5,97

Short circuit

short circuit (and)

Use Brain  
be conscious

(int with C and E)

Ques - main()

int i;

for(i=1; i<25; i++)

{

switch(i)

{

case '1': i+=7;

case '2': i+=5;

case 3: i+=3;

default: i+=5;

break;

}

if(i);

}

}

Note - In switch statement, after every case there is requirement of break otherwise it will execute remain cases also.

After default, there is no requirement of break as it is last stmt.

i = 8

= 13 + 5

8 + 5 + 3 + 5 =

27

22

i = 29 + 5 + 3 + 5

42

42

22 + 13

35

i = 21 i = 1 → i = 21

21

i = 22 i = 21

22

[O/P = 21]

[O/P - 21, 27]

Ques - main()

{ int i=-1, j=-1, k=-1, l=2, m;

m = ((i++ && j++ && k++) || (l++));      ( ) - left to right associativity

if(i, j, k, l, m),

{ (T && T) && (T) - || T {T} }

0 0 -1 0 21 T

(0 0 0 21 1)

-1 ... -∞, 1 ... ∞

↓ True on

zero or

false

i = 2 will remain 2 as no change is done to l due to '1' bracket is (1)

(1 or -) = 1  
so no need to check. (1)

main()

{

int i=-1, j=-1, k=0, l=2, m;

m = ((i++ && j++ && k++) || (l++))

printf(i, j, k, l, m)

}

m =  $\frac{i++}{\downarrow} \cdot \frac{j++}{\downarrow} \cdot \frac{k++}{\downarrow}$  then check  $\frac{l++}{\downarrow}$   
 $\underbrace{i}_{1} \cdot \underbrace{j}_{1} \cdot \underbrace{k}_{0} \cdot \underbrace{l}_{1}$

$\therefore m = 1$

O/P = 0, 0, 1, 3, 1

If int i=0, j=-1, k=-1, l=2, m;

m = ((l++ && j++ && k++) || (l++));

printf(i, j, k, l, m)

i no need to check others (j & k)

$\downarrow$  check for l

$\therefore O/P = 1, -1, -1, 3, 1$

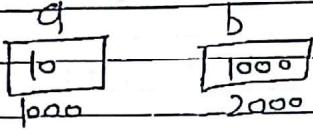
P

Pointers - int a = 10 - a 2Byte M/M is allocated to a to store integer data

int b = &a - 2B M/M is allocated to b to store address integer data which may go n wrong as address may be more than 2B.

int \*b = &a - a 8 byte M/M is allocated to b to store add of integer data

$\downarrow$  OS will take it as declaration to store integer address



whenever you wanna store address use \*

$$\text{pif}(a, b) \Rightarrow (10, 1000)$$

$$\text{pif}(*a, *b) \Rightarrow 10, 10$$

↳ User  $\Rightarrow$  value at point by value stored at

`int *b = a` may get wrong as  $a > b$  then error

`int b = 'a'` Yes

`int b = a` Yes

`int b = 0.0` No (4 byte cannot be mapped to 2 byte.)

↳ to do this type casting is used.

Ex-2

`int a = 10`

`int *b = &a` (for operating system declarations are done)

`int **c = &b` (when b is also storing address of another variable)

8Byte

c [2000]  
3000

2Byte

double pointer is used when value getting stored also points to another address

$*\star$  - address of a single pointer.

c  $\rightarrow$  double pointer.

`pif(a, b, c) = 10, 1000, 2000`

↳ directly we point value stored in a, b, c

`pif(a) = 10` (immediate addressing mode)

`pif(*b) = 10` (Direct address mode)

`pif(**c) = 10` (Indirect address mode)

`pif(c) = 10` (Indirect address mode)

single pointer - \*  
double pointer - \*\*

- if b is used with double(\*:) & c with (\*\*) It will generate an error message
- the retrieval of data must be done same way of storage

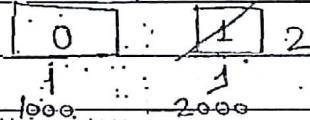
Problem 1- Consider the following C program -

```
#include<stdio.h>
void f(int *p, int *q)
```

```
b=q;
```

```
*p = 2;
```

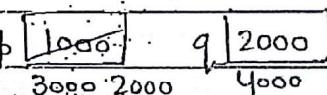
```
}
```



```
int i=0, j=1;
```

```
int main()
```

```
{
```



```
f(&i, &j);
```

```
by ("./d, /d, i, j), O/P: 0, 2
```

```
return 0;
```

Call by address

Call by value - when values are needed for the processing

Call by reference - when actual originals are to be changed while processing.

(preprocessor will take the preprocessor code definition of  $f^n$  to program)

int c, \*b, \*\*xq;

c = 4;

b = &c

a = \*b

bf ("./d", jcc, b, a))

jcc (int &x, int \*py, int \*\*pz)

int y, z;

\*\*pz += 1;

z = \*\*pz

\*py += 2

y = \*py;

x = 3

return (x + y + z);

O/P - 19

\*py = \*py + 2

= \*1000 + 2

= 5 + 2 = 7

7 5 1 4	1000	2000
c	b	a

1000 2000 3000

7 1 4	1000	2000
x	py	pz

7 1 5

19

Steps-

1. check for static variable
2. check for static & dynamic scope
3. go on

\*\*pz = \*\*pz + 1

2000

\* 2000

→ 1000

↓

4 + 1

= 5

(\* has priority over +) as unary op  
 has priority more than Binary op  
 as to complete unary 1 is enough.

local pointer - M/M pointer  
by that  
pointer has  
local va

Ques - Consider the following C program

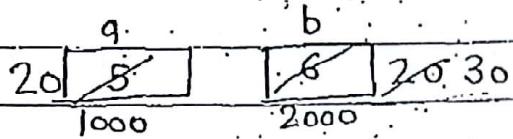
f (int  $\alpha$ , int \* $\beta$ )

\* $\beta$  =  $\alpha$ ;

$\alpha$  = 10;

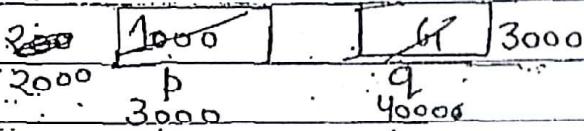
main()

int  $a=5, b=6;$



int \* $\beta = \&a, **q;$

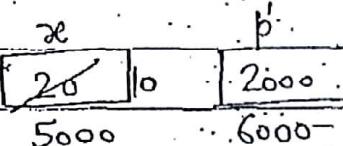
\* $\beta = 20, q = \&\beta;$



f (a, &b)

\* $q = \&b$

\* $\beta = 30;$



b) f (a, b);

x q = 2000

O/P - (20, 30)

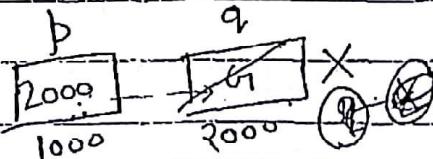
\* 3000

\* $\beta$

Ques - main()

{

int \* $\beta;$



int  $q;$

O/P - X+10

$\beta = \&q;$

scanf ("%d", p);

| If value entered - X

p) ("%d", q+10);

}

Ques - Consider the following C program-

main()

{ int b=1;

b=f(b);

b=f(b);

b=f(1+f(b));

}

f(int q)

printf("%d", q++);

return (++q);

b	1	3's
to		

O/P = 1, 3, 5

f(8)  
f(3)

a=123  
f(1)

b=1  
main

O/P = 1, 3, 5, 8

b=10 ; q=10

(final)

Ques - main()

{

if a is array  
it will keep address

by default int is the  
with main, no need

int a[] = { 12, 7, 13, 4, 11, 6 };

write it but if don't

by ("%.d", f(a, 6));

return anything, er-

return(0);

a

12	7	13	4	11	6
----	---	----	---	----	---

Let x

x	n
2000	3000

f(int \*a, int n)

{

if (n <= 0) return 0;

else

if (\*a / 2 == 0) return \*a + f(a+1, n-1);

else

\*a = f(a+1, n-1);

12 / 2 = 0

then 12 + f(1, 5),

x+1, 5

$$12 + 7 - (13 - (4 + 11 - (6)))$$

$$= 19 - (13 - (4 + 5))$$

$$12 + 12 + f(x+2,$$

$$12 + 12 + 13 + 7 + 13.$$

• Base Address of an array cannot be changed, it will remain same  
 $a = a + 3$  initially  $a = 1000$   
 $a' \neq 1006$   $a$  can not be changed.

1006



initially  $a = x$  (base address)

$f(x, 6)$



$x$       6

$a$

$n$

$m$

$$*q = 12 / 2 = 0$$

$$12 + f(a+1, 5)$$

$$12 + (7 - (f(q+2, 4))$$

↓ 3

$$12 + (7 - (f(13 - (f(a+3, 3))))$$

↓ 3

$$12 + (7 - (13 - (4 + f(a+4, 2))))$$

↓ 4

$$12 + 7 - (13 - (4 + 11 - (f(a+5, 1))))$$

↓ 5

$$12 + 7 - (13 - (4 + 11 - (6 + f(a+6, 0))))$$

$$12 + 7 - (13 - (4 + 11 - 6))$$

$$12 + 7 - (13 - 9)$$

$$\Rightarrow 12 + 7 - 4$$

$$= 15$$

O/P = 15

for integer pointer  $a = 1000$

$$a + 1 = 1002$$

for float pointer

$$a + 1 = 1004$$

for char pointer

$$a + 1 = 1001$$

27-MAR

Ques - main()

gatc2011 string end with  
(it is a string null)  
double course  
number  
1000 0 1 2 3  
 $P + P[3] - P[1]$

char P[] = "gate2011";  
=  $P + *(P+3)e-2*(P+1)$

pf ("%s", P + P[3] - P[1]) = "t000 e-9"

5. 1000 + 101 - 97 = 1004 (after  
1000 + 1003 = 1001 / 1004)

g | a | t | e | 2 | 0 | 1 | 1 |  $\square$   
1000 . 01 . 02 . 03 . 04 . 05 . 06 . 07 . 08  
10 point

If don't want to have null  
 $P + P[3]$

{'g', 'a', 't', 'e', '2', '0', '1', '1'} O/P = 2011

printf ("%s", p) = 1000

printf ("%s", P+2)  $\Rightarrow$  te2011  
1000 + 2  
1002

printf ("%s", "gate2011")  $\Rightarrow$  ga

printf ("%s", P)  $\Rightarrow$  gate2011

b) it will print string from

printf ("%s", P+5)  $\Rightarrow$  011  
1000  
1005  
address stored in array P  
Null character

P + P + 3

printf ("%c", \*(P+3))  $\Rightarrow$  e (only one char)  
1003  
(for printing one value put \*).  
P + \* (P+3) = (P + 1003) = \* 100

printf ("%d", \*(P+1))  $\Rightarrow$  97 (ASCII value of a)  
1000  
a

pf ("%c", P+2)  $\Rightarrow$  error as you cannot print string by %c)

pf ("%s", \*(P+5))  $\Rightarrow$  error as string cannot accept char.

Note - If a char value from given array has to be pointed, pointers are used i.e. a[i]  
else base address is used i.e directly a.

Ques 2 - main()

Every array is a pointer

char a[] = "gotohell";

char a[10];

char \*b = "gate2011";

int i, length = strlen(b);

for (i=0; i<length; i++)

a[i] = b[length - i];

pf("y.s", a);

O/P = No output

as starting char itself  
is null character

0 1 2 3 4 5 6 7 8 9

0 1 2 0 2 e t a l

temp (by CPU)

g a t e 2 0 1 1 \ 0

base of 02 03 04 05 06 07 08

, 1000

b

8-0

strlen("abcd")

↓ 4

sizeof("abcd")

↓ 5

a & temp & b are stored  
in diff. location. but they  
are link.

but a & a[0] are strongly  
connected they cannot be  
stored separately

pf("y.s", a+1)

→ 1 1 0 2 e t a l i c

(upto time  
any null  
char occur).

Note: Here, a = a + 5 (Wrong. Base address can't be changed)

a[5] = 'b' - possible      ↴ (base address constt.)

b = b + 5 (possible)

\* (b + 5) = 'a' (Wrong. we cannot change string  
here)

these are known as string constant

format.

It will not give error msg but no  
change occur.

### Note -

Array Base Address cannot be changed, but array content can be changed. String Constant base address can be changed but contents cannot be changed.

### Ques : main()

```
char a[] = "hello";    a | 1000 01 02 03 04 05  
char b[] = "hello";    b | 2000 h e l l o \0
```

if (a == b) → 0/P → bi (as 1000 ≠ 2000)

else

bf(bi);

if (\*a == \*b)  
\*a = h; \*b = h.

→ O/P → bi.

### main()

```
{ char a[] = "hello";
```

```
char b[] = "hello";
```

if (a == b) bf(bi); error as you are trying to change  
else : bf(bi); base address of array.

if (\*a == \*b) → print hi!

`%u` - Unsigned integer (used to print address)

Ques -

main()

{

int a[] = {10, 20, 30, 40, 50, 60};

int \*b[] = {a+3, a+4, a+5, a+2, a, a+1};

int \*\*c = b;

\*C++;

pf ("%u %u %d", c-b, \*c-a, \*\*c);  
++c;

pf ("%u %u %d", c-b, \*c-a, \*\*c);  
++c;

pf ("%u %u %d", c-b, \*c-a, \*\*c);  
\* (++(++c));

pf ("%u %u %d", c-b, \*c-a, \*\*c);

}

a	0	1	2	3	4	5
	10	20	30	40	50	60

1000 02 04 08 08 10

b	0	1	2	3	4	5
	1006	1008	1010	1004	1000	1002

2000 2008 2016 2024 2032 2040

2000	2008	2016	2008	*C++
C	3006	3012	outside	

①  $c-b = 2008-2000 = 8 / 8B = 1$  i.e. it will be dividing the n

$*c-a = *c = 1008-1000 = 8 / 2B = 4$  bit it take to store one value i.e. it will give

$**c = 50$

②  $\downarrow *C++ = *1008$

↓ no of element present b.  
as size of array  
is 2B  
two pointer

Whenever two pointers are subtracted it will give no. of elements b/w those two ptrs incl first one & excluding last.

2)  $\ast C++ \quad C++;$

$\checkmark \quad = \ast C$

$C - b = 2016 - 2000 = 16 / 8B \text{ (size of array elmt is } 8B)$

$= 2B$

$\ast C - a = 1010 - 1000 = 10 / 2B \text{ (size = } 2B \text{ of a)}$

$= 5$

$\ast \ast C = \ast C = 100$

$(2, 5, 60)$

3)  $++ \ast (++C);$

$\checkmark \quad (\ast 2024)$

③  $\ast C++$

$\downarrow$

$2008$



$\ast C$

$2008$

$1008 + 1$

$1010$

Note -  $\ast, ++$  both are unary operators with same priority but associativity is right to left.

3)  $\ast C + \sharp$

$C - b = 2016 - 2000$

$= 16 / 8B \text{ (size of array elmt. is } 8B) = 2B$

$\ast C - a = 1010 - 1000 = 10 / 2B = 5$

$\ast \ast C = 60 \quad (2, 5, 60)$

3)  $++ \ast ++ C =$

$\downarrow$

$2016 \quad 2024$

$\downarrow$

$1004$

$1006 \quad 1006$

00	08	16	24	32	40
1006	1008	1010	1016	1000	1002

$C - b = 24 / 8 = 3$

$\ast C - a = 1006 - 1000 = 6 / 2 = 3$

$\ast \ast C = 40$

$\boxed{1} \quad *f + (f + c)$        $\boxed{2024} \quad 2032 \quad 2040$   
 C  
 ↓  
 $2032$   
 $2040$   
 $C - b = 40/8 = 5$   
 $*c - a = 1062 - 1000 = 1/2 = 1 \quad (5, 1, 20)$   
 $*c = 20$

$\rightarrow P \rightarrow$   
 1, 4, 50  
 2, 5, 60  
 3, 3, 40  
 5, 1, 20

- e1
- We can add integer constant to pointer variable.
  - Ex-  $P$  - pointer,  $P + i \rightarrow$  skipping  $i$  slots/element including  $P$  (in forward direction).
  - $P$  - pointer,  $P - i \rightarrow$  skipping  $i$  slots/element including  $P$  (in backward direction).
  - We cannot add float constant to pointer variable.
  - We can subtract integer constant from pointer variable.
  - We cannot subtract float constant from pointer variable.
  - We can subtract two pointers pointing to same array.
- $P_1 - P_2 = \text{No of element b/w } P_1 \text{ & } P_2 \text{ excluding } (P_1) \text{ incl } (P_2) \quad (P_1 \text{ is greater than } P_2)$
- To subtract two pointers —
  - $P_1 > P_2$
  - $P_1, P_2$  both should point to same array.
  - Subtraction of two pointer give no.of element b/w the
  - Pointer division & subtraction has no use  
(We can't perform add, multiplication & division b/c they don't have any meaning)

main()

{

float c[] = { 50, 20, 30, 10, 40, 60 };

float \*b[] = { c+3, c+1, c+2, c, c+4, c+5 };

float \*\*a = b;

\*a++;

pf(a-b, \*a-b, \*\*a);

++\*a;

pf(a-b, \*a-c, \*\*a);

++\*\*a;

pf(a-b, \*a-c, \*\*a);

++\*++a;

pf(a-b, \*a-c, \*\*a);

}

1000	04	08	12	16
50.0	20.0	30.0	10.0	40.0
0	1	2	3	4

b	2000	2008	16	24	32	44
1012	1004	1008	1000	1016	1020	
8	1	1008	2	1012	3	4

2000	2008	2016
a		

1, 1, 20.0

1) \*q++ a-b = 8/8 = 1

\*a-c = 1004 - 1000 = 4

\*\*q = 20.0

2) ++(\*q) a-b = 8/8 = 1 1, 2, 30.0  
2008 1004 1006

\*q-c = 1008 - 1000 = 8/8 = 2

\*\*q = 30.0

3) ++ \* \*q a-b = 8/8 = 1 1, 2, 31.0  
1008 30.0 31.0

\*q-c = 1008 - 1000 = 2

\*\*q = 31

4) ++ \* ++a a-b = 16/8 = 2 2, 3, 1016  
2016 1008 1012

\*a-c = 12/4 = 3

\*\*q = 10.6

main()

{

char \*a[] = {"baba", "stupid", "gotohell", "break", "chacha", "chach"};

char \*\*b[] = {a+2, a+3, a+4, a, a+1, a+5};

char \*++c = b;

\*c++;

pf("./%s", \*++c); chacha

pf("./%s", ++\*c+2); achi

pf("./%s", \*(++c+2)); b

b a b a \0  
1000 01 02 03 04

a  
b 1 2 3 4 5...  
1000 2000 3000 4000 5000 6000  
1000 08 96 92 32 40  
24

stupid \0  
2000 01 02 03 04 05

gotohell \0  
3000 01 02 03 04 05 06 07 08

b 1 2 3 4 5...  
7016 7024 7032 7000 7008 7040  
8000 08 16 7040 24 32 40

break \0  
4000 01 02 03 04 05

chacha \0  
5000 01 02 03 04 05 06

8000 8008 8016 8024

chachi \0  
6000 01 02 03 04 05 06

g000

\*++c

↓

8008

→ 8016

→ 7032

→ 5000

\*++c +2

8024

\*++c +2

7032

→ 7040

→ 6000 + 2 = 6002

1000 + 2 → 1002

↓

O/P - chacha

achi

b

Ques

main()

{

int a[5][3] = { 10, 20, 30, ..., 140, 150 }

pf ("%d", &a[0][0] + (\*a + \*(a+0))) );

pf ("%U", \*(&a[0][0] + 2));

pf ("%U", \*(&a[0][0] + 2));

pf ("%U", \*(&a[0][0] + 2));

1011 = 11 (unsigned)

1011 = -3 (signed)

}

a	Row 1	Row 2	Row 3	Row 4
	10 20 30	40 50 60	70 80 90	100 110 120

1000 02 04    06 08 10    12 14 16    18 20 22    24 26 28

$$a = 1000$$

$$a[0] = *(&a) = 10$$

$$3) *a + 2$$

$$\dots 10 + 2 = 12$$

4.

$$1002 + 2 \quad 1004$$

32

↓ It can also be represented as

0      1      2      3

$$(3) *(&a + 2) + 2$$

$$*1004 + 2 = 32$$

0      1      2      3

1      40      50      60

06      08      10

2      70      80      90

12      14      16

3      100      110      120

18      20      22

4      130      140      150

24      26      28

Vote - In 2-D Array

$a + 1 \Rightarrow$  skipping 1-D array

If  $a = 1000$

$$a + 1 = 1000 + \text{no of column in row} = 1000 + 3 \times 2$$

$$= 1006 \text{ (base address of } 1^{\text{st}} \text{ element)}$$

$$*(a + 1) = 1006 \text{ (1st row is selected)} = 1006$$

$*(&a + 1) + 2 \rightarrow$  skipping 2 elements in 1 Row

$$= 1006 + 2 \times 2 = 1010$$

$$*(a + 1) + 2 = 1010$$

$$*\{*(a + 1) + 2\} = 60 \}$$

$a \rightarrow 1000$  (base address of whole array)

$a+i \Rightarrow$  skipping  $i$  rows i.e. no of column  $\times i$

$a+2 \Rightarrow 1012 \rightarrow$  (2 row skipped)

$a+2+2 \Rightarrow 1024$  (Again 2 row skipped)

$a[4][3][5]$



$a+2 \Rightarrow$  skipping 2  $\neq$  2-D table

$$= 1000 + 2 * \text{size of data} * \text{size of table}$$

$$= 1000 + 2 * 2 + 15$$

$$= 1060$$

$* (a+2) \Rightarrow$  2nd table selected

$* (a+2) + 2 \Rightarrow$  skipping 2 row in second table

$* (* (a+2) + 2) \Rightarrow$  selecting 2nd row in table selected

$(* (* (a+2) + 2) + 2 \Rightarrow$  skipping 2 element in 2nd row of second table

$* (* (* (a+2) + 2)) = \text{Value}$

(n-D vector array, require n pointer to access value)

i)  $(a == a[0]) \& \& (*a == *(a+0))$

$$\begin{array}{cccc} \downarrow & \downarrow & \downarrow & \\ 1000 & * (a+0) & * 1000 & 1000 \\ & \downarrow & \downarrow & \\ & 1000 & 1000 & = 1 \end{array}$$

(ii)

$* (a+2) + 2$

$\downarrow$   
1000  
 $\downarrow$   
1000 + 2  $\times$  3  $\times$  2

$$\begin{array}{c} \downarrow \\ * 1012 \\ \downarrow \\ 1012 + 2 \times 2 \\ = 1016 \end{array}$$

$\nearrow$  skipping two element

3)  $*q+2$

$$a = 1000 \quad *a = 1000$$

b 0th row selected

$$*q+2 = 1000 + 2 \times 2$$

$$= 1004$$

3)  $*((a+1)+2)+2$

↓

$$a = 1000 \Rightarrow a+1 = 1000 + 3 \times 2 = 1006 + 2$$

b skip 1 row

b again skip 2 row

$$= 1006 + 2 \times 3 \times 2$$

$$= 1018$$

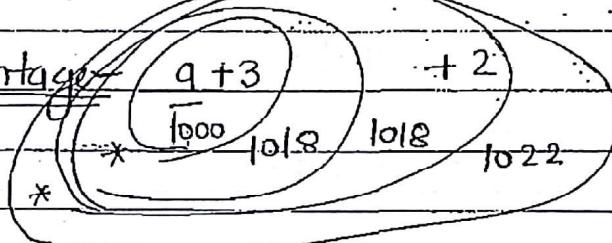
$*1018 \Rightarrow 3$  row selected

+ 2 (skip two element in 3 row)

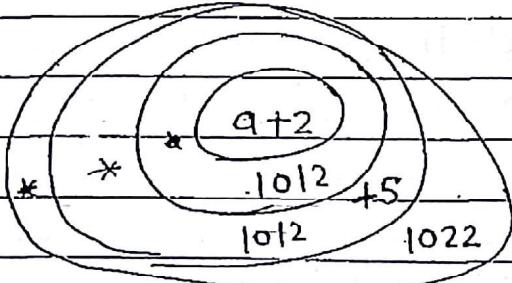
$$\Rightarrow 1018 + 2 \times 2 = 1022$$

O/P - 1, 1016, 1004, 1022

DisAdvantage



$$\Rightarrow 120 q [3][2]$$



$$120 \Rightarrow q [2][5]$$

If it is not actually  
possible but  
occurring here

as there is no row &  
column present in the  
array.

Ques main()

$$BA = 1000$$

float  $a[5][2][3] = \{10, 20, 30, \dots, 300\}$

: pf ("%U", \*(a+3)+5))

: pf ("%U", \*(\*(a+5)+2))

: pf ("%U", \*a[3]+5)

: pf ("%U", \*(\*\*a+2)+5))

i)

$$a = 1000$$

3 2D skipped

$$a+3 = a + 3 * (2 \times 3 \times 4)$$

$\downarrow$  size of element

$$= 1000 + 72$$

$$= 1072$$

$$*(1072) = *(a+3) = 1072$$

$$*(a+3) + 5 = 1072 + 5 * (2 \times 3 \times 4)$$

$\downarrow$  skip 5 1-D

$\downarrow$  size of 1-D element

$$= 1072 + 60 = 1132$$

(ii)  $*(*a+5)+2$

$$*a = 1000 \text{ (table selected)} \quad \text{as } (*a+0) = *a$$

$$*a+5 = 1000 + 5 * (2 \times 3 \times 4) \text{ (row selected)}$$

$$= 1020 + 1060$$

$$*(1060) = 1060$$

$$*(a+5) + 2 = 1060 + 2 \times 4 = \text{column}$$

$$= 1068$$

(iii)  $*a[3]+5$

$$a[3] = *(a+3)$$

$$= *(1000 + 3 * 2 \times 3 \times 4)$$

$$= *(1072) = 1072 \text{ (table selected)}$$

$$*a[3] = 1072 \text{ row selected}$$

$$*a[3]+5 = 1072 + 5 \times 4 = 1092$$

(IV)  $*(*a+2)+5$   
 $*a = \text{table selected} = 1000$   
 $**a = \text{row selected} = 1000$   
 $**a+2 = 1000 + 2 \times 4 = 1008$

$(*a)**a+2 = 1008 \del{1008} 30$   
 $*(*a+2)+5 = 35$

O/P = 1132, 1068, 1092, 35

(V)  $*(*a+7)+5$  (10kml)  
1000  
 $1000 + 7 \times 3 \times 4$   
1084  
 $1084 + 5 \times 4 = 1104$

# Structure & Unions

- Structure -
- user defined data type
  - a variable is defined

struct node:

(upto the time no variable is declared  
no M/M is allocated)

int a;  $\Rightarrow$  a datatype in compiler is defined of 4 bytes

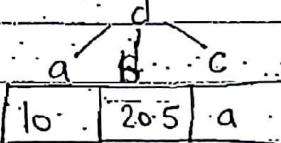
float b;  $\Rightarrow$  name of datatype = struct node

char c;  $\Rightarrow$  size of datatype = 1 Byte

}

struct node d;  $\Rightarrow$  declared a variable of type struct node

printf("%d", &d)  $\Rightarrow$  1000



d = {10, 20.5, 'a'};  $\Rightarrow$  1000 (base address of d)

printf("%d", d.a)  $\Rightarrow$  10  $\quad \cdot$  operator is used to access the element inside the structure.

printf("%f", d.b)  $\Rightarrow$  20.5

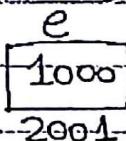
element inside the structure.

printf("%c", d.c)  $\Rightarrow$  a

printf("%d", d)  $\Rightarrow$  by default 1st value will come  
 $\Rightarrow$  10

struct node \* e;  $\Rightarrow$  e a variable, i.e. pointer variable of 4 bytes  
store address of struct node type data

\* e = &d;



printf("%d", e)  $\Rightarrow$  1000

printf("%d, (\*e).c)  $\Rightarrow$  'a'

printf("%d, (\*e).a)  $\Rightarrow$  10

$(\ast e) \cdot q \Rightarrow \downarrow$  \*e  $\Rightarrow$  \*1000 will point to d.  
 1000  
 .q will select a field d.

$$(\ast e) \cdot q \equiv e \rightarrow q$$

$$e = e + 1 \Rightarrow \text{if } e = 1000$$

$$e = e + 1 = 1000 + 1 = 1007$$

printf("%d", &(d.b)).

$\downarrow$  1002

size of 1 variable of type struct node.

Ex 2 - struct s

```
int a;
float b; 7Byte
char c;
```

Here data type is created only, no allocation to memory is done so, in structure, you cannot initialize the data.

struct s2

{

float d;  $\rightarrow 4$

struct s1 e;  $\rightarrow 7$

int f;  $\rightarrow 2$

13 Byte

}

//struct s2

float d;  $\rightarrow 4B$

struct s1 \*e;  $\rightarrow 8B$

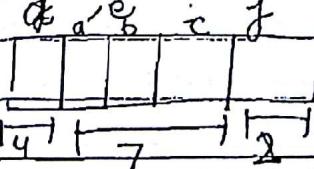
int f;  $\rightarrow 2B$

}

struct s2 g;

g = {10.25, 120, 20.5, 'a', 10};

14 7 2



g is able to watch  
 & access 3 variables  
 d, e, f only

$\text{pf}("f", g) \Rightarrow 10\cdot25$  (by default d is pointed)

$\text{pf}("f", g.d) \Rightarrow 10\cdot25$

$\text{pf}("d", g.f) \Rightarrow 10$

$\text{pf}("d", g.e.a) = 20$

↳ h to access element inside e

struct s2 \*h

size = 8 byte

$h = &g$

will store the address of struct s2 data



$\text{pf}("d", h) \Rightarrow 1000$

$\text{pf}("d", *h)$

$\text{pf}("f", *h.d) \Rightarrow 10\cdot25$

$\text{pf}("f", *h.d) = 10\cdot25$

$\text{pf}("d", *h.e.a) = 20$

$\text{pf}("d", (h \rightarrow e).a) = 20$

$*(*h) \Rightarrow \text{removed by } \rightarrow$

$\text{pf}("f", (*h.d)) \Leftarrow \text{pf}("f", (h \rightarrow d))$

$\text{pf}("f", *h) \Rightarrow 10\cdot25$

$h = h + 1 \Rightarrow \text{if } h = 1000$

$h + 1 = 1000 + 13 = 1013$

increasing from a slot

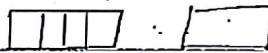
- declare first,
- do not declare more than one,
- if operation to be performed, In two variables, they both have to be of same type

## TypeCasting - Data

```
int a; float b; float c;
```

- Implicit      Explicit → when there is chance of losing the data
- (done by compiler)      (done by user)
- (when a data of smaller size is getting stored in greater size then compiler will do it)
- Typecasting is done because to any operation to be performed, data type of variable must be same.

$b = a$



↓ done by compiler

$b = (int) a$

$b = (float) a$



→ Implicit

Typecasting

& Warning msg

$a = b$



↓ error generated

user will do casting

$a = (int) b$

(loss of data occur here)

$a = (float) b$

(user write it)



↓ explicit typecasting

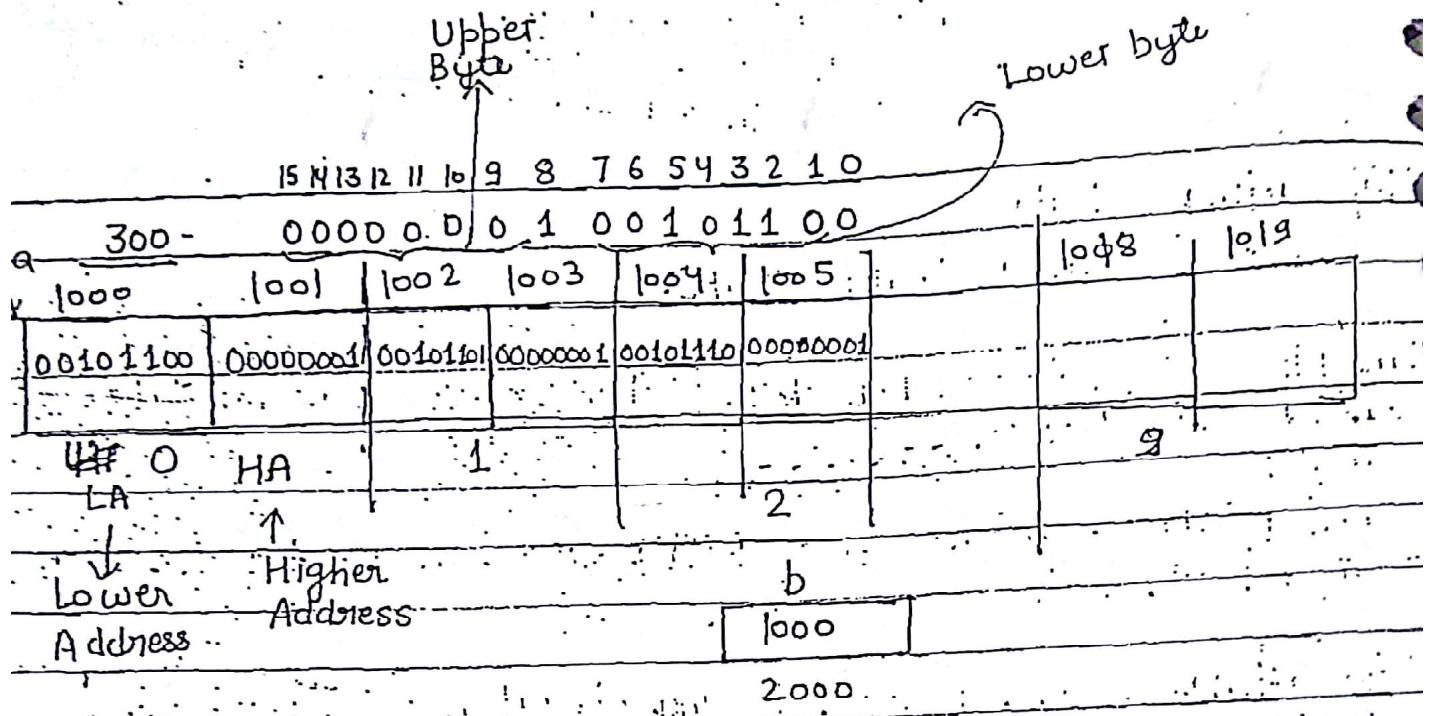
## Typecasting in Pointers

300 ~~444~~ 100101100

$\text{int } q[10] = \{ 300, 301, \dots, 309 \}$

	1000	1001	1002	1003	1004	1005		1018	1019
$q$	1	2	3	4	5	6	7	8	9
	LA	HA							
	0	1	2	3	4	5	6	7	8

Lower byte get stored  
in lower address &  
High Byte is MSB & is  
stored in Upper By



int \*b; b is a variable which will store address of integer data.  
size of b = 8 Byte

int \*b=a (no error no typecasting required as b was expecting integer type address)

char \*c=a (problem) as it was expecting character pointer.

char \*c=(char \*)a

1000                      C  
1000

(Explicit typecasting)

this is added

3000

as you have to change for typecasting has to be done for pointer.

the value stored in c will be 1000 as it is in a because a & c both are of 8 bytes.

char float \*d=(float \*)a              d  
1000

\* is kept with float as we have to do typecasting for pointer

d will also store 1000 as in 4000

a as they both have

same size but if float + 1 = 1004

void - it will allow any type of data so no conflict  
and no need of type casting but typecasting is required  
at time of retrieval.

void \*e = a [1000] e

if  $b = 1000$

$*b \Rightarrow$  since it is an integer pointer it will read 2 Byte of  
data

= 300

if  $b + 1 = 1000 + 2 = 1002$

$\uparrow$  integer size

if  $b + c = 1000$

$*c =$  it is an char pointer it will read 1 byte  
= 44

$c + 1 = 1000 + 1 = 1001$

$\downarrow$  size of char

$d = 1000$

$*d =$  it will read 4 Byte

(1003 1002 1001 1000) bits

$d + 1 = 1000 + 4 = 1004$

$e = 1000 \therefore$

$*e =$  error as it is of void type; it donot have any info to  
read bit.

if you want to use e, then typecasting of e must be  
done

$\text{int } *f = (\text{int } *)e;$

ques - a file is provided, how to read it char by char.

- open the file, it will return pointer to start of file (fp)
- type cast it to char type

char \* c = (char \*) fp;

then do  $c = c + 1$

ques - to read file bit by bit - typecast to bool value

note - when typecasting of pointers is done, it seems as they have same address but the method of accessing data get changed here.

problem - main()

{

int i = 300;

1000 1001  
00101100 00000001 00000010

300  $\rightarrow$  2B

char \* p = (char \*) & i;

1000

P++;

b 1000  $\rightarrow$  8B  
2000 1001

\* P = 2;

pf ("%.d", i);

p++ (due to char it will

\* P = \* P + 2      be increased

1 byte)

i = 812

3 \* P = 2

O/P - i = 512 + 44 = 556

as 2

i = 812

int a = 20

int a = 20

int \*\*c

int \* b = (int \*) a

= 20

c = 20

b = 20

but

as 2 byte data get

+ + c = 28

mapped in 8 byte

as it is

+ + a = 21

pointing to

+ + b = 22 (as it will

ptr as it

treated like

is of 8 Byte

Ques

Consider the following C program

main()

```
{ void *vp;
char ch='g';
```

```
char *cp="goofy";
```

```
int j=20;
```

```
vp=&ch;
```

```
pf ("%c", *(char *)vp);
```

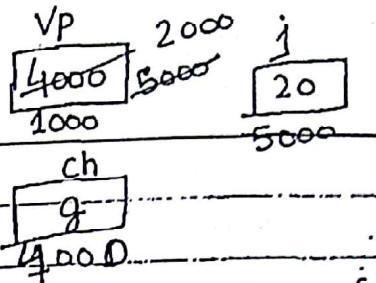
```
vp=&j;
```

```
pf ("%d", *(int *)vp);
```

```
vp=cp;
```

```
pf ("%s", (char *)vp+3);
```

```
}
```



typecasting  
by void  
pointer

O/P - g, 20, fy O/P - goofy

$$\text{value} + 1 = (\text{value} + 1)$$

$$\text{int pointer} + 1 = (\text{int pointer} + 2)$$

Use of typecasting - Body scanning in hospital.

→ DeepTilliw w/ Shekhchilli :P :P ☺

Linked List - List - collection or group of elements

Linked list - when many lists are linked.

- Self Referential data structure

### 1) Struct node

{

```
int data;  $\Rightarrow$  2B
```

```
struct node *next;  $\Rightarrow$  8B
```

};

// it will create a data type

of 10B with name struct

node:

(storing addresses of struct node type  
data)

### 2. struct node

{

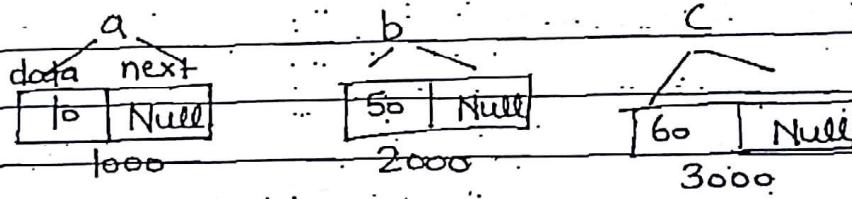
```
int data;
```

```
struct node next; you cannot use it, it will give
```

};

error as undefined datatype

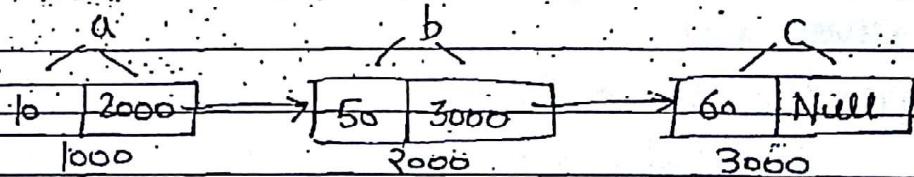
- 3: struct node a = {10, Null}  $\Rightarrow$  allocation of memory while declaration
- struct node b = {50, Null}
- struct node c = {60, Null}



4. linking is done at this stage

a. next = &b

b. next = &c



5. through linked list, from starting node it is ~~easy~~ possible to access each node.

struct node \*s = &a

s

[1000]

400

pj(s)  $\Rightarrow$  1000

50

5. }  $\text{bf}(" \%d", (*s).\text{data}) \Rightarrow 10$

↓ it can also be written as

$\text{bf}(" \%d", s.\rightarrow\text{data}) \Rightarrow 10$

{  $\text{bf}(" \%d", (*s).\text{next}) = 2000$   
 $s.\rightarrow\text{next}$

∴ When linked list is created,  $s$  is actually storing base address i.e. starting address of list then return  $s$  & you can access the whole list.

6.  $\text{return}(s);$

$\text{int} * q()$  // it is a function which donot take any i/p  
 { but return pointer ie integer address

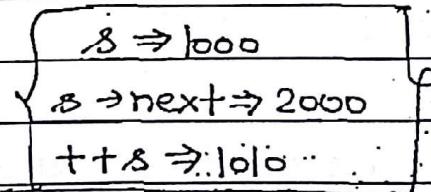
So for creating linked list,  $s$  will be a pointer of struct node type.

$\text{struct node} * \text{Create\_LL}()$

{

$\text{return}(s);$

}



Problem 1 - Write a C program to print every node data in given list:

~~void Point( $\text{struct node} *s$ )~~

~~{  $\text{printf}(" \%d", s.\rightarrow\text{data});$~~

~~$\text{while}(s.\rightarrow\text{next} \neq \text{NULL}) \text{ do }$~~

~~$\text{While}(s.\rightarrow\text{next} \neq \text{NULL}) \{$~~

~~{~~

void PRINT(struct node \*s)

{

// if ( $s == \text{NULL}$ ) (No need to  
print ("No data"); write if  
return; but it is not  
wrong).

while ( $s \neq \text{NULL}$ )

{

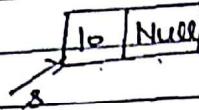
printf ("%d", s->data);

$s = s \rightarrow \text{next};$

}

Time Complexity - O(n)

(BC, WC, AC)



if ( $s \rightarrow \text{next} == \text{NULL}$ )

LL contain 1 element

if ( $s == \text{NULL}$ )

LL is empty  $\rightarrow \text{NULL}$

if  $s == \text{NULL}$

(\*s will be segmentation  
error)

as  $s == \text{NULL}$

\*NULL - No memory  
there.

known as segmen-  
tation error

Ques - Write a C program to return address of second last  
node in the given linked list.

struct node \* address (struct node \*s)

{ if ( $s == \text{NULL}$ ) return NULL; else if ( $s \rightarrow \text{next} == \text{NULL}$ )

while ( $s \rightarrow \text{next} \neq \text{NULL}$ )

return NULL;

{ p = s

$s = s \rightarrow \text{next};$

if

$(s == \text{NULL})$

return NULL

return p;

else if ( $s \rightarrow \text{next} == \text{NULL}$ )

return NULL;

else while ( $s \rightarrow \text{next} \neq \text{NULL}$ )

$p = s;$   
 $s = s \rightarrow \text{next};$

return (p);

Time Complexity - O(n)

(BC, WC, AC)

Topological Sorting  
call by

Activation Record (Do More & More)

VAL

Variable declaration in Pascal

Aliasing

Anagram ADT

go to basics like ADT, Aliasing, etc.