

21

## Programming & Data Structure

08

Aug mark:

13 marks

- Teaching schedule.
- Basic: Operators, loops, function, switch)
- Storage classes, scope
- pointers String
- Array
- Recursion
- Structure, unions
- Dynamic Memory Allocation

- Data structure
  - linked List
  - stack
  - Queue
  - Tree (BST, AVL)
  - Hashing

Text Books.

1. "C Test your Aptitude" By Venugopal & N Chandra Kanth
2. "C Language" By Dennis Ritchie
3. DS By Sahani
4. DS By Mark Allen Weiss.

► Implementation of queue using stack & stack from queue

- enum datatype
- compile time & runtime
- Hashing question.
- all test

precedence table

## Basic of C programming

### Operators :-

Operators	Precedence	Associativity
()	1(High)	L → R
↑	2	R → L
* %	3	L → R
+ -	4	L → R
. =	5(Slow)	R → L

### Associativity

If two or more operators are having same precedence then the expression will be evaluated by using associativity

ex:  $\underline{\underline{()()()}}$   
 $2^3^4$

Expression	Result
5/2	2
5.0/2	2.5
5.0/2.0	2.5
2/5	0
2.0/5	0.4
2.0/5.0	0.4

### Note

• If both are integers the output will be integers  
 & if any one is float the output will be float

### Relational & logical operators.

1. All the relational & logical operators return 0 or 1.

If the expression is true then it returns 1 & if the expression is false then it returns 0.

2. All non-zero are considered as true & zero(0) is considered as false.

True	False
1	0.0
10	0
-10	10
6.5	null
-3.5	
0.6	

10 ASCII value is 0.

$$\text{Eg 1: } a = 5 > 4; \quad \begin{array}{c} \text{T} \\ \downarrow \\ 1 \end{array}$$

$$a = 1$$

$$\text{Eg 2: } a = (5 > 4) + (5 > 3) \quad \begin{array}{c} \text{T} \quad \text{T} \\ \downarrow \quad \downarrow \\ 1 \quad 1 \end{array}$$

$$a = 2$$

$$\text{Eg 3: } a = \overline{5 > 4 > 3} \quad \begin{array}{c} \text{T} \\ \downarrow \\ 1 > 3 \\ \text{F} \\ \downarrow \\ 0 \end{array}$$

$$a = 0$$

Expression	Assigned to int	Assigned to float
1. 5	5	5.0
2. 5/2	2	2.0
3. 5.0/2	2	2.5
4. 5.0/2.0	2	2.5
5. 2/5	0	0.0
6. 2.0/5	0	0.4
7. 2.0/5.0	0	0.4

Eg:  $\text{int } a = 2 * 3 / 4 + 2.0 / 5 + 8 / 5$   
 $6 / 4 + 2.0 / 5 + 8 / 5$   
 $2 + 0.4 + 1$

$$\text{int } a = 2.4 \Rightarrow 2$$

#### • Modulus operator (%):

1.  $15 \% 7 = 1$

$$+7) -15(-2$$

$$\frac{-14}{-1}$$

$$-7) +15(\cancel{-2}$$

$$\frac{+14}{+1}$$

$$-7) -15(\cancel{+2}$$

$$\frac{-14}{-1}$$

2.  $-15 \% 7 = -1$

3.  $15 \% -7 = 1$

$$+7) -15(-2$$

$$\frac{-14}{-1}$$

$$\frac{+14}{+1}$$

$$\frac{-14}{-1}$$

4.  $-15 \% -7 = -1$

5.  $-15 \% +25 = -15$

6.  $+15 \% -25 = 15$

smaller than divisor

7.  $+15.5 \% 7 = \text{error}$

8.  $-15.7 \% 7 = \text{error}$

- Note**
- modulus always give numerators sign
  - modulus doesn't work on float value, it work only on the integers.

- ↳ If the value is small without sign, then it give same value as OLP.

ex: void main()

```
int a=5;
if (a>8)
    printf("Raj");
else
    printf("kishore");
    return 0;
```

a % 8

Ex:  $15.0 / 2 * 2 \% 7$

$$\begin{array}{r} 7.5 * 2 \% 7 \\ \hline 15 \downarrow \\ 15.0 \% 7 \\ \hline \text{error} \end{array}$$

```
y
else {
    printf("kishore");
}
printf("%d", a);
y
```

• assignment operator assign the value & return the assign value

Format  
Specifies.

- %d int
- %f float
- %c char
- %s string
- %u address

1. Ex: **Void main()** {  
    int x;  
    x = printf("Hello");  
    printf("%d", x);  
}

O/p: 5

→ Note

- printf always return integer i.e. No of symbols displayed on the screen by that particular printf only

2. Ex: **Void main()** {  
    int a;  
    a = printf("Made %d Easy", printf("delhi"));  
    printf("%d", a);  
}

a 9

O/p: delhi Made %d Easy 9  
                Made 5 Easy

return 5 → these 5 is take care by %d

- If we want we can collect that value

Ex: write a program to  
read two proper integers  
and print them on the screen.

Void main() {

int x,y,sn;

pf("Enter two proper integers");

sn = scanf("%d,%d", &x, &y);

if (sn == 2) {

pf("proper inputs given that is %d,%d", x, y);

else

printf("wrong I/p given").

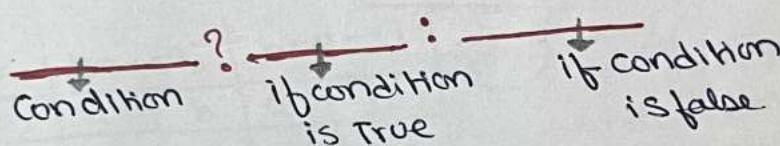
y  
y

### Scarf

- Scarf always returns integer i.e. no. of proper I/p given A/c to format specifiers.

I/p	sn
* ; *	0
# ; 0	1
\$ - \$, \$	0
10, 20	2

### Ternary Operator



### Note

- In the ternary operators there should be equal no. of colon's & question marks otherwise it give error.

- Every question mark followed by colon not immediately but following.

Every colon should match with the just before question marks.

Ex1:  $a = 3 > 4 ? 10 : 8 > 7 ? 20 : 30$

O/p:  $a = 20$

$8 > 7 ? 20 : 30$

Ex<sub>2</sub>:  $a = \underline{5 > 4} ? \underline{6 > 7} ? \underline{10 : 20 : 30};$  O/p:  $a = 20$

$\underline{6 > 7} ? \underline{10 : 20}$

Ex<sub>3</sub>:  $a = \underline{2 > 3} ? \underline{5 > 4} ? \underline{10 : 30 : 8 > 7} ? \underline{30 : 40}$

$\underline{2 > 3} ? \underline{10 : 30}$

$\Rightarrow 30 //$

Ex<sub>4</sub>:  $a = \underline{6 > 7} ? \underline{2 > 3} ? \underline{10 : 6 > 8} ? \underline{-2 : 3 : 0} ? \underline{-1 : 6 > 8} ? \underline{10 : 3 > 4} ? \underline{20 : 20}$

$3$

$\underline{6 > 7} ? \underline{3 : 0} ? \underline{-1 : 20}$

$\boxed{0 ? -1 : 20}$

$20 //$

Ques) write a program to find max. of 3 no.  
using ternary operators?

int a, b, c, max;

max =  $(a > b \& a > c) ? a : (\underline{b > a \& b > c}) ? b : c;$

$a > b \& a > c ? b : c ? b : c$

Ques) write a program to find min of 4 no.

int a, b, c, d, min;

min =  $a < b \& a < c \& a < d ? a : (\underline{b < c \& b < d}) ? b : (\underline{c < d}) ? c : d;$

i.w] write the above two program without using ( $&$ ) and ( $&&$ )

max of 3

variable:

max:  $a > b ? a > c ? a : c : b < c ? b : c$

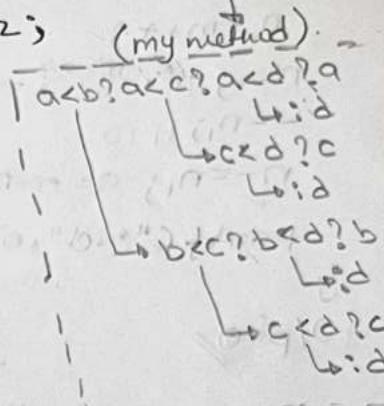
min value  
out of ~~a~~:  $a < b ? acc ? acc ? a : d : c < d ? c ; d : b < c ? b < d : c < d ? c : d ;$

Ques: Given combinations of the integer variable  $x, y, z$  make variable tree method

'a' get the value  $a$  in the following expression.

$$a = x > y ? x > z ? x > z : y > z ? y > z ;$$

- X a)  $x=6, y=5, z=3$
- X b)  $x=6, y=3, z=5$
- c)  $x=5, y=4, z=5$
- d)  $x=3, y=4, z=2$



### Pre/post Increment/Decrement

Pre	Post	Pre	Post
$a = 10;$	$a = 10;$	$a = 10;$	$a = 10;$
$b = a++;$	$b = a + 1;$	$b = --a;$	$b = a - 1;$

a	b	a	b	a	b	a	b
10	11	10	10	9	9	10	10
11	11	9	9	9	9	10	10

Eg:  $a = 10;$

Pf( $a++;$ )

O/p: 11

$a = 10;$

Pf( $a + 1;$ )

O/p: 10

$b = a++;$   
• first assign the value &  
increment the value in  
next line.

$a++$ ,  $a + 1$ ,  $a$  are same  
until  $a$  unless you are  
not assigning the value  
to any other variable.

Ex:  $\underline{+--- * \hat{a}++ \times ++-1- - -}$

post increment  
applied only  
on variable 'a'

Ex: $a = 10;$	$a = 10;$	$a = 10;$
$f(a++);$	$f(a + 1);$	$f(a + 1);$
$p.v = 11$	$p.v = 10$	Passing value ( $p.v$ ) = 11
$a.v = 11$	$a.v = 11$	assign value ( $a.v$ ) = 10

Date 2017

Void main()

int m=10;

int n,n,i;

n = ++m;

n = m++;

n--;

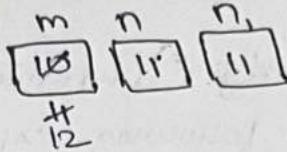
--n;

$$n = n_i; n = n - n_i \Rightarrow n = 10 - 10$$

printf("%d", n); //

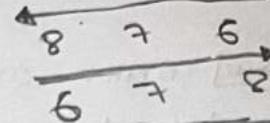
y

O/p = 0



Ex<sub>1</sub>: int a=5

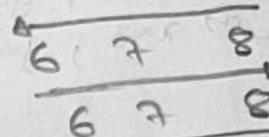
printf(++a, ++a, ++a);



bad program

Ex<sub>2</sub>: int a=5;

printf(a+1, a+2, a+3);



good program

Note

1. program should work based on logic not based on compilers
2. Incrementing same variable multiple time in a single line is considered bad programming.

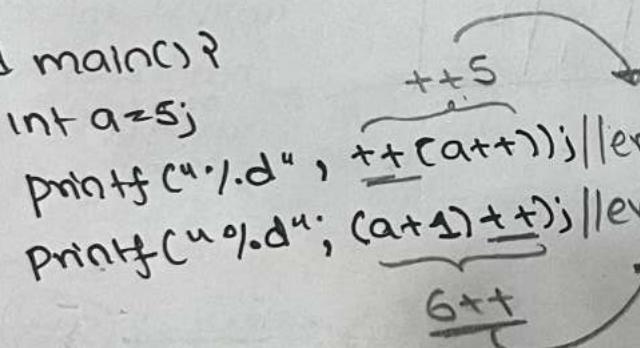
Ex<sub>3</sub>: void main()

int a=5;

printf("%d", ++(a++)); //error

printf("%d", (a++)++); //error

y



Note

- the pre increment & post increment will work only on the variable but not on the constant, but not on the expression

- Loops.
  - 1. for
  - 2. while
  - 3. do-while

Ex. WAP to print 1 to 10 using above loops

1) void main() {
 int i;
 for(i=1; i<=10; i++)
 printf("%d", i);
 }

2) void main() {
 int i=1;
 do {
 printf("%d", i);
 } while(i<=10);
 }

also be written in

3) void main() {
 int i;
 for(i=1; i<=10; printf("%d", i++));
 }

$$\begin{matrix} 3 \leq 10 \\ 2 \leq 10 \\ 1 \leq 10 \end{matrix}$$

y

i = 1, 2, ..., 10

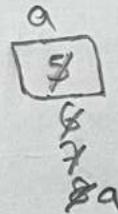
O/p: 1, 2, ..., 10

Ex. void main() {
 int a=5;

while (a++ <= 7) {
 printf("%d", a);
}

y

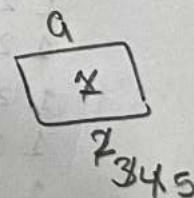
O/p: 9



Ex. void main() {
 int a=1

do {
 while (a++ < 2);
 while (a++ <= 2);
 printf("%d", a);
}

y  
O/p: 5



Note

```
int a() {
  ...
  ...
  return x;
}
```

return value is not having any impact

for (an; ac; ac)

it will execute exactly once

it will execute min once & return value is having impact

it will execute min zero time & return value is having no impact

H-W ① input n=6

Output should  
be

1  
1 2  
1 2 3  
1 2 3 4  
1 2 3 4 5  
1 2 3 4 5 6

```
void main()
{
    int n=6, i;
    for(i=1; i<n; i++)
        for(int j=1; j<=i; j++)
            printf("%d", j);
}
```

y y y

② if n=6

6  
5 6  
4 5 6  
3 4 5 6  
2 3 4 5 6  
1 2 3 4 5 6

```
void main()
{
    int n=6, i;
    for(i=1; i<n; i++)
        for(j=1; j<=n-i; j++)
            printf(" ");
    for(j=1; j<=n; j++)
        printf("%d", j);
}
```

③ n=7

Output:

1	0	0	0	0	0	0	0	0	1
1	2	.	.	.	.	.	.	0	2 1
1	2	3	.	.	.	.	.	0	3 2 1
1	2	3	4	.	.	.	.	0	4 3 2 1
1	2	3	4	5	.	.	.	0	5 4 3 2 1
1	2	3	4	5	6	.	.	0	6 5 4 3 2 1
1	2	3	4	5	6	7	.	0	7 6 5 4 3 2 1

Word mains?

```

int n=9, i;
for(i=1; i<n; i++) {
    j=1;
    for(j; j<=i; j++) {
        printf("%d", j);
    }
    for(j; j<=2*(n-i); j++)
        printf(" ");
}

```

Q/P

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----

\*switch case:-

- switch case:
    - 1. case and label should have space
    - 2. case label must end with colon(:)
    - 3. case label must be constant
    - 4. case label can't be floating point no.
    - 5. Empty switch case is allowed
    - 6. only one "default" is allowed if it can be placed anywhere in the switch
    - 7. Default will be execute only when no case is making
    - 8. after every case there is a requirement of break otherwise it will execute all further cases. till the occurs of break or till the end of switch.

Ex:-

```

void main() {
    int x;
    scanf("%d", &x);
    switch(x) {
        Case 0: x=x+1;
        Case 1: x=x+2;
        default: x=x+3;
    }
    printf("%d", x);
}

```

IIP	OIP
x=0	0486
x=1	X86
x=2	25

Ex:- void main() {

```

int x;
scanf("%d", &x);
switch(x) {
    Case 0: x=x+1;
    break;
    default: x=x-1;
    Case 1: x=x+10;
    break;
    Case 2: x=x+100;
}
printf("%d", x);
}

```

Keyword

Should have gap

IIP	OIP
x=0	012
x=1	X11
x=2	X102
x=3	X212

Ex: void main()

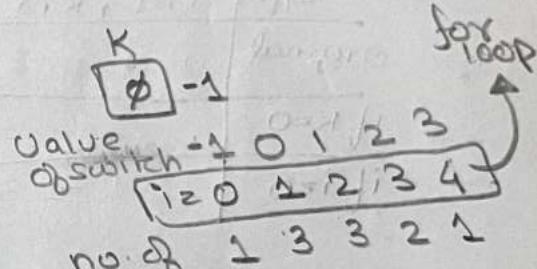
```
int i;  
for (i=1; i<=25; i++) {  
    switch(i) {  
        case 0: i+=5;  
        case 1: i+=3;  
        case 2: i+=4;  
        default: i+=6;  
    }  
    printf("%d", i); // 57  
}
```

- the sum of all value of printf is 57 //

Ex: void main()

```
int i, j, k=0;  
j = 2 * 3 / 4 + 2 * 0 / 5 + 8 * 5; // 24 2  
k = -i - j // -1  
for (i=0; i<5; i++) {  
    switch (i+k) {
```

Case 0:  
Case 1: printf("%d", i+k); call  
Case 2: printf("%d", i+k);  
default: printf("%d", i+k);



- the no. of time printf statement execute is 10

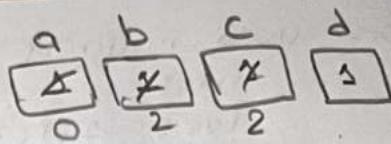
### \* Short circuit evaluation

Ex:  $a=1, b=1, c=1, d=1$

$$a = (\cancel{b++} > 1 \text{ } || \cancel{c++ > 1}) \text{ } \&& \cancel{d++ > 1};$$

`printf(a,b,c,d);`

O/P:  $\Delta 222$

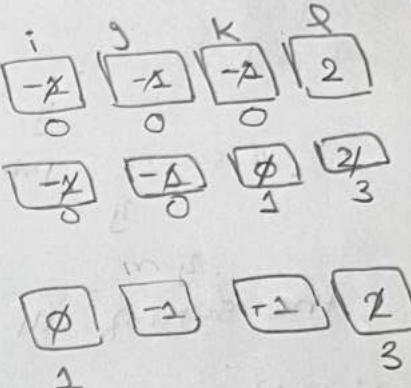


↳ not evaluated

Ex:  $\text{int } i=-1, j=-1, k=-1, l=2, m;$

$$m = ((\cancel{r++} \&& \cancel{j++} \&& \cancel{k++}) \text{ } || \text{ } (l++));$$

`pf(r,s,k,l,m);`



SIP	O/P
Original	r, j, k, l, m
	0 0 0 2 1
if k=0	0 0 1 3 1
if i=0	1 -1 -1 3 1

### Function

- the purpose of function is code reutilization
- the purpose of loop is code repetition.
- the purpose of header file is function reutilization
- if we want to reuse the function then we need to create header file of that particular functn & include that in the required program.

- the function will have the following attributes.
  - Function prototype / declaration
  - Function definition
  - Function calling.

- Prototype contains
  - 1) Name of the function
  - 2) no. of arguments
  - 3) type of arguments
  - 4) return type of argument

```

int add(int, int); // Function prototype
void main()
{
    int x=10, y=20, z;
    z = add(x, y); // Function calling
    printf("%d\n", z); // Actual parameters
}
  
```

y

```

int add(int a, int b)
{
    int c;
    c=a+b;
    return c;
}
  
```

Function definition

Formal parameters & they are called as local variable in the add function

Ex. void mec(int x)

```

printf("%d", x);
if (x <= 2)
    mec(x+1);
printf("%d", x);
  
```

y

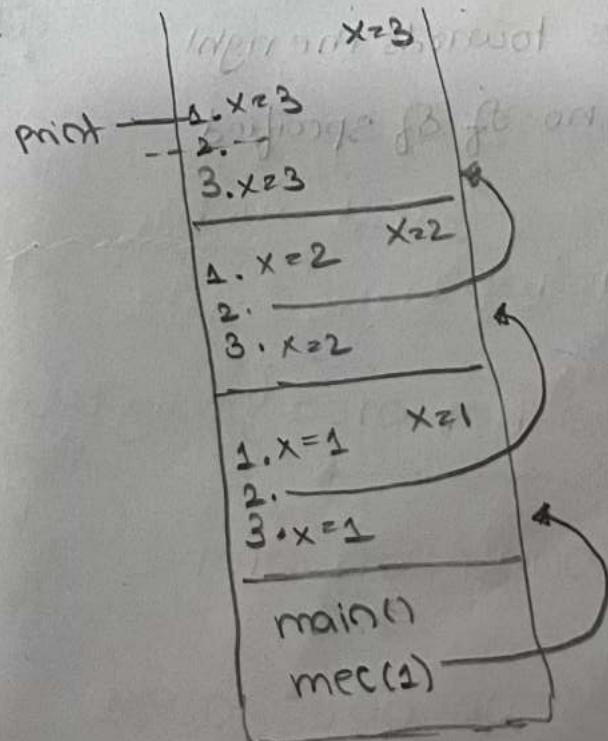
```

Void main()
{
    mec(2); // 123321
}
  
```

m(1) → pf(1)  
m(2) → pf(2)  
m(3) → pf(3)  
pf(1)      pf(2)      < pf(3)

what is o/p printed?

123321



## Recursion

- funcn calling itself is known as recursion.
- Recursion should definitely have the termination condition (Base condition) otherwise it will lead to stack overflow.
- Activation Record: the <sup>(memory)</sup> stack frame allocated to a recursive call is called as activation record.
- the creation & the deletion of the activation record depend on the actual funcn calling sequence.

## Bitwise operators

operator	meaning
&	Bitwise AND
	Bitwise OR
>>	Right shift
<<	Left shift.

- Right Shift (">>")
- right-shift operator shift all the bits towards the right by certain no. of specified bits.

Eg: 212 : 11010100

⇒ 0000 0000 1101 0100 why 16bit  
BLC Integer take 2bytes.

212>>2 : 0000 0000 0011 0101  
⇒ 53

212>>7:  
↓ 0000 0000 0000 0001  
⇒ 1

Ex:

```
void main()
int a=25, b=12;
printf("%d", a&b);
printf("%d", a|b);
```

y  
25: 00011001  
12: 00001100  
a&b 8: 00001000

25: 00011001  
12: 00001100  
a|b 29: 00011101

- left shift ("<<"):
- left shift operator Shift all bits towards left by certain no. of Specified bit.

Ex:  $212 = 0000\ 0000\ 1101\ 0100$   
 $212 \ll 2 = 0000\ 0011\ 0101\ 0000$   
 $\rightarrow 848$   
 $212 \ll 3 = 0000\ 0110\ 1000\ 0000$   
 $\rightarrow 1696$

Q:- void main() {

```
int x=4096;
while(x)
    if(x & 1)
        printf("Hello");
    x >>= 1;
```

y  
y

How many time string "Hello" will be printed //

X	4096	1
	2048	2
	1024	3
	512	4
	256	5
	128	6
	64	7
	32	8
	16	9
	8	10
	4	11
	2	12
	1	13

print the Hello only once time

H.W] write a program to store Date, month, year.

of a person in a single Integer variable(Read & print)?

int dob;  $\Rightarrow$  28  $\Rightarrow$  16bit

Date  
1 to 31

5bit

month  
1 to 12

4bit

year

i.e. 2000

-1900

100

7bit

- Pre-processor statements.
- Pre processor will make only substitution & it will not make any calculation
- It will recognize those statement by hash symbol.
- It will done before compilation of the program

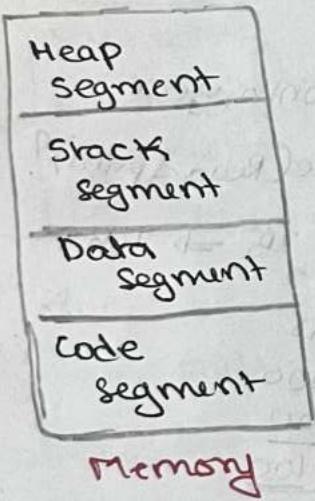
Ex: `# define f1 x****x`

```
void main()
{
    int x=3,y;
    y=f1// x *** x
    printf("%d",y); // 3*3*3
}
```

y

O/p: 27

### Memory Segments



- Data segment
- In the Data segment all the global variable & static variable will be store.

• Main Memory is divided into various segment

#### • Code segment

→ In the Code segment of the memory all the general instruction which will not change throughout the program execution will be stored

Ex: Editors

#### • Stack segment

→ In the stack segment all the recursive functn call, local variable & formal parameters will be stored

#### • heap segment

→ In the heap segment all the dynamic memory allocation will be done, the memory allocation by using the standard functn like malloc(), calloc(), realloc().

Ex: int x=10;

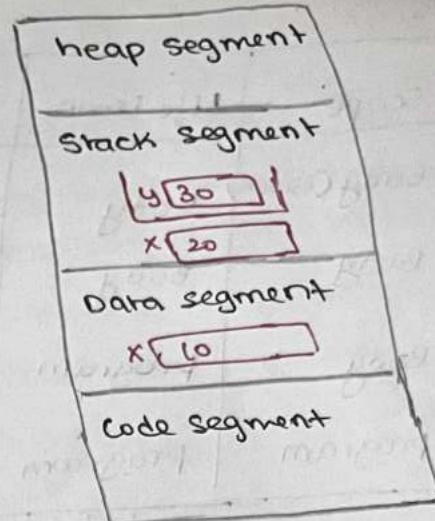
void main()

int x=20;

{  
int y=30;

printf("%d,%d", x,y);  
10,30

y  
printf("%d", x); //20



### Note

- For the global variable the memory will be allocated at compile time, if it is not initialized then default contain 0.
- For the local variable the memory will be allocated at run time if it is not initialized then default contain garbage value.
- Local variable is having high precedence as compare to global variable

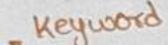
### Storage classes

- In the C language every variable have some characteristics like data type, scope, lifetime & default value.
- Storage classes mainly deal with scope & lifetime
- Scope:**  
the region in which variable is available & accessible is called as scope of a variable

it's default for local variable  
auto } used for local variable  
register } it's not default.  
static }  
extern } they are for both local & global.

Type	Scope	Lifetime	Default Value	Storage	Memory Segment
auto	Body (Block)	Body	Garbage value	RAM	Stack Segment
register	Body	Body	Gr-V	Registers	CPU registers
<del>AMP</del> static	Body	Program	0	RAM	Data (Static) Segment
extern	Program	Program	0	RAM	Data Segment

- auto.

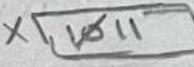
Syntax:  Keyword  
auto datatype  
variable-name;

Ex: auto int x;

Ex: void main()

```
test(); // 10
test(); // 10
test(); // 10
```

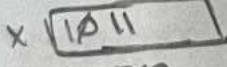
y

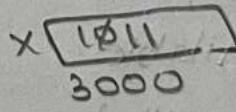
x   
1000

void test() {  
 auto int x=10;

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

y

x   
2000

x   
3000

extern  
↓  
is available  
to linker

• Auto  
register } no linker  
static

• Object with the auto class are not available to the linker

• you can't use for global variable.

• you can't use auto  
register for global  
variable

## • Registers

Syntax: `register` → Keyword

`register` data-type variable-name;

not a cmd. it's a request

Ex:  
`register int x;`

- the keyword `register` is request by the compiler to allocate the memory in CPU registers

- If the free CPU register are available then it will be allocated to the register otherwise it will be allocated to RAM only
- register variables are faster as compare to auto variables. Hence they will be used in loops & functions.
- By default register variables will behave like auto variables.

## • Static

Syntax:

`static` data-type variable-name

Ex: `static int x;`

- the variable persists its previous state value even though the destruction of several functions call is called as

- Static variable is created at memory will be allocated only once in the data segment at compile time.
- If it is not initialized the default contains zero.
- It has a local scope but lifetime is entire program.

Ex: void main() {

```
    k();
    k();
    k();
```

```
void test() {
    static int x=10;
    printf("%d",x);
    x++;
}
```

3

O/P: 10, 11, 12

### Data Segment

x 10 11 12 13

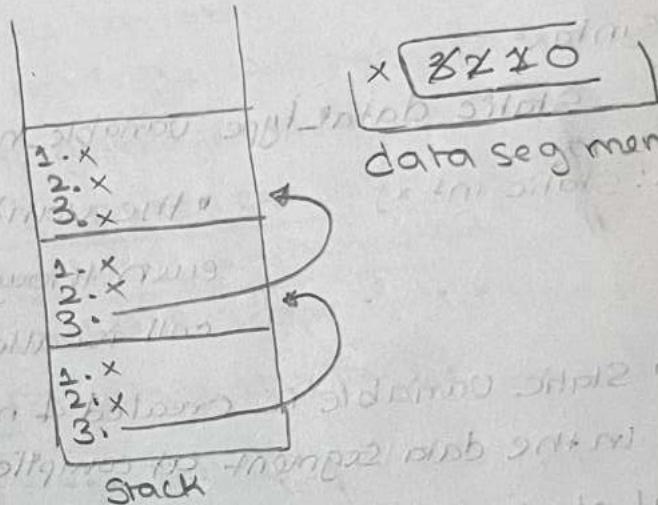
expires  
when  
program  
terminates

Ex: void main() {

```
1. static int x=3;
2. printf("%d",x--);
```

```
3. if(x)
   main();
```

O/P: 8 2 1



x 8 2 0

data segment

Same program without static.

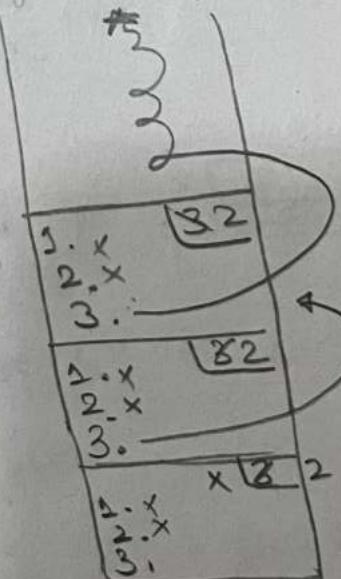
void main() {

```
1. int x=3;
2. printf("%d",x--);
```

```
3. if(x)
   main();
```

y

O/P: 3 3 3 . . . . .



Abnormal termination with  
Stack overflow msg

```

Q) int f(int n)?
    static int i=1;
    if(n>5) return n;
    n=n+i;
    i++;
    return f(n);

```

y

```

void main()
    printf("%d",f(1));

```

y

**Note**  
Returns means Coming out of the Activation record

Ans)

```

int f(int n)?
    static int r;
    if(n<=0) return 2;
    if(n>2)?
        r=n;
        return f(n-2)+2;

```

y  
return f(n-1)+r

y

what is the return value

of  $f(5)$  18?

i  
x  
2/3/4

```

void main()
    while(true)
        printf("3");

```

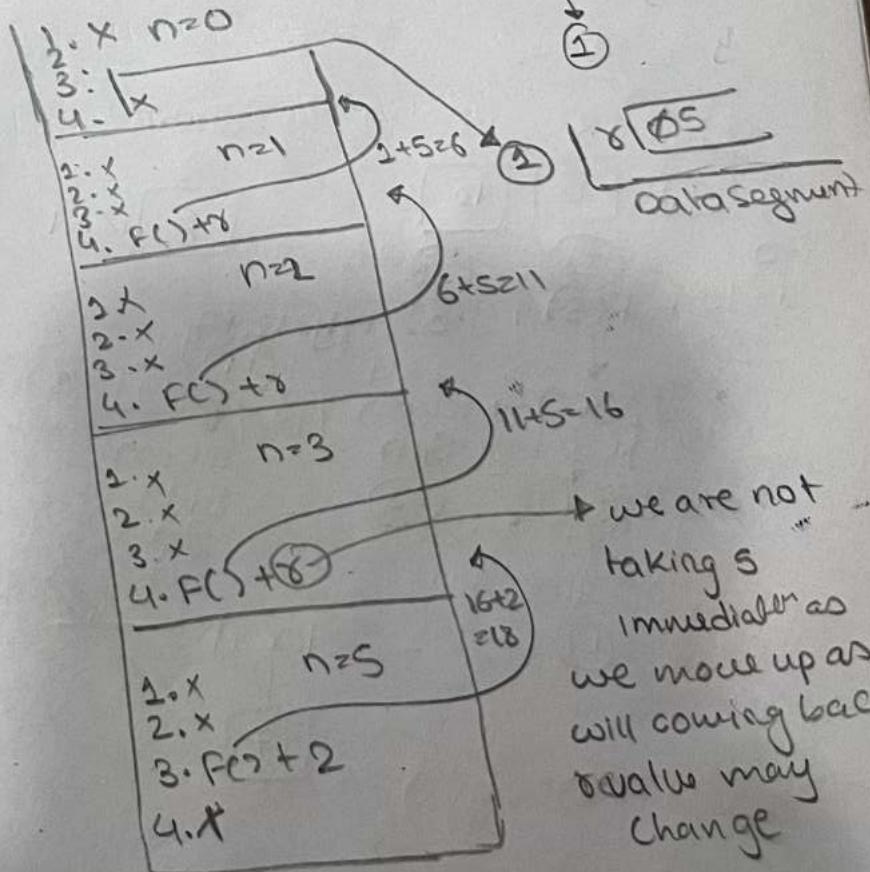
y  
y

0/p: 3 3 3 - - -

Infinite loop.

$F(1) \rightarrow 1+1$   
 $F(2) \rightarrow 2+2$   
 $F(4) \rightarrow 4+3$   
 $F(7) \rightarrow 7 //$

5  
 $F(5) \rightarrow 18$   
 $\textcircled{16} \rightarrow F(3) \rightarrow \textcircled{2}$   
 $\textcircled{11} \rightarrow F(2) \rightarrow \textcircled{5}$   
 $\textcircled{6} \rightarrow F(1) + 5$   
 $\rightarrow F(0) + 5$   
 $\downarrow$   
 $\textcircled{1}$



Q) int incr(int i)?  
 static int count;  
 count = count + i;  
 return count;

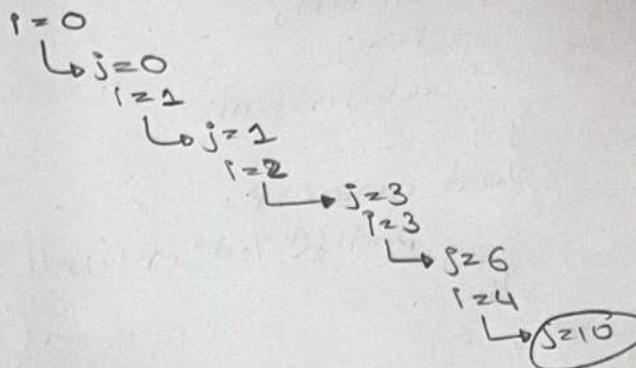
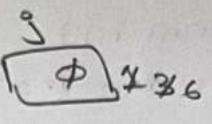
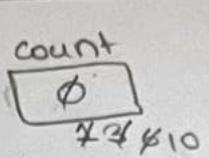
y

void main()?

```
int i, j;
for (i = 0; i <= 4; i++)
    j = incr(i);
printf("%d", j);
```

y

what is o/p printed 10 ?

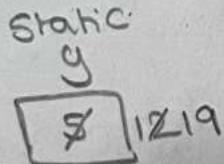
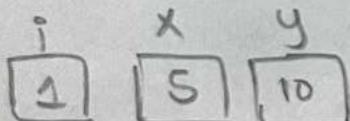


Q) void main()?

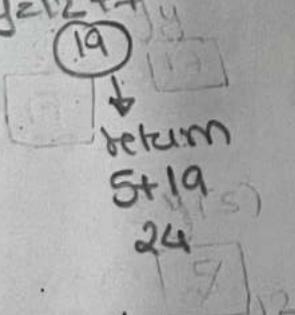
```
int i; x = 5, y = 10;
for (i = 1; i <= 2; i++) {
    y = y + f(x) + g(x);
    printf("%d", y);
}
```

y

y



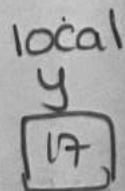
i = 0  
y = 10 + f(5) + g(5) → y = 12 + 7



y

y

int g(int x) ?  
 static int y = 5;  
 y = y + 7;  
 return (x + y);



83

y = 6 + f(5) = 8

```
Q) int r() {  
    static int n = 7;  
    return n--;
```

Ans

6 5 4 3 2 1

```
void main() {  
    for(i=1; i<7; i++)  
        printf("%d", r());
```

Ans

O/P printed?

- a) 4 1
- b) 6 3 0
- c) 6 3
- d) 5 2

W.B  
41  
Pg no.  
132

Solve it  
#include <stdio.h>  
int total(int v) {  
 static int count = 0;  
 while(v) {  
 count += v % 10;  
 v /= 10;  
 }  
 return count;  
}

Ans

```
void main() {  
    static int x = 0;  
    int q = 5;  
    for(; i > 0; i--)  
        x = x + total(i);
```

```
Ans  
printf("%d\n", x);
```

Ans

• Extern.

- no memory will be allocated for extern variable they make use of memory of outside variable (global variable)

Ex<sub>1</sub>: int a=8

main() {

    extern int a;

    printf("%d", a);

These don't have  
memory

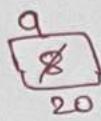
    a=20;

    printf("%d", a);

}

O/p: 8 20

a is a local variable  
in the main functn & it  
will make use of memory  
of a global variable.



- Ex<sub>2</sub> main() {

    extern int a;

    printf("Hello");

}

O/p: Hello.

Ex<sub>3</sub>:

main() {

    extern int a;

    printf(a);

}

no variable  
outside  
so it will  
throw  
error

O/p: error

Ex<sub>4</sub>: int a=5;

main() {

    extern int a;

    extern int a;

    extern int a;

    printf(a);

}

multiple declaration of  
some variable in the same  
functn is allowed only if  
it is extern.

O/p: 5

Ex<sub>5</sub>: main() {

    extern int a;

    printf(a);

    int a=10;

if these  
variable  
not there

O/p: 10

then  
these  
will not  
work

Ex<sub>6</sub>: main() {

    int a; } X

    int a;

    int a;

O/p: error

Ex<sub>1</sub> main() {  
    extern int a=10;  
    printf();  
}

y  
O/P: error

Initialization of a extern  
Variable is not allowed  
only assignment is allowed

Ex<sub>2</sub>: int a;

main() {  
    extern int a;  
    printf();  
}

y  
O/P: 0

Ex<sub>3</sub> int a=20; → D,I  
main() {  
    int a=30;  
    int b; → D  
    b=20 → A  
}

B() {  
    Static int c=30;  
    a=10; → D,I  
    y → A  
}

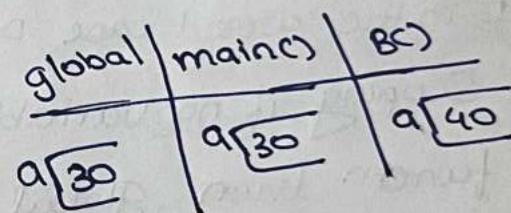
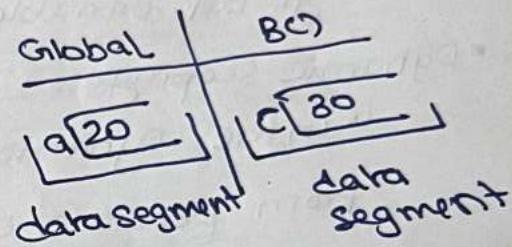
y  
∴ purpose of declaring the variable is to  
allocate the memory.

Ex<sub>4</sub> int a=30; → D,I

main() {  
    Static int a=30;  
    B();  
}

B() {  
    Static int a=40;  
    C(); → D,I  
    y  
    C() {  
        extern int a;  
        y → D  
    }

Declaration: D  
Initialization: I  
Assignment: A



Ex<sub>5</sub> a=10; → D,I  
main() {  
    a; → D  
    a=20; → A  
}

B() {  
    a=20; → D,I  
    b=40; → D,I  
    b=50; → A  
}

y

## Scope

- It is categorized into two type
  - Static scoping (Compile time)
  - Dynamic scoping (Run time)

Scoping rules are applicable only to the "free variable"

- Free variable
  - ↳ the variable which is not declared in the function but used in the function is called as free variable

- Static Scoping (faster than Dynamic Scoping)
  - ↳ In the static scoping free variable are resolved by using global variable

- Dynamic Scoping
  - ↳ In the dynamic scoping free variable are resolved from previous call function.

### Note

1. In the worst case dynamic scoping also behave like static scoping if no variable is present in the previous calling function then global variable will be printed (used).
2. C language By default follow static scoping

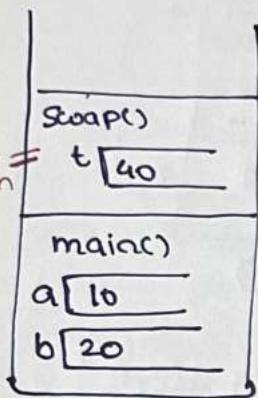
```
Ex: int a=40, b=80;  
void main(){  
    int a,b;  
    a=10, b=20;  
    Swap();  
    pf(a,b);  
}
```

```
Swap(){  
    int t;  
    t=a; // free variable  
    a=b;  
    b=t;  
    pf(a,b);  
}
```

What is O/P using static scoping & dynamic scoping?

Static

delete  
after  
operation



Global variable

a	40	30
b	30	40

Dynamic

O/P: 30,40  
10,20

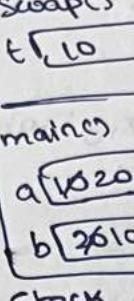
global

a	40
b	30

data segment

O/P: 20,10  
20,10

Scopes



Stack

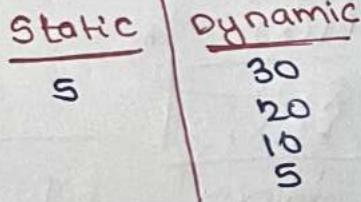
Ex: int a=5;

main()  
int a = 10;  
BC();

BC()  
int a=20;  
C();  
y

C()  
int a=30;  
DC();  
y

DC()  
int a=50;  
Pf(a);  
y



Show doesn't have its own variable so it back  
but main also doesn't have so it refers  
global

Ex: Variable x : real  
procedure show()

begin  
Pfx)

end

procedure small()

begin

variable x:real  
x = 0.125;  
show();

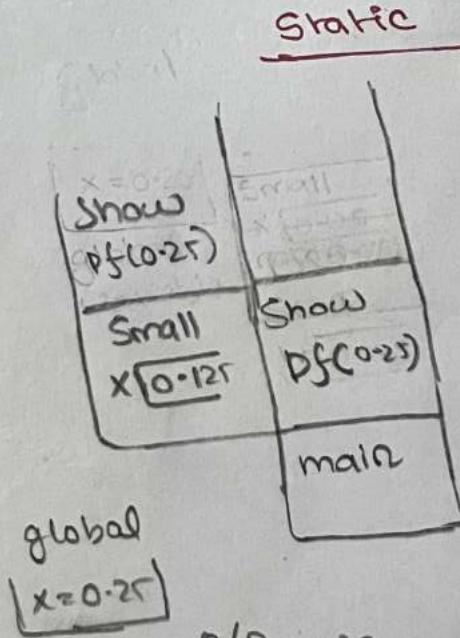
end

begin

x = 0.25

Show  
Small

end



O/P 0.25

0.25

O/P: 0.25  
0.125

Static

Dynamic

Show

Pf(0.25)

Show

Pf(0.25)

main()

global

x = 0.25

Small  
x = 0.125

main()

Ques] Variable x,y : integer

procedure pc variable n : integer)

begin

x = (n+2) \* (n-3);

end

procedure Q()

begin

variable x,y : integer

x=3

y=4

pc(y);

printf();

end

begin

x=7, y=8;

Q();

printf();

main

end

Static

$$x = (4+2) * (4-3)$$

x(2)  
y(4)  
pc(4)  
pc(2)

(L1)

Q()  
Pf(6)

D/P : 3,6

global  
variable

Dynamic

$$x = (4+2) * (4-3)$$

x(86)  
y(4)  
pc(y)  
Pf(x)

(L1)

Q()  
Pf(x)  
main()

D/P : 6,7

global  
variable

Q1 int a=1, b=2;

main {

int a=20, b=30;

pf(a,b);

c();

pf(a,b);

d();

}

c() {

int b=6;

pf(a,b);

a=7, b=8;

}

.

O/P 20 30 50 2 1 2 3 4 3 6 50 4

20 30 7 8 3 4 3 6

c() {

int a=50;

pf(a,b);

a();

pf(a,b);

y

a [28 78 7]

b [24 84 8]

global  
variable

EC()

DOA

c()

main

d() {

pf(a,b);

a=3, b=4;

pf(a,b);

EC();

y

b [6

pf(3,6)

b [6

pf(3,6)

pf(1,2);

pf(3,4);

EC();

a [50

pf(50,1,2);

DC();

pf(50,8)

a [20

b [30

pf(20,30);

c();

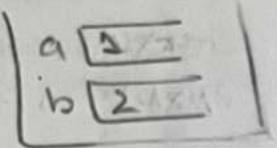
pf(20,30);

DC();

EC();

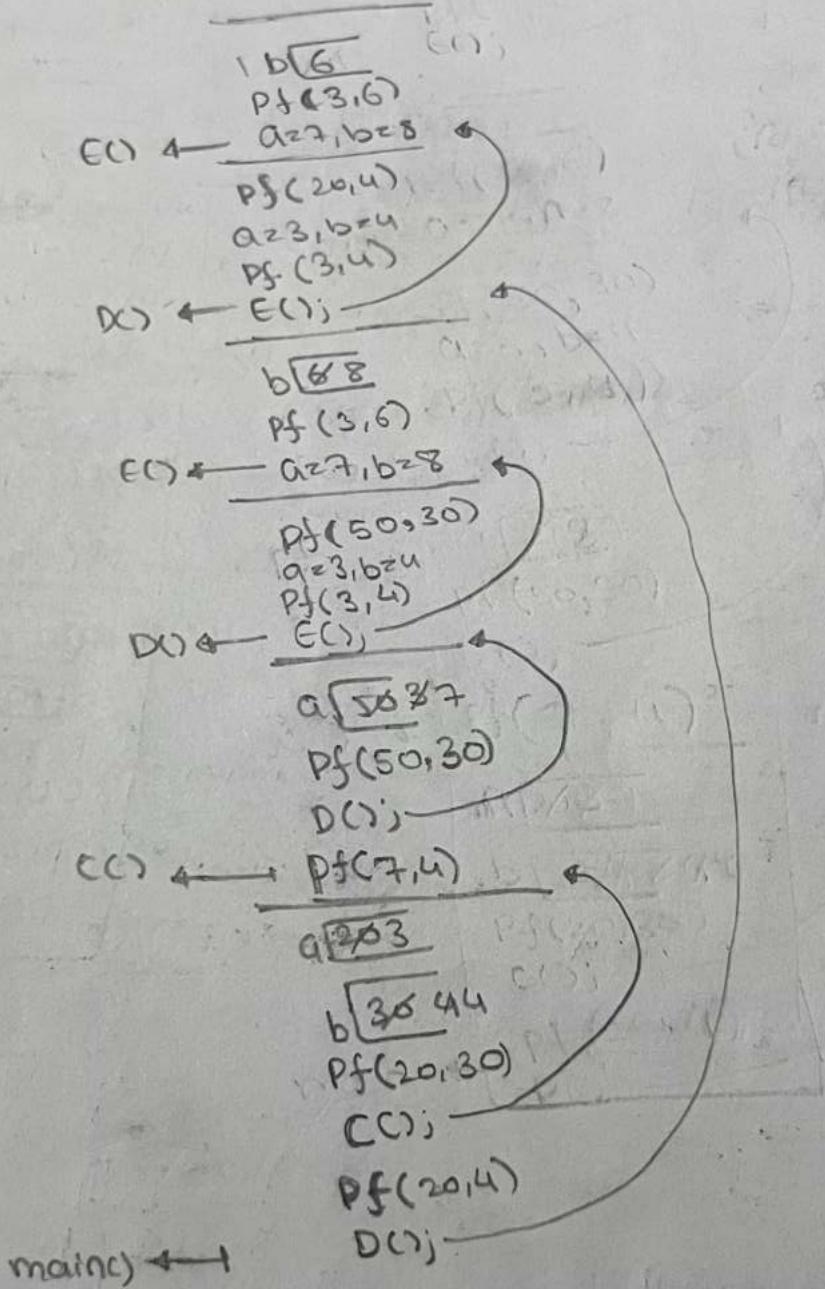
main

## Dynamic



global

O/P: 20 30  
50 30  
50 30  
3 4  
3 6  
7 4  
20 4  
20 4  
34  
36





Q) int a=5, b=6;

main() {

    auto int a=1;

    pf(a,b);

    c();

    a=2, b=3;

    e();

    pf(a,b);

y

e();

    static int b=27;

    pf(a,b);

    a=59, b=69;

y

Static

a: 5 59 4x159  
b: 6 55 2 69

global variable

O/P: 5 6

c()  
static b

0 5

dc()  
extern a

5 44

e()

static b

2 7 69.

c()

registers int a=25;

static int b;

b=3;

pf(a,b);

a=4, b=5;

y

d()

extern int a

pf(a,b)

e();

a=4, b=55;

y

extern is referring to global variable so if you change in extern then global is changes

O/P: 2 6

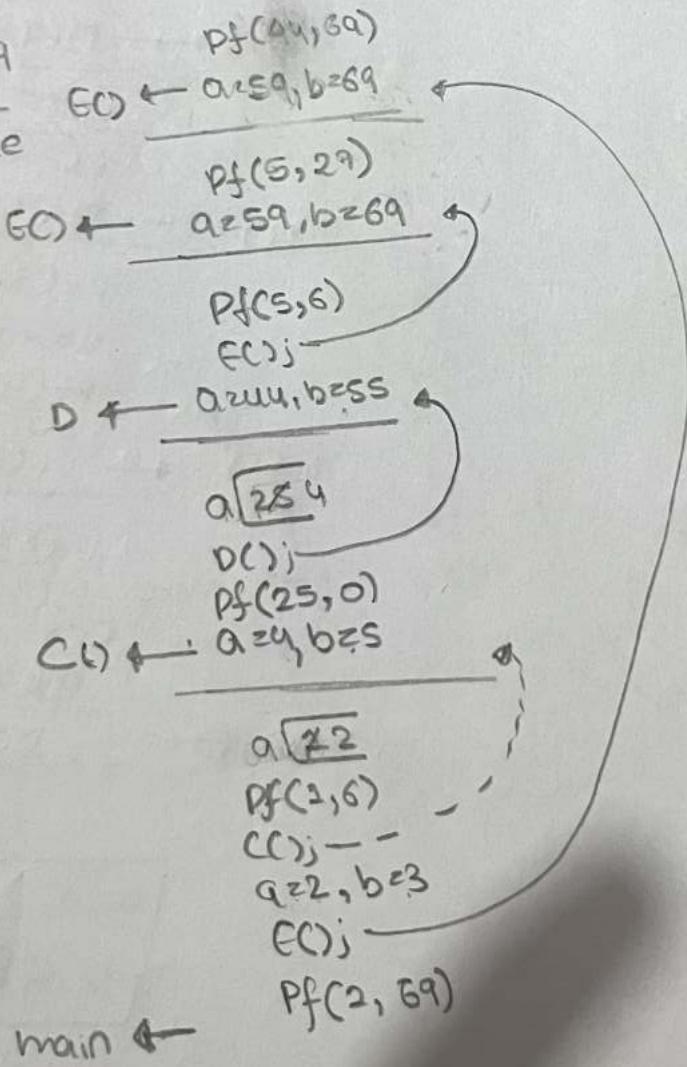
5, 6

5, 27

25, 0

44, 69)

2, 69



Dynamic

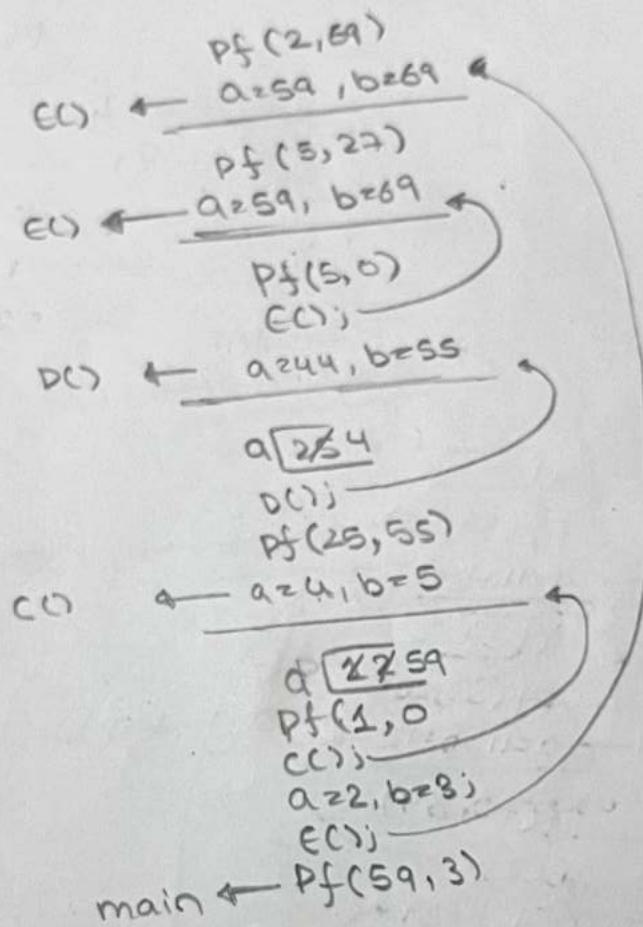
a: 5  
b: 63

global

CC  
Static b  
0 555

DC  
extern a  
2169

EC  
Static b  
246969



O/P: 1, 0

5, 0

5, 27

25, 55

2169

59, 3

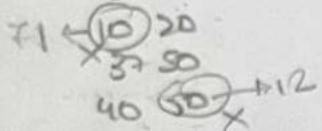
Q1) int a=10, b=20;  
 main()  
 static int a=30;  
 pf(a,b);  
 cc();  
 a=40, b=50;  
 EC();  
 pf(a,b);

3

O/P: 30 10

10 30

30 20



C()?  
 extern int a;  
 D();  
 pf(a,b);  
 a=44, b=55;  
 y

D()?  
 static int b=37;  
 pf(a,b);  
 EC();  
 a=21, b=81;

y

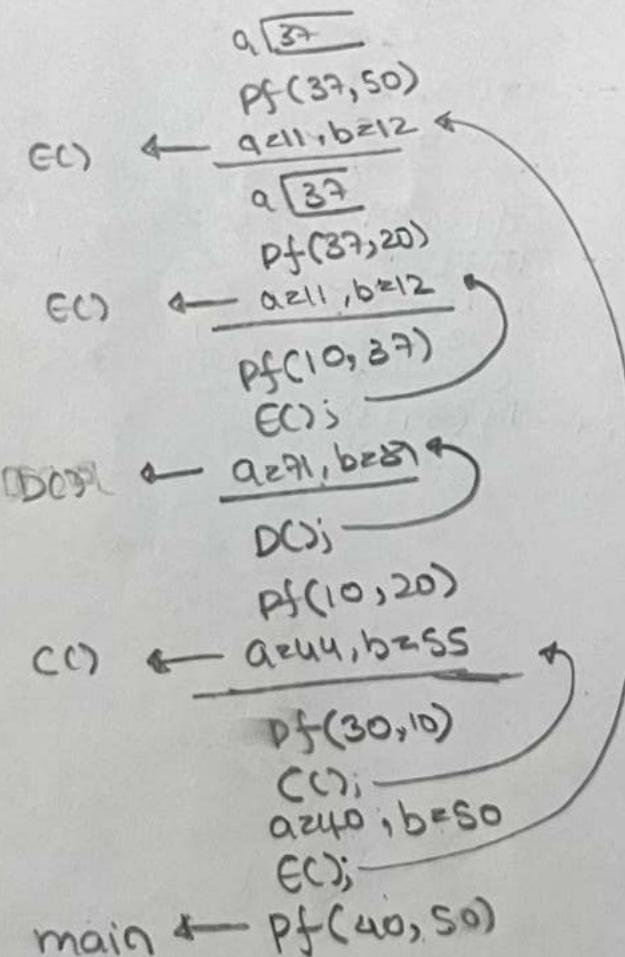
E()?  
 int a=37;  
 pf(a,b);  
 a=11, b=12;

y

DC)  
 static  
 b [34] 21

CC)  
 extern  
 a [10] 44

main  
 static  
 a [36] 40



a: 10 21 44  
 b: 20 55 50

global variable

## Dynamic

0/P: 30, 20  
 10 37  
 37 37  
 71 20  
 37 50  
 40 12

global variable

a: 16x44x11  
 b: 20x56x12

main

state

a [38x40]

CC

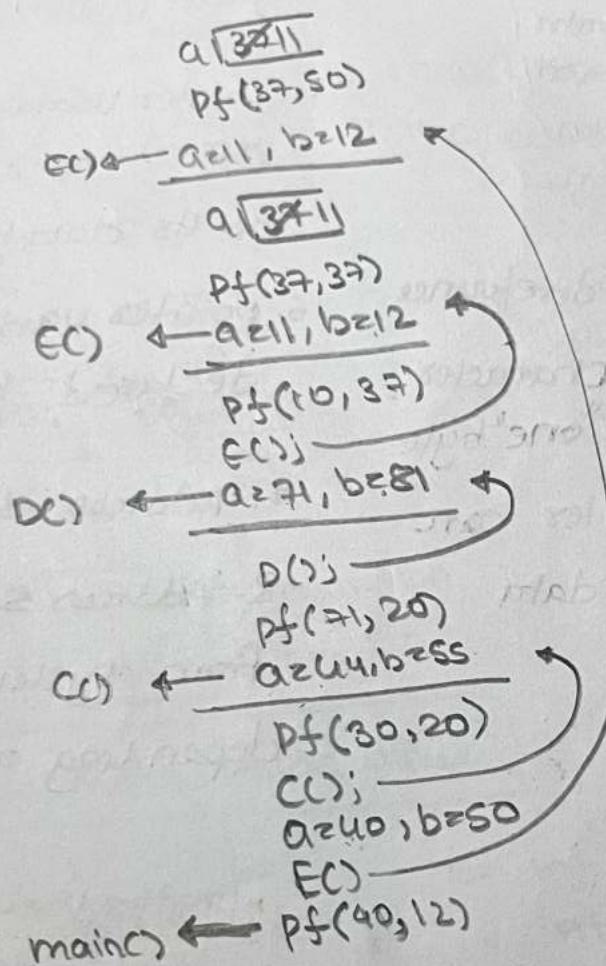
extern

a [16x44]

DC

Static

b [34x28]



## Pointers

1. pointer is a variable which is used for to store the address
  2. the two different operation are used in the pointers
    - a)  $\&$  → address of operator
    - b)  $*$  → indirection operator/dereference operator/object at location/value at location
  5. Integer pointer can dereference "two" bytes of data/character pointer can dereference "one" byte of data & float pointer can dereference "4" byte of data
3. pointer variable takes same byte of memory irrespective of its datatype  
(It can be integer pointer, character pointer, float pointer).
4. pointer variable can dereference (access) no. of bytes depending on its data type
  6. pointer variable can be declared by using ("\*")
  7. Address can't be negative
  8. Address size will change from system to system depending on O.S.

### • Pointers :

- Basic
- pointer to pointer
- pointer to array
- Array of pointers
- multi-dimensional Array
- pointer to string
- pointer to structure & funct'

- pointer variable takes same byte of memory w.r.t data-type
- $\&$  pointer Variable can dereference/access no. of byte depending on the datatype

• Basic:

Ex: int \*bj

b is a pointer variable of type integer coherence we can store address of some integers (any)

(or)

b is an integer pointer

int \*bj

↓ Second first

- operand has high precedence  
i.e b is a integer variable

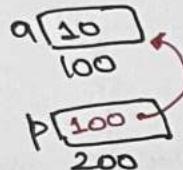
Ex: int a=10; a [10] 100

int b;  
b = &a // ~~(x)~~ b [ ] 200

∴ we can't store address of a variable in the normal integer variable.

Ex:

int a=10;



$$Pf(a) = 10$$

$$Pf(&a) = 100$$

$$Pf(p) = 100$$

$$Pf(&p) = 200$$

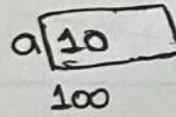
$$Pf(*p) = 10.$$

Declaration & Initialization

int \*p = &a;  
int \*p;  
p = &a

"Variable name means inside value"

Ex: int a=10;



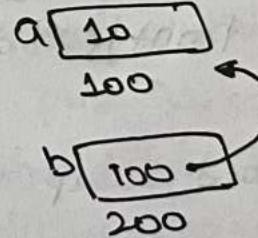
$$Pf(a) = 10$$

$$Pf(&a) = 100$$

$$Pf(*\&a) = 10$$

note  
 $a = *\&a$

Ex: int a=10;



$$Pf(b) = 100$$

$$Pf(a) = 10$$

$$Pf(*b) = 10$$

$Pf("%d", a * b);$  // error

Compiler ask b  
if the left \* is yours  
or not do "b" said  
check my declaration

$Pf("%d", a * *b);$  // 100

Ex: int a=10;  
int \*b=&a;

$$Pf(b) = 100$$

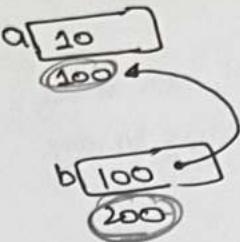
$$Pf(*b) = 10$$

$$Pf(**b) = \text{error}$$

$$Pf(*\&b) = 100$$

b

$$Pf(*\&*b) = 10.$$



### parameter passing techniques:

- ▶ call by value
- ▶ call by reference ↗  
call by Address

#### Call By Value

Void main()

int a=10; b=20;

Swap(a,b); *actual parameters*

printf("%d %d", a, b);

y

void Swap(int x, int y);  
int t;  
t=x;  
x=y;  
y=t;

↳ Formal  
Parameters

y

O/P : 10,20

main()

a 10  
100

b 20  
200

x 20  
300

y 10  
400

t 10  
500

swap()

note

Whatever the value change in the formal parameters are not reflected into actual parameters

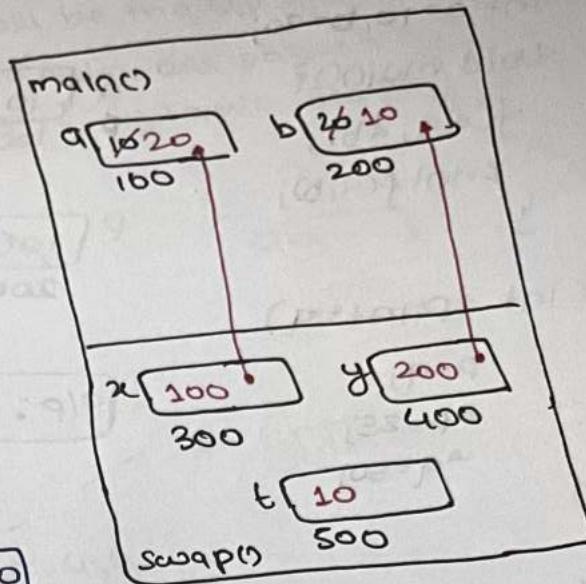
## Call by Reference

```
void main() {
    int a=10,b=20;
    swap(&a,&b);
    printf("%d %d",a,b);
}
```

y

```
Void swap(int *x,int *y) {
    int t;
    t = *x // 10
    *x = *y // 20
    *y = t; // 10
}
```

Olp: 20,10



### Note

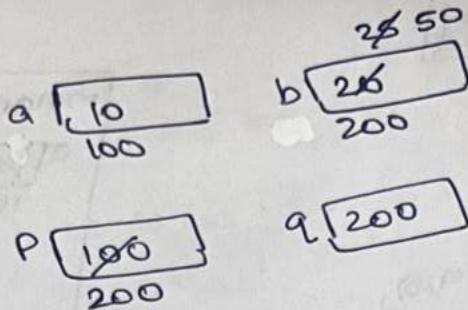
- If the Address is been passed then receiver should be pointer variable
- C language support only call by value but we can achieve call by reference By passing the address

- In the above program actual parameter are getting change b/c of call by reference.

- \* \* in the question if the call by reference is given then whenever parameter is passed in the function we need to understand implicit that address is been passed receiver is a pointer variable & actual parameter will be change

Ex:

```
int a=10, b=20;  
void main()  
{  
    f(&a, &b);  
    printf("%d %d", a, b);  
}
```



Q1 int \*p, int \*q;

p=q;  
\*p=25;  
\*q=50;

O/p: 20,50

Q1 int x;

void Q(int z){

z=z+x;

printf("%d", z);

y

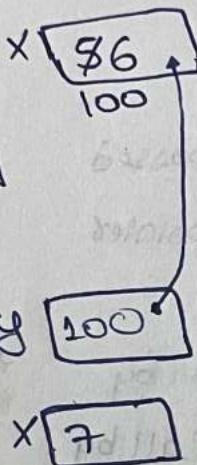
void p(int \*y){

int x=\*y+2; // 5+2

Q(x); // 7

\*y=x-1; // 6

printf("%d", \*y);



main()

x=5

p(&x);

printf("%d", x);

y

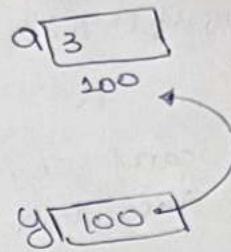
O/p: 12

```

Q1) a = 3;
D,S void f(x) { x=200; }
x = 2*a; // x=2*3*4
printf("%d", x); // x=24
y
void main() {
    a=1;
    a=y-a; // a=4-y=4
    nca(j)/200
    printf("%d", a); // a=4
}
y
void main() {
    mca(); // 200
}

```

what will be the O/p of the program  
when parameters are passed by  
reference & dynamic scoping.



O/P  $\neq$  4,4

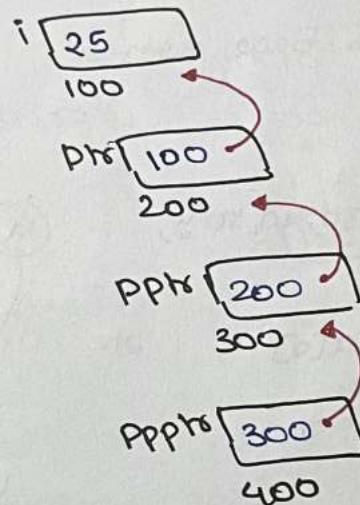
### Pointers to pointers

Ex: void main()

```

int i;
1. int *ptrj
2. int **pptr;
3. int ***ppptr;
i = 25;
ptr = &i;
pptr = &ptr;
ppptr = &pptr;

```



3

1. These is called as single pointer,  
in these we need to store address  
of the integer & it can access upto  
1 level
2. It is called as double pointer or "pointer to pointer",  
In these we need to store address of single pointer & it  
can access upto 2 level
3. It is called as triple pointer or "pointer to pointer to pointer", In these we  
need to store address of double pointer & it can access upto 3 level.

1. i = 25
2. ptr = 100
3. &ptr = 200
4. &i = 100
5. \*ptr = 25
6. \*\*ptr = Error
7. pptr = 200
8. &pptr = 300
9. \*pptr = 100
10. \*\*pptr = 100
11. ppptr = 300
12. &ppptr = 400

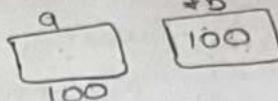
13. \*ppptr = 200
14. \*\*ppptr = 100
15. \*\*\*ppptr = 25
16. \*\*\*\*ppptr = Error
17. \*\*\*\*\*ppptr = 25

gate:

```
Void main() {
    int a;
    int *b;
    b = &a;
}
```

int \*b  
b = &a;

D.S.  $\leftarrow$  int \*b = &a;



0  
1  
2  
3  
4  
5  
6  
7  
8  
9

Scansf("%d", b); can be replaced by &a  
printf("%d", \*b+10); // value entered + 10

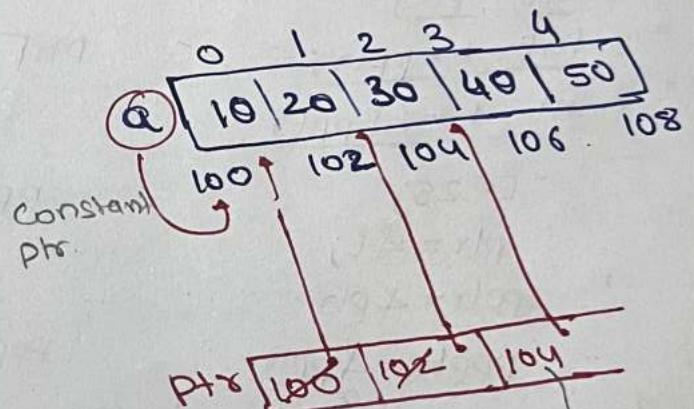
&  
0  
1

### Note

In the Scanf we need to provide address of the variable in which the data should be stored.

- pointer to array.
- Array name without subscript ([ ]), contains constant base address of the array, thus base address can't be modified
- Array name is considered as constant pointer
- Array name means base address.

```
Void main() {
    int a[5] = {10, 20, 30, 40, 50};
    int *ptr;
    ptr = a; // ptr = &a[0]
    ++ptr;
    ++ptr;
    -- *ptr;
    printf("%d", *ptr); // 29
}
```



$$pf(a) \approx 100$$

$$pf(ptr) \approx 100$$

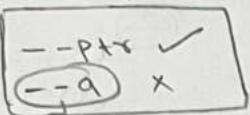
$\boxed{++ptr \checkmark}$   
 $\boxed{++a X}$

y  
ptr increase with respect  
of the size it is stored

Unary operators association  
is right to left

### Note

- 1 when the ptr. variable is incremented then it will increment with respect to the size of element to which it is pointing to
- 2 unary associativity is from right to left
- 3 array name is given as constant ptr b/c that is the only source to the memory that how the array contain can be access.
- 4 In the above example 'ptr' & 'a' both are ptr to array but 'ptr' is a pointer variable & 'a' is the pointer constant.



↳ here we are trying to assign the increment value

$$ptr(a+2)=102 \quad \checkmark$$

allow b/c here we are not assign the value, we are just trying to print the value.

### Pointer Arithmetic

Ex: void main()

int a[5] = {10, 20, 30, 40, 50};

O/p: 108-100 Int \* p1 = a;

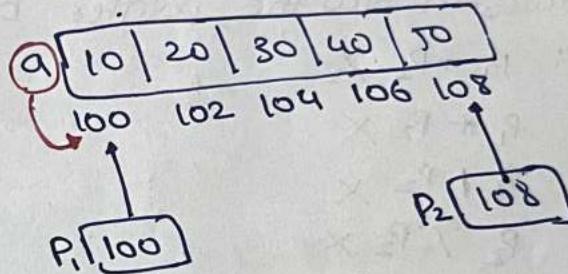
→ 8

→ 8/2  
→ 4

Int \* p2 = a+4;

printf("%d", p2 - p1);

y



① Ex: a+2

$$p_1 = p_1 + 2$$

$$= 100 + 2$$

$$\Rightarrow 100 + 2 * 2$$

↳ size of element

$$\Rightarrow 104.$$

- 1) we can add integer constant to an ptr. it means skip the element in the forward direction
- 2) we can subtract integer constant from the pointer it means skipping the element in the backward direction.

$$Ex_2: \quad p_2 = p_2 - 2$$

$$= 108 - 2$$

$$\Rightarrow 108 - 2 * 2$$

↳ size of Element  
⇒ 104 //

3. we can perform subtraction b/w 2 pointers,  
these is allowed only when both are pointing  
to same array &  $P_1 \leq P_2$  & "size of  
the element should be same".

$$\begin{aligned} \text{Ex: } P_2 - P_1 \\ = 108 - 100 \\ = 8 \\ \Rightarrow \frac{8}{2} = 4 \end{aligned}$$

[these operation gives no. of elements from  
 $P_2$  before  $P_1$ .]

$$\text{Ex: } P_2 - P_1$$

$$\begin{aligned} &= 108 - 100 \\ &\Rightarrow 8 \\ &\Rightarrow \frac{8}{2} \xrightarrow{\text{Size of Element}} 4 \end{aligned}$$

4. we can't add floating point no. to the  
pointers b/c it has no meaning.

$$\text{Ex: } P_1 + 2.5 \times$$

$$\Rightarrow 4$$

$$P_2 - 2.5 \times$$

5. we can't perform addition, multiplication &  
division b/w the pointers b/c it has no meaning

$$\text{Ex: } P_1 + P_2 \times$$

$$P_1 * P_2 \times$$

$$P_1 / P_2 \times$$

$$P_2 \% P_2 \times$$

Passing an array to the function

in the C language if we want to pass an array elements  
to the functn. then it is not a good idea to pass all the array  
element instead by simply passing the base address of the array  
we can access all the array element

Ex: void main()

int a[5] = {10, 20, 30, 40, 50};

display(a, 5); // 100, 5

y

void display (int \*p, int n) {

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

printf("%d", p[i]);

y

O/P : 10, 20, 30, 40, 50

P[0]

\*(P+0)

\*(100+0)

+ (100)

10

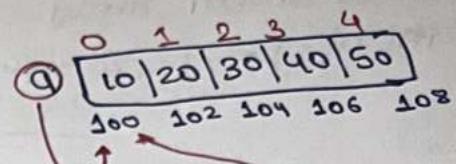
P[1]

\*(P+1)

\*(100+1)

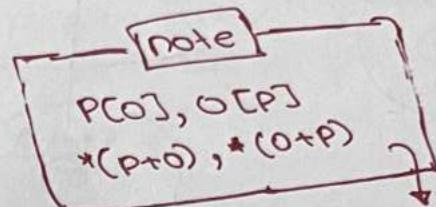
\*(102)

20



pfa = 100  
p+(a) = 100

p | 100



all are equal.

### Q) Gate

int f(int \*a, int n) {

if (n ≤ 0) return 0;

else if (\*a % 2 == 0)

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

else

return \*a - f(a-1, n-1);

y

main()

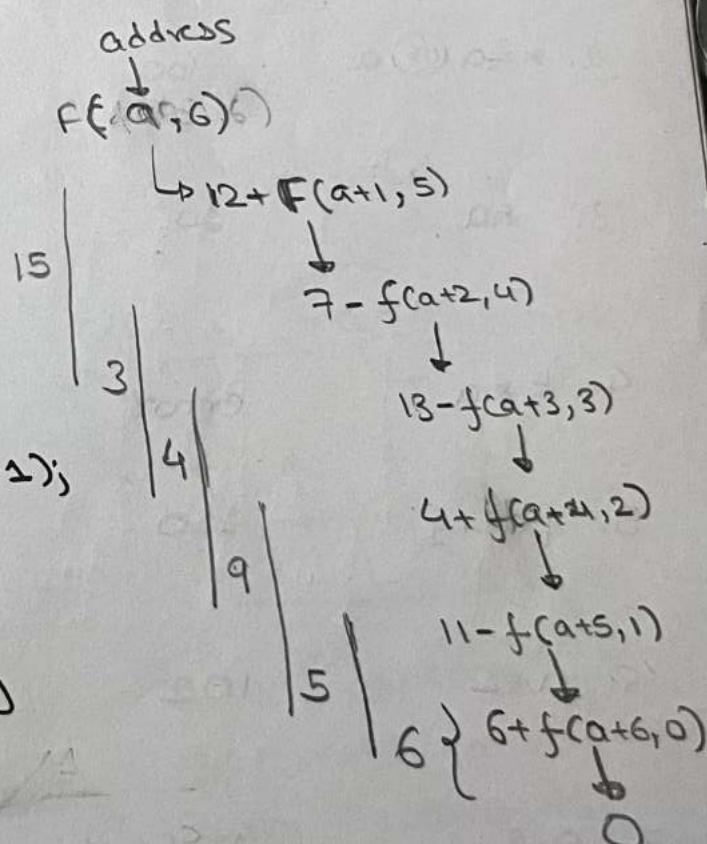
int a[6] = {12, 7, 13, 4, 11, 6};

printf("%d", f(a, 6));

y

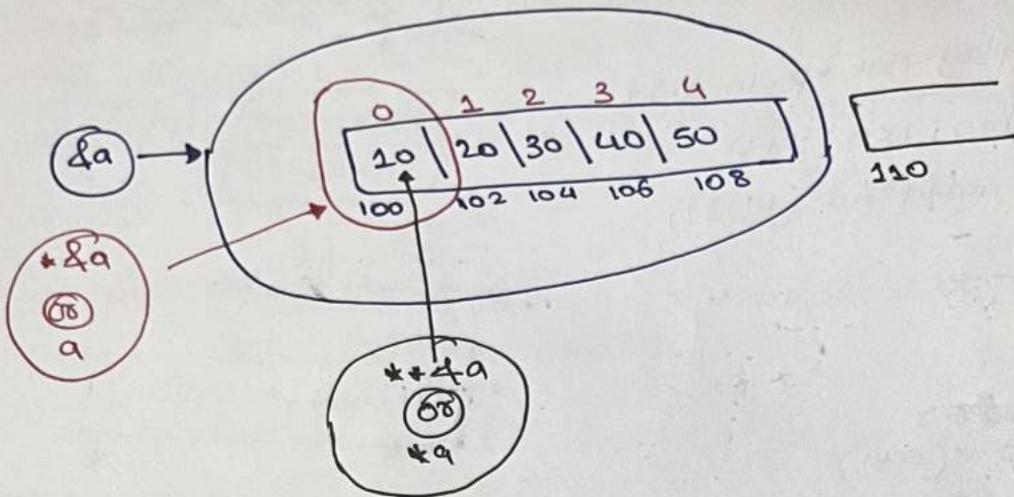
O/P : 15

⇒ 12 - 7 - 13 + 4 - 11 + 6 = 0



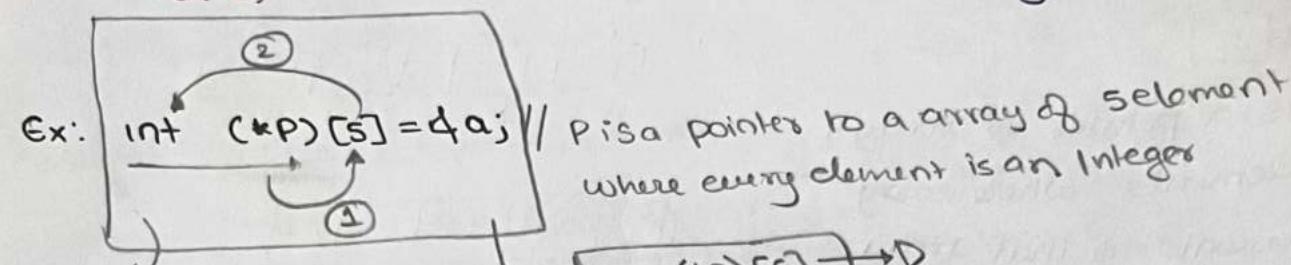
• 1-Dimensional Array:

Eg:  $\text{int } a[5] = \{10, 20, 30, 40, 50\}$



Expression	Print value	Type	meaning
1. $\&a$	100	$\text{int } (*)[];$	Base address of the entire 1D array
2. $*\&a$ or $a$	100	$\text{int } *;$	Base address of the zeroth element of the array
3. $*a$	10	$\text{int}$	Value of the zeroth element of the array
4. $**a$	error	?	referring illegal location
5. $\&a + 1$	110	$\text{int } (*)[];$	Base address of the next continuous 1D array
6. $a + 1$	102	$\text{int } *;$	Base address of the 1st element of array
7. $a + 3$	106	$\text{int } *;$	Base address of the 3rd element of array

Expression	Print value	Type	meaning
8. $*(\text{a}+3)$	40	int	value of the 3rd element of an array
9. $\text{a}[2] \neq *(\text{a}+2)$	30	int	value of the 2nd element of an array



Both Declaration & Initialization

$$\text{pf}(\text{P}+1) = 110$$

↓ Same as da

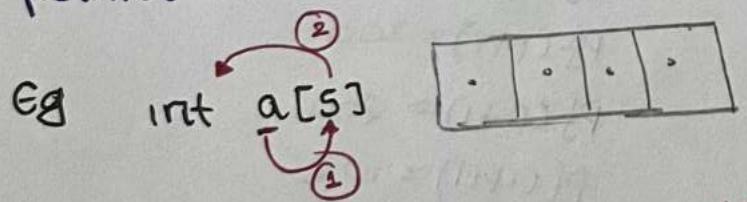
$$\text{Ex}: *(\text{da}+1) - \text{a}$$

$$\Rightarrow 110 - 100$$

$$\Rightarrow \underline{10}$$

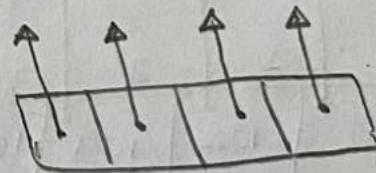
### Array of Pointers

- Just like we had array of integers
- Similarly we also have array of pointers



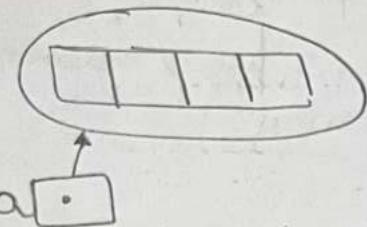
a is an array of 5 elements where every element is an integer

$$\text{Ex: } \text{int } * \text{a}[5]$$



a is an array of 5 elements where every element is an integer pointer

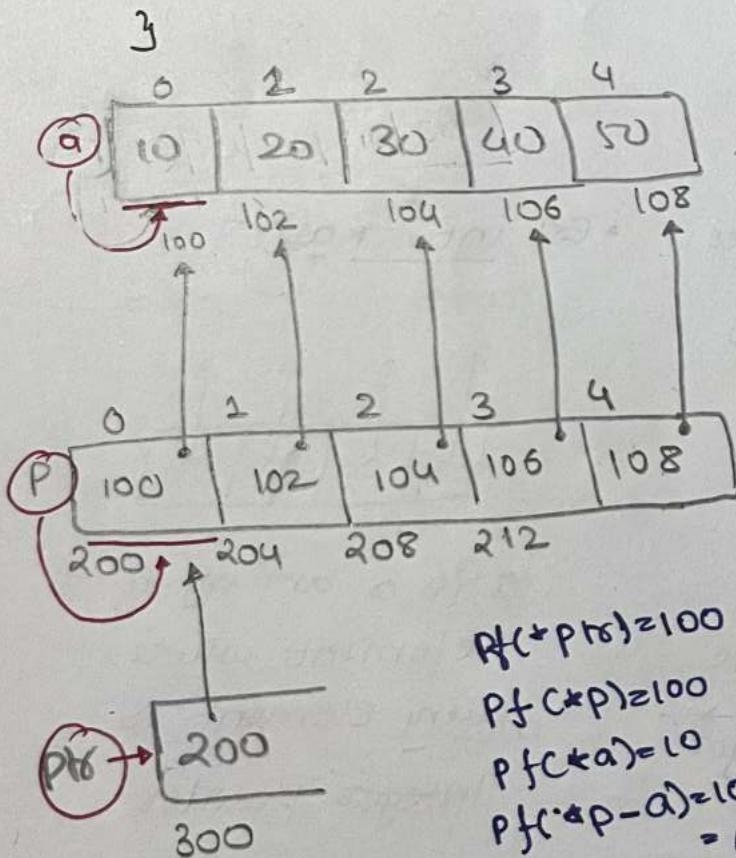
Ex:  $\text{int } \underline{\text{c}}[\ast \text{a}][5];$



$a$  is a pointer to array of 5 elements where every element in that array is an integer.

Ex: main() { int  $a[5] = \{10, 20, 30, 40, 50\};$   
 $\text{int } \ast p[5] = \{a, a+1, a+2, a+3, a+4\};$

$\text{int } \ast \ast \text{ptr};$   
 $\text{ptr} = p;$



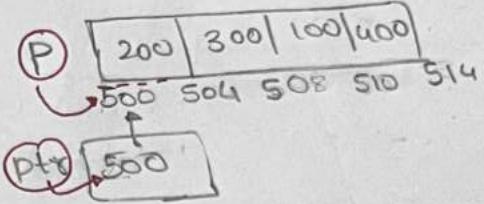
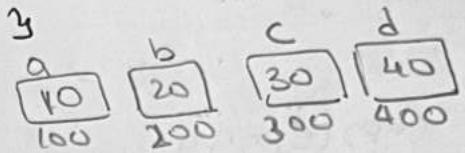
$$\text{pf}(\ast \text{ptr} - 9)$$

$$\Rightarrow 100 - 100$$

$$= \frac{0}{8.0 \text{ E}}$$

Ex: void main() {

$\text{int } a = 10, b = 20, c = 30, d = 40;$   
 $\text{int } \ast p[4] = \{4b, 4c, 4a, 4d\};$   
 $\text{int } \ast \ast \text{ptr} = p;$



$$\begin{aligned} \text{pf}(a) &= 10 \\ \text{pf}(P) &= 500 \\ \text{pf}(p[0]) &= 500 \\ \text{pf}(p[1]) &= 504 \\ \text{pf}(p[2]) &= 508 \\ \text{pf}(\ast p[0]) &= 200 \\ \text{pf}(\ast p) &= 200 \end{aligned}$$

$$\begin{aligned} \text{pf}(a) &= 100 \\ \text{pf}(P) &= 200 \\ \text{pf}(\ast p) &= 200 \\ \text{pf}(\ast \ast a) &= 10 \\ \text{pf}(\ast \ast p - a) &= 100 - 100 \\ &= \frac{0}{8.0 \text{ E}} = 0 \\ \text{pf}(\ast \text{ptr} - 9) &= 100 - 100 \\ &= \frac{0}{8.0 \text{ E}} = 0 \end{aligned}$$

$$\begin{aligned} \text{size of } &\Rightarrow \frac{0}{4} = 0 \\ &\text{size of } \end{aligned}$$

Ex: void main()

int arr = {0, 1, 2, 3, 4};

int \*ptr = &arr[0];

int \*\*kptr = &ptr;

ptr++; // 504

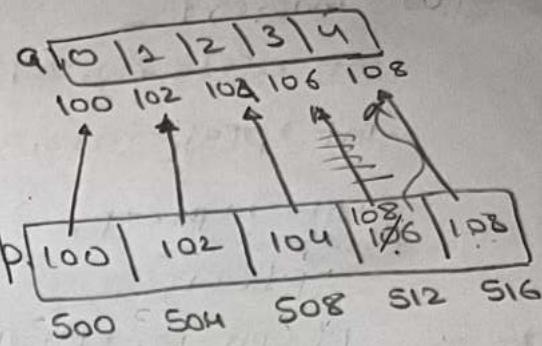
1, 1, 1  $\leftarrow \text{pf}(\text{ptr} - \text{P}, * \text{ptr} - \text{a}, ** \text{ptr});$   
 $* \text{ptr}++; // *(\text{ptr} = \text{ptr} + 1)$

2, 2, 2  $\text{pf}(\text{ptr} - \text{P}, * \text{ptr} - \text{a}, ** \text{ptr});$   
 $* \text{ptr}++; // *(\text{ptr} = \text{ptr} + 1)$

3, 3, 3  $\text{pf}(\text{ptr} - \text{P}, * \text{ptr} - \text{a}, ** \text{ptr});$   
 $++ * \text{ptr}; * \text{ptr} = * \text{ptr} + 1$

3, 4, 4  
y  
 $\Rightarrow 508 - 500 | 104 - 100 | 2$   
 $\Rightarrow \frac{8}{4} = 2 | \frac{4}{2} = 2$

$\Rightarrow 512 - 500 | 106 - 100 | 3$   
 $\Rightarrow \frac{12}{4} = 3 | \frac{6}{2} = 3$



ptr  $\boxed{500 | 504 | 508 | 512}$

$$504 - 500 \Rightarrow \frac{4}{4} = 1$$

$$4 | 102 - 100 \Rightarrow \frac{2}{2} = 1$$

4 | 1

$$\Rightarrow 512 - 500 | 108 - 100 | 4$$
$$\Rightarrow \frac{12}{4} = 3 | \frac{8}{2} = 4$$

Q) void main() {

int a[5] = {10, 20, 30, 40, 50};

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

int \*\*ptr = p;

\*\*ptr++; 500

2, 1, 20 pf(ptr - p, \*ptr - a, \*\*ptr);

\*++ \*ptr;

1, 2, 30 pf(ptr - p, \*ptr - a, \*\*ptr);

++ \*ptr;

1, 2, 30 pf(ptr - p, \*ptr - a, \*\*ptr);

y

504-100 | 104-100 | 31  
 $\frac{4}{4}=1$        $\frac{4}{2}=2$

q	10	20	30	40	50
	100	102	104	106	108

p	104	102	106	100	108
	500	504	508	512	516

ptr [500 504]

504-100 | 102-100 | 20  
 $\frac{4}{4}=1$        $\frac{2}{2}=1$

504-100 | 104-100 | 50  
 $\frac{4}{4}=1$        $\frac{4}{2}=2$

note

- if the array is initialized remaining all will be stored as 0.
- if it is not initialized then it will contain garbage value.

Ex: void main() {

int a[5] = { };

pf(a[2]); O/P: 0

y

Ex1. void main() {

int a[5];

pf(a[2]);

O/P:  
Garbage  
Value

y

void main() { }

int a[5] = {10, 20};

pf(a[2], a[3]);

y O/P: 20, 0

- multi dimensional Array.
- array name without subscript Contains base address of 0th element

### 0th element

- If it is 1D array then it means Base address of the normal element  
eg: int a[5];  
 $a \Rightarrow \text{int } *j;$
- If its 2D array then 0th element means Base address of 2D array  
Ex: int a[3][4][5]  
 $a \Rightarrow \text{int } (*\text{ })[ ][ ];$

- If its a 2D array then 0th element means Base address of the 0th 1D array  
eg: int a[3][4];  
 $a \Rightarrow \text{int } (\text{ } *)[ ],$

### 2D array

1. the best example for "pointers to pointers" is 2D array

2. In the above example it represent 3 row & 3 column

3. In the 2D array  $a[i] = \text{constant base address of } i^{\text{th}} \text{ row & } 0^{\text{th}} \text{ element}$

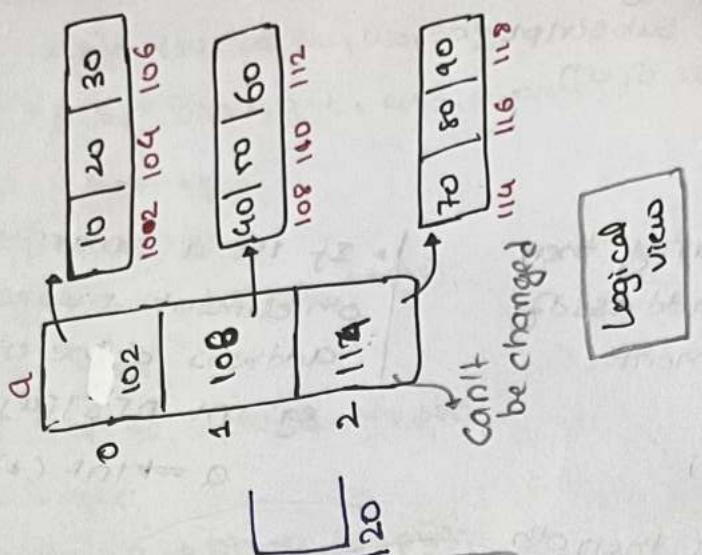
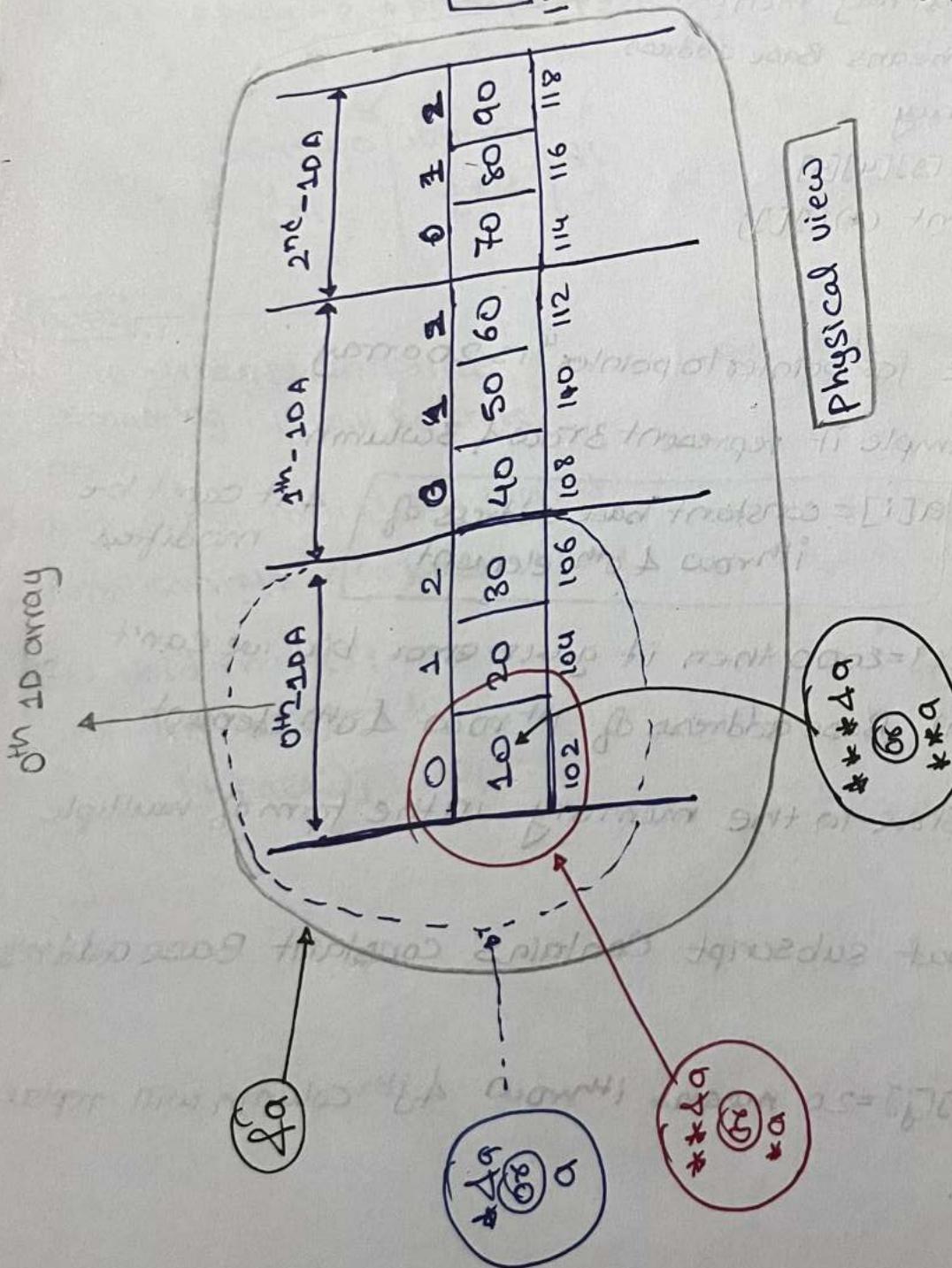
4. it can't be modified

4. if we perform  $a[1]=2000$  then it gives error b/c we can't modified constant Base address of 1st row 1<sup>st</sup> element

5. 2D array will be stored in the memory in the form of multiple 1D array

6. array name without subscript contains constant Base address of 0th 1D array

7. if we perform  $a[i][j]=20$  means  $i^{\text{th}} \text{ row } j^{\text{th}} \text{ column will replace the value with } 20.$



can't  
be changed

Logical views

$a[i][j]$        $\star (c * (c * c + 1) + 1)$        $\star (c * (c * 10^2 + 1) + 1)$        $\star (c * (c * 10^4 + 1) + 1)$

50 elements of  $\mathbb{C}^{100}$  are drawn from  $\mathbb{C}^{100 \times 100}$ .

Expression	Print Value	Type	meaning.
1. $\&a$	102	$\text{int}(\star)[]\[]$	Base address of the entire 2D array
2. $\&*a @ 0$	102	$\text{int}(\star)[]$	Base address of the 0th 1D array
3. $*a$	102	$\text{int} \star$	B.A of the 0th element of the 0th 1D array
4. $**a$	10	$\text{int}$	value of 0th element of the 0th 1D array
5. $***a$	error	?	referring unknown locat'
6. $\&a+1$	120	$\text{int}(\star)[]\[]$	B.A of the next continuous 2D array
7. $a+1$	108	$\text{int}(\star)[]$	B.A of the 1st 1D array
8. $*(a+1)$	108	$\text{int} \star$	B.A of 0th element of the 1st 1D array
9. $*a+1$	104	$\text{int} \star$	B.A of the 1st element of 0th 1D array
10. $*(a+2)+1$	116	$\text{int} \star$	B.A of the 1st element of 2nd 1D array
11. $a[0]$	102	$\text{int} \star$	B.A of the 0th element of 0th 1D array
12. $*(*(a+1)+2)$	60	$\text{int}$	value of 2nd element of 1st 1D array
13. $***\&a$	10	$\text{int}$	value of 0th element of 0th 1D array
14. $*(a+2)$	114	$\text{int} \star$	B.A of the 0th element of 2nd 1D array

Ex:  $\ast(\&a+1) - a$

$$\rightarrow 118 - 100$$

$$\not= \frac{18}{6} = 3$$

↳ SOE  
1D array

Ex:  $\ast\ast(\&a+1) - \ast a$

$$118 - 100$$

$$\not= \frac{18}{2} = 9$$

↳ SOE

Ex:  $\text{int } (\ast p)[8][3] = \&a;$

①      ②

• p is the pointer to the 2D array  
of size 3x3 where every element  
is an integer

Ex:  $\text{pf}(" \%d", p+1) // 120$

H.W

1)  $\text{int } a[4][5] = \{0, 1, \dots, 15\}; BA = 200, \text{SOE} = 2B$   
print all expression

2)  $\text{int } a[5][6] = \{0, 1, 2, \dots, 29\}, BA = 3000, \text{SOE} = 4B$

a)  $\text{pf}(\ast a + 4); : 3096$

b)  $\text{pf}(\ast a + 3) + 2; : 3080$

c) expand  $a[4][5]$

d)  $\text{pf}(\ast(\ast(a+2)+3)); : 215$

for address  
for these  
is 3060

expand  $\rightarrow a[4][2]$

$\ast(\ast(a+2)+2)$

$\ast(\ast(102+2)+2)$   
↳ 0<sup>th</sup> element of  
1D array

$\ast(\ast(112)+2)$

↳ 2<sup>nd</sup> 2D array

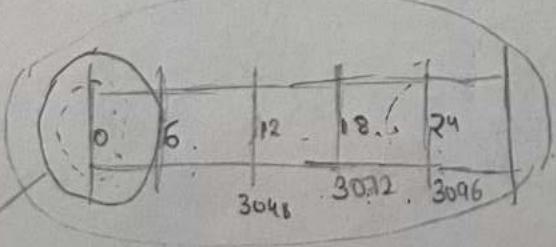
$\ast(112+2)$

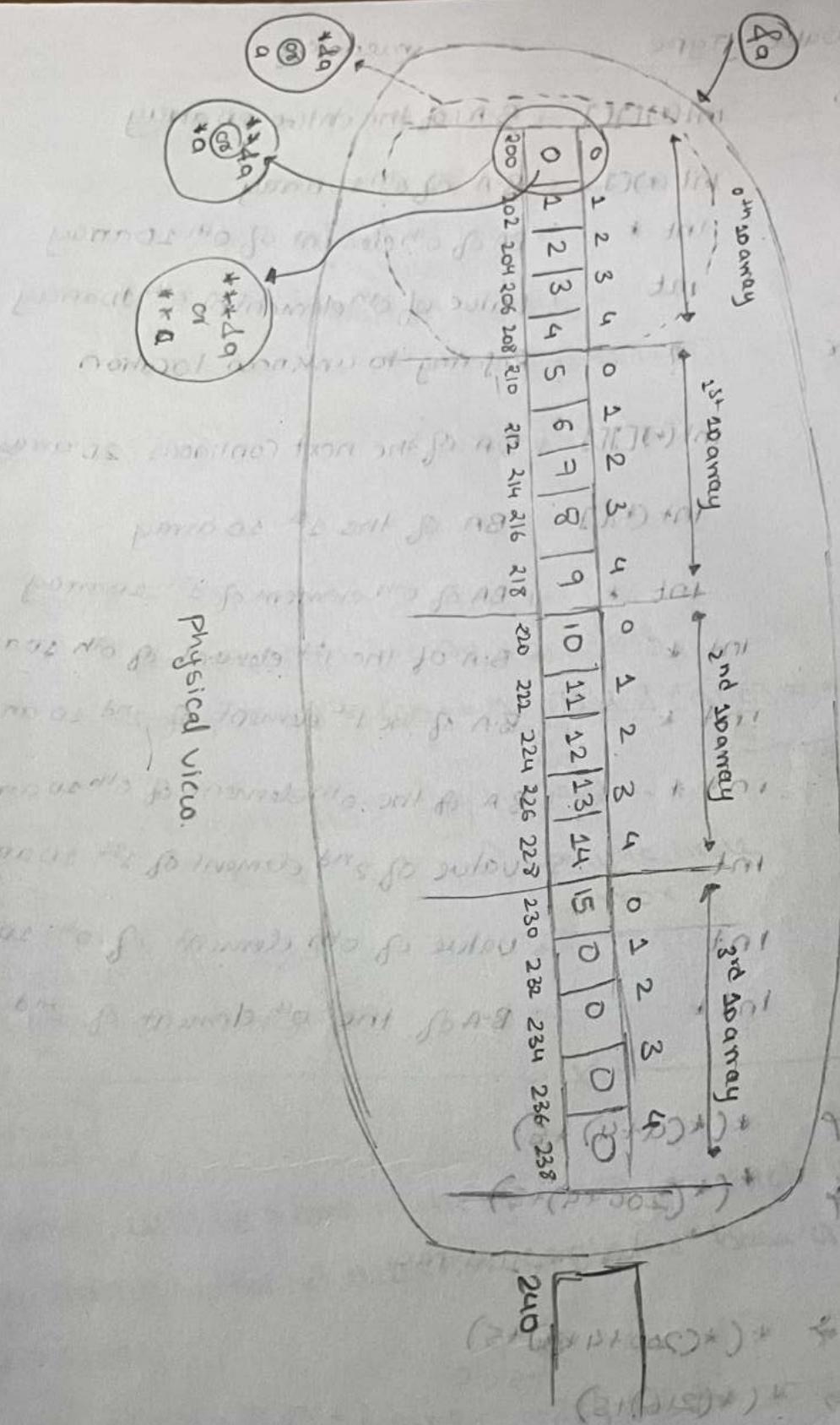
↳ 0<sup>th</sup> element of  
2<sup>nd</sup> 2D array

$\ast(118)$

↳ 2<sup>nd</sup> element of  
2<sup>nd</sup> 2D array

(40) //





$$(2+2+0.5)* \leftarrow$$

$$(0.5)* \leftarrow$$

$$(0.1)* \leftarrow$$

Expression	Print Value	Type	meaning
1. $\&a$	200	int(*)[][]	$\rightarrow$ B.A of the entire 2D array
2. $\&*a/a$	200	int(*)[]	$\rightarrow$ B.A of 0th 1D array
3. $\&a$	200	int *	$\rightarrow$ BA of 0th element of 0th 1D array
4. $*a$	10	int	$\rightarrow$ value of 0th element of 0th 1D array
5. $**a$	error	?	$\rightarrow$ Referring to unknown location
6. $\&a+1$	240	int(*)[][]	$\rightarrow$ B.A of the next continuous 2D array
7. $a+1$	210	int(*)[]	$\rightarrow$ BA of the 1st 1D array
8. $*(*a+1)$	210	int *	$\rightarrow$ BA of 0th element of 1st 1D array
9. $*a+1$	202	int *	$\rightarrow$ B.A of the 1st element of 0th 1D array
10. $*(*a+2)+1$	222	int *	$\rightarrow$ B.A of the 1st element of 2nd 1D array
11. $a[0]$	200	int *	$\rightarrow$ B.A of the 0th element of 0th 1D array
12. $*(*a+1)+2$	7	int	$\rightarrow$ value of 2nd element of 1st 1D array
13. $**\&*a$	0	int	$\rightarrow$ value of 0th element of 0th 1D array
14. $\&(*a+2)$	220	int *	$\rightarrow$ B.A of the 0th element of 2nd 1D array

② Expand  $a[4][5]$

$$\Rightarrow *(*(*a+4)+5)$$

$$\Rightarrow *(*(*3000+4)+5)$$

$\hookdownarrow$  0th - 1D array

$$\Rightarrow *(*(*3000+4*24)+5)$$

$\hookdownarrow$  S.O.E

$$\Rightarrow *(*3096+5)$$

$\hookdownarrow$  3rd - 1D array

$$\Rightarrow *(3096+5)$$

$\hookdownarrow$  0th element of 3rd 1D array

$$\Rightarrow * (3096 + 5 * 4)$$

$$\Rightarrow$$

$$\star(3116)$$

$\hookdownarrow$  5th element of 3rd 1D array

Ques  
gate 2020

```

int main()
    int a[4][5] = {{1, 2, 3, 4, 5},
                    {6, 7, 8, 9, 10},
                    {11, 12, 13, 14, 15},
                    {16, 17, 18, 19, 20}};
    printf("%d", *(*(a + *x + 2) + 3));
}

```

O/P: 19 //

y

$$*(a+3)+3 \Rightarrow a[3][3] = 19$$

gate

Ques

void main()

```

int a[5][3] = {10, 20, 30, -100};
printf("%d", ((a == a[0]) & (a[0] == *a)));

```

y

what is O/P printed?

a) 1

b) 0

c) Compile time error

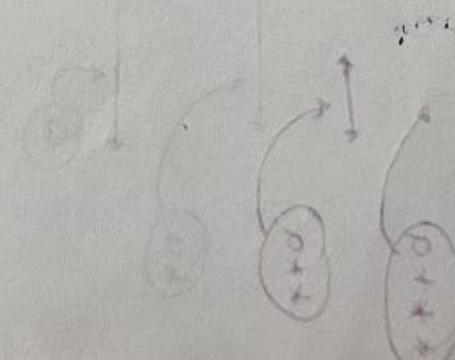
d) Run time error.

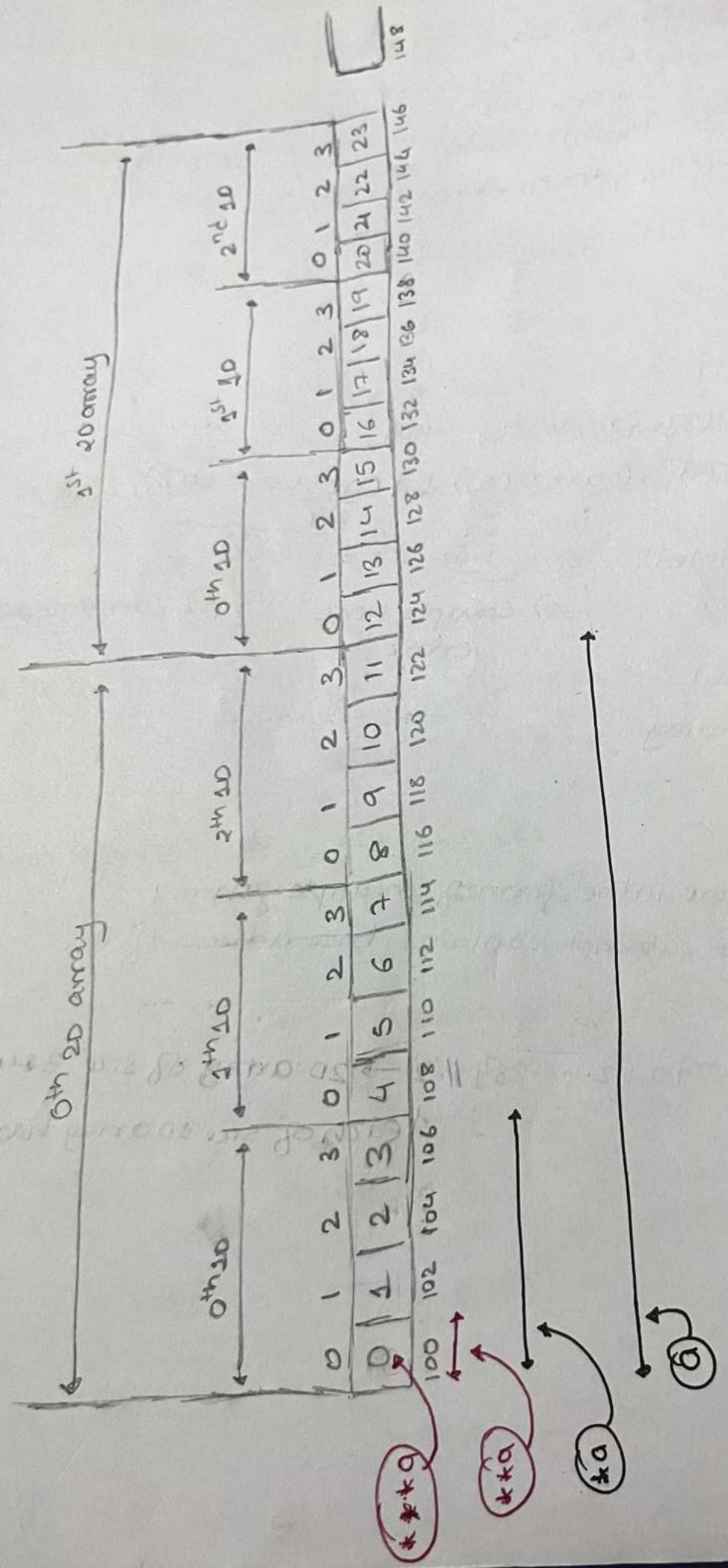
• Comparing with print  
Value will comparing

3D array

- 3D array will be stored in the form of multiple 2D array
- array name without subscript contains base address of the 0th 2D array

Ex: int a[2][3][4] = {0, 1, 2, ..., 23}; // '2' → 2D array of size 3\*4  
 Each of size 1D array has 4 elements





Expression	Print Value	Type	meaning
1. $\&a$	400	$\text{int}^*$ [ ] [ ] [ ]	B.A of the entire 3D array
2. $\&&a/a$	400	$\text{int}^*$ [ ] [ ]	B.A of 0th 2D array
3. $\&a$	300	$\text{int}^*$ [ ]	B.A of 0th 1D array of 0th 2D array
4. $\&*a$	100	$\text{int}^*$	B.A of 0th element of 0th 1D array of 0th 2D array
5. $\&**a$	0	int	Value of 0th element of 0th 1D array of 0th 2D array
6. $\&***a$	error	?	referring to unknown location
7. $\&a+1$	148	$\text{int}^*$ [ ] [ ] [ ]	B.A of the continuous next 3D array
8. $a+1$	124	$\text{int}^*$ [ ] [ ]	B.A of the 1st 2D array
9. $*(\&a+1)$	124	$\text{int}^*$ [ ]	B.A of 0th 1D array of 1st 2D array
10. $\&**(\&a+1)$	124	$\text{int}^*$	B.A of the 0th element of 0th 1D array of 1st 2D array
11. $\&a+1$	108	$\text{int}^*$ [ ]	B.A of the 1st 1D array of 0th 2D array
12. $\&*a+1$	102	$\text{int}^*$	B.A of the 1st element of 0th 1D array of 0th 2D array
13. $*(\&a+1)+1$	132	$\text{int}^*$ [ ]	B.A of the 1st 1D array of 1st 2D array
14. $*(*(\&(a+1))+2)$	128	$\text{int}^*$	B.A of the 2nd element of 0th 1D array of 1st 2D array
15. $*(*(\&(a+1)+1)+3$	138	$\text{int}^*$	B.A of the 3rd element of the 1st 1D array of 1st 2D array.
16. $*(\&a+2)+2$	120.	$\text{int}^*$	
17. $\&***\&a$	0	int	
18. $*(*(\&a+1)+2)$			
19. $*(*(\&(a+1)+2)+1)$			

Ex:  $a[1][2][1]$

$$\Rightarrow *(*(*(<(a+1)+1)+1)$$

$$\Rightarrow *(*(*(*((100+2)+1)+1)$$

↳ 0th - 2D array

$$\Rightarrow *(*(*(*((224)+1)+1)$$

↳ 1st - 2D array

$$\Rightarrow *(*(*((124)+1)+1)$$

↳ 0th 1D array,  
1st 2D array

Ex:  $a[1][2][3]$

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

$$\Rightarrow *(*(*(*((100+1)+2)+3)$$

↳ 0th - 2D array

$$\Rightarrow *(*(*(*((124)+2)+3)$$

↳ 1st - 2D array

$$\Rightarrow *(*(*((124+2)+3)$$

↳ 0th 1D array,  
1st 2D array

$$\Rightarrow *(*((140)+3)$$

↳ 2nd 1D array, 1st 2D array

$$\Rightarrow *(*((132)+1)$$

↳ 1st 1D array, 1st 2D array

$$\Rightarrow *((132)+1)$$

↳ 0th element, 1st 2D array

1st 2D array

$$\not\Rightarrow *((134))$$

↳ 1st element, 1st 1D array

1st 2D array

$$\Rightarrow *((140+3))$$

↳ 0th element of 2nd 1D array

of 1st 2D array

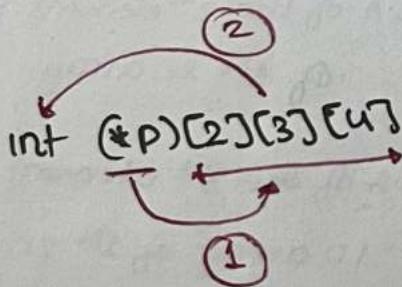
$$\Rightarrow *(146)$$

↳ 3rd element, 2nd 1D array  
of 1st 2D array.

Q 17

Note

①



| P is a pointer to a 3D array of  
Size ~~2x3x4~~ 2x3x4 where every element  
is integer.

Pf("...").d<sup>u</sup>, p+D) // 148

$$\textcircled{2} \quad *(\&a+1) - a \quad \textcircled{2}$$

$$\Rightarrow 148 - 100$$

$$\Rightarrow \frac{48}{24} = 2$$

Ex:

void main() {

int arr[3][2] = { {1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10}, {11, 12} };

printf ("%d %d %d %d", arr[1][0], arr[1][1], arr[0][0] - arr[0][1], arr[0][0] - arr[0][1]);

y

what is o/p printed?

- a) 3,3,1
- b) 3,6,1
- c) 6,6,1
- d) 1,1,1

$$\begin{aligned} &arr[1][0] - arr[0][0] \\ &*(arr + 1) - *arr \\ &\Rightarrow 112 - 100 \\ &\Rightarrow \frac{12}{2} = \textcircled{3} // \end{aligned}$$

$$\begin{aligned} &arr[1][0] - arr[0][0] \\ &*(arr + 1) - *arr \\ &**arr - ***arr \\ &\Rightarrow 112 - 100 \\ &\Rightarrow \frac{12}{2} = \textcircled{6} // \end{aligned}$$

H.W

Ex: ① int arr[3][4][5] = { { { 1, 2, 3, 4, 5 } }, { { 6, 7, 8, 9, 10 } }, { { 11, 12, 13, 14, 15 } } };

$$B.A = 3000, S0E = 28$$

Print all expressions. 16

② int arr[4][5][6] = { { { 1, 2, 3, 4, 5, 6 } }, { { 7, 8, 9, 10, 11, 12 } }, { { 13, 14, 15, 16, 17, 18 } }, { { 19, 20, 21, 22, 23, 24 } } };

$$B.A = 40000, S0E = 48;$$

a) pf(\*arr + 2) = ?

b) pf(\*arr + 2 + 3) = ?

c) pf(\*arr + 2 + 3 + 4) = ?

d) expand arr[3][2][3].

## Strings

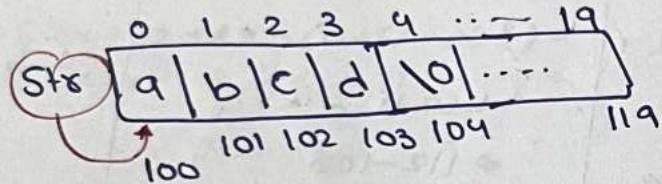
1. Character array is called string.  
String always end with a "NULL" character.
2. null character representation is "\0" & ASCII value "0".
3. String name always give Base address.
4. every character of a string store in the memory sequential  
in continuous location in the form of their ASCII value

Ex: void main()

Char str[20] = { 'a', 'b', 'c', 'd', '\0' };

printf("%c.%c.%c.%c.%c", str[0], str[1], str[2], str[3]);

Output: abcd

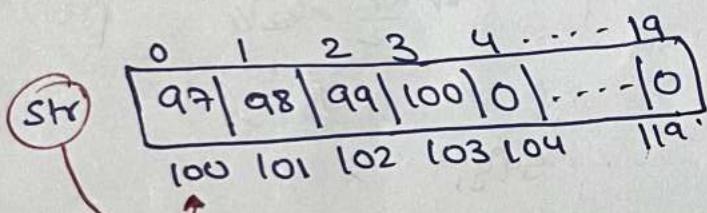


not compulsory

\*(str+1)  
\*(100+1)  
\*(101)  
98

ASCII

A → 65  
a → 97



printf(str) = 100

printf(str+1) = 101

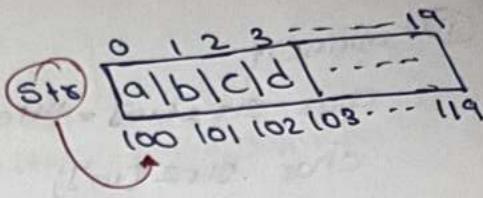
① %c → will print character → we should provide ASCII value

② %.d → will print ASCII value

str[0]  
\*(str+0)  
\*str  
97

③ If we read str[0]. In any question we need consider ASCII value of the respective character.

② void main() {  
 char str[5] = "abcd";  
 printf("%s", str); }  
 100  
 y  
 O/P: abcd



For %s we need to provide B.A from the BA provided it will print all the characters till the null character until the NULL character.

- Initialization of string:

1. char s[6] = "INDIA";
2. char s[] = "INDIA";
3. char \*p = "INDIA"; // string constant

char s[5] = "INDIA";  
 pf("%s", s);  
 O/P: INDIA\0

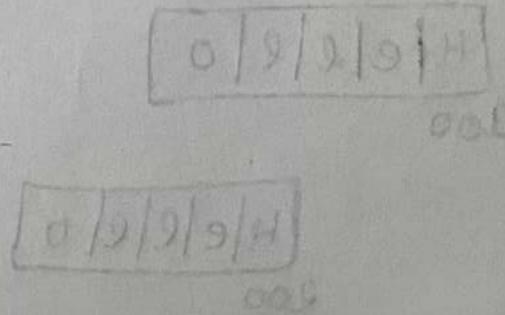
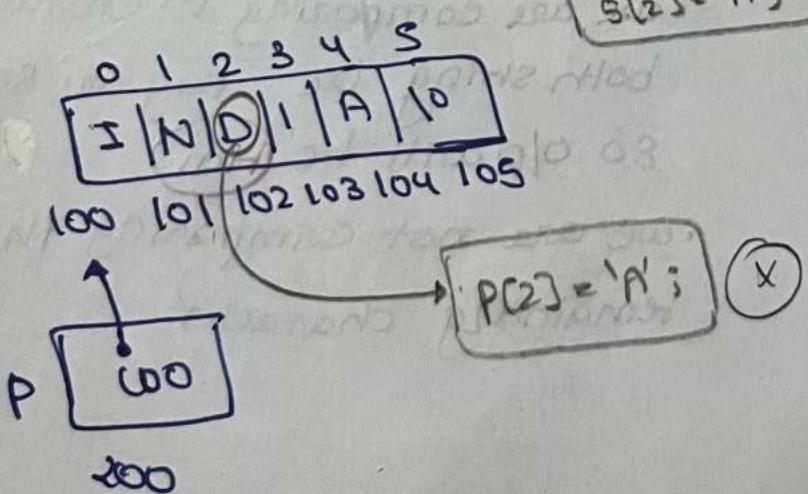
- Explanation

1. Explicitly we need to provide size for null character otherwise it will print some additionally garbage data/value.
2. Implicitly it calculate the size & allocate the memory accordingly

3. The meaning of 3rd statement means we are requesting the compiler to allocate the memory for the string "INDIA". & B.A of the 1st character of the string will be stored in the pointer

Variable P.

- It is considered as string constant & content of string constant can't be modified



### ① main()

```
char str1[10] = "Hello";
```

```
char str2[10];
```

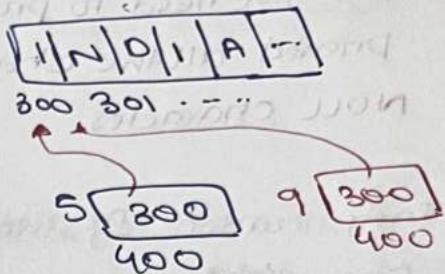
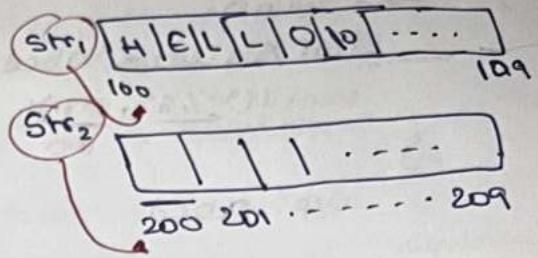
```
char *s = "INDIA";
```

```
char t[9];
```

```
str2 = str1; // X
```

$\checkmark \& q = s;$  we can't  
modify constant  
base address

y



### ② Main()

```
char str1[10] = "Hello";
```

```
char str2[10] = "Hello";
```

if ( $str_1 == str_2$ ) || we are comparing B.A of 2 different  
string b/c they are different O/p will be

```
pf("Hai");
```

else

```
pf("Bye");
```

y

"Bye"

↓  
O/P

### ③ main()

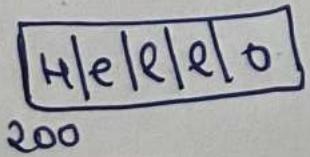
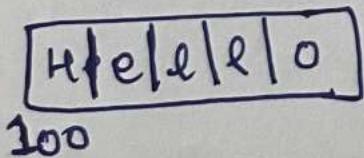
```
if (*"Hello" == *"Hello")
```

```
pf("Hai");
```

```
else pf("Bye");
```

y

OP : Hai



- If the string is given in double quotes it means BA is given

- we are comparing 1st character of both string b/c they are same so O/P will be **Hai**
- we are not comparing the remaining character

- general functn available in <String.h>

1. `strcpy(T, S)`

target      ↗ source string  
String      B.A

### 3. `strlwr(S)`

- Convert all the character to lower case

Ex: void main() {

    char \*str[20] = "abcdef";

    Pf("%s", str); // abcdef

    str[3] = 'A';

    Pf("%s", str); // Abcdef

    str[3] = 0;

    Pf("%s", str); // abc

    Pf("%s", \*str); // a // error      str[3] = '0';

    Pf("%c", str + 3); // 00+3=103 // Pf("%s", str); // abcDef      error

    Pf("%s", str + 3); // 00+3=103 // Def

    Pf("%c", \*(str + 3)); // 0

    str[3]

3 It was expecting ASCII value we got some address

### 2. `strupr(S)`

- Convert all the character to upper case.

### 4. `strrev()`

- reverse the given string

### 5. `strlen(s)`

- the below functn return the actual length of the string

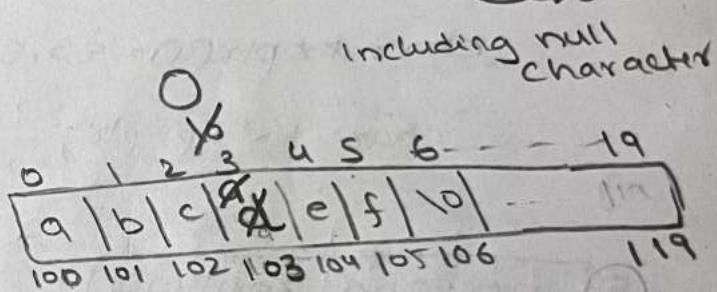
Ex: char \*s = "INDIA";

`strlen(s);` // 5

length

if you calculate size the

size



Pf("%s", str); // abcDef

str[3] = '0';

error

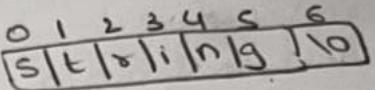
Def

gate:

```

Q) char p[20]
    char s = "String";
    int length = strlen(s); //6
    for( int i=0; i<length; i++)
        p[i] = s[length - 1];
    printf("%s", p);

```



Array of pointer  
to strings

Ex: void main()

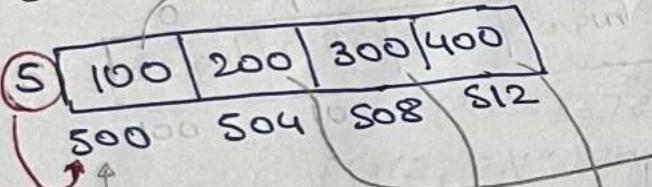
②  
char \*s[4] = { "Bhopal", "Delhi", "Pune", "Jaipur" };

s is an array of 4 elements where every element is a character pointer

char \*\*ptr() = { s, s+1, s+2, s+3 };

char \*\*kp = ptr;

y.

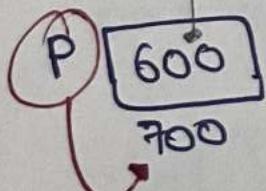
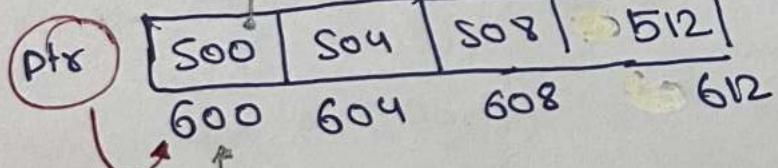


Bhopal \0  
100

Delhi \0  
200

Pune \0  
300

Jaipur \0  
400



$$Pf(s) = 500$$

$$Pf(ptr) = 600$$

$$Pf(p) = 600$$

$$Pf(s+1) = 504$$

$$Pf(ptr+1) = 604$$

$$Pf(p+1) = 604$$

Ex: void main() {

    char \*s[s] = {"London", "Birmingham", "Houston", "milpitas",  
        "macau"};

Char \*k = p[s] = {s, s+1, s+2, s+3, s+4};

Char \*\*p = p+s;

.Pf ("%.S", \*\*++p); // Birmingham

Pf ("%.\*.S", \*\*--++p+3); mingham

Pf ("%.\*.S", \*(p+2)+3); don

Pf ("%.\*.S", p[-1][-1]+1); ondon

Pf ("%.\*.S"; \*\*p++) // Birmingham

Pf ("%.\*.C", \*\*++p+2); // O

Pf ("%.\*.C", s[1][2]);

(\*(s+1)+2)

y

Unary  
operators  
have high  
precedence  
than Binary

add +2 to get "O"

matcha[u]s

Macau

500

London

Birmingham

Houston

Milpitas

100

200

300

400

500

pot [100 | 200 | 300 | 400 | 500]

500 504 508 512 516

600

526 504 508 512 516

P

Ex: void main() {  
    char \*s[5] = {"Qatar", "Dubai", "Bangkok", "California", "Texas"};

char \*\*ptr [5] = &s, s+3, s+1, s+4, s+2};

Char \*\*\*p = ptr;

pf("%.\*s", \*\*ptr); // Qatar

pf("%.\*s", &--\*++ptr+3); // ifornia

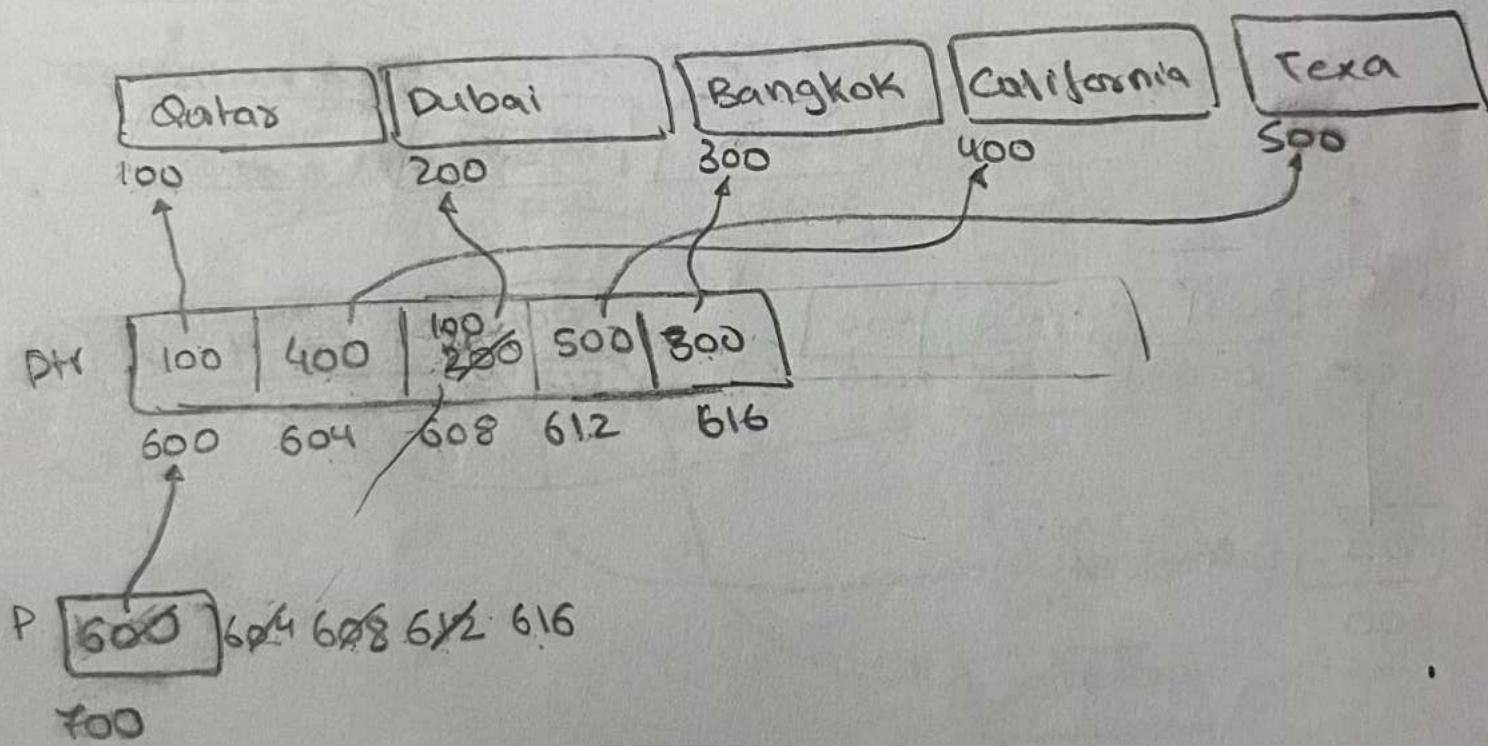
pf("%.\*s", \*\*++ptr); // Texa

pf("%.\*s", PC[-1]-1+1); // error

pf("%.\*s", \*(PC-2)+3); // ifornia

pf("%.\*s", \*\*\*++ptr+2); // D<sup>Kok</sup>

pf("%.\*s", SC1[2]); // b



Ex: void main() {

char \*s[s] = {"Bahubali", "Katappa", "Ballaldev", "Rajamatha",  
"Deusena"};

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

char \*k = p+ptr;

printf("%s", \*k++); // Katappa

printf("%s", \*(p[-2])+3); // error or Garbage value.

printf("%s", \*k++); // bahubali

printf("%s", \*--k++ + 3); // amatha

printf("%s", p[-1][-2] + 2); // error

printf("%s", s[1][2]); // t

printf("%c", \*k++ + 2); // T

Bahubali

100

Katappa

200

Ballaldev

300

Rajamatha

400

Deusena

500

ptr [ 200 | 300 | 100 | 500 | 400 ]

600 604 608 610 612

P [ 605 ] 604 608 610 612  
700

## multi dimensional arrays on Strings

Ques] void main()

```
char mess[6][20] = { "koi bhi Lakshya",  
                     "Bada nahi",  
                     "jeeta wahi",  
                     → "jo dara, nahi",  
                     "hara wahi",  
                     "Jo Lada nahi" };
```

```
printf("%s", mess+3); // Jo dara nahi
```

```
printf("%s", *(*(mess+2)+3)); // t
```

```
printf("%s", *(mess+4)); // hara wahi
```

↳

H.W

Q. write a program to implement string copy function

void main()

```
char str1[20] = "made easy";
```

```
char str2[20];
```

```
strcpy(str2, str1);
```

```
printf("%s", str2);
```

↳

void strcpy (char \*t, char \*s) {

```
while (*t++ = *s++) ; } } after performing the last operation  
copying the "t" & then return  
0) while returning "0" it get  
false
```

Assignment value assign  
value & return that value

## Type Casting

- In order to perform any operation b/w two variable type of both the variable should be same otherwise we need to perform type casting

Ex:

- int i; → 2B
- char c; → 1B
- float f; → 4B

### Implicit Type Casting: (coercion)

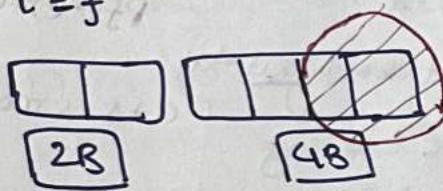
- No problem in size
- problem in type
- no error

f = (float)i;

temporarily treated as float  
variable

### Explicit Type Casting:

i = f



l = (int)f;

treated  
as int.

- problem in size
- problem in type
- error ✓
- typecasting required
- user will do type casting
- no error after type casting
- data loss will happen

## Typecasting in pointers

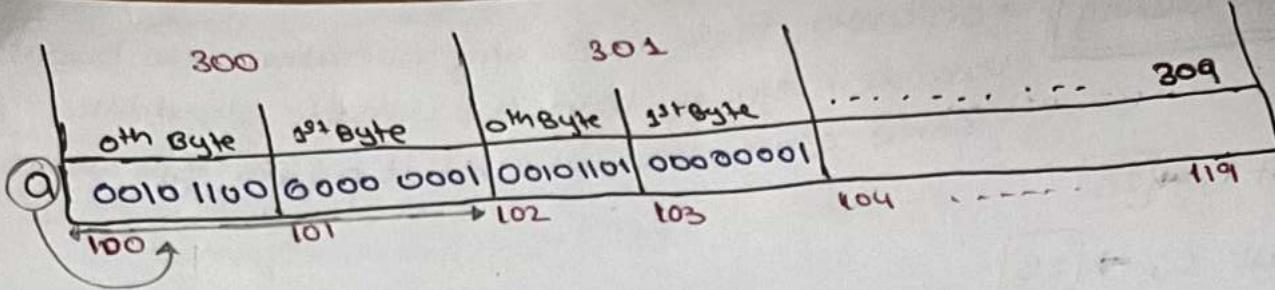
int a[10] = {300, 301, 302, ..., 309};

Ex:  $(300)_{10} = \underline{\text{0000}}_2 \underline{\text{0001}}_2 \underline{\text{0010}}_2 \underline{\text{1100}}_2$

we use  
in all  
systems

Little Endian  $\Rightarrow L.V = L.A \neq H.V = H.A$

Big Endian  $\Rightarrow L.V = H.A \neq H.V = L.A$



b-100

- no problem in size
  - no problem in type
  - no type casting required
  - no error

$$Pf(\frac{kb}{t}) = 2B \text{ data}$$

$$\downarrow \quad b+1=102$$

It has capacity  
to access only  
1 Byte

C 100

char \*czaj;

- no problem in size
  - problem in type casting required

$$\text{char} \leftarrow C = \underbrace{\text{char} \cdot}_{J} a_j$$

## Type Casting to Character pointers

- no errors after type casting

$$P_f(kc) = 18 \text{ data}$$

C+1=101

f100

4B 4B

- no problem  
size
  - problem in  
type
  - error will  
come

type casting required

float \* = (float \*) a;

type casting  
floating  
pointers

- no error after type casting

$$Pf(*f) \approx 48$$

data

$$f+1=104$$

e100

void \*e = a;

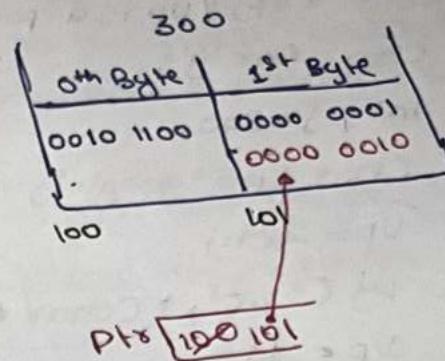
- no problem in size
  - no problem in type
  - no errors
  - no type casting required

$$Pf(\gamma e) = \text{error}$$

\* W.B  
page no.  
128  
Q23

int i=300;

```
char *ptr=(char*)&i;
++ptr
printf("%d",*ptr);
```



$$i = (0000 \ 0010 \ 0010 \ 1100)_2$$

$$(556)_{10}$$

### Void pointer

- Just like we had integers & character pointers similarly we can also have void pointers.
- In the void pointers we can store address of any type of variable but will accessing the data we need to perform respective type casting A/C
- Type casting is required b/c void pointer doesn't know how many bytes of data it has to dereference
- Void pointers are used in device driver programming.

### Note

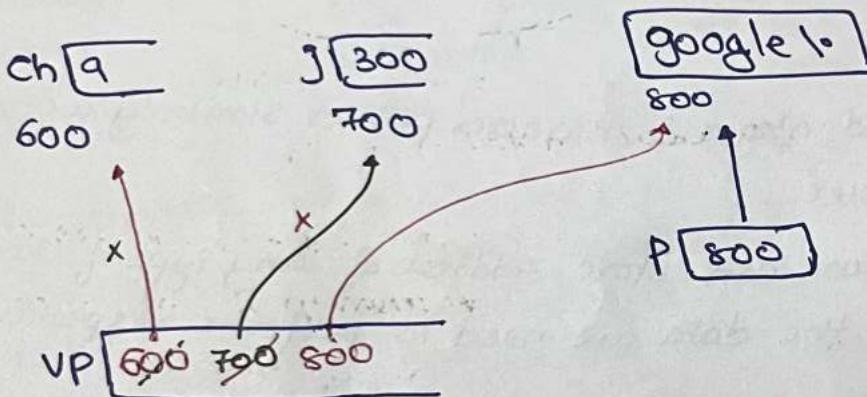
- The main purpose of type casting is that it will clarify that how many bytes of data it has to dereference

```

void main() {
    void *vp; // vp is a pointer variable of type void where we can store
    char ch='g';
    int j=300;
    char *p="google";
    vp=&ch;
    printf("%c", *(char *)vp); // g
    vp=&j;
    printf("%d", *(int *)vp); // 300
    vp=p;
    printf("%s", *(char *)vp+3); // 800+3 → 803 → gle
}

```

y



### Pointer to function

```

void test();
void main() {
    1. void (*p)(); // ②
    2. p=test; // ①
    3. (p)();
}

```

y  
void test()

printf("Hello");

y

pointer variable P

1. this is known as declaring function pointer, p is a pointer to a functn which doesn't take any arguments and return void

2. assigning the B.A of the test functn to the pointer variable P

3. calling the test functn by using pointer variable P

• Terminology for various expression:

1. `int (*p)(int, int);` // p is a pointer to the functn which takes two integer argument & return integer
2. `int *p(int *);` // p is a function which take integer pointer as a argument & return integer pointer
3. `char **(*a)(*, *);` // a is a pointer to a function which takes two pointers as a argument & return characters double pointer.

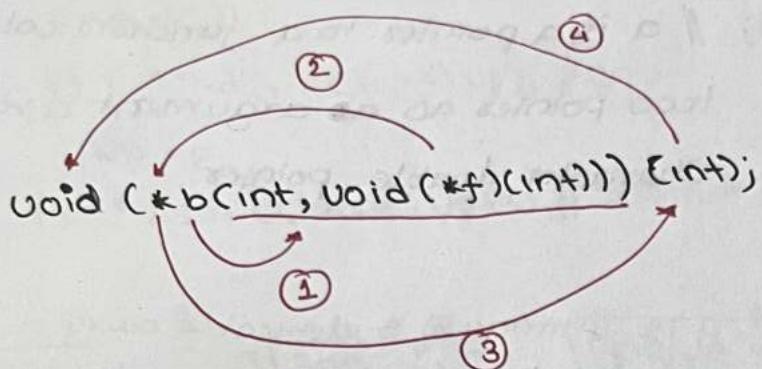
- Ex:
4. `int (*a[5])(());` // a is an array of 5 element & every element is pointer to a function which don't take any argument & return integer.

5. `char (*(*x[3]))[5];` // x is function which doesn't take any argument to an pointer of an array of 30 element & every element is pointer to the function which doesn't take any argument & return char.

6. C

- `char (*(*x[3]))(())[5];` x is an array of 3 element, where each element is a pointer to a function which doesn't take any argument & return a pointer to an array of 5 element which return char.

Ex: void a(\*b(int, void (\*f)(int)))(int);



► B is a functn which is taking two arguments a, b & return pointer, & this pointer is pointing to function which is taking integer as argument & return void

a: a is an integer

b: f is a pointer to functn which takes integer as argument & return void.

## Structure

- Structure is used to create user-defined datatype.
  - In the structure to operators are used.
    - dot(.) : Structure variable to member variable
    - Arrow("→") : pointer variable to member variable

**Note**

- Structure contains member variable
  - size of the structure is sum of all member variable
  - we can't define a structure without any member variable
  - we can define a structure inside the functn.
  - we can't define the functn inside the structure
  - we can create existing struct as member to other struct
  - we can't create same structure variable as member to the structure b/c logical size is undefined
  - we can create same structure pointer as member to the structure & that is called as self referential structure (used in **linked list**).

Ex: Struct Keyword name of  
emp } Structure

$$5t \text{ Idj} \rightarrow 28$$

Int sal; → 20

W W S A Y → Z B

Char name[36]; → 36b

۴۰

typedef Struct emp Emp;

```
void main()
```

emp e<sub>1</sub>,e<sub>2</sub>,e<sub>3</sub>; // X      struct keyword is . . . . .

struct emp e<sub>1</sub>,e<sub>2</sub>,e<sub>3</sub>; //✓

there are called structure variable

Emp e<sub>4</sub>, e<sub>5</sub>; // ✓

~~Emp~~ will be replaced by  
struct. emp.

e<sub>1</sub>. id = 1427 // ✓  
e<sub>1</sub>. sal = 2500 // ✓

e<sub>1</sub>. name = "Rahul" // ✗ not allowed since we can't  
modified constant base  
address.

e<sub>1</sub>. name = "Rahul" // ✗ —————|| Same reason

strcpy(e<sub>1</sub>.name, "Rahul"); // ✓

Emp \*ptr; // ✓ ptr is a pointer variable of type  
struct Emp where we can store  
address of any struct variable

ptr = &e<sub>1</sub>; // ✓

pf(ptr); // 100 ✓

pf(e<sub>1</sub>); // ✗

pf(e<sub>1</sub>. sal); // ✓ 2500

pf(\*ptr); // ✗

pf((\*ptr). sal); // ✓ 2500

allow  
same

pf((ptr[0]). sal); // ✓ 2500

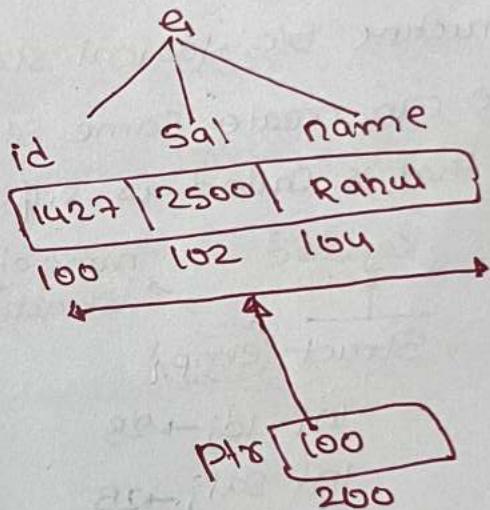
pf(ptr → sal); // ✓ 2500

e<sub>2</sub> = e<sub>1</sub>; // the variable operation is allowed but  
not suggestible to perform practically.

e<sub>1</sub> \* e<sub>2</sub>; // ✗

e<sub>1</sub> + e<sub>2</sub>; // ✗

y



① Struct node {

    int data; → 2B

    Struct node \*n<sub>1</sub>; // (\*)

y refer to point no.

⑦

② Struct node {

    int data;

    Struct data \*n<sub>1</sub>; // (\*)

y refer to point no. 8.

[As we are creating pointers we are clearly define that pointers take 4B i.e used in linked list]

H.W

Difference b/w typedef & #define

③ void main() {

    Struct emp {

        int id;

        float b;

y;

    mec();

    Void mec();

    Struct emp e<sub>1</sub>, e<sub>2</sub>, e<sub>3</sub>; // (\*)

y;

► Structure is defined inside the main & we are creating structure variable outside the main.

► If we want to create structure variable outside the main() then we need to define the structure outside the main.

Eg. Struct S<sub>1</sub> {

    int a;

    char b;

    float c;

y;

Struct S<sub>2</sub> {

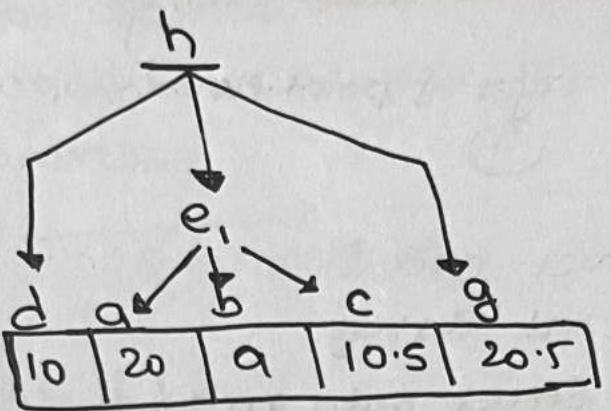
    int d;

    Struct S<sub>1</sub>, e;

    float g;

y

Struct  $S_2$  h = {10, 20, 'a', 10.5, 20.5}



$$Pf(h.d) = 10$$

$$Pf(h.e_1.a) = 20$$

Ex: Struct test?

```
int i;  
char *c;  
}
```

Struct test st[s]= {"5, "become", 4, "better", 6, "Jungle", 8, "ancestor",  
7, "brother"}, i;

ptr[506] = 64508

void main()?

```
Struct test *p = st;
```

p++;

t+p → c;

```
Pf("%c", p++ -> c); // better
```

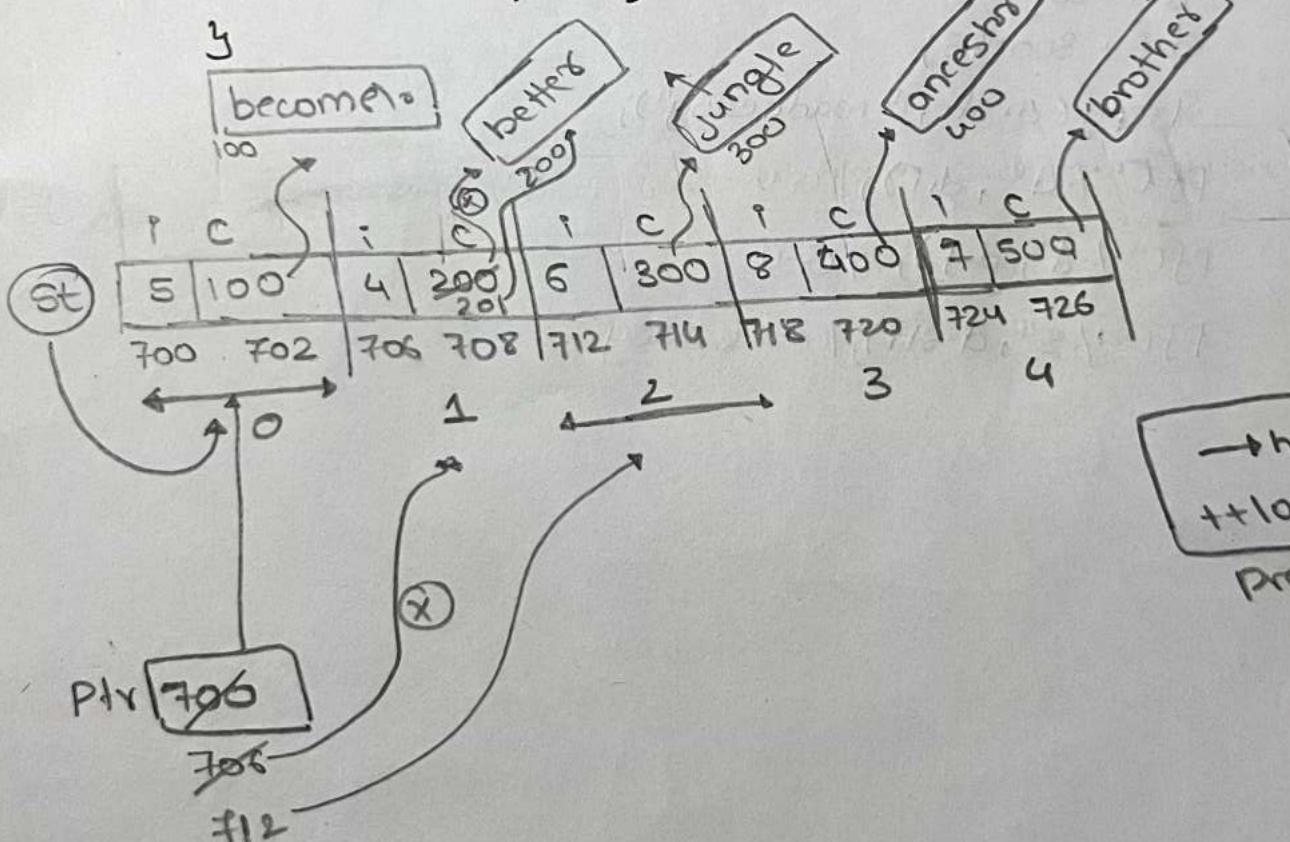
```
Pf("%c", *(t + p -> c)); // o
```

```
Pf("%c", p[3].i); // l
```

```
Pf("%c", p->c);
```

Op:

better, o, l, ungle.

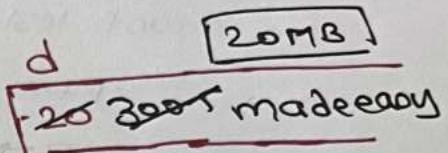


## Unions

- the size of the union will be the max size of the member variable

- In the structures we can access all the members any time but in case of union we can only access one member variable & all the remaining member variable will contain garbage value.
- In the union same memory space will be shared by all the member variable

Eg: union data {  
    int i; → 2B  
    char str[20]; → 20B  
    float f; → 4B  
};  
main() {  
    union data d; // 20B  
    d.i = 20;  
    d.f = 300.5;  
    strcpy(d.str, "madeeasy");  
    printf("%d", d.i); // 20  
    printf("%f", d.f); // 300.5  
    printf("%s", d.str); // madeeasy



y

## Array

- array is used to store homogenous / similarly type of element
- all the element will be stored in continuous location
- array support random access

### • 1D Array

$\text{int } a[0 \dots 9] = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

Lower bound      Upper bound

0	1	2	3	4	5	6	7	8	9	10
100	102	104	106	108	110	112	114	116	118	

$$\text{loc}(a[8]) = 100 + (8 - 0) * 2$$

↳ S.O.E  
↳ lower bound

$$\Rightarrow 100 + 16$$

$$\Rightarrow 116$$

$$\text{loc}(a[4]) = 100 + (4 - 0) * 2$$

↳ S.O.E  
↳ L.B

$$108$$

Q1)  $a[-636 \dots +817], B \cdot A = 4700, S.O.E = 9$ ,

$$\text{loc}(a[799]) = 4700 + (799 - (-636)) * 9$$

$$\Rightarrow 4700 + 1435 * 9$$

$$\Rightarrow 17615$$

2)  $a[-843 \dots +916]$

$$B \cdot A = 6800$$

$$S.O.E = 8$$

$$6800 + (847 - (-843)) * 8$$

$$\Rightarrow 6800 + 1690 * 8$$

$$\Rightarrow 6800 + 13520$$

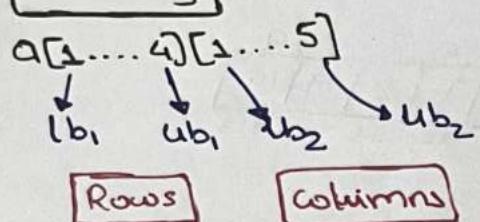
$$\Rightarrow 20320$$

**Note** • array index always start with 0. otherwise we need to perform <sup>one</sup> additional subtraction operation

$a[lb \dots ub]$ ,  $B \cdot A$ ,  $soe = 'S'$

$$\text{loc}(a[i]) = B \cdot A + (i - lb) * S$$

2D array



$$\text{no. of rows } (n-r) = ub_1 - lb_1 + 1$$

$$4 - 1 + 1 = 4$$

$$\text{no. of columns } (n-c) = ub_2 - lb_2 + 1$$

$$5 - 1 + 1 = 5$$

• 2D array will be stored in the form of 1D array & it is stored in

2 different way

→ row-major order

→ column-major order

• C language By default follow row-major order

Row major order (R.M.O.) :

$a_{11}$	$a_{12}$	$a_{13}$	$a_{14}$	$a_{15}$
$a_{21}$	$a_{22}$	$a_{23}$	$a_{24}$	$a_{25}$
$a_{31}$	$a_{32}$	$a_{33}$	$a_{34}$	$a_{35}$
$a_{41}$	$a_{42}$	$a_{43}$	$a_{44}$	$a_{45}$

4x5

$a_{11}$	$a_{12}$	$a_{13}$	$a_{14}$	$a_{15}$	$a_{21}$	$a_{22}$	$a_{23}$	$a_{24}$	$a_{25}$	$a_{31}$	$a_{32}$	$a_{33}$	$a_{34}$	$a_{35}$	$a_{36}$	$a_{41}$	$a_{42}$	$a_{43}$	$a_{44}$	$a_{45}$
100	102	104	106	108	110	112	114	116	118	120	122	124	<u>126</u>	128	130	132	<u>134</u>	136	138	

Ex:  $\text{loc}(a[3][4]) = 100 + [(3-1)*5 + (4-1)] * 2$

$\downarrow b_1 \quad \downarrow n.c \quad \downarrow b_2$

S.O.C

$$\Rightarrow 100 + [10+3] * 2$$

$$\Rightarrow 126$$

Ex:  $a[-157 \dots -963][-468 \dots 819]$

B.A = 7200

S.O.C 7

①  $\text{loc}(a[839][727])$

$$\Rightarrow 7200 + [(839 - (-157)) * 1288 + (727 - (-468))] * 7$$

$$7200 + [996 * 1288] + 1195 * 7$$

$$7200 + 12,82,848 + 1195$$

Ex:  $a_{43} = \text{loc}(a[4][3]) =$

$$\Rightarrow 100 + [(4-1) * 5 + (3-1)] * 2$$

$$\Rightarrow 100 + 34$$

$$\Rightarrow 134$$

Q)  $a[699 \dots 718][543 \dots 597]$ .  $BA = 6500, SOE = 6$

$\text{loca}(a[i][j])$

$$\Rightarrow 6500 + [(691 - (-699)) * 1141] * 6 + (597 - (-543)) * 6$$

$$\Rightarrow 6500 + [1390 * 1141 + 1090] * 6$$

$$\Rightarrow 6500 + 1587080 * 6$$

$$\Rightarrow 9528980$$

Note:

$a[lb_1 \dots ub_1][lb_2 \dots ub_2]$ ,  $BA, SOE = S, ROMO$

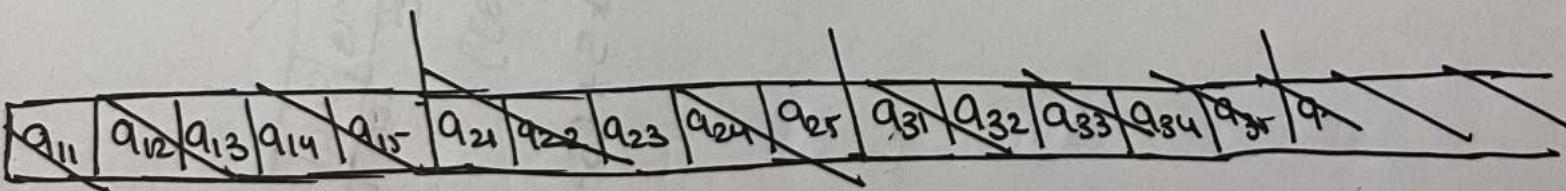
$$\text{loca}(a[i][j]) = BA + [(i - lb_1) * (ub_2 - lb_2 + 1) + (j - lb_2)] * S$$

Column major  
Order (CMO)

$a_{11}$	$a_{12}$	$a_{13}$	$a_{14}$	$a_{15}$
$a_{21}$	$a_{22}$	$a_{23}$	$a_{24}$	$a_{25}$
$a_{31}$	$a_{32}$	$a_{33}$	$a_{34}$	$a_{35}$
$a_{41}$	$a_{42}$	$a_{43}$	$a_{44}$	$a_{45}$

$a[lb_1 \dots ub_1][lb_2 \dots ub_2]$ ,  $BA, SOE = S$   
CMO

$$\text{loca}(a[i][j]) = BA + [(j - lb_2) * (ub_1 - lb_1 + 1) + (i - lb_1)] * S$$



$a_{11}$	$a_{21}$	$a_{31}$	$a_{41}$	$a_{12}$	$a_{22}$	$a_{32}$	$a_{42}$	$a_{13}$	$a_{23}$	$a_{33}$	$a_{43}$	$a_{14}$	$a_{24}$	$a_{34}$	$a_{44}$	$a_{15}$	$a_{25}$	$a_{35}$	$a_{45}$
100	102	104	106	108	110	112	114	116	118	120	122	124	126	128	130	132	134	136	138

$$\text{Loca}[a[3][5]] = 100 + [(5-1)*4 + (3-1)] * 2$$

$\nwarrow b_2 \quad \searrow n \cdot r$

$$\Rightarrow 100 + [16 + 2] * 2$$

$$\Rightarrow 136$$

Ex:  $a[-611 \dots +711][-811 \dots +911]$ ,

$$B_R = 7850, S_0 E = 9$$

$$\text{Loca}[a[633][844]] = 7850 + [(844 - (-811)) * 1323 + (633 - (-611))] * 9$$

$$\Rightarrow 7850 + [1655 * 1323 + 1244] * 9$$

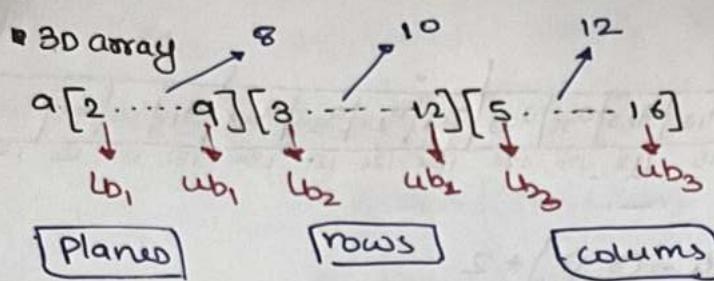
$$\Rightarrow 7850 + [2189565 + 1244] * 9$$

$$\Rightarrow 1972513$$

$$= [1 + 2141 + 214000] * 001$$

$$= [(1 - 11) + 214 * (2 - 11) + 214 * 01 * (2 - F)] * 001 = (1111111111111111) * 001$$

$$= [1 + 214 * 8 + 214 * 02] * 001$$



no. of rows ( $n \cdot r$ )  $\Rightarrow$   
 $ub_2 - lb_2 + 1$

no. of column ( $n \cdot c$ )  $\Rightarrow$   
 $ub_3 - lb_3 + 1$

R.M.O

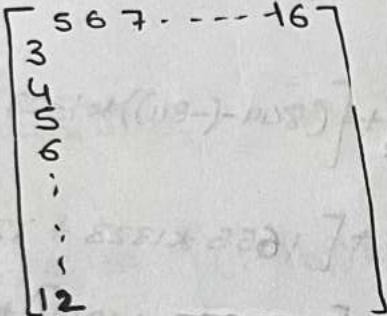
► 8  $\Rightarrow$  2D arrays

$\hookrightarrow$  each of size  $= 10 * 12$

► 10  $\rightarrow$  1D arrays

$\hookrightarrow$  each array has 12 elements

$$\begin{aligned} 2 &\rightarrow 10 * 12 \\ 3 &\rightarrow 10 * 12 \\ 4 &\rightarrow 10 * 12 \\ \vdots & \\ 5 &\rightarrow 10 * 12 \end{aligned}$$



Ex:

$$\text{loc}(a[5][7][9]) = 100 + [(5-2)10 * 12 + (7-3) * 12 + (9-5)] * 2$$

$\downarrow$        $\downarrow$        $\downarrow$        $\downarrow$

lb<sub>1</sub>    nr.    nc.    ub<sub>3</sub>

S.O.F

$$\Rightarrow 100 + [30 * 12 + 4 * 12 + 4] * 2$$

$\Rightarrow 924$

$$\text{loc}(a[7][11][14]) = 100 + [(7-2)10 * 12 + (11-3) * 12 + (14-5)] * 2$$

$$\Rightarrow 100 + [50 * 12 + 8 * 12 + 9] * 2$$

$\Rightarrow 1510$

Note

$a[1b_1 \dots ub_1][1b_2 \dots ub_2][ub_3 \dots ub_3]$ , B.A, soe = \$

R.M.O:

$$\text{Loc}(a[i][j][k]) : BA + \left[ \underbrace{(i-1b_1) * (ub_2 - 1b_1 + 1)}_{n \cdot r} * \underbrace{(ub_3 - 1b_3 + 1)}_{n \cdot c} + (j-1b_2) * \underbrace{(ub_3 - 1b_3 + 1)}_{n \cdot c} \right] + (k-1b_3) + \$$$

C.M.O:

$$\text{Loc}(a[i][j][k]) : BA + [i-1b_1 * n \cdot r * n \cdot c + (k-1b_3) * n \cdot r + (j-1b_2)] + \$$$

Lower triangular matrix (L.T.M)

Lower triangular matrix is possible only for square matrix.

Ex:  $a[1 \dots 4][1 \dots 4]$

$$i < j \Rightarrow 0$$

$i \geq j \rightarrow$  stored  
non-zero.

R.M.O.

$$\begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}_{4 \times 4}$$

∴ no of non-zero element  
 $\rightarrow n \frac{n(n-1)}{2}$

$a_{11}$	$a_{21}$	$a_{22}$	$a_{31}$	$a_{32}$	$a_{33}$	$a_{41}$	$a_{42}$	$a_{43}$	$a_{44}$
100	102	104	106	108	110	112	114	116	118

$\text{loc}(a[4][3]) =$

$$100 + \left[ \frac{(4-1) \times (4-1+1)}{2} + (3-1) \right] * 2$$

$\uparrow b_1$   
 $\uparrow b_2$

$$\Rightarrow 116.$$

i.e  $\text{loc}(a[3][2]) \neq$

$i > j$

$$100 + \left[ \frac{(3-1)(3-1+1)}{2} + (2-1) \right] * 2$$

$\uparrow b_1$

$$\Rightarrow 100 + \left[ \frac{2 \times 3}{2} + 1 \right] * 2$$

$$\Rightarrow 108.$$

Ex:  $a[137 \dots 961][137 \dots 961]$ ,  $B \cdot A = 4900$ , so  $E = 7$ .

$$\text{loc}(a[893][745]) = 4900 + \left[ \frac{(893-137)(893-137+1)}{2} + (745-137) \right] * 7$$

$$\Rightarrow 4900 + \left[ \frac{756 \times 757}{2} + 608 \right] * 7$$

$$\Rightarrow 4900 + [378 + 757 + 608] * 7$$

$$\Rightarrow 4900 + [285546] * 7$$

$$\Rightarrow 2012178$$

Note:  $a[lb_1 \dots ub_1][lb_1 \dots ub_1] \rightarrow B \cdot A$ , SOE = S, R.M.O., L.T.M

$$\text{loc}(a[i][j]) \Rightarrow B \cdot A + \left[ \frac{(1-lb_1)(i-lb_1+1)}{2} + (j-ub_1) \right] * 2$$

i > j

C.M.O. :-

$$\begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}_{4 \times 4}$$

$a_{11}$	$a_{21}$	$a_{31}$	$a_{41}$	$a_{12}$	$a_{22}$	$a_{32}$	$a_{42}$	$a_{13}$	$a_{23}$	$a_{33}$	$a_{43}$	$a_{14}$	$a_{24}$	$a_{34}$	$a_{44}$
100	102	104	106	108	110	112	114	116	118						

$$\text{loc}(a[4][2]) = 100 + \left[ \frac{(4-1+1)*(4-1+1+1)}{2} + \frac{-(4-2+1)*(4-2+1+1)}{2} + (3-2) \right] * 2$$

$$\Rightarrow 100 + \left[ \frac{4*5}{2} - \frac{3*4}{2} + \right] * 2 \Rightarrow 100 + (10 - 6) * 2 \Rightarrow 110$$

$$\text{loc}(a[4][3]) = 100 + \left[ \frac{(4-1+1)(4-1+1+1)}{2} - \frac{(4-3+1)(4-3+1+1)}{2} \right] * 2 + (4-3) * 2$$

↓      ↓  
i      j

→ 116.

Ex:  $a[117 \dots 489][117 \dots 489]$        $B.A = 8800, \text{ soe} = 9.$

$\text{local}[a[473][389]]$

$$\Rightarrow 8800 + \left[ \frac{(489-117+1)(489-117+1+1)}{2} - \frac{(489-389+1)(489-389+1+1)}{2} + (473-389) \right] * 9$$

$$\Rightarrow 8800 + \left[ \underbrace{\frac{373 * 374}{2}}_{69751} - \underbrace{\frac{101 * 102}{2}}_{5151} + 84 \right] * 9$$

→ 590956

Note:  $a[lb_1, \dots, ub_1][lb_1, \dots, ub_1], B.A, S.O \in \mathbb{Z}^+$ . LTM, CMAO.

$$\text{loc}(c[i:j]) = B \cdot A + \left[ \frac{(ub_1 - lb_1 + 1)(ub_1 - lb_1 + 1 + 1)}{2} - \frac{(ub_1 - j + 1)(ub_1 - j + 1 + 1)}{2} - (i - j) \right] * \xi$$

### Recursion

- Recursion are categorized into 4 type
- Tail Rec.: In the program very last statement is only one recursive call.

Ex:  $TR(\text{int } n)$ ?

```
if ( $n == 0$ ) return;  
else {  
    P( $f(n)$ );  
    return  $TR(n - 1)$ ;  
}
```

y  
y

### Note

1) Break;  $\neq$  ?

y // come out of bracket

2) return  $\rightarrow$  function

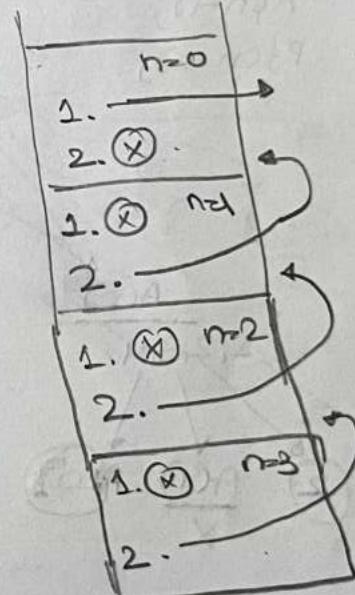
3) exit(0)  $\rightarrow$  Program,  
successful  
exit

4) exit(1)  $\rightarrow$  program,  
unsuccessful  
exit

5) exit()  $\rightarrow$  program, not  
informing  
anything

what is the O/p of  $TR(3)$ ?

O/p: 3' 2' 1'



• In tail Recursion we are miss using concept of recursion  
b/c it is wasting a lot of stack space

• the advantage is that we can easily write equivalent non-Recursion program by using for loop & while loop

### non-tail Recursion

• in the program after the recursive call there are some statement to execute then it is called as non-tail Recursion

Ex①  $A(\text{int } n)$

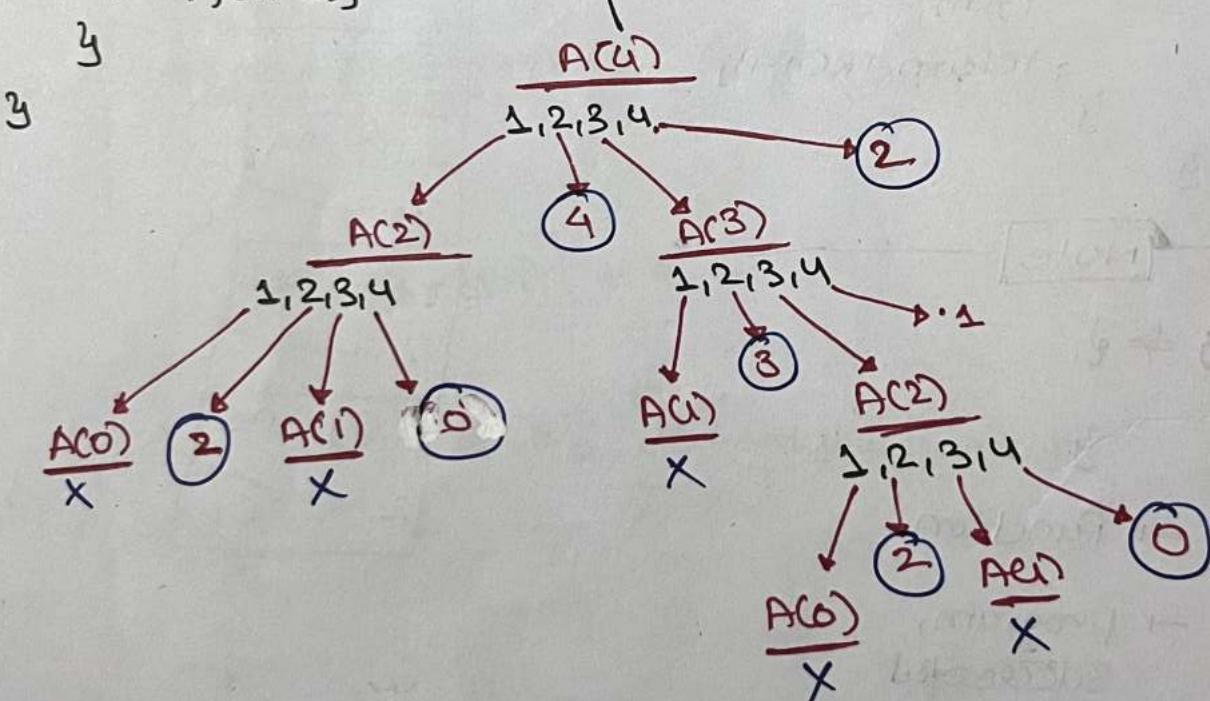
if ( $n \leq 1$ ) return;

else?

1.  $A(n-2)$ ;
2.  $Pf(n)$ ;
3.  $A(n-1)$ ;
4.  $Pf(n-2)$ ;

what is O/p of  $A(4)$ ?

O/p: 2,0,4,3,2,0,1,2.



Q1 AC(n)

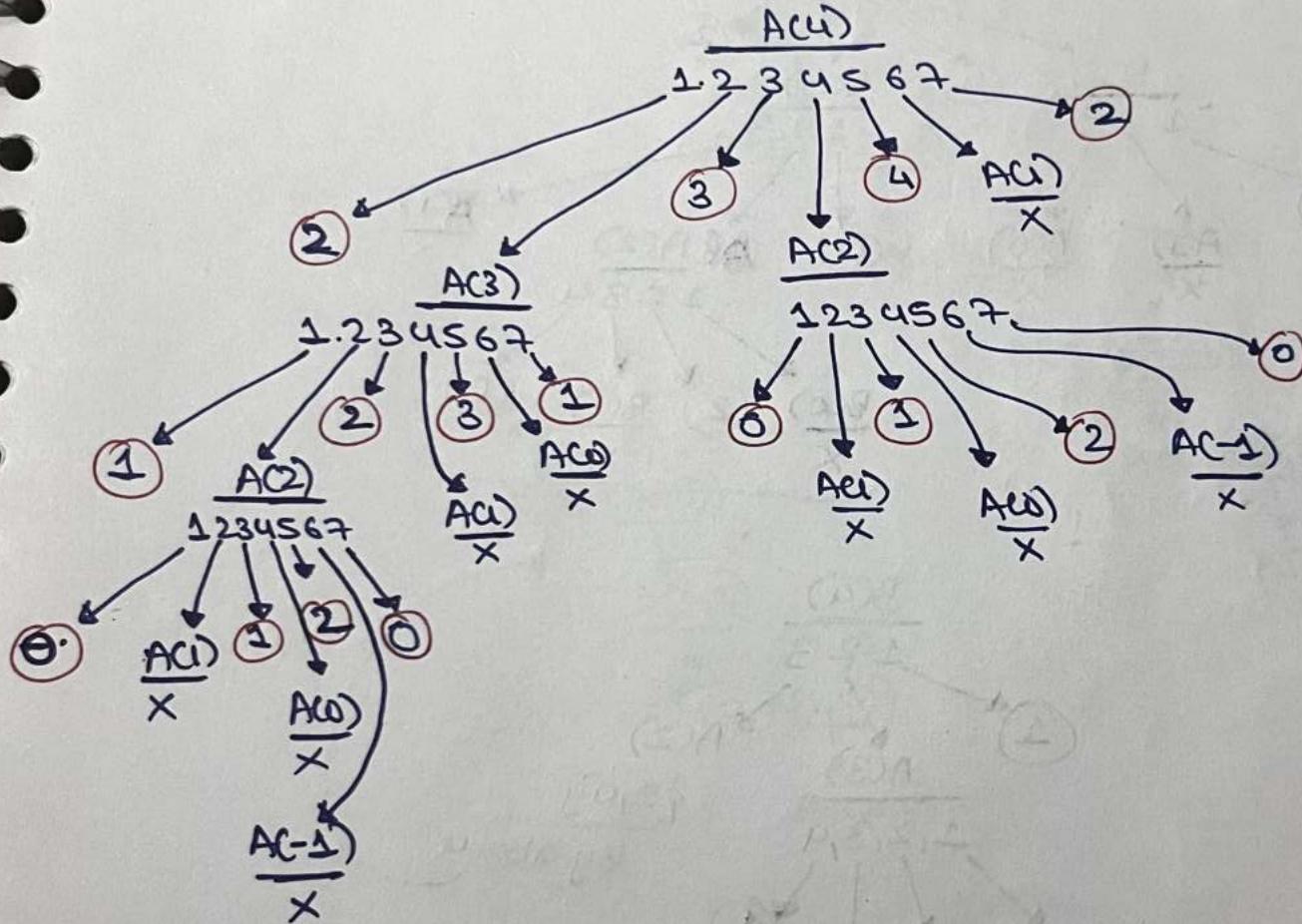
```
if (n ≤ 1) return j  
else {  
    1. Pf(n-2)  
    2. AC(n-1);  
    3. Pf(n-1)  
    4. AC(n-2);  
    5. Pf(n);  
    6. AC(n-3);  
    7. Pf(n-2);  
    }  
}
```

O/P: 2, 1, 0, 1, 2, 0, 2, 3, 1, 3, 0, 1, 2, 0, 4, 2

Note

in non tail Rec. writing an equivalent non. Recursive program is possible but difficult.

3



Indirect Recursion

Two or more function calling each other is called indirect recursion.

$A(\text{int } n)$

if ( $n \leq 1$ ) return;

else {

1.  $B(n-2)$ ;

2.  $P(n)$ ;

3.  $B(n-1)$ ;

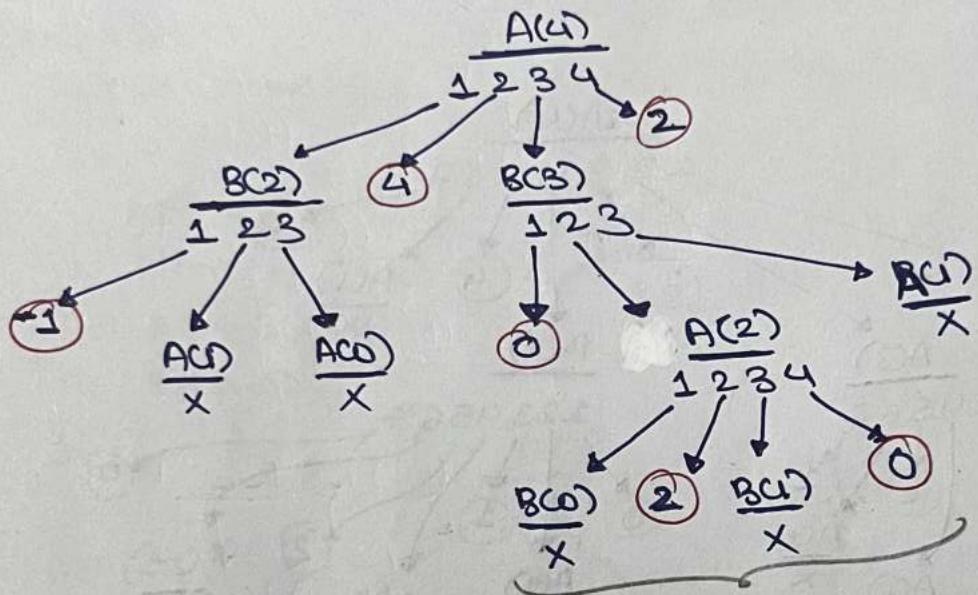
4.  $P(n-2)$ ;

}

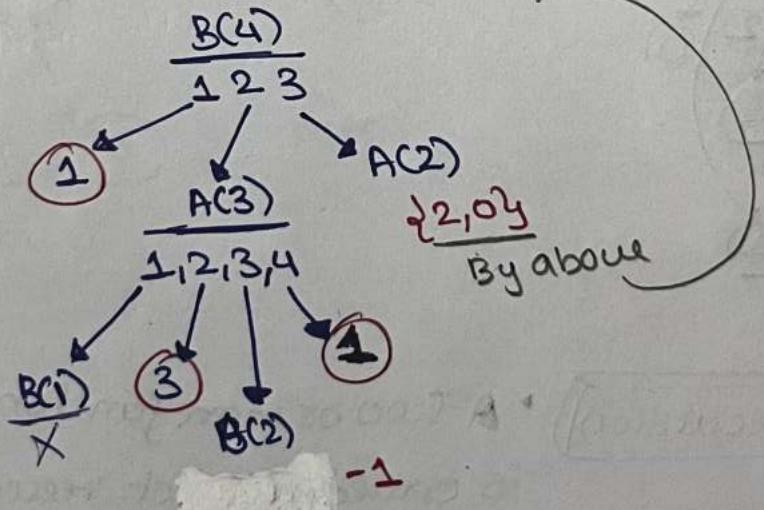
y

what is O/p of  $A(4)$ ?

→



→



O/p: 1, 3, -1, 1, 2, 0

```

BC int n?
if (n ≤ 1) return;
else {
    1. P(n-3);
    2. A(n-1);
    3. A(n-2);
}
y
y
y
    
```

O/p: -1, 4, 0, 2, 0, 2

## Nested Recursion

(acker mains functn).

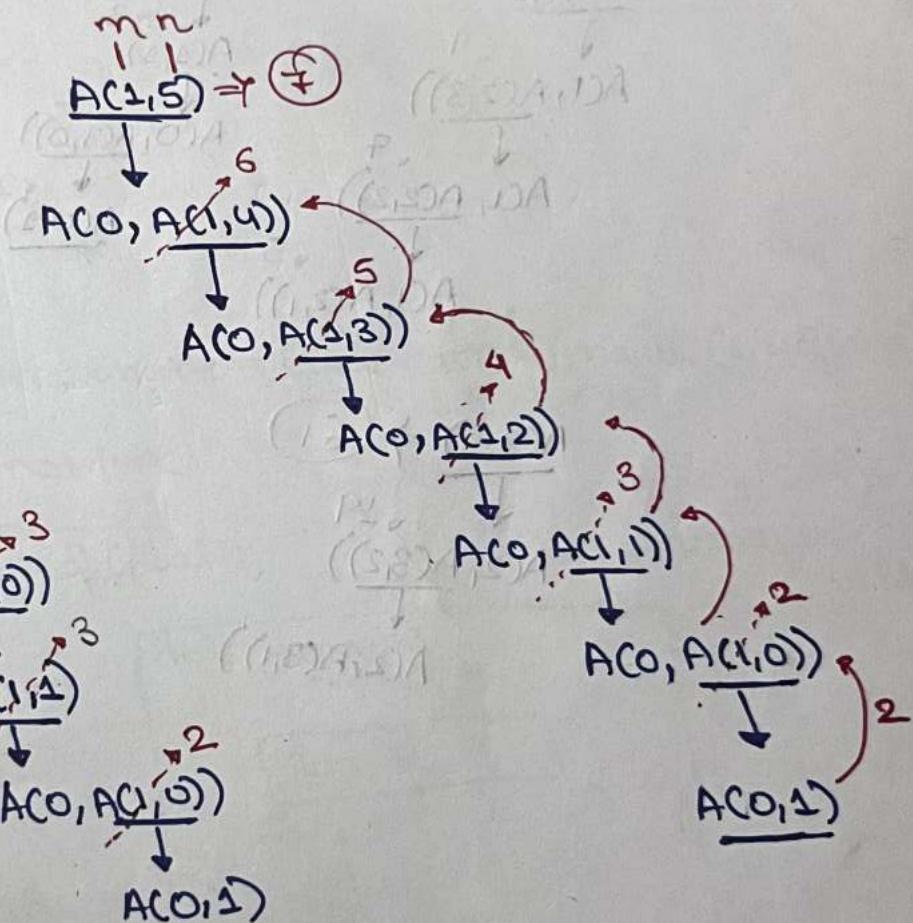
- A recursion function which is passing itself as a parameter then it is called as nested recursion.

$$A(m,n) = \begin{cases} n+1, & \text{if } m=0 \\ A(m-1, 1), & \text{if } n=0 \\ A(m-1, A(m, n-1)), & \text{otherwise} \end{cases}$$

what is o/p of  $A(1,5)$ ?  $\oplus //$

$$\begin{array}{c} m \quad n \\ \downarrow \quad \downarrow \\ A(2,3) = 9 \\ \downarrow \quad \downarrow \\ A(1, A(2,2)) \\ \downarrow \quad \downarrow \\ A(1, A(2,1)) \\ \downarrow \quad \downarrow \\ A(1, A(2,0)) \\ \downarrow \quad \downarrow \\ A(1, 1) \\ \downarrow \quad \downarrow \\ A(0, A(1,0)) \\ \downarrow \quad \downarrow \\ A(0, 1) \end{array}$$

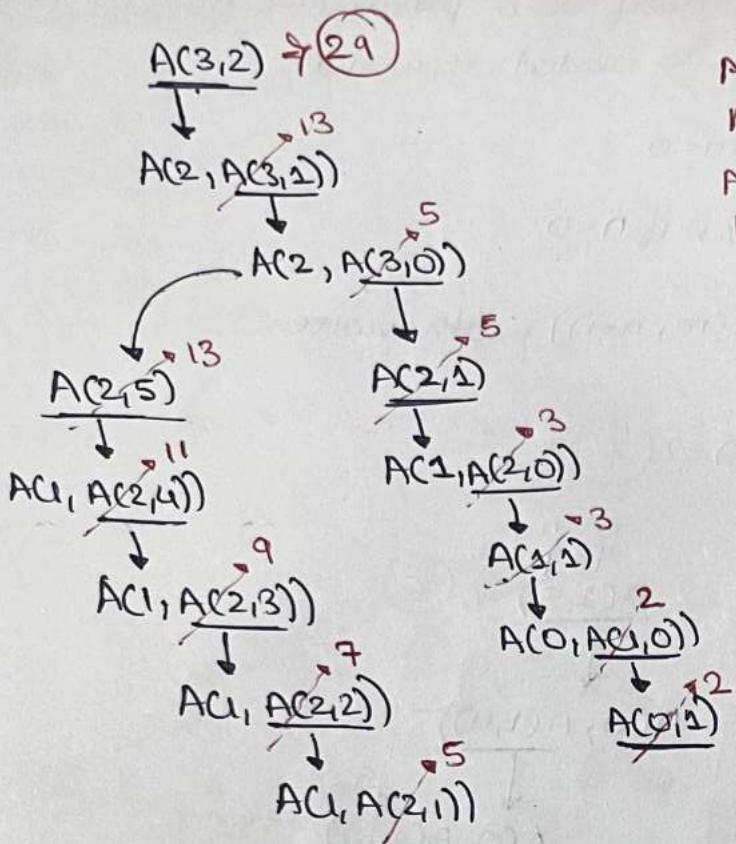
O/P: 9.



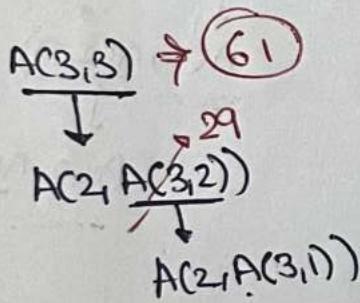
H.W

$$\textcircled{1} \quad A(3,2) = 29$$

$$\textcircled{2} \quad A(3,3) = 61$$



$$\begin{aligned}
 A(2,1) &= 5 \\
 A(2,3) &= 9 \\
 A(2,5) &= 13 \\
 A(2,7) &= 17 \\
 A(2,9) &= 21 \\
 A(2,11) &= 25 \\
 A(2,13) &= 29 \\
 A(2,29) &\Rightarrow 2 \times 29 + 3 \\
 &\Rightarrow 58 + 3 \\
 &\Rightarrow 61
 \end{aligned}$$



- Dynamic memory allocation (D.M.A)
  - array follow static memory allocation which means that we should the size of array prior to the declaration.
  - If the requirement is dynamical allocating the memory at runtime, then we need to use the concept of DMA.
  - By using the concept of DMA we can allocate the memory whenever we want & How ever we want at runtime
  - these will allocated at the heap segment.
  - the lifetime of Dynamic Memory (DMA) is entire program.
- Following funct<sup>n</sup> are used in the DMA
1. malloc():
  2. calloc():
  3. realloc():
  4. free():

**Malloc()**

Syntax: Pointer-variable = (type-cast \*) malloc(sizeof(size));

Ex: float \*ptr;

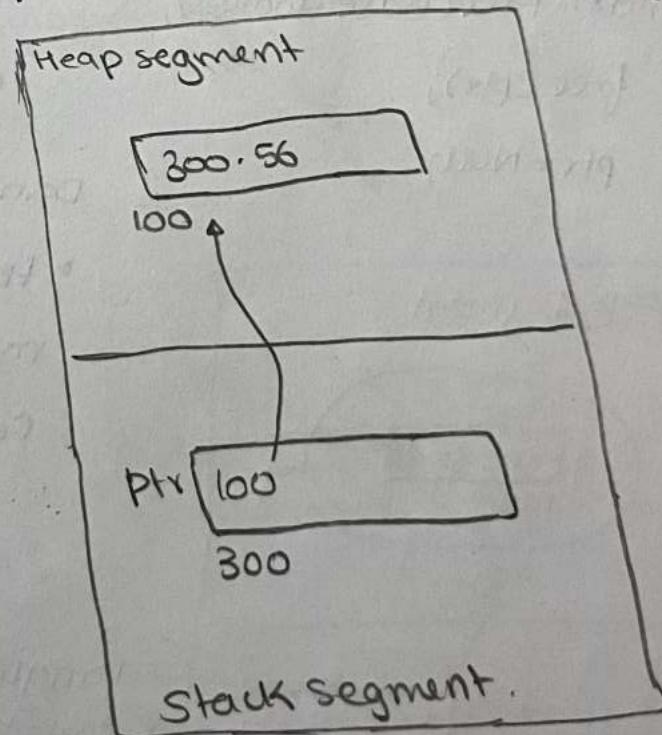
ptr = (float \*) malloc(sizeof(float));

\*ptr = 300.56;

printf ("%f", \*ptr);

↑  
not funct<sup>n</sup>, it's just an operator

//malloc use  
#include  
<stdlib.h>



//malloc only take Byte as an argument.

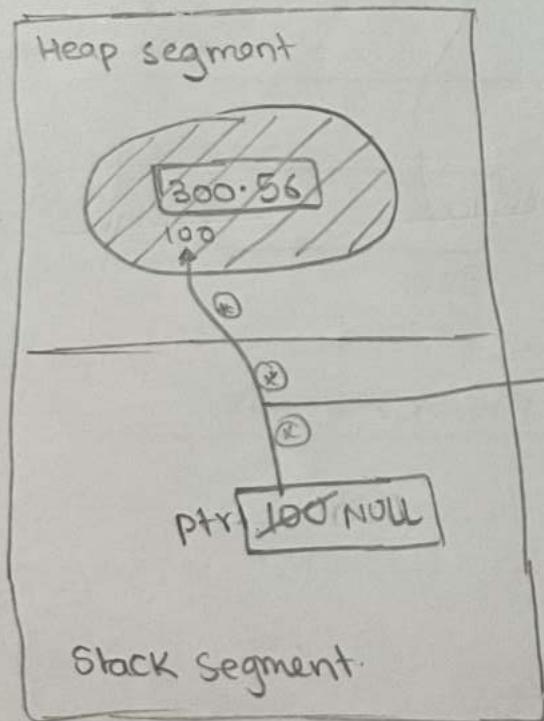
- malloc Dynamical allocate memory in the heap segment & it take one argument that is in Byte
- sizeof is an operator and it calculate the size in Byte
- malloc allocate the memory in heap segment & return its Base address & these Base address is an integer pointer.
- B/c it is returning B-A receivers should be an pointers, b/c it is returning it is integer pointer then respectively type Casting is required to access the required no. of Bytes.
- whatever the memory allocation in the stack segment can be accessed using variable names the memory is allocated in the heap segment can only be accessed through pointer
- malloc return NULL if it is not able to allocate memory if memory is full

`free()`

Syntax: `free(Base address);`

Ex: `free(ptr);`

`ptr=NULL;`



- `free()` deallocated the memory in heap segment & it takes one argument that is B-A.

### Dangling Pointers:

- the pointer pointing to deallocated memory (unknow location) is called as Dangling Pointer

after  
`ptr=NULL;`

Dangling  
Pointers

- After completion of the usage of dynamically allocated memory it must be deallocated otherwise it is called "memory leakage".

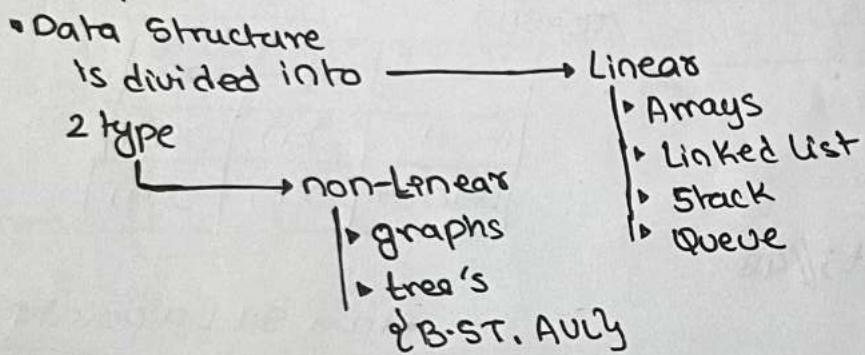
dangling ptr is removed too.

## Data Structure

- Data Structure is a way of organizing the data element by considering its relationship
- Data structure mainly deal with
  1. how efficiently data can be organized & stored in the computer memory
  2. how efficiently data can be retrieved & manipulated

Program = Algorithm + Data Structure

Data Structure = organized + operation Data



• Abstract Data type (ADT): In order to simplify the process of solving the problem data structure combined along with a operation is called as "ADT".

- ADT mainly deal with
- Declaration of Data
- Declaration of operation

## Linked List

- It's an Data Structure used to store data element, the successive element are connected by pointers & last element will contain "null"
- We can increase the size of Linked List dynamically until the memory is full
- If the requirement is frequently inserting & deleting at vary's position then Linked List is best suitable data structure
- The main Drawback of Linked List is that it only allow Sequential access.

Construction of Linked List:

①. Struct node {

    int data; //2B

    Struct node \* next; //4B

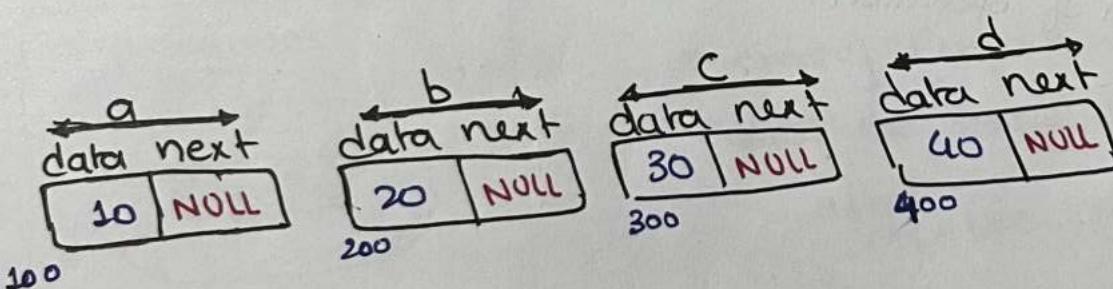
y;

typedef struct node nodej

②. node a = {10, null};  
 node b = {20, null};  
 node c = {30, null};  
 node d = {40, null};

Declaration  
4 Initialization

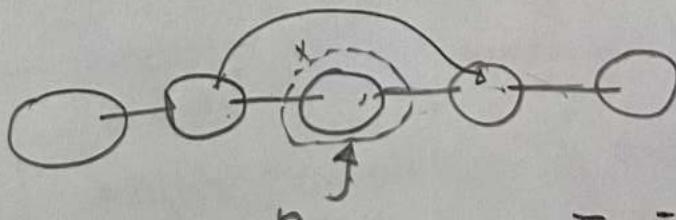
a,b,c,d are  
Structure  
variable



Note

In the S.L.L last node next pointer is always NULL

In the S.L.L we can traverse only in one direction.



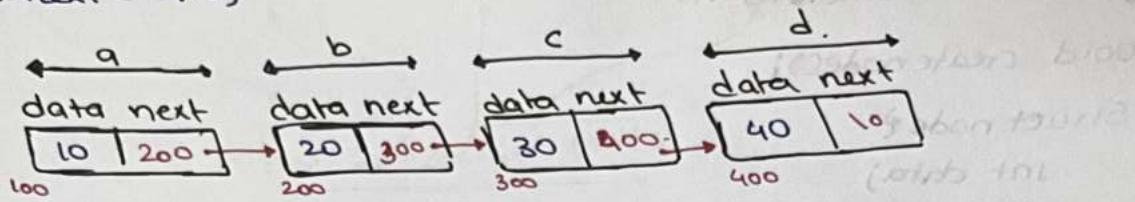
TC = O(n)

TC = SLL

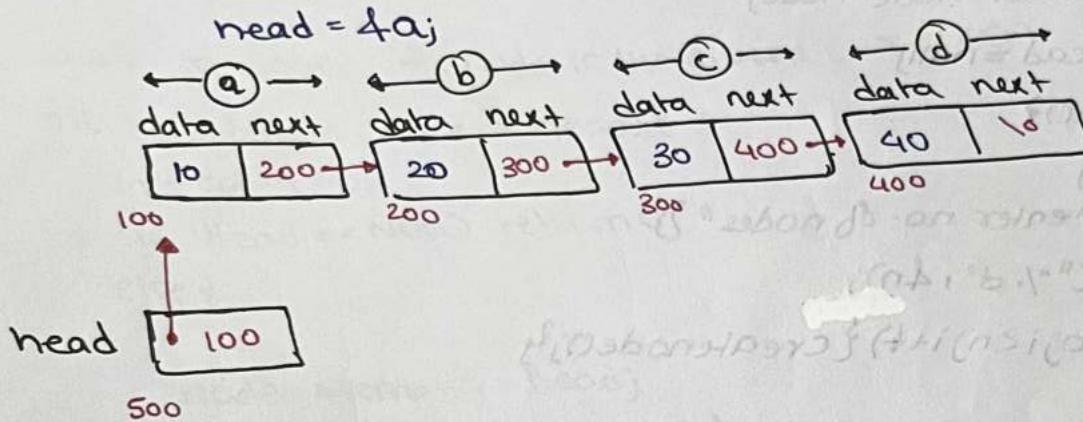
	B.C.	WC
Insert	O(1)	O(n)
Deletion	O(1)	O(n)

In a S.L.L a pointer is given, point to some node. So to delete that node what is the TC.

- ③ a. next = & b;  
 b. next = & c;  
 c. next = & d;



- ④ node \* head = NULL;



### Note

1. pf(head) = 100
2. pf(head → data) // 10
3. if(head == NULL)  
 // if above condition is true  
 then linked list is empty

④ pf(head → next); // 200

- 5) pf(head → next == NULL); // if above condition is true then L.L have only one element.
- 6). pf(head → next → next → data); // 40

- Creation of linked list is done by self referential structure variable
- If the requirement is frequent insertion & deletion in various position then linked list very much useful compare to array.

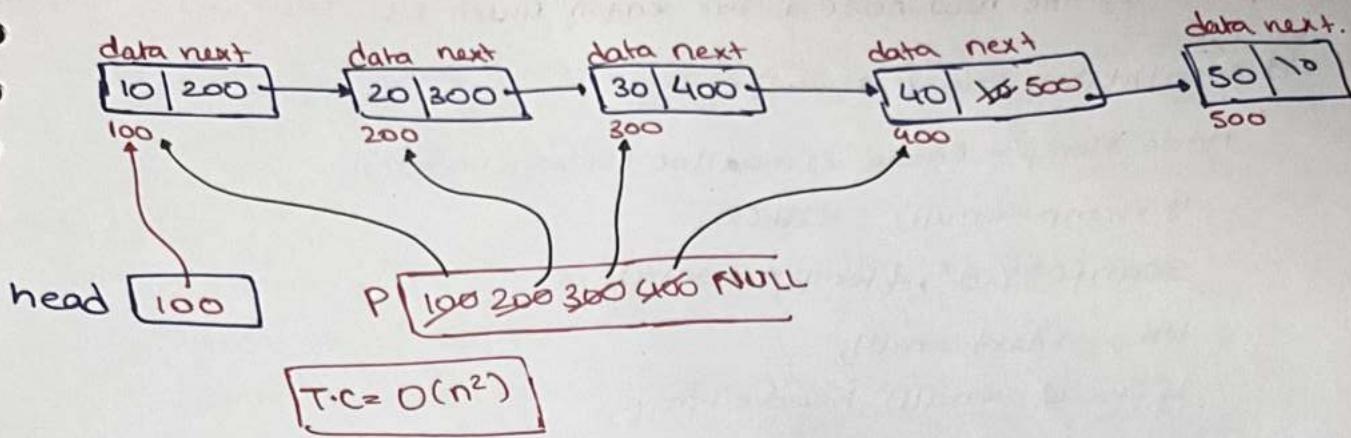
• Write a program to create L-L Dynamical with n nodes.

```
#include <Studio.h>
#include <StudioIB.h>
void createnode();
Struct node {
    int data;
    Struct node *next;
};

typedef Struct node node;
node *head = null;
Void main()
{
    int i, n;
    printf("enter no. of nodes");
    scanf("%d", &n);
    for(i=0; i<n; i++) {createnode();}
}
```

y

```
void createnode()
{
    node *temp = (node*) malloc (sizeof(node));
    if (temp == null) exit(1);
    printf("enter data element");
    scanf("%d", &temp->data);
    temp->next = null;
    if (head == null)
        head = temp;
    else {
        node *p = head;
        while (p->next != null) {
            p = p->next;
        }
        p->next = temp;
    }
    p = null; // optional
    temp = null; // optional
}
```



W.A.P to count # of node in the given L.L. T.C:  $O(n)$

```
int Listlength(node * head){  
    int count = 0;  
    if(head == NULL) return 0;  
    else {
```

```
        node *temp = head;  
        while(temp->next != NULL){  
            count++;  
            temp = temp->next;
```

}  
Return count;

}  
WAP to insert new node at the begin of L.L T.C.  $O(1)$

InsertAtBegin(node \* head){

Creating node }  

```
    node *temp = (node *)malloc(sizeof(node));  
    // if (temp == NULL) exit(1); // if not written then will cause.  
    scanf("./l.d", &temp->data);  
    temp->next = NULL; // Segmentation error @ NULL  
    // (null->next)  
    if(head == NULL) head = temp;  
    else {  
        temp->next = head; // 1st connect external link then  
        head = temp; // break internal link
```

}  
{

Q.A.P Insert the new node at the end of given L.L T.C. O(n)

Void insertAtEnd (node \*head) {

    node \*temp = (node \*) malloc (sizeof(node));

    if (temp == null) exit(1);

    scanf("%d", &temp->data);

    temp->next = null;

    if (head == null) head = temp;

    else {

        node \*curr = head;

        while (curr->next != null) {

            curr = curr->next;

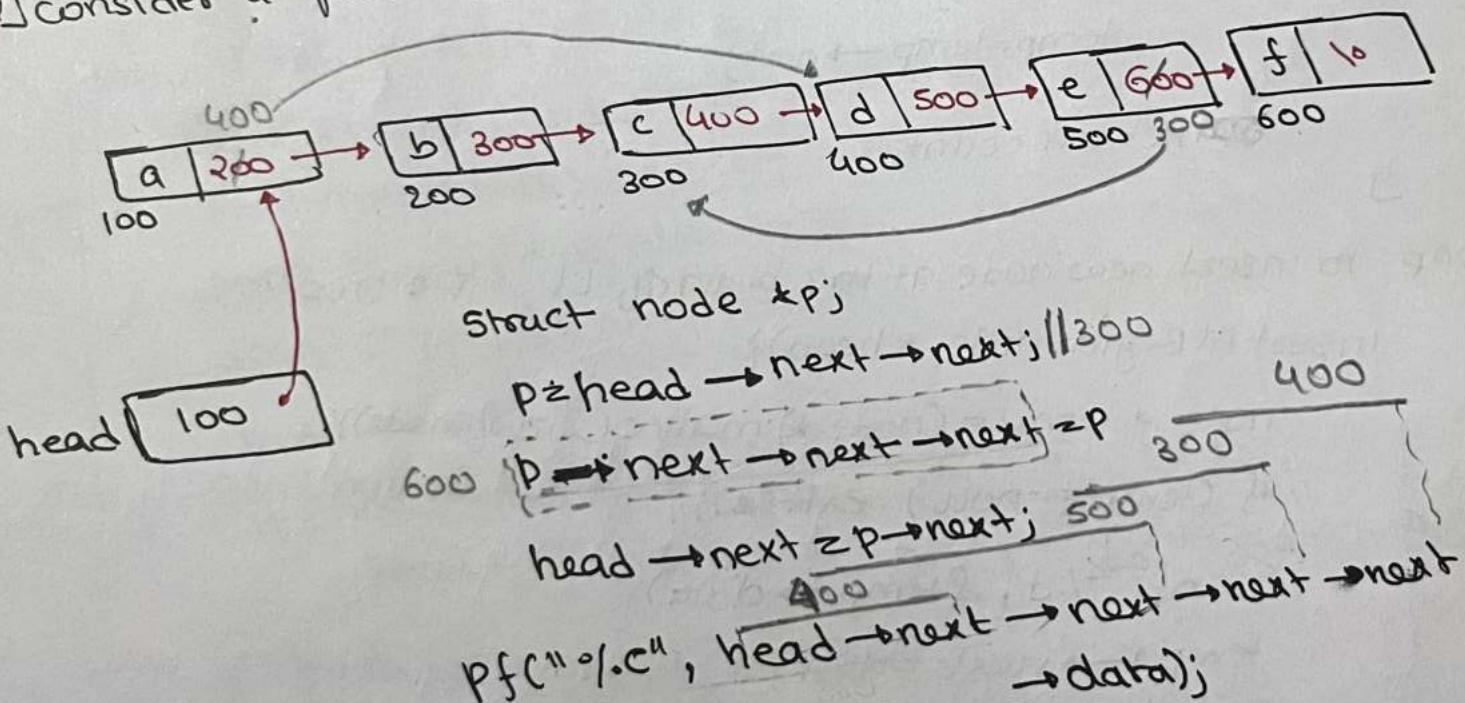
        y

        curr->next = temp;

    y

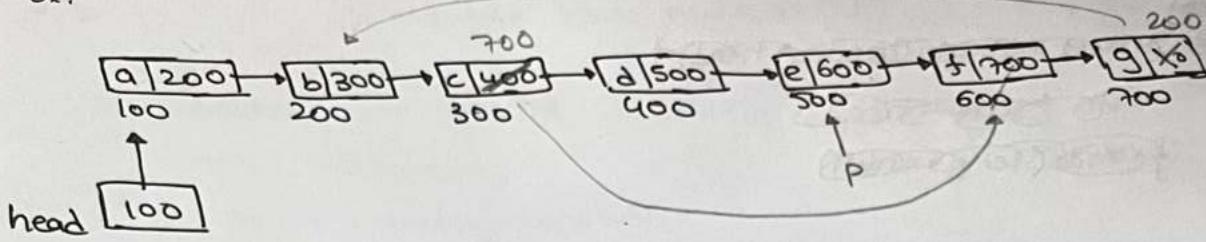
    y

Q) consider a following LL with given statement.



O/p: d//

Gx:



Struct node \*p;

p = head → next → next → next → next; // 100

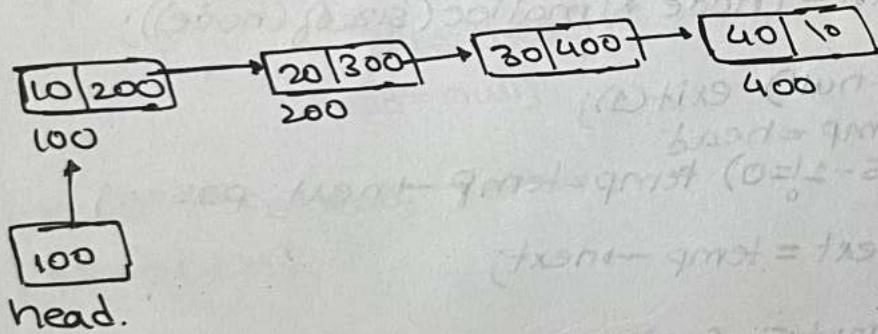
head → next → next → next = p → next → next;

p → next → next → next = head → next  
600  
700

pf("%d", p → next → next → next → next → next → next → data);

y  
O/p: c

Gx: WAP to insert the new node at the given position



```
int listlength(node *head){  
    int count=0; node *temp=head;  
    while(temp!=null){
```

count++;  
 temp=temp→next;

y

return count;

y

```

insertAtPos( node *head , int pos ) {
    int length = listLength( head );
    if ( pos <= 0 || pos > length + 1 ) {
        pf("invalid position");
        return;
    }
    if ( pos == 1 ) { insertAtBegin( head ); return; }
    if ( pos == length + 1 ) {
        insertAtEnd( head );
        return;
    }
    else {
        node *curr = ( node * ) malloc( sizeof( node ) );
        if ( curr == null ) exit( 1 );
        node *temp = head;
        while ( pos - 1 != 0 ) temp = temp->next; pos--;
        curr->next = temp->next;
        temp->next = curr;
    }
}

```

Scanf("%d",  
 &curr->data);

← node \*temp = head  
 while (pos - 1 != 0) temp = temp->next; pos--;

Q. WAP to delete the last node.  $T(C(n))$

```
deleteAtEnd (node *head) {
    node *p = head;
    int len = listLength(head);

    if (p == null) return;
    if (p->next == null) { free(p);
        head = null;
    }
    while (len - 1 != 0) { p = p->next; len--;
        free(p->next);
        p->next = null;
    }
}
```

Q] WAP to delete the 1st node.

```
deleteAtBegin (node *head) {
    if (head == null) return;
    if (head->next == null) {
        free(head);
        head = null;
    }
}
```

```
node *p = head;
p = p->next;
```

```
free(head);
```

```
head = p;
```

}

Q1 WAP to delete element of a given position. TC: O(n)

```
deleteAtPos (node * head, int pos) { int len = listLength(head);  
    if (pos <= 0 || pos > length) { printf("Invalid"); return; }  
    if (head == null) return;  
    if (head->next == null) {  
        free(head); head = null; } }
```

{

```
if (pos == len) { deleteAtEnd(head); return; } }
```

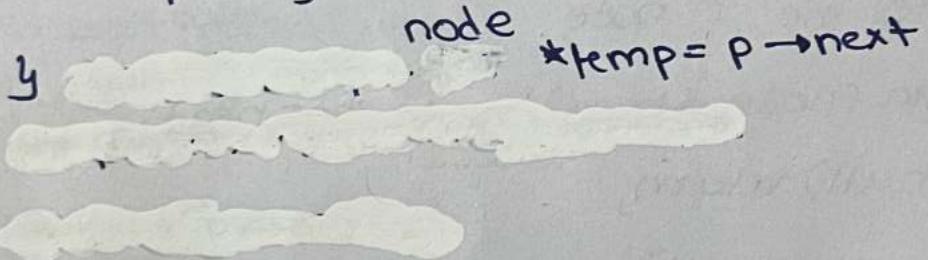
```
if (pos == 1) { deleteAtBegin(head); return; } }
```

```
else { node * p = head; }
```

```
while (pos - 1 != 0) {
```

```
    p = p->next; }
```

```
    pos--; }
```



```
    p->next = p->next->next; }
```

```
free(temp); }
```

{

}

Q.WAP to reverse a given LL. T.C: O(n)

node \*reverse(node \*head) {

    if(head == null) return;

    if(head->next == null) return;

    node \*curr = head,

        \*prev = null;

    while(curr->next != null || curr == null) {

        node \*temp =

        curr->next;

        curr->next = prev;

        prev = curr;

        curr = temp;

    }

    return prev;

}

Q] WAP to find whether a cycle exist or doesn't exist.

LLcyclepresentornot(node \*head) {

    if(head == null || head->next == null)

        Pf("no cycle");

    node \*slow = head, \*fast = head->next;

    while(slow != fast) {

        slow = slow->next;

        fast = fast->next->next;

}

    if(slow == fast)

        Pf("cycle is present");

    else Pf("no cycle present");

}

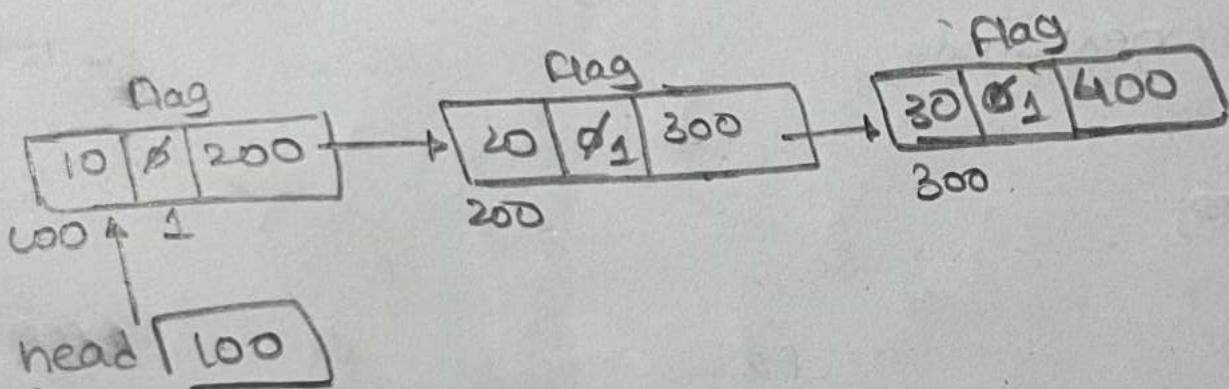
Solution : 2

[changing the struct]

```
Struct node{  
    int data;  
    int flag;  
    struct node *next;}
```

y:

In every node Initialize flag value '0'  
while creating L.L



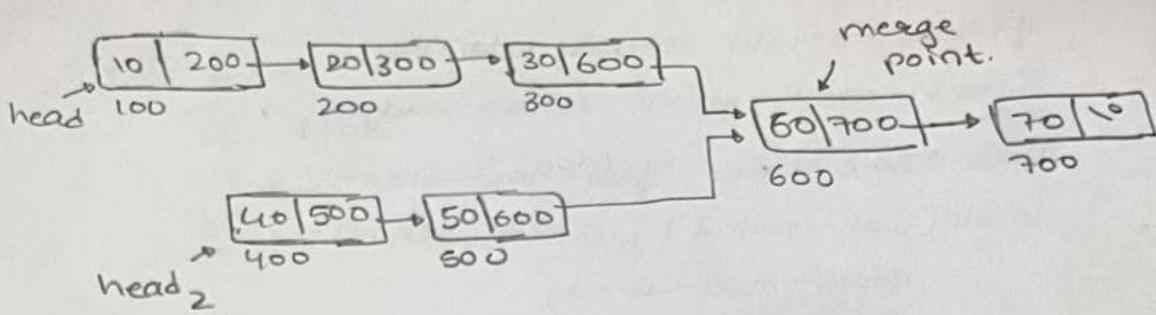
Q) WAP to find middle of linked list

```
node* middleLL(node *head){  
    if(head == null) return null;  
    if(head->next == null) return head;  
    node *slow = head, *fast = head->next;  
    while(fast != null && fast->next != null){  
        slow = slow->next;  
        fast = fast->next->next;  
    }  
    return slow;  
}
```

Q) WAP to find middle of the LL using listlength().

```
node* middleLL(node *head){  
    int mid = listlength() / 2;  
    if(head == null) return null;  
    if(mid == 0) return head;  
    if(head == null) return null;  
    if(head->next == null) return head;  
    int mid = listlength() / 2; node *temp = head;  
    while(mid-- != 0){  
        temp = temp->next;  
    }  
    return temp;  
}
```

Q) WAP to find merge point. In give two L.L :- T.C O(m+n)  
or O(len<sub>1</sub>+len<sub>2</sub>)



mergepoint (node \*head<sub>1</sub>, node \*head<sub>2</sub>) {

    if (head<sub>1</sub> == null || head<sub>2</sub> == null) return;

    if (head<sub>1</sub> → next == null && head<sub>2</sub> → next == null) return;

    int len<sub>1</sub> = listlength(head<sub>1</sub>); len<sub>2</sub> = listlength(head<sub>2</sub>); diff = 0;

    if (len<sub>1</sub> > len<sub>2</sub>) diff = len<sub>1</sub> - len<sub>2</sub>

    else diff = len<sub>2</sub> - len<sub>1</sub>

    while (len<sub>1</sub> > len<sub>2</sub> && diff != 0) head<sub>1</sub> = head<sub>1</sub> → next; diff--;

    while (len<sub>2</sub> > len<sub>1</sub> && diff != 0) head<sub>2</sub> = head<sub>2</sub> → next; diff--;

    while (head<sub>1</sub> != head<sub>2</sub>) {

        head<sub>1</sub> = head<sub>1</sub> → next;

        head<sub>2</sub> = head<sub>2</sub> → next;

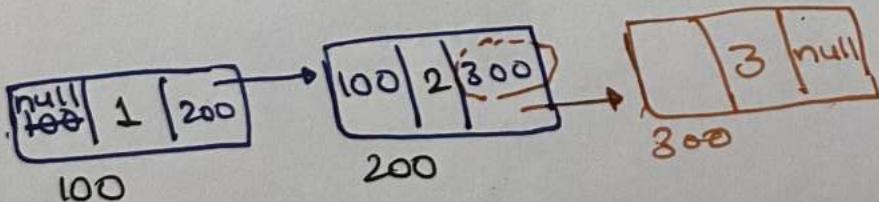
    }

    return head<sub>1</sub>;

}

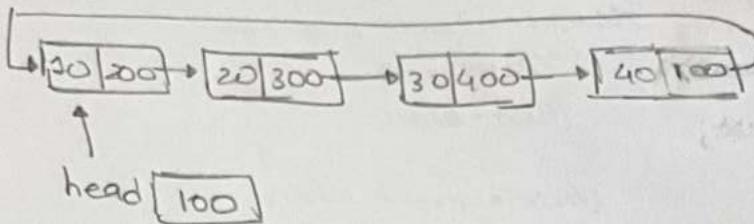
In a doubly linked list organization, after insertion of record in end  
involve modification of one pointer for existing list.

ex:



## Circular Single Linkedlist (C.S.L or S.L.L):

- In CSLL, last node next pointer will point to head node.



**note**

while ( $p \rightarrow \text{next} \neq \text{head}$ )

↳ termination condition

- WAP to find length of a C.S.L

```
int ListLengthCSLL(node *head){
```

```
    if(head == null) return 0;
```

```
    if(head->next == head) return 1;
```

```
    int len = 1; node *temp = head;
```

```
    while (temp->next != head) {
```

```
        temp = temp->next; // len++
```

```
        len++;
```

y

```
    return len;
```

y

- WAP to insert new node at the begin of the C.S.L.

```
insertAtBeginCSLL(node *head){
```

```
    node *curr = (node *) malloc(sizeof(node));
```

```
    if(curr == null) exit(1);
```

```
    scanf("%d", curr->data);
```

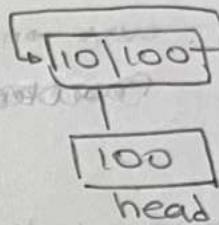
```
    curr->next = null;
```

```
    if(head == null) {
```

```
        head = curr;
```

```
        curr->next = head;
```

y



```

if(head->next == head) {
    head->next = curr;
    curr->next = head;
}

node *temp = head;
while(temp->next != head) {
    temp = temp->next;
}

temp->next = curr;
curr->next = head;
temp = NULL;

```

y

• WAP to insert the new node at the end of the C.L.L.

```

InsertAtEnd CLL(node *head) {
    node *curr = (node *) malloc(sizeof(node));
    if(curr == NULL) exit(1);
    scanf("%d", curr->data);
    curr->next = NULL;
    curr->prev = NULL;
    if(head == NULL) head = curr;
    curr->next = head;
}

node *temp = head;
while(temp->next != NULL) {
    temp = temp->next;
}

temp->next = curr;
curr->next = head;
temp = NULL;

```

y

• WAP to delete the last node of the C.L.L.

```
deleteAtEnd(node *head) {
    if(head == null) return;
    if(head->next == head) {
        free(head);
        head = null;
    }
    node *temp = head;
    while(temp->next->next != head) {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = head;
    temp = null;
}
```

• WAP to delete the 1<sup>st</sup> node of the C.L.L.

```
DeleteAtBeginCLL(node *head) {
    if(head == null) return;
    if(head->next == null) {
        free head->next = null;
        free(head);
        head = null;
    }
    node *curr = head, *temp = head;
    curr->prev->next = curr->next;
    curr->prev = null;
    head = curr->next;
    curr->next = null;
    free(curr);
}
while(temp->next != head) {
    temp = temp->next;
}
temp->next = curr->next;
head = curr->next;
curr->next = null;
free(curr);
}
```

these code  
Section is for  
Circular Double  
Linked List.

note T.C: C.S.L.L

	Begin	End
Insert	O(n)	O(n)
Delete	O(n)	O(n)

### Double Linked list (D.L.L)

- in double L.L every node contains 2 pointers pointing to previous node & next node
- we can traverse in both direction,
- insertion & deletion will take more amount of time b/c of more pointer updation.
- the main application D.L.L is, it is used in tree programming

### struct DLLnode{

    Struct DLL node \*prev;

    int data;

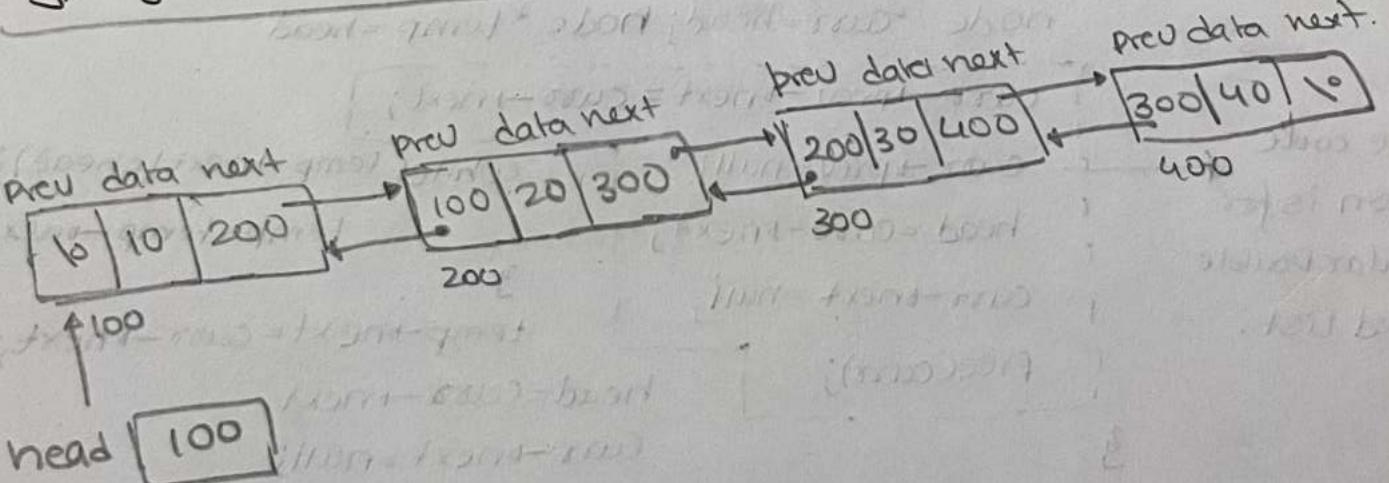
    Struct DLL node \*next;

y;

typedef struct DLL node node;

T.C:D.L.L

	Begin	End
Insert	O(1)	O(n)
Delete	O(1)	O(n)



- WAP to insert new node at the end of DLL

insertAtEnd DLL (node \*head)?

node \*curr = (node \*) malloc (size of (node));

if (curr == null) exit (1);

scanf (" . . d ", curr → data);

curr → prev = null;

curr → next = null;

if (head == null) {

    head = curr;

y

    if (head → next == null) {

        head → next = curr;

        curr → prev = head;

y

    node \*temp = head;

    while (temp → next != null) {

        temp = temp → next;

y

        temp → next = curr;

        curr → prev = temp;

        temp = null;

y

• WAP to insert a node at the beginning of the D.L.L

```
inserttheNodeATBegin(node *head)
node *curr = (node *) malloc(sizeof(node))
if(curr == null) exit(1)
scanf("%d", curr->data)
curr->next = null;
curr->prev = null;
if(head == null) return;
if(head->next == null)
    free(head);
head = null;
```

y  
node \*curr = head;
head = curr->next
curr->next = null;
free(curr);

y

- WAP to delete 1st node of D.L.L.

```

delete AtBegin DLL(node *head)
{
    if(head == null) return;
    if(head->next == null){ free(head); head = null; }
    node *temp = head; *temp->next = temp->next;
    while(temp->next->next != null){
        temp = temp->next;
        temp->next = temp->next->next;
    }
    temp->next = null; temp->next->prev = null;
    free(temp->next);
}

```

- WAP to delete the last node of a D.L.L.

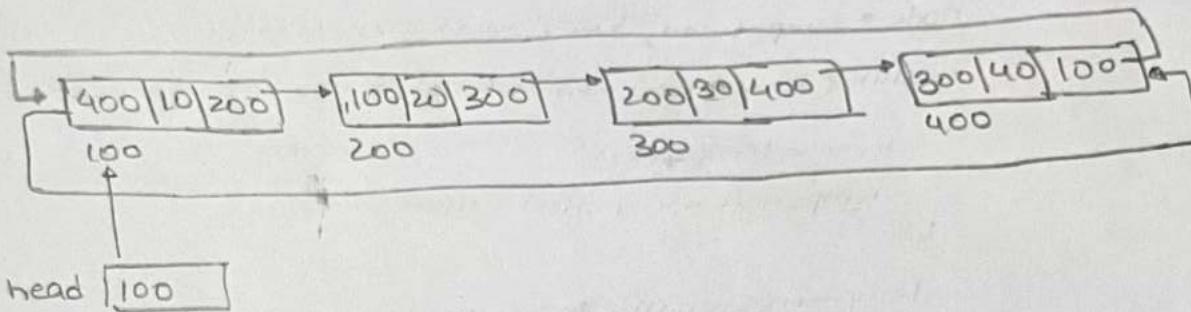
```

delete AtEnd(node *head)
{
    if(head == null) return;
    node *temp1 = head; *temp2 = head->next;
    while(temp2->next != null){ temp1 = temp1->next;
        temp2 = temp2->next;
    }
    if(temp2->next == null){
        free(head);
        head = null;
    }
    temp1->next = null;
    free(temp2);
}

```

- Circular Double Linked List (C.D.L.L)
- C.D.L.L 1st node previous will point to last node & last node next pointer will point to 1st node

T.C. : C.D.L.L		
	Begin	End
insert	O(1)	O(1)
delete	O(1)	O(1)



- WAP to insert at Begin & End in CDLL

Begin : InsertAtBeginCDLL(node \*head){

```

node *curr = (node *)malloc(sizeof(node));
if(curr==NULL) exit(1);
scanf("%d", curr->data);
curr->next = NULL;
curr->prev = NULL;
if(head == NULL) {
    head = curr; head->next = head;
}
if(head->next == head){
    curr->next = head;
    head->prev = curr;
    head = curr;
}
node *temp = head;
temp->prev->next = curr;
curr->prev = temp->prev;
temp->prev = curr;
curr->next = temp;
head = curr;
}

```

```

End: InsertionAtEnd (node *head) {
    node *curr = (node *) malloc (sizeof(node));
    if (curr == null) exit(1);
    scanf("%d", curr->data); curr->next=null;
    curr->prev=null;
    if (head == null) head=curr; head->next=head;
    node *temp = head;
    while (temp->next != head) temp=temp->next;
    temp->next = curr;
    curr->prev = temp;
    curr->next = head;
    head->prev = curr;
    temp=null;
}

```

\* WAP to delete from Begin & End of a CDLL.

Begin:

```

DeleteATBeginCDLL (node *head) {
    if (head == null) return;
    if (head->next == head) {
        head->next=null; free(head);
    }
    node *temp = head;
    head->prev->next = head->next;
    head->prev = null;
    head = temp->next;
    temp->next = null;
    free(temp);
}

```

end:

```
DeleteATEndCDLL( node *head) {
    if (head == null) return;
    if (head->next == null) { free(head); head=null; }

    node *prev-node = head->prev;
    head->prev = prev-node->prev;
    prev-node->next = null;
    prev-node->prev = null;
    free(prev-node);
}
```

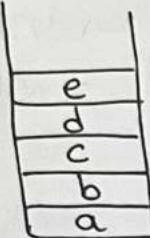
Note:

- next level Data Structure after
- linked list is "skip list".

## Stack

- Stack is a Data Structure used to store Data element with the restriction of first in last Out, or last in first out
- the elements are inserted and deleted from same end called "Top"
- One side open & one side close.

Ex:



Insertion: a, b, c, d, e

deletion: e, d, c, b, a

### Application of stack:

- recursive functn call
- balancing parenthesis.
- HTML & XML tag checking.
- Tower of Hanoi

- page back in browsers

- Infix to postfix conversion

- Postfix evaluation

- Fibonacci Series

### Stack ADT:

#### Declaration of data:

1. Space to Store Stack Element

→ array

→ linkedlist.

2. a Variable "TOP" pointing to  
Top most element in stack

#### Declaration of operation:

• 1. push():

↳ element to insert

• the above operation **insert**  
the element in the stack with  
the help of top.

2. pop(): the operation delete the  
element from stack using "TOP"

3. isFull(): the operation return  
true if the stack is full

4. isEmpty(): the operation return  
true if stack is Empty

- Implement stack:
- using array:

Initially  
Top = -1

16	4
26	3
31	2
27	1
14	0

Stack

overflow condition  
Top == N-1

underflow condition  
Top == -1

Push:  
Top: -1 or 4 or 3 or 2 or 1  
Top++;  
S[Top] = x;

Pop:  
Top: 4, 3, 2, 1 or -1  
x = S[Top--];

- WAP to perform push op<sup>n</sup> in the stack;

push(S, n, Top, x);  
Stack → Top  
size → element to insert

if (Top == N-1) Pf("Overflow condition"); return;  
else S[++Top] = x; y

y

- WAP to perform pop op<sup>n</sup> in the stack;

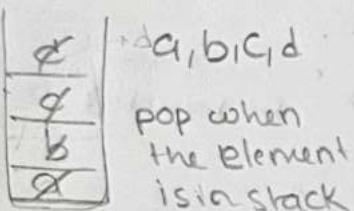
int pop(S, N, Top);  
int x = 0;  
if (Top == -1) Pf("Underflow condition"); return -1;  
else x = S[Top--];  
return x;

y

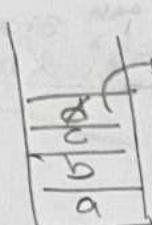
gate

Q) A program attempts to generate as many strings as possible of the string "abcd" by pushing the characters a,b,c,d. in the same order onto a stack, but it may pop off the top character. Which one of the following string can't be generated using the program?

(a) abcd.



(b) dcba



(c) cbad

push till  
C & the  
POP 4  
at last  
pushd

(d) cabd

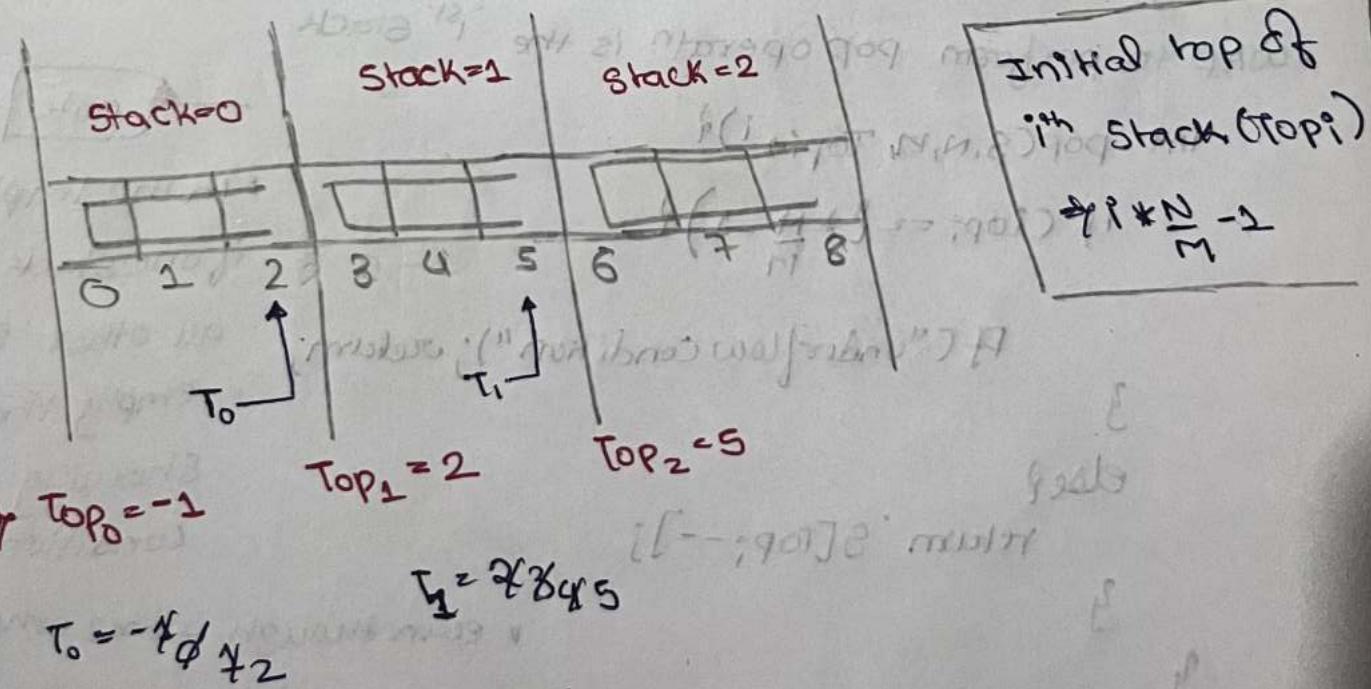
can't be  
generated

1. a  
2. b  
3. c  
4. d

### • Implementing multiple stacks in a single arrays:

In order to execute any program we required stack, if we want to execute multiple program then we need multiple stack in the memory. These concept is ~~stacks for multiple~~ similarly to implementing multiple stack in a single array.

array size ( $N$ ) = 9, no. of stack ( $m$ ) = 3, size of each stack  $= \left(\frac{N}{m}\right) = \frac{9}{3} = 3$



Initial Top of 0th stack ( $\text{Top}_0$ ) =  $0 \times \frac{N}{M} - 1 = -1$

Initial Top of 1st stack ( $\text{Top}_1$ ) =  $1 \times \frac{N}{M} - 1 = 2$

Initial Top of 2nd stack ( $\text{Top}_2$ ) =  $2 \times \frac{N}{M} - 1 = 5$

**Note**

- i<sup>th</sup> stack overflow condition arises when "Top<sub>i</sub>" reaches to

Initial Top of (i+1)<sup>th</sup> stack

- WAP to perform push operatr in the i<sup>th</sup> stack.

Stack size, no. of stack, Top<sub>i</sub>, stack no., element to be inserted  
push(S, N, M, Top<sub>i</sub>, i, x);

If ( $\text{Top}_i == (i+1) * \frac{N}{M} - 1$ ) {

Pf("Overflow condition");

return;

y

else

• Top<sub>i</sub>++

S[Top<sub>i</sub>] = x;

y

y

- WAP to perform pop operatr in the 1<sup>st</sup> stack

int pop(S, N, M, Top<sub>i</sub>, i){

If ( $\text{Top}_i == i * \frac{N}{M} - 1$ ) {

Pf("Underflow Condition"); return;

y

else

return S[Top<sub>i</sub>--];

y

y

**Note**

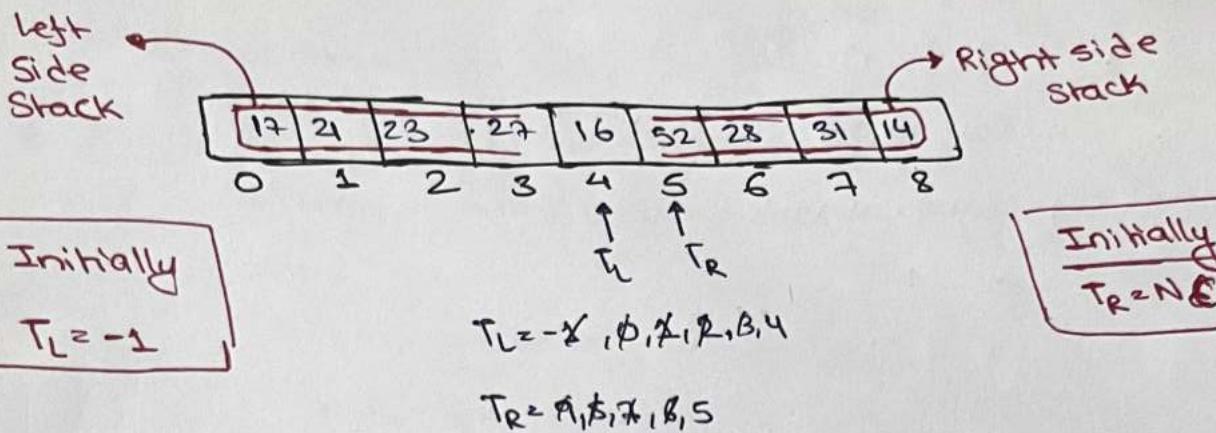
- In the implementation if one stack is full & all other stack are empty that still it is showing overflow condition.

• even though some empty slot are

available they are not efficiently utilized.

- To avoid these problem we will use another approach.

- Implementing multiple stacks in a single array efficient ways.
- array size ( $N$ ) = 9      no. of stacks ( $M$ ) = 2



In above implementation we are using two stack in a single array growing in the opposite direction.

WAP to perform push & pop operation in both the stack.

Overflow condition

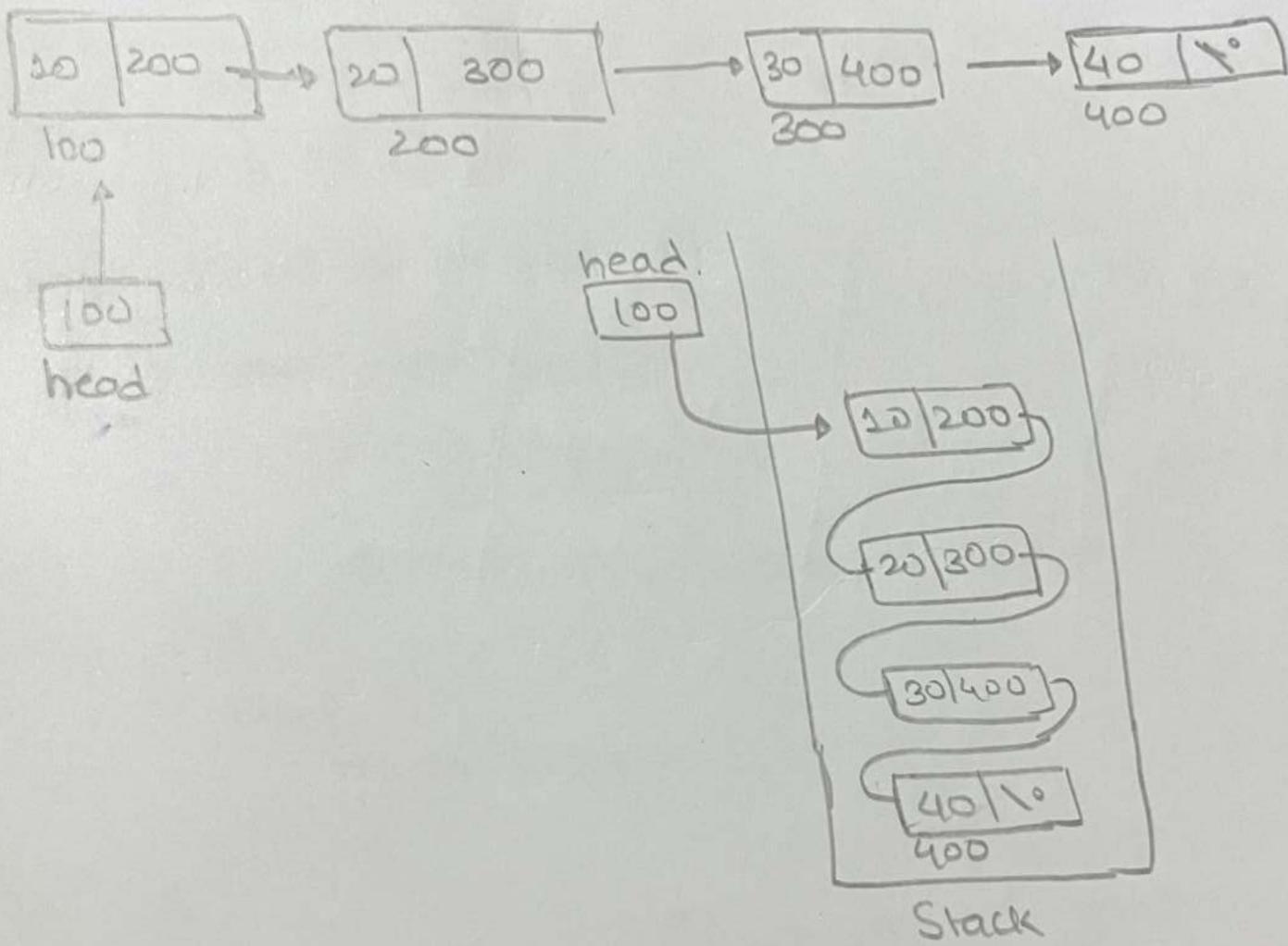
$$T_R - T_L = 1$$

$$T_R = T_L + 1$$

$$T_L = T_R - 1$$

↓  
for both  
the stack.

- Implementing Stack using Linked list



- Insertion & deletion will always happen from the head node.
- WAP to perform push operation in the stack using linkedlist.
 

```

        ↗ element to insert
push(node *head, int x) {
    node *temp = (node *) malloc(sizeof(node));
    if (temp == null) {"Stack overflow condition"; exit(1);}
    else {
        temp->data = x;
        temp->next = head;
        head = temp;
    }
}
      
```

- WAP to perform pop operatn in the stack using linked list? TC: O(1)

```

int pop(node *head) {
    if (head == null) {"Underflow condition"; return -1;}
    else {
        node *p = head;
        int x = head->data;
        head = head->next;
        free(p); p=null; return x;
    }
}
      
```

y

DD+

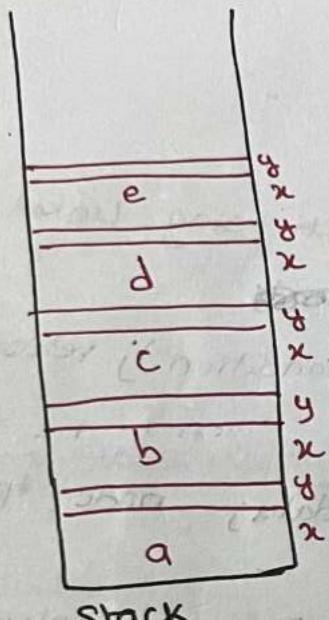
d + n : n : x : f :  
104  
104  
104  
104  
104

borrow 2 extra memory units.

1-2

- Find avg. life time of an element in the stack:
- The time for insert / delete an element take 'x' & there is a delay of "y" b/w every such operation.
- The life time is considered as time elapsed from end of Push to start of pop operation that removes an element from Stack.
- Find avg. lifetime of an element.

PGQ  
3.8  
Pg No. 356



element = abc, ide

element	Life Time
e	y
a	$2x + 3y$ ( $y + x + y + x + y$ )
c	$y + x + y + x + y + x + y \Rightarrow 4x + 5y$
b	$6x + 7y$
d	$8x + 9y$ .

### Infix, prefix, postfix

- Binary operation b/w 2 operand.

Infix: ex:  $a + b$ .

```

    a   +   b
    |   |   |
OP1   Operator   OP2
    |   |
    |   |
    +---+
        Operand
    
```

Prefix:

- Binary operatin before 2 operands.

```

    + ab
    |   |
    |   |
    +---+
        Operand
    
```

OP<sub>2</sub>      OP<sub>1</sub>

Postfix: Binary operatin after 2 operand

```

    abt
    |   |
    |   |
    +---+
        Operand
    
```

OP<sub>1</sub>      OP<sub>2</sub>

• Infix:  $A + B * (C + D) / F + D * E$

Prefix:  $A + B * (C + D) / F + D * E$

$$\Rightarrow A + B * \underline{CD} / F + D * E$$

$$\Rightarrow A + \underline{*B + CD} / F + D * E$$

$$\Rightarrow A + \underline{B + \underline{CD}} / F + D * E$$

$$\Rightarrow A + \underline{B + CDF} + \underline{DE}$$

$$\Rightarrow \underline{+A} \underline{B + CDF} + \underline{DE}$$

$$\Rightarrow + + A \underline{B + CDF} \underline{DE}$$

Postfix:  $A + B * (C + D) / F + D * E$

$$\Rightarrow A + B * \underline{CD} / F + D * E$$

$$\Rightarrow A + \underline{BCD} * / F + D * E$$

$$\Rightarrow A + \underline{BCD} * \underline{F} / + D * E$$

$$\Rightarrow A + \underline{BCD} * \underline{F} \underline{I} + \underline{DE}$$

$$\Rightarrow ABCD * \underline{F} \underline{I} + \underline{DE}$$

$$\Rightarrow ABCD * \underline{F} \underline{I} + \underline{DE} *$$

Infix:  $a + b * c - d / e \uparrow f * g + h$

Prefix:  $a + b * c - d / e \uparrow f * g + h$

$$\Rightarrow a + b * c - d / \underline{e} \uparrow \underline{f} * g + h$$

$$\Rightarrow a + \underline{bc} - d / \underline{ef} * g + h$$

$$\Rightarrow a + \underline{bc} - \underline{d} \uparrow \underline{ef} * g + h$$

$$\Rightarrow a + \underline{bc} - \underline{d} \uparrow \underline{efg} + h$$

$$\Rightarrow - + a * bc * / d \uparrow e f g h + h$$

$$\Rightarrow + - + a * bc * / d \uparrow e f g h h$$

Postfix:  $a + b * c - d / e \uparrow f * g + h$

$$\Rightarrow a + b * c - d / \underline{ef} \uparrow * g + h$$

$$\Rightarrow a + \underline{bc} * - \underline{def} \uparrow * g + h$$

$$\Rightarrow a + \underline{bc} * - \underline{def} \uparrow \underline{g} * h$$

$$\Rightarrow abc * + - def \uparrow lg * h$$

$$\Rightarrow abc * + def \uparrow lg * - h$$

$$\Rightarrow abc * + def \uparrow lg * - ht$$

4.00 Infix:  $f * (d - b) ^ {(a/c)} ^ {(e+g)} * h$

Prefix:  $f * (d - b) ^ {(a/c)} ^ {(e+g)} * h$       Postfix:  $f * (d - b) ^ {(a/c)} ^ {(e+g)} * h$

$\Rightarrow f * (- \underline{db} ^ {\underline{ac}}) ^ {\underline{eg}} * h$        $\Rightarrow f * (\underline{db} - \underline{ac}) ^ {\underline{eg}} * h$

$\Rightarrow f * ^ {-} \underline{db} / \underline{ac} ^ {\underline{eg}} * h$        $\Rightarrow f * \underline{db} - \underline{ac} / ^ {\underline{eg}} * h$

$\Rightarrow f * ^ {^{\wedge \wedge}} \underline{db} / \underline{ac} + \underline{eg} * h$        $\Rightarrow \underline{fdb} - \underline{ac} / ^ {\underline{eg}} * h$

$\Rightarrow * f ^ {^{\wedge \wedge}} \underline{db} / \underline{ac} + \underline{eg} * h$        $\Rightarrow fdb - ac / eg + * h$

$\Rightarrow * * f ^ {^{\wedge \wedge}} \underline{db} / \underline{ac} + \underline{eg} h$

#### • Prefix to postfix conversion:

• In order to convert from prefix to postfix we need

- 1) operators followed by
- 2) two immediate operands.

then, convert it into postfix.

ex: prefix:  $* + cd - ba$

postfix:  $* \underline{cd} + \underline{ba} -$

$\times \underline{cd} + \underline{ba} -$

$\curvearrowleft \underline{cd} + \underline{ba} - *$

Ex: prefix:  $* ^ / + c a e * d b ^ f g$

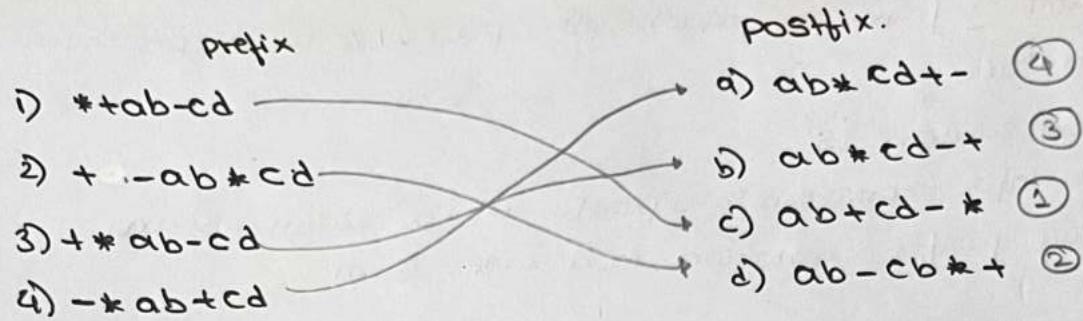
postfix:  $* ^ / \underline{c a} \underline{e} \underline{d} \underline{b} * f g ^$

$\Rightarrow * ^ \underline{cate} / \underline{db} * f g ^$

$\Rightarrow * \underline{cate} \underline{db} * ^ f g ^$

$\Rightarrow cate / db * ^ f g ^ *$

Q] match the following



→ Reverse Polish notation  
Postfix to infix conversion [we use bracket]

- to convert from postfix to infix
- 1) two operands followed by
- 2) immediate operator.

then convert it into infix expression.

postfix: ab\*c d |-

$$\begin{aligned}\text{Infix: } & \underline{(a * b)} \underline{c} \underline{d} \mid - \\ \Rightarrow & \underline{(a * b)} \underline{(c \mid d)} \mid - \\ \Rightarrow & ((a * b) - (c \mid d))\end{aligned}$$

prefix to infix conversion

- 1) operator followed by

- 2) two immediate operands

then convert the prefix into infix expression

prefix: \*+ad-cd

$$\begin{aligned}\Rightarrow & \underline{*} \underline{(a+d)} \underline{(c-d)} \\ \Rightarrow & ((a+d) * (c-d))\end{aligned}$$

### Note

In order to evaluate Infix expression we need to scan multiple time jumping from one place to other place. A/c to the precedence of the operator.

The best solution is

- 1) Convert Infix expression into postfix in a single scan.
- 2) Evaluating postfix expression in a single scan

### Convert Infix to postfix:

(using operator stack):

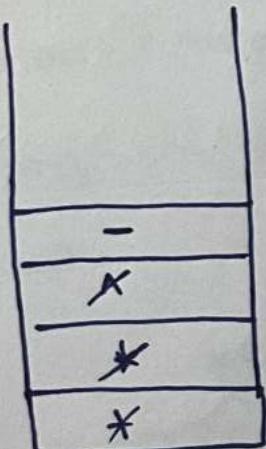
Top of stack	Next operators	operator
Low	high	push
high	low	pop
Same	Same(L → R)	pop
Same	Same(R → L)	push

If any braces come treat it as new stack until the closing braces

Ex: Infix:-  $a + b * c ^ d - e$

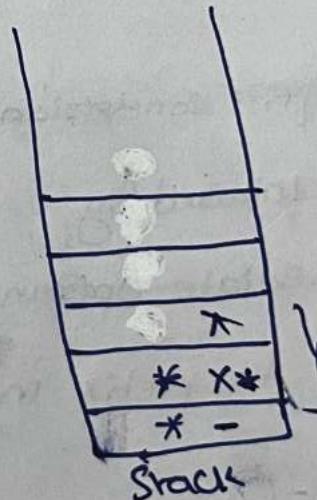
Postfix:  $abcd^*+e-$

Infix:  $a + b * c - d / e ^ f * g + h$   
postfix:  $ab + cd^* + ef ^ / gh +$



Stack

max stack size used  
(M.S.U)  
 $\geq 3$

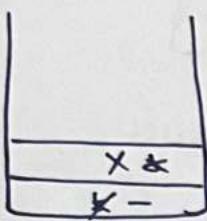


max stack size used  
 $\geq 3$

Ex: when we will reach symbol "f" while something during conversion from infix to postfix what will be the content of stack

Infix string:  $a * b - c / e * f + g$

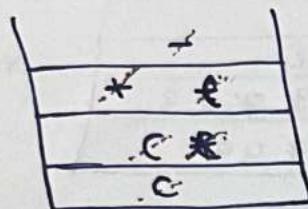
postfix:  $ab * ce / f$



Ex: Infix  $((a+b) * (c-d))$

postfix:

$ab + cd - *$

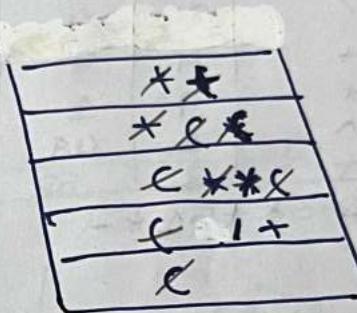


M.S.U = 4 //

Ex: Infix:

$((P+Q) * (R+S)) / T) + (A * (B+C))$

Postfix:  $PQ + RS + * T / ABC + *$



M.S.U = 5.

## ▪ postfix evaluation: [using operand stack]

procedure:

1) if (symbol == operand)  
    push (symbol);

2) if (symbol == operator)

    a)  $OP_2 = \text{pop}();$

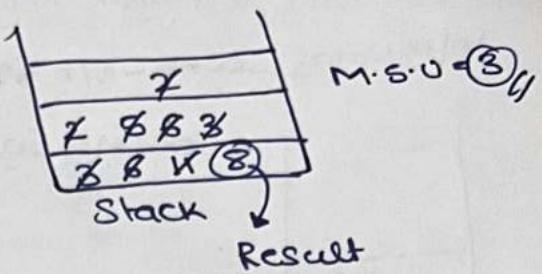
    b)  $OP_1 = \text{pop}();$

    c)  $\text{push}(OP_1 < \text{operator} > OP_2);$

3) At the end, Result will be in the stack.

Postfix:  $32 * 5 + 62 / -$

OP <sub>1</sub>	operator	OP <sub>2</sub>	Result
3	*	2	6
6	+	5	11
6	/	2	3
11	-	3	(8) //



M.S.U = 3 //

Postfix:  $623+-382/++2^3+ : 52$

OP <sub>1</sub>	operator	OP <sub>2</sub>	Result
3	+	3	5
5	-	5	1
8	/	2	4
3	+	4	7
1	*	7	1
7	<sup>2</sup>	2	49
49	+	2	52

4
3 8 4
2 8 3 7 2 3
8 4 7 4 9

M.S.U = 4

0] 8 2 3 ^ / 2 3 \* + 5 1 \* -

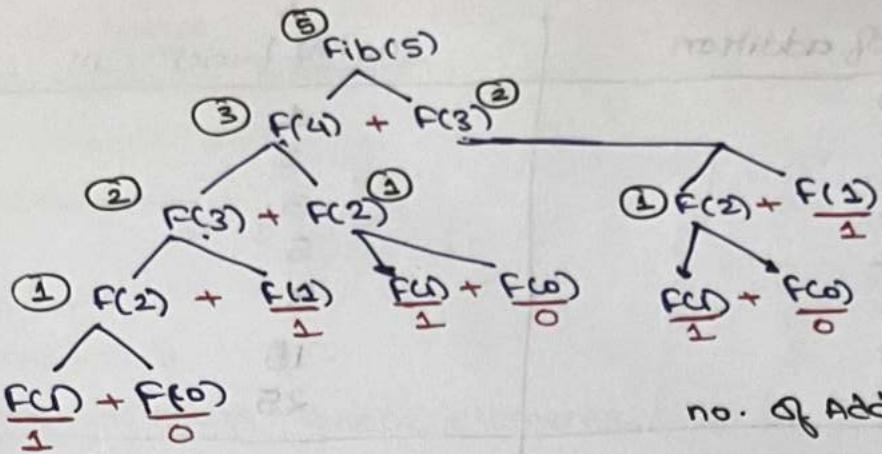
$\Rightarrow 2$   
4 M.S.U = 3 //

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

### Fibonacci Series.

n	0	1	2	3	4	5	6	7	8	9	10
Fib(n)	0	1	1	2	3	5	8	13	21	34	55

$$\text{Fib}(n) = \begin{cases} n, & \text{if } n=0 \text{ or } n=1 \\ \text{Fib}(n-1) + \text{Fib}(n-2), & \text{otherwise} \end{cases}$$

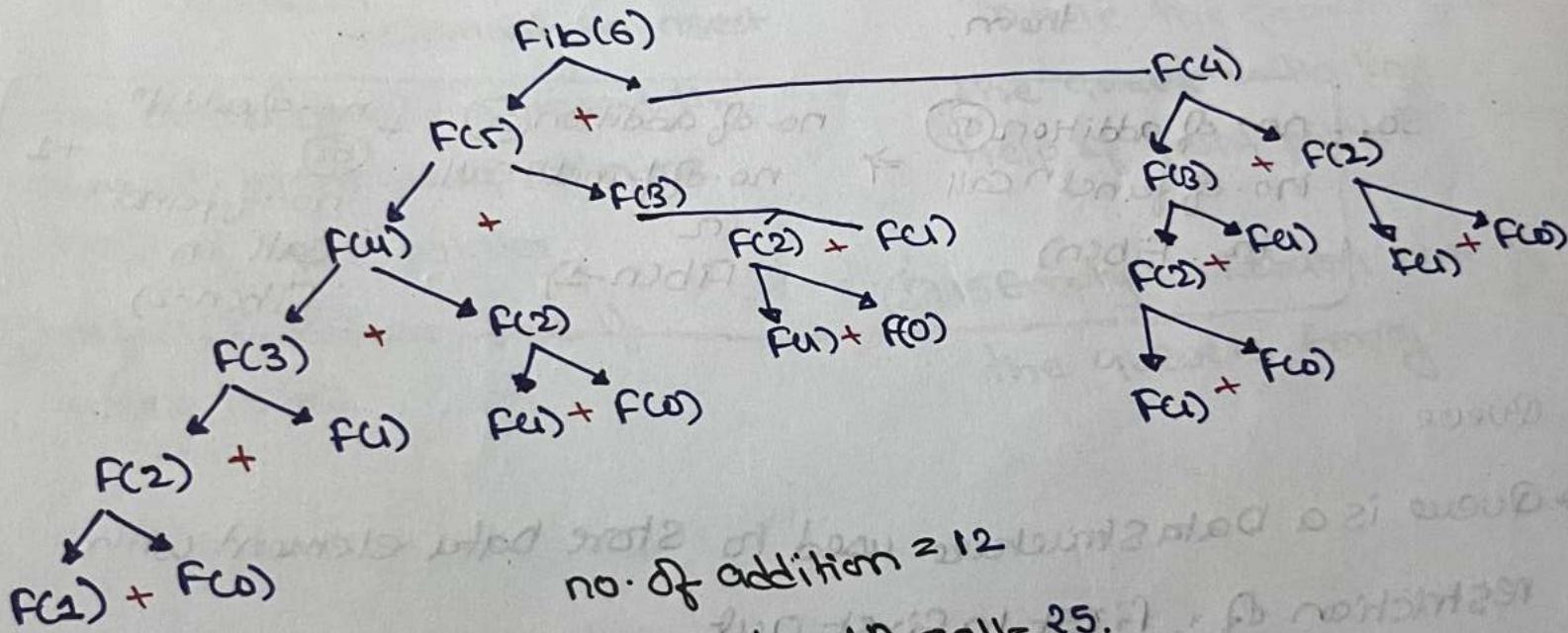


no. of addition = 7

no. of functn call = 15

### Note

- functn calling is preorder
- functn evaluation is postorder



no. of addition = 12

no. of functn call = 25.

Fib(n)	no. of addition	no. of functn call.
0	0	1
1	0	1
2	1	3
3	2	5
4	4	9
5	7	15
6	12	25

general "formal"

$$F(n) = F(n-1) + F(n-2)$$

↓  
no. of addition  
required for individual  
+ 1 extra for combining  
them.

so, no. of addition  $\textcircled{6}$   $\Rightarrow$  no. of addition  $\textcircled{6}$  + no. of addition  $\textcircled{6}$   
 no. of functn call in Fib(n)  $\Rightarrow$  no. of functn call in Fib(n-1) + no. of functn call in Fib(n-2) + 1

## QUEUE

- Queue is a Data Structure used to store data element with restriction of first in first out.
- the elements are inserted from one end & deleted from another end
- it uses ~~two~~ pointers
  - 1) Front → used for deleting an element.
  - 2) Rear → used for inserting an element.

- Application of Queue.
  1. Printer Spooler Daemon
  2. Ticket Reservation.
  3. Job scheduling.

- Queue A.D.T :-

D Declaration of data:

- a) Space to store Queue elements.
  - Array
  - Linked list

- b) two pointers front & rear.

- 2) Declaration of operation:

- a) enqueue(x)

↳ element to insert  
the above operation insert the  
element in the queue with the  
help of rear pointer.

- c) isfull(): Return true if  
queue is full.

- ⑥ dequeue()

Delete the element from  
the queue with the  
help of front pointer.

- d) isEmpty(): Return true if  
the queue is empty

## Implementing Queue:

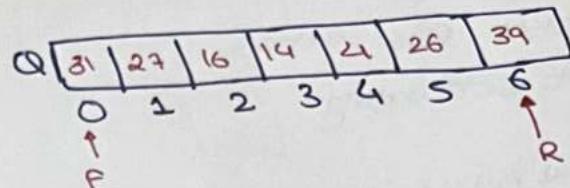
① using array:

```
int Q[N], N=7
```

Initially  
Front = -1  
rear = -1

Overflow condition:  
 $R == (N-1)$

Underflow condition  
 $F == -1$



enqueue:

$F = -1$   
 $R = -1, 0, 1, 2, 3, 4, 5, 6$

$R++;$   
 $Q[R] = x$

WAP to enqueue

- ① Initially check for overflow condition
- ② if it is not overflow & if it is 1st insertion then increment both front & rear otherwise increment only rear pointer & then insert the element in the queue using rear pointer.

Queue  
 enqueue(Q, N, F, R, x);  
 ↑ size front rear  
 element to insert.

```
if( $R == (N-1)$ ) Pf("overflow condition"); return;
```

```
if( $R == -1$ ) {  

    F++; R++; } } 1st insertion.
```

else

$R++;$

Q[R] = x;

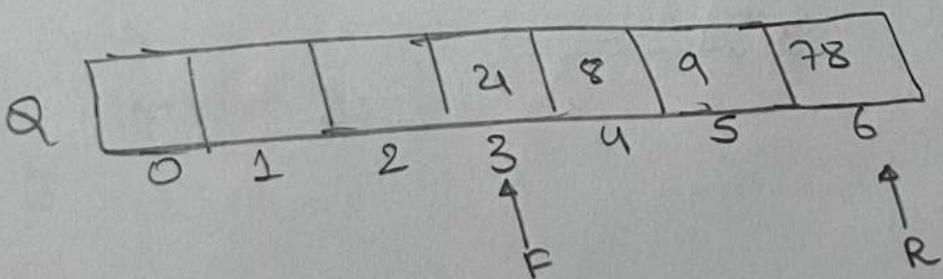
y

• WAP to dequeue.

```
int dequeue(Q, N, F, R) {
    if (F == -1) {
        pf("underflow condition");
        return;
    }
    int y = Q[F];
    if (F == R) {
        F = -1;
        R = -1;
    } else {
        F++;
    }
    y;
    return y;
}
```

**note**

- in the above implementation of linear queue, when the rear pointer reaches to right most pointer & front pointer is some where in the middle then still it is showing overflow condition.
- even though queue is showing some empty slot still they are not efficiently utilized.
- to avoid these problem we will go for circular queue.



int Q[N] // N=7

$$F = -1, \emptyset, 2, 4, 3$$

$$R = -1, \emptyset, 2, 2, 3, 4, 5, 6$$

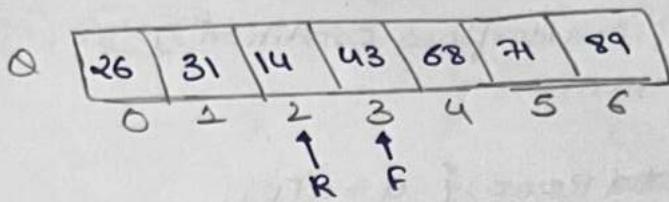
• Circular Queue

int Q[N], N=7

Initially  
F=-1  
R=-1

$$R = (R+1) \% N$$

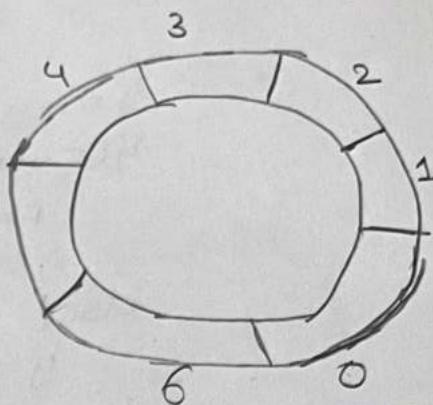
$$F = (F+1) \% N$$



enqueue:

$$F = -1, 0, 1, 2, 3$$

$$R = -1, 0, 1, 2, 3, 4, 5, 6, 7, 8$$



$$(R+1) \% N == F$$

$$(6+1) \% 7 == 3$$

$$0 == 3$$

(F)

$$(R+1) \% N == F$$

$$(1+1) \% 7 == 3$$

$$2 == 3$$

(F)

$$(R+1) \% N == F$$

$$(0+1) \% 7 == 3$$

$$1 == 3$$

(F)

$$(R+1) \% N == F$$

$$(2+1) \% 7 == 3$$

$$3 == 3$$

(T)

Overflow.

overflow condition

$$(R+\Delta) \% N == P$$

underflow condition

$$F = -1$$

• WAP to perform enqueue operation in circular queue.

```
int enqueueCircular(Q, N, f, R, x) {
    if ((R+1)%N == f) printf("overflow condition"); return;
    if (R == -1) {
        f++;
        R++;
    } else {
        R = (R+1)%N;
    }
    Q[R] = x;
}
```

• WAP to perform dequeue operation in circular queue.

```
int dequeueCircular(Q, N, f, R) {
    if (f == -1) printf("underflow condition"); return;
    int y = 0; y = Q[f];
    if (f == R) {
        y = Q[f];
        f = -1, R = -1;
    } else {
        f = (f+1)%N;
    }
    return y;
}
```

Q1 Consider the following program which uses 3 data structures.  
Queue(Q1), Queue(Q2), & stack(St).

- the program takes 'Q1' value as input & perform some operation

$$Q_1 = 13, 71, 6, -12, 9, 0, 4, 15, 12, 8, -16, 0, 18, 13$$

xyz(Queue Q1)?

```
int s = 0;
while (!isEmpty(Q1)) {
    x = dequeue(Q1);
    if (x == 0) {
        st = pop(st);
        enqueue(Q2, s);
        s = 0;
    }
    push(st, x);
}
```

1) what is sum of all value of stack after completion of  
above program. 148

IS
4
9
-12
6
71
13
St

Q2
9 -16

2) what is m.s.v = -12

3) what is sum of all value of Q2 = -7

4) what is max Q2 size used = 2.

Q<sub>1</sub> = 31, 7, 16, 22, 9, 0, 45, 15, -29, 8, 6, 0, 8, 13

ABC (Queue Q<sub>1</sub>)?

```
while(! isEmpty(Q1)) {
    x = dequeue(Q1);
    if (x == 0) {
        pop(st);
        pop(st);
        while (! isEmpty(st)) enqueue(Q2, pop(st));
    }
    y
    push(st, x);
}
```

y  
3.

Q<sub>2</sub> [16 | 7 | 31 | -29 | 15 | 45 | 0]

6
9 8
22 -29
16 15 13
7 45 8
31 0 0

① what is sum of all value of stack after completion of above program : 21

② what is M.S.U : 6

③ what is sum of all value of Q<sub>2</sub> ? 85

④ what is max Q<sub>2</sub> size used : 7.

Ex:

$$Q_1 = 21, 19, -19, 8, 2, 0, 14, 51, 12, 8, -16, 0, 18, 3$$

ABC (Queue Q<sub>1</sub>) {

```
    static int s;  
    while (!isEmpty(Q1)) {  
        x = dequeue(Q1);  
        if (x == 0) {  
            while (!isEmpty(st))  
                st = pop(st);  
            st = s;  
            enqueue(Q2, st);  
        }  
        push(st, x);  
    }
```

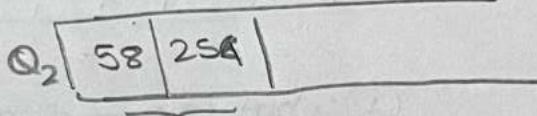
Static  
S

Q

2  
10  
-19  
9  
29  
58  
42  
90  
62  
113  
127  
25A

-16	
2	8
8	12
-14	51
14	18
10	0

st



312

1. what is sum of all value of stack

(2) //

2. what is M.S.U

(6) //

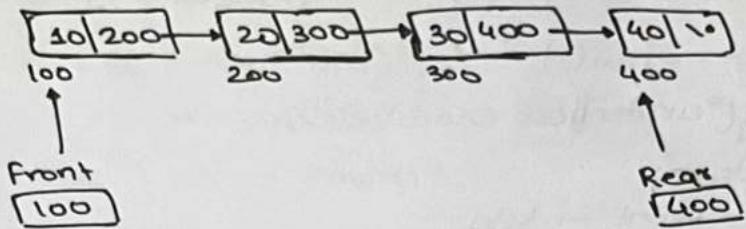
3. what is sum of all values of Q<sub>2</sub>

(312) //

4. what is max. Q<sub>2</sub> size used.

(2) //

• Using linked list :



Initially

- $\text{Front} = \text{null}$
- $\text{Rear} = \text{null}$

• WAP to perform enqueue operation using linked list.

```
enqueueLL(node * front, node * rear, int x) {
```

```
    node * curr = (node *) malloc(sizeof(int));
```

```
    if (curr == null) { printf("Overflow condition"); return; }
```

```
    curr->data = x;
```

```
    curr->next = null;
```

```
    if (rear == null) {
```

```
        Front = curr; Rear = curr;
```

```
y
```

```
else {
```

```
    Rear->next = curr;
```

```
    Rear = curr;
```

```
y
```

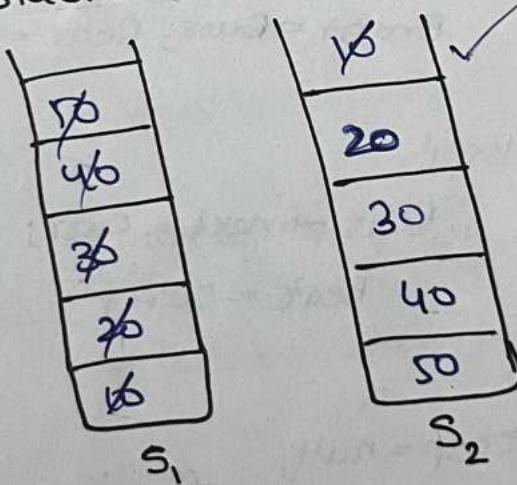
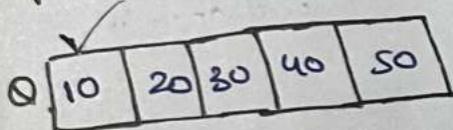
```
temp = null;
```

```
y
```

• WAP to perform dequeue operation

```
int dequeueLL(node * front, node * rear) { node * p = front;
    if (front == null) {
        cout("underflow condition");
        return;
    } int y = front->data;
    if (front == rear) {
        Front = null; Rear = null;
    }
    else {
        front = front->next;
    }
    free(p);
    p = null;
    return y;
}
```

• Implementing Queue using stacks:



• WAP to perform enqueue & dequeue operation.

int dequeue(S<sub>1</sub>, S<sub>2</sub>) {

    if (S<sub>2</sub> is NOT Empty) {

        return pop(S<sub>2</sub>);

    else if (S<sub>1</sub> is Empty) {

        Pf("Queue is Empty");

        return -1;

    }

    else {

        while (S<sub>1</sub> is not Empty) {

            x = pop(S<sub>1</sub>);

            push(S<sub>2</sub>, x);

        }

        return pop(S<sub>2</sub>);

    }

}

enqueue(S<sub>1</sub>, x);

push(S<sub>1</sub>, x);

y

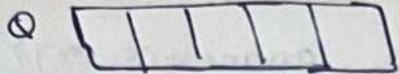
~~Ques~~ Consider the above pseudo code to implement queue using stack.  
Let 'n' insert & m ( $\leq n$ ) delete operation be performed in an arbitrary order on an empty Queue Q. Let 'x' & 'y' be the no. of push & pop operation performed respectively.  
which one of the following is true for all 'm' & 'n'?

(a)  $n+m \leq x \leq 2n$  &  $2m \leq y \leq n+m$ .

(b)  $n+m \leq x \leq 2n$  &  $2m \leq y \leq 2n$

(c)  $2m \leq x \leq 2n$  &  $2m \leq y \leq n+m$

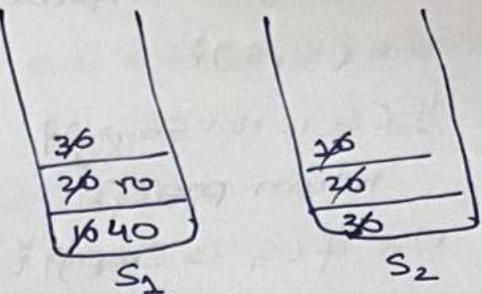
(d)  $2m \leq x \leq 2n$  &  $2m \leq y \leq 2n$ .



$n=5$   
 $m=3$

$$\max = \text{push} = x = 5+5 = 10$$

$\frac{x}{2n}$



$\text{push}=5$   
 $\text{pop}=3$

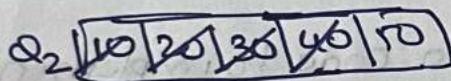
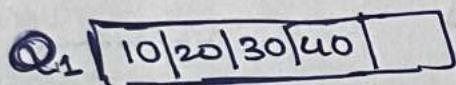
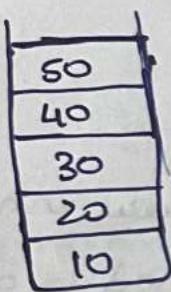
$\text{push}=3$   
 $\text{pop}=3$

$$\max = \text{pop} = y = 5+3 = 8 = n+m$$

$$\min = \text{push} = x = 5+3 = 8 = n+m$$

$$\min = \text{pop} = y = 3+3 = 6 = 2m.$$

### • Implementing stack using queues:



Stack

$$\min \geq 0 \geq m.s. + m.s \geq m.s$$

$$m.s \geq m.s + m.s \geq m.s$$

$$m.s \geq m.s + m.s \geq m.s$$

$$m.s \geq m.s + m.s \geq m.s$$

- write psuedo code for push & pop operatn.

```

int POP(Q1, Q2) {
    if (Q1 is empty) {
        if (Q2 is empty) {
            pf("stack is empty");
            return -1;
        }
    }

```

```

    else {
        while (Q2 does not contain one element) {

```

```

            x = dequeue(Q2);
            enqueue(Q1, x);

```

```

        }
    }

```

```

    else {
        while (Q does not contain one element) {

```

```

            x = dequeue(Q);
            enqueue(Q2, x);

```

```

        }
    }

```

```

    return dequeue(Q);
}

```

```

}

```

```

}

```

```

push(Q1, Q2, x) {

```

```

    if (Q1 is Empty)

```

```

        enqueue(Q2, x);

```

```

    else

```

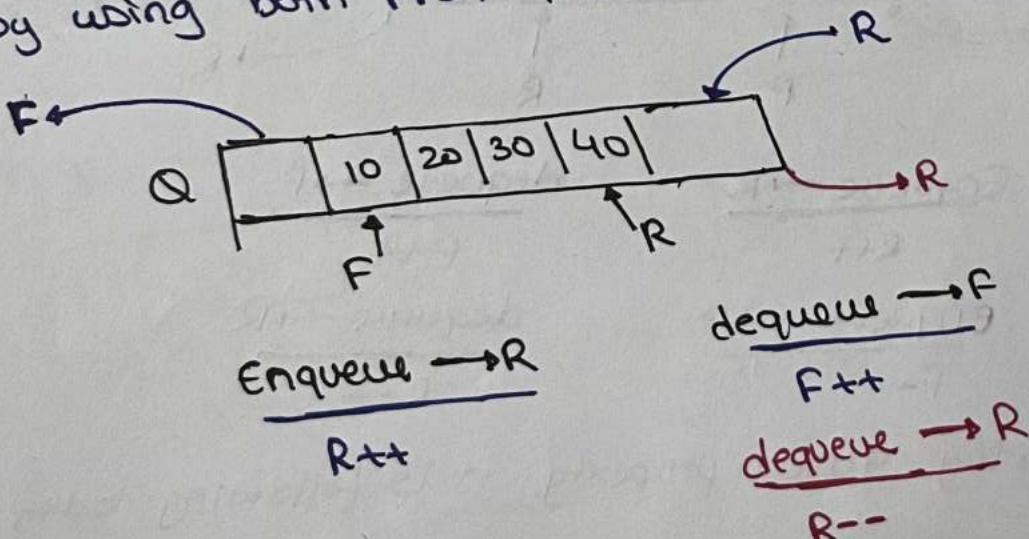
```

        enqueue(Q1, x);
    }
}

```

### • Input Restricted Queue:

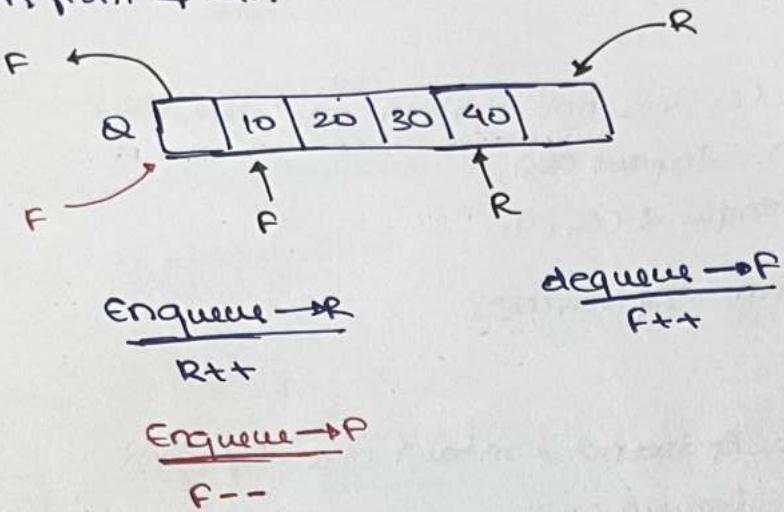
- enqueue operatn are restricted but the dequeue operatn can be done by using both front & rear.



• It is not following queue property. It is following both FIFO & LIFO  
(Last In First Out).

• Q/p Restricted Queue.

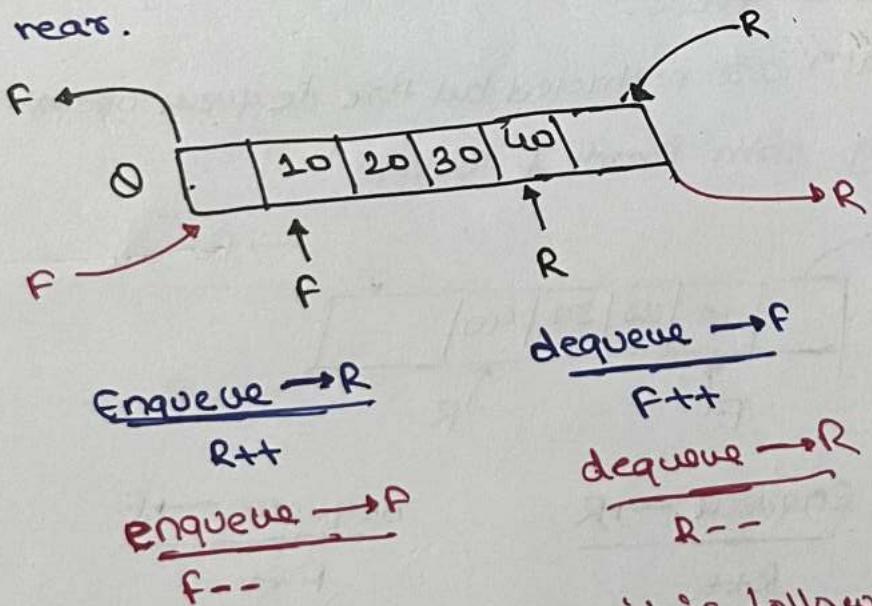
• Dequeue operation is restricted but enqueue operation can be done by using both front & rear.



• It is not following queue property, it is following both First In First Out & Last In First Out.

• Double Ended Queue.

• Enqueue & dequeue operation can be done by using both front & rear.



• It is not following queue property, it is following both FIFO & LIFO

• priority Queue:

element: 85, 30, 16, 10, 25, 70, 90, 45.

1) Ascending priority Queue (delete min)

10, 16, 25, 30, 45, 70, 85, 90

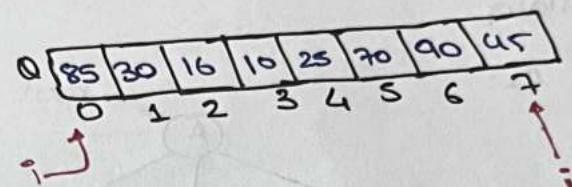
2) Descending priority Queue (delete max)

90, 85, 70, 45, 30, 25, 16, 10

• Implementing Ascending priority Queue (delete min):

element : 85, 30, 16, 10, 25, 70, 90, 45

int Q[N]; N=8



Initially.

i = -1

j = -1

enqueue:

i = -1 0  
j = -1 0 1 2 3 4 5 6 7

i++  
Q[j] = x

'j' will always point to last element.

Dequeue:

1. Scan the entire array & find the position of minimum element

2. y = Q[min] // 10

3. Q[min] = Q[j]; j

4. j--;

5. return y;

T.C =  $\Theta(n)$

B/C of unsorted array

enqueue  
T.C =  $O(n)$

using sorted array then  
dequeue  
T.C =  $O(1)$

## Tree

- Tree is non-linear data structure used to store the data element by following (parent-child) hierarchical relationship b/w the data element

### Type of tree

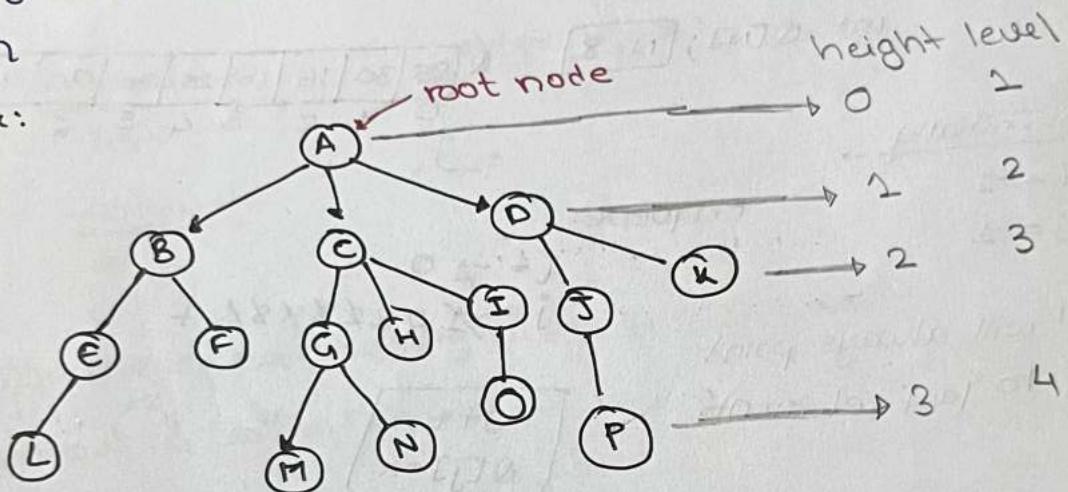
- n-ary tree
- Binary tree.

### Degree of tree:

the degree of the tree represent the max. no. of child node : any node

can contain

Ex:



1) Leaf node: L, F, M, N, H, O, P, K

2) Internal node: A, B, E, C, G, I, D, J

3) Ancestors of 'P': J, D, A

height of the tree: 3

level of the tree: [height + 1] 4

• siblings of "B": C, D

• left most child of 'C': G

• Right sibling of "J": No

• Right sibling of 'E': F

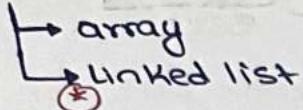
• Right sibling of 'A': No

Height of the tree.

the longest path from root node to any leaf

node. (root node height is generally considered as "0").

• Representation of tree:



• to find max. element in binary tree  
T.C.: O(n)

• Binary tree

• A tree in which every node can contain atmost 2 child is known as Binary tree.

① Struct BTnode

```
Struct BTnode *left;  
int data;  
Struct BTnode *right;
```

y;

typedef Struct BTnode node

② node a = {NULL, 10, NULL};

node b = {NULL, 20, NULL};

③ a.left = &b;

a.right = &c;

b.left = &d;

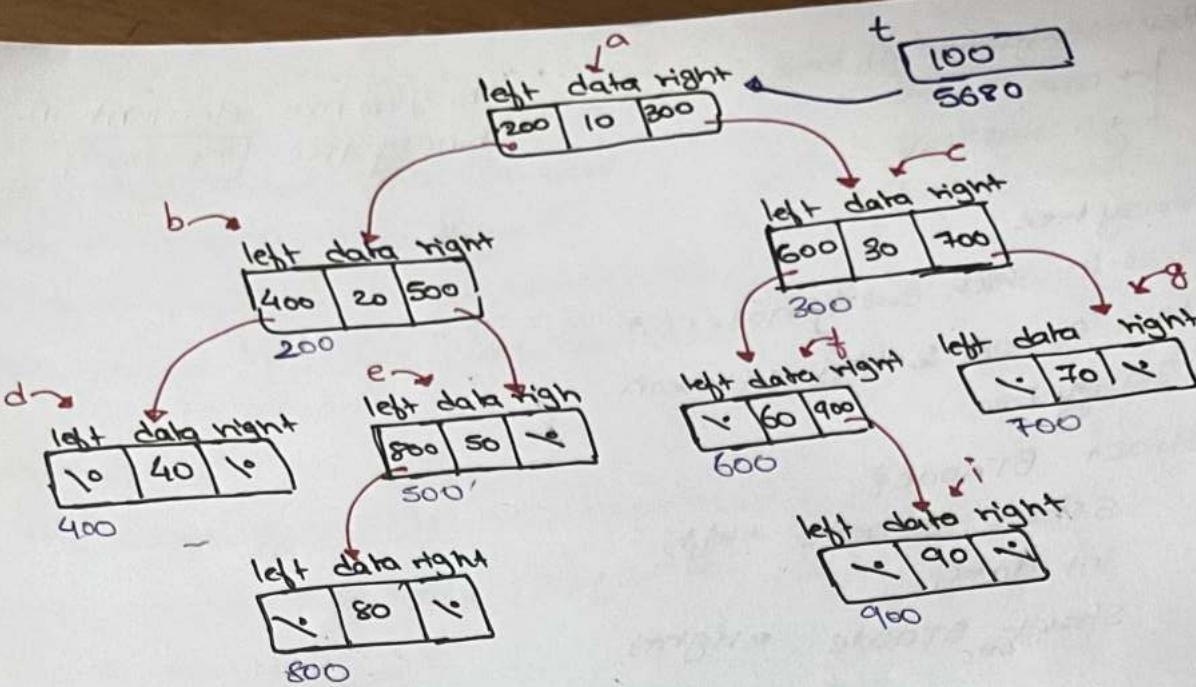
b.right = &e;

;

!

④ node \*t = null

t = &a;



**note**

- $Pf(t) = 100$

- if ( $t == \text{null}$ )

↳ if above condition is true then it means that Binary tree is empty

- $(t \rightarrow \text{left})$  is given

↳ it means that left subtree is given

- $(t \rightarrow \text{right})$  is given

↳ it means that right subtree is given

- $Pf(t \rightarrow \text{left} \rightarrow \text{right} \rightarrow \text{data}) ; // 50 //$

► in the tree program will be done using recursion b/c every node is having some behaviours.

- $Pf(t \rightarrow \text{data}) ; // 10 //$

- $Pf(t \rightarrow \text{left}) ; // 200 //$

- if ( $t \rightarrow \text{left} == \text{null}$  &  $t \rightarrow \text{right}$ ) ;

↳ if above condition is true that means Binary tree has only one node & that node is known as leaf node.

- $(t)$  is given

↳ it means that entire tree is given

Ex: Write a program to find no. of leaf node in the given Binary tree.

```
int NL(node *t) { if(t == null) return 0;
    if(t->left == null & t->right == null) return 1;
    return (NL(t->left) + NL(t->right)); }
```

Ex: WAP to find height of the Binary tree.

```
int HL(node *t) {
    if(t == null) return 0;
    return max(HL(t->left), HL(t->right)) + 1;
}
```

Ex: WAP to find wheather Binary tree is a strict Binary tree.

```
int StrictBT (node *t) {
    if(t == null) return -1;
    if((t->left != null & t->right == null) || (t->right == null & t->left == null))
        return 0;
    if(t->left == null & t->right == null) return 1;
    return (Strict(t->left) && Strict(t->right));
}
```

3

- WAP to find no. of internal node in the binary tree.

```

int NI(node *t)?
    if(t == null) return 0;
    if(t->left == null && t->right == null) return 0;
    return (1 + NI(t->left) + NI(t->right));

```

y.

- WAP to find total no. of node in the given binary tree

```

int Totalnode(node *t)?
    if(t == null) return 0;
    return (Totalnode(t->left) + Totalnode(t->right) + 1);

```

y

to make these good program  
you should add  
these line

i. if(t->left == null & t->right == null) return 1;

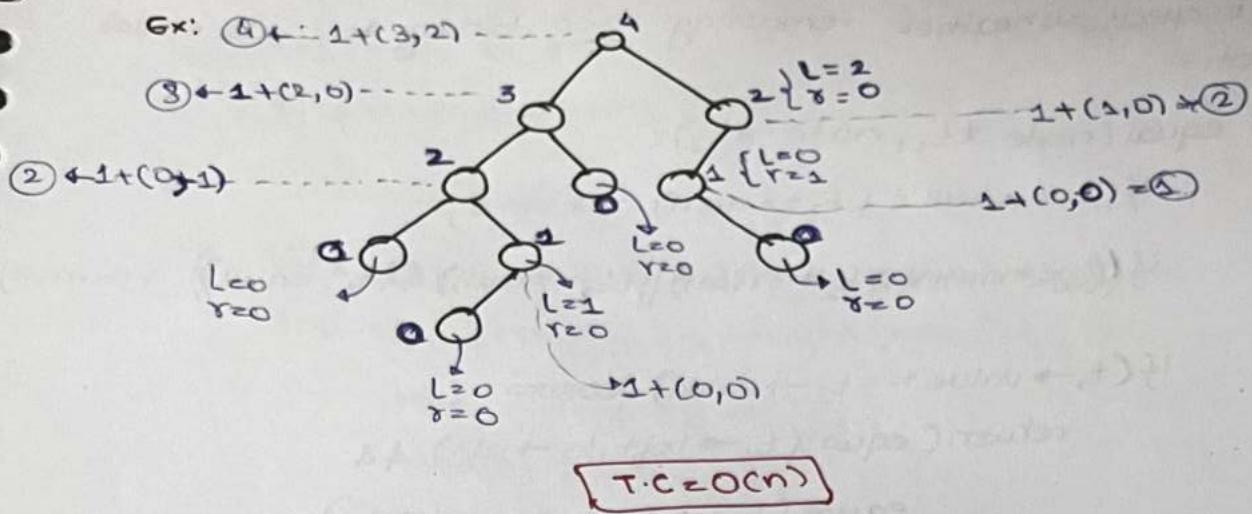
- WAP to find height without using max funtn.

```

int height(node *t)?
    if(t == null) return 0;
    if(t->left == null && t->right == null) return 0;
    int L = height(t->left);
    int R = height(t->right);
    return (1 + (L > R ? L : R));

```

y



- WAP to find level in BT.

```

int levels(node *t) {
    if (t == null) return 0;
    if (t->left == null && t->right == null) return 1;
    else {
        int l = levels(t->left);
        int r = levels(t->right);
        return (1 + (l > r ? l : r));
    }
}

```

$T.C = O(n)$

- WAP to check whether remaining two binary tree are equal or not.

```

int equal(node *t1, node *t2) {
    if (t1 == null && t2 == null) return 1;
    if ((t1 == null && t2 != null) || (t2 == null && t1 != null)) return 0;

    if (t1->value == t2->value)
        return (equal(t1->left, t2->left) &&
                equal(t1->right, t2->right));
}

```

3

### • Strict Binary tree (S.B.T)

- a tree in which every node has exactly zero or two child node is called as Strict Binary tree

- WAP to convert the given binary tree into its mirror image.   
Swapping data.

```

void MI(node *t) {
    if (t == null) return;
    if (t->left == null && t->right == null) return;

    if (t->left == null && t->right != null) {
        int *temp = t->left->data;
        t->left->data = t->right->data;
        t->right->data = temp;
    }
}

```

4

return (MI(t->left) && MI(t->right));

5

```
void MI(node *t) {
    if (t == null) return;
```

∴ Swapping node  
not data..

```
if (t → left == null & t → right == null) return;
```

```
if (t → left != null & t → right != null) {
```

```
    node *temp = t → left;
```

```
    t → left = t → right;
```

```
    t → right = temp;
```

y

```
MI(t → left);
```

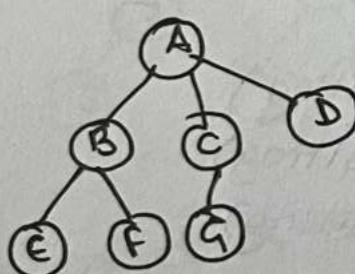
```
MI(t → right);
```

y

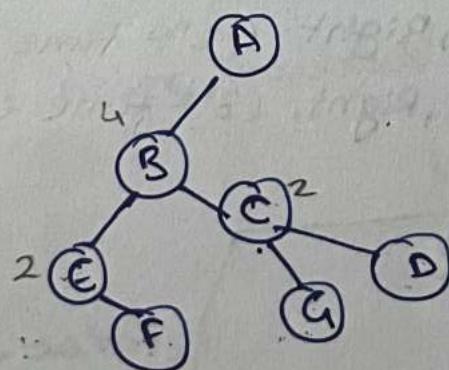
Pg: 151

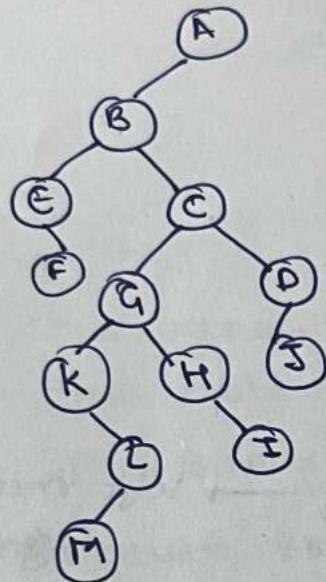
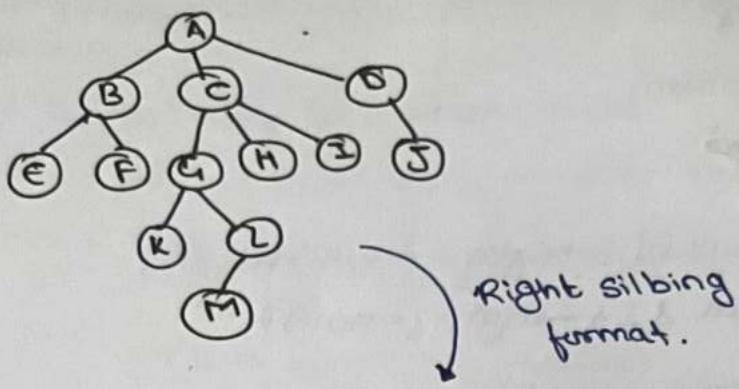
Q: 12

[Arbitrary tree → Binary tree → left most child, Right sibling.]



Right sibling  
Format





## Tree traversals.

postorder: left, Right, Root (1<sup>st</sup> time visiting)

inorder: left, Root, Right (2<sup>nd</sup> time visiting)

preorder: Root, left, Right. (3<sup>rd</sup> time visiting)

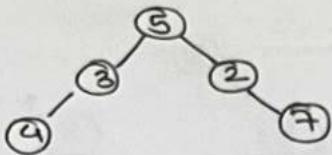
Preorder (node \*t) {

```

if(t) {
    pf(t->data);
    Preorder(t->left);
    Preorder(t->right);
}
  
```

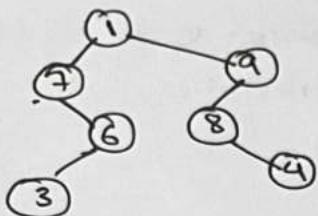
}

Ex:



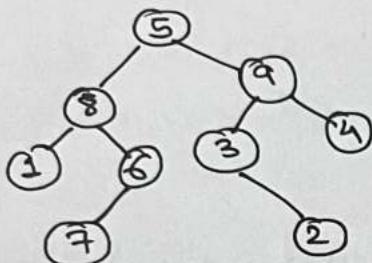
Preorder: 5, 3, 4, 2, 7  
Inorder: 4, 3, 5, 2, 7.  
Postorder: 4, 3, 7, 2, 5.

Ex:



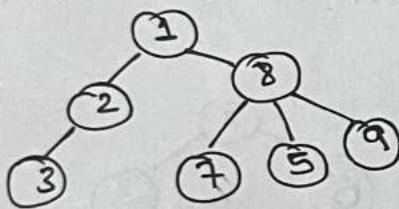
Preorder: 1, 7, 6, 3, 9, 8, 4  
Inorder: 7, 3, 6, 1, 8, 4, 9  
Postorder: 3, 6, 7, 4, 8, 9, 1

Ex:



Preorder: 5, 8, 1, 6, 7, 9, 3, 2, 4  
Inorder: 1, 8, 7, 6, 5, 3, 2, 9, 4  
Postorder: 1, 7, 6, 8, 2, 3, 4, 9, 5

Ex:

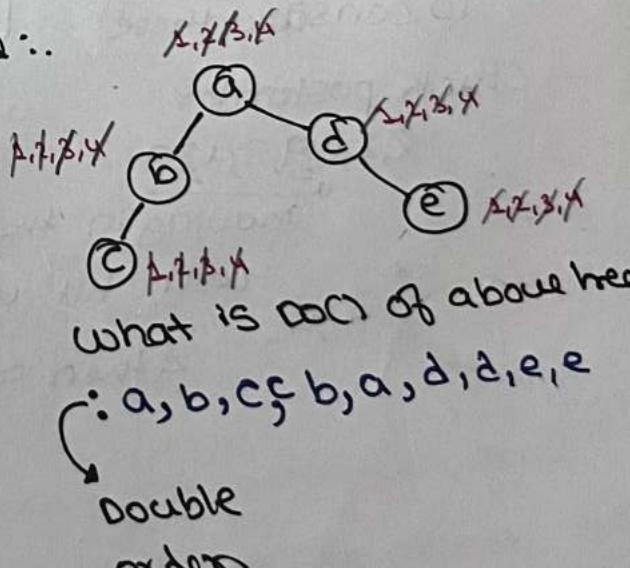


Inorder: 3, 2, 1, 7, 8, 5, 9

\* Double order & Triple order Traversal ..

```
Do(node *t)
    if(t)
        1. pf(t->data);
        2. DO(t->left);
        3. pf(t->data);
        4. DO(t->right);
    }
```

y



• Indirected Recursion on trees:-

A (node \*t) ?

if(t) ?

1. Pf(t → data);
2. B(t → left);
3. B(t → right);

↳  
↳

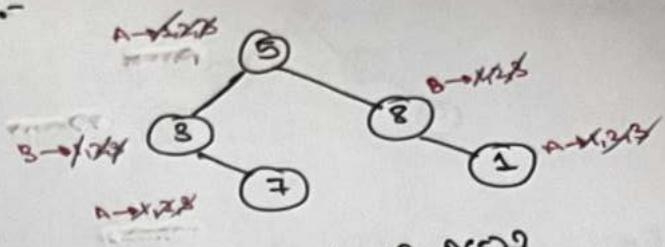
B(node \*t) ?

if(t) ?

1. A(t → left);
2. Pf(t → data);
3. A(t → right);

↳  
↳

↳



what is OLP of A(5)?

5, 3, 7, 8, 1

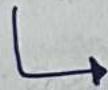
• Constructing unique Binary tree

► A unique Binary tree can be construct if any two traversal are given But one of them should be Inorder.

Ex: Postorder: 8, 2, 6, 9, 5, 4, 3, 7

Inorder: 2, 8, 6, 7, 9, 5, 3, 4.

To construct these



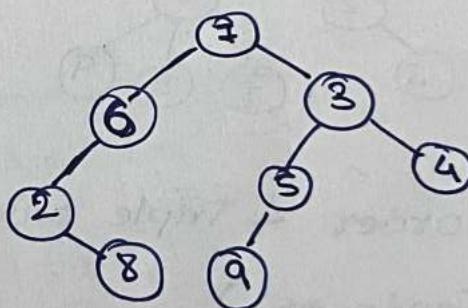
Check postorder

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

moving in these direction

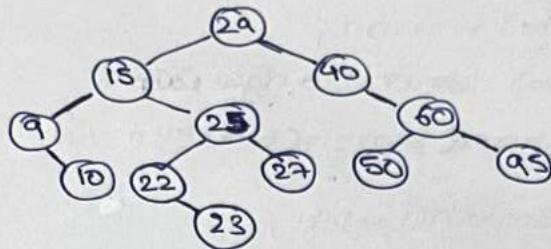
will tell us root node

then compare with inorder.



Date:

Post order: 10, 9, 23, 22, 27, 15, 50, 45, 60, 40, 29.  
Inorder: 9, 10, 15, 22, 23, 25, 27, 29, 40, 50, 60, 45

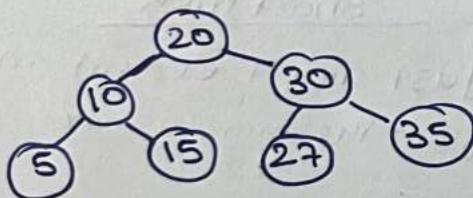


- Binary search tree (B.S.T.):

- Binary Search tree is also a Binary tree But with the restriction that all the element in the left subtree should be smaller to the parent & all the elements on the right should be greater than parent. (these rule is applicable for all node).
- BST is used for searching an element.
- All the element in the BST should be unique.

Ex:-

Element : 20, 10, 15, 30, 35, 27, 5.



Preorder : 20, 10, 5, 15, 30, 27, 35

Inorder : 5, 10, 15, 20, 27, 30, 35

postorder : 5, 15, 10, 27, 35, 30, 20.

**Note**

- Inorder traversal of BST is a sorted order.
- only Inorder traversal of the BST is given, can we construct unique binary tree? No.

- only postorder traversal of BST is given. can we construct unique binary tree? Yes.

- last element in the preorder & last element in the inorder is same (this is true only for complete binary tree).

\* WAP to Search an element in an BST.

\* search

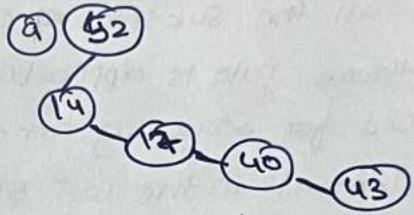
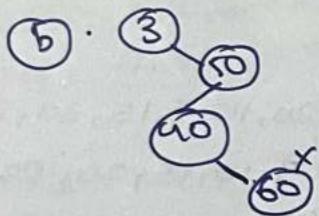
```
Node (node *t, int x)?  
if(t == null) return;  
if(x == t->data) return t;  
if(x > t->data)  
    return(search(t->right, x))  
else return search(t->left, x);
```

y

Ex: Consider the no. (1 to 100) in BST

when searching for element 43, which of the following sequence can't be the sequence of node examined?

- a) 52, 14, 12, 40, 43
- b) 3, 50, 40, 60, 43
- c) 10, 65, 31, 48, 37, 43
- d) 81, 61, 52, 14, 41, 43



Search for 55

- a) 10, 25, 64, 43, 60, 57, 55
- b) 90, 12, 68, 34, 62, 45, 55
- c) 9, 85, 47, 68, 43, 57, 55
- d) 79, 14, 72, 56, 16, 53, 55

Smaller ---

90 12 68 34 62 45 55  
  \---  
  greater ---

Note

WC: O(n)

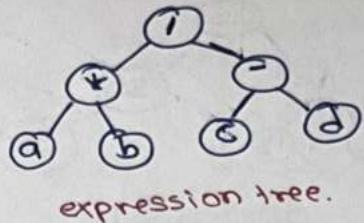
↳ left skewed  
or Right skewed.  
BST

- Short trick
- (a) just insert element in a sequence from the min to max.
  - (b) check for adjacent if it is lower than the current then all following should be lower than that or if adjacent is greater than all following should be greater.

10 43 57 55 60 64 75 } no issue

- Construct unique binary tree of given reverse polish notation.

postfix:  $ab * cd - /$   
 convert it  
 Infix:  $(a * b)(c - d) /$   
 $((a * b) / (c - d))$



Ques:

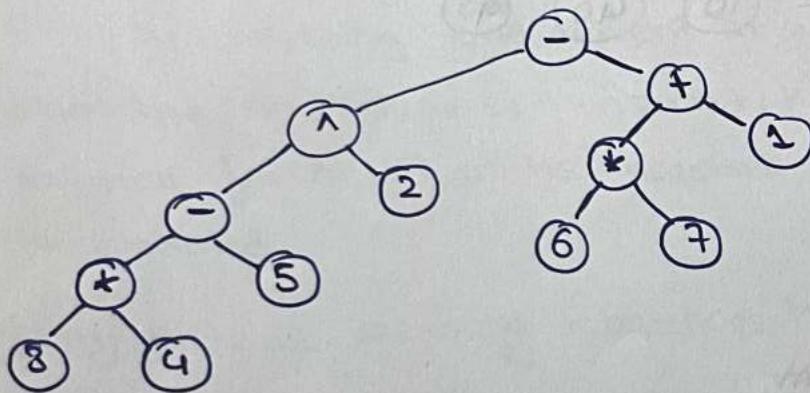
postfix:  $- 3 4 * 5 - 2 ^ 6 7 * 1 + -$

Infix:  $(3 * 4) 5 - 2 ^ (6 * 7) 1 + -$

$((3 * 4) - 5) 2 ^ (6 * 7) 1 + -$

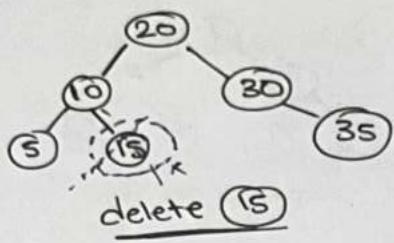
$((3 * 4) - 5)^2 (6 * 7) 1 + -$

$\neq ((3 * 4) - 5)^2 - (6 * 7) 1 + -$



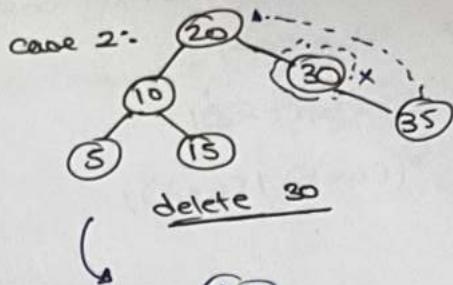
### \* Deletion in the BST

Case 1:

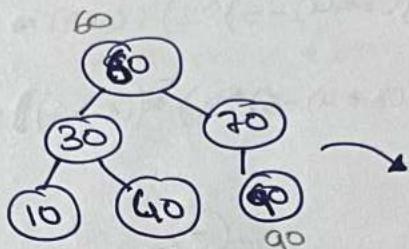
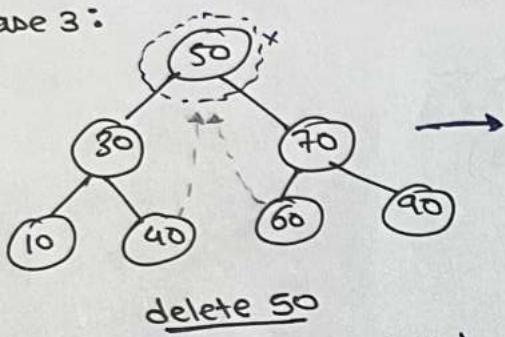


- if it is a leaf node then simple delete that node & update its parent pointer as null.

Case 2:



- child of 30 will be connected to parent of 30



- new root will be updated with inorder predecessor (largest element in the left subtree) or inorder Successor (smallest element in the Right subtree).

Inorder:

10 30 40 **50** 60 70 90

*inorder*

*successor*

(smallest successor)

{ left child will  
be always null }

{ Right child will be  
always null }

35  
null

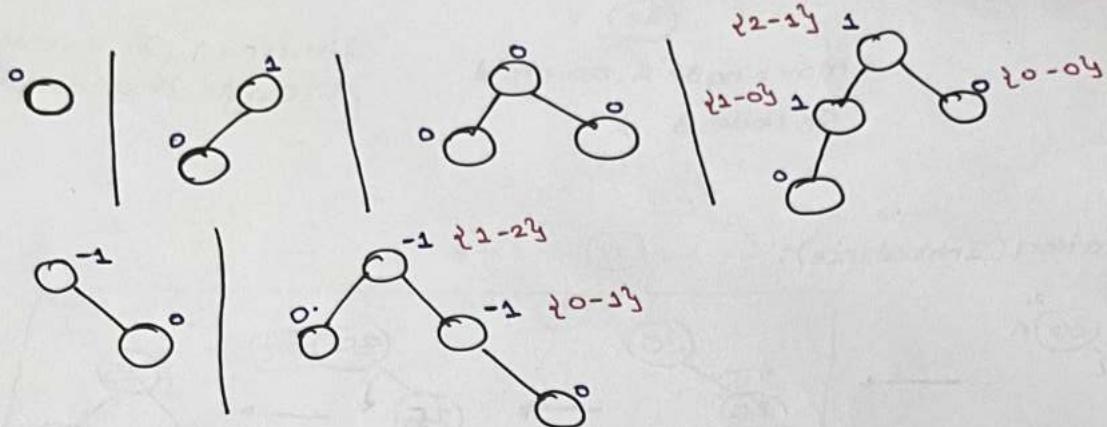
*Inorder  
predecessor  
(largest element  
l.s.f.)*

- AVL Tree's (Height Balanced B.S.T):-

(Adelson, Valsky, Landis).

$$[\text{Balanced Factor} = \text{height of L.S.T} - \text{height of R.S.T}]$$

∴ Accepted  
Balance factors  
= 0, -1, 1



- Construction of AVL Tree's

the procedure is same as constructing BST by inserting one element at a time

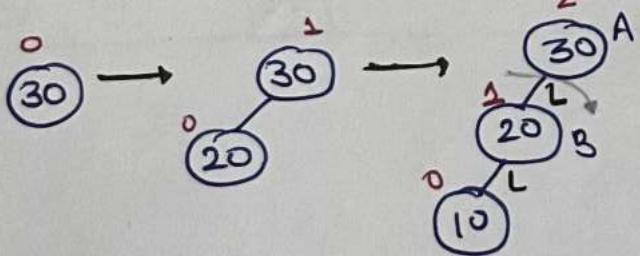
when the new node is inserted, the Balance factor of all its ancestor's will be updated.

If any node is becoming imbalance's then we need to apply proper rotation technique to make it balance

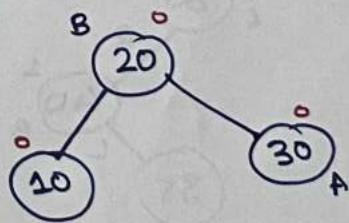
- Rotation technique:

- LL Imbalance:

{Left, Left}

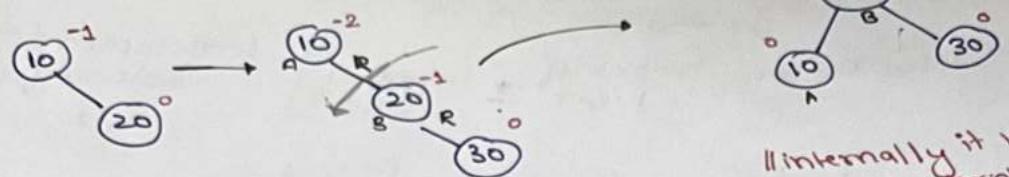


[Make node 'A' as child of node



|| Internally it involve single Rotation.

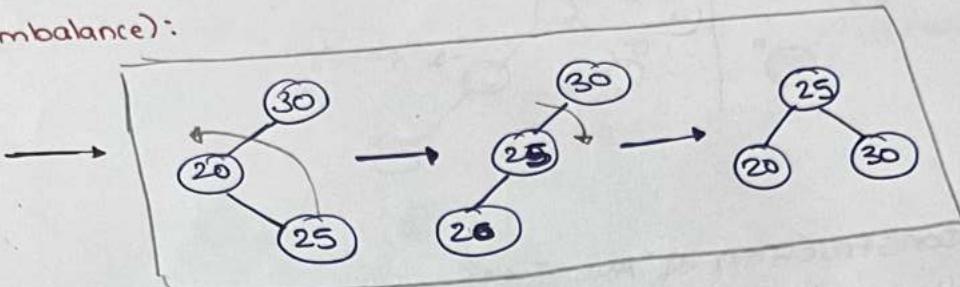
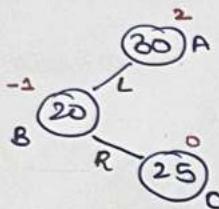
• RR Imbalance:



// make node A as child  
of node B

// internally it involve  
single rotation.

• LR Rotation (Imbalance):

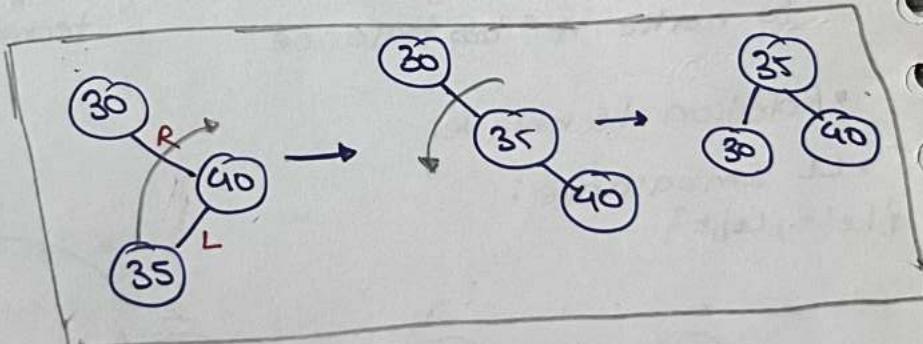
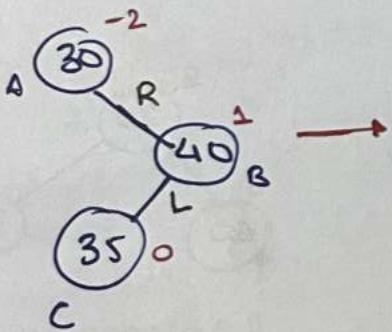


// Make node A & B as a  
child of node C.

// internally it involve  
double rotation.

{ go back towards newly  
insert node if there is an  
imbalance}

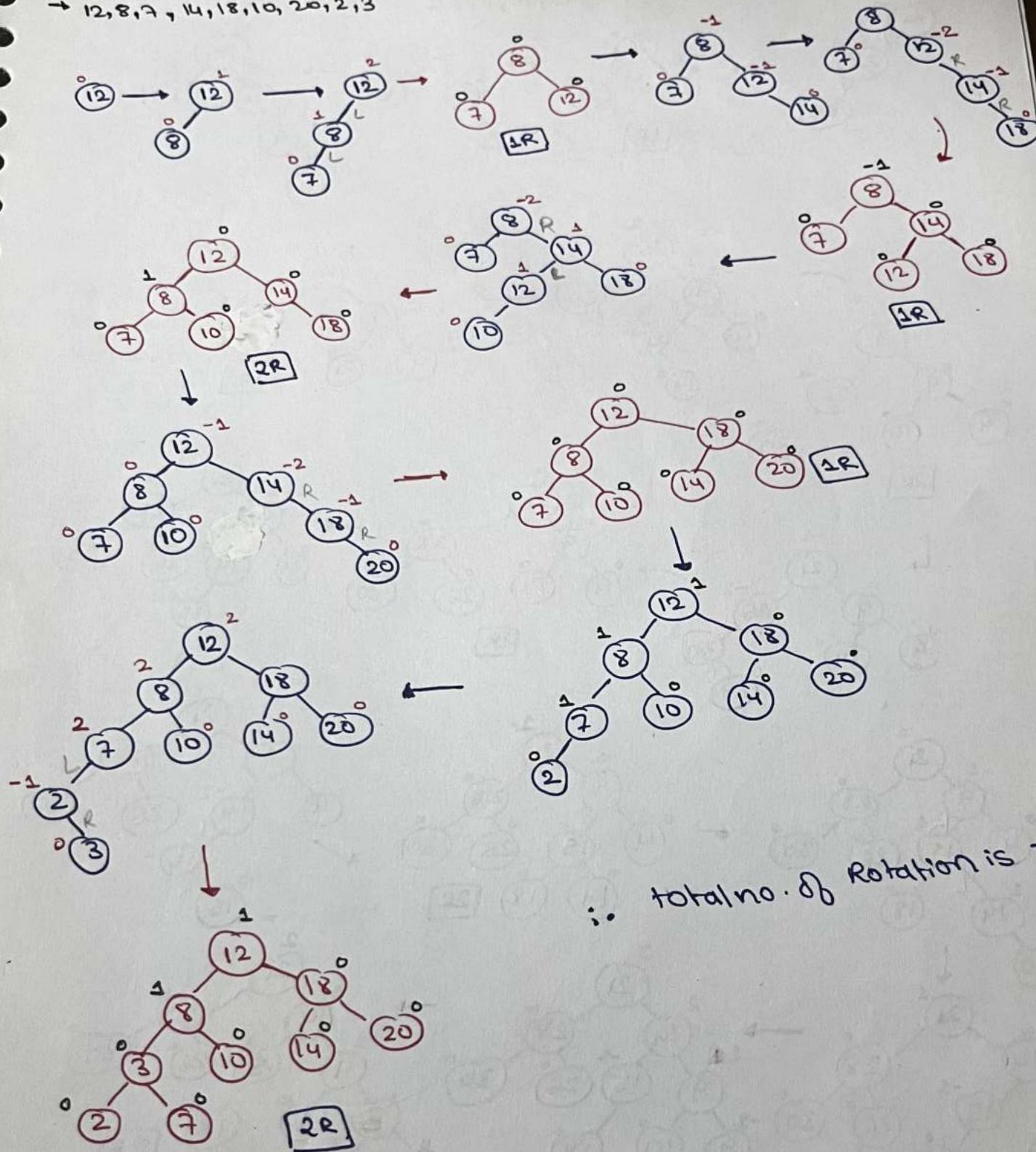
• RL Imbalance:



// make node A & B as a  
child of node C.

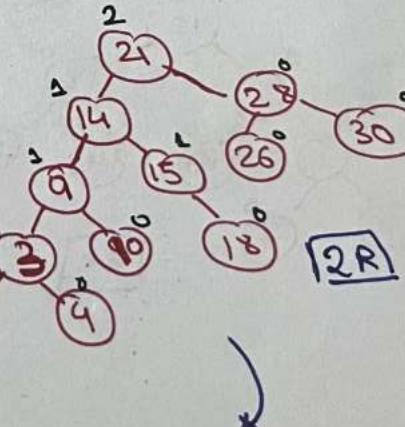
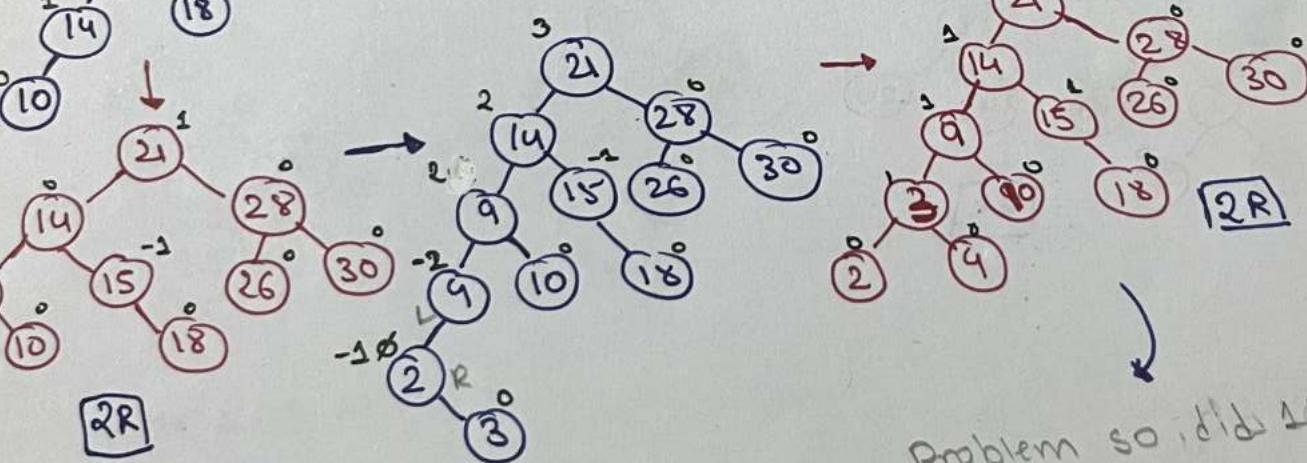
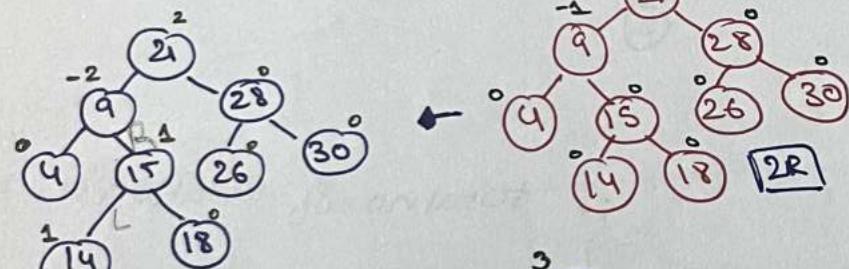
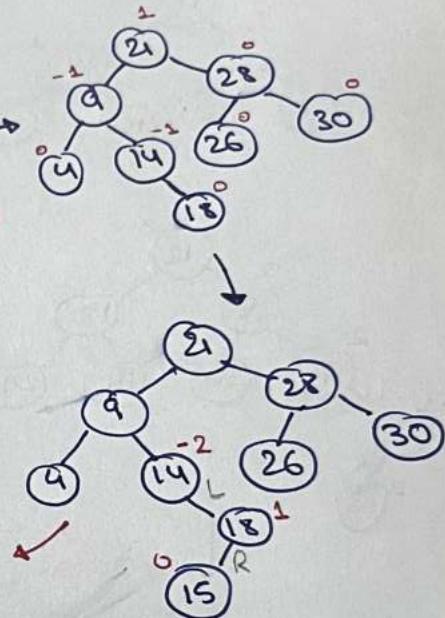
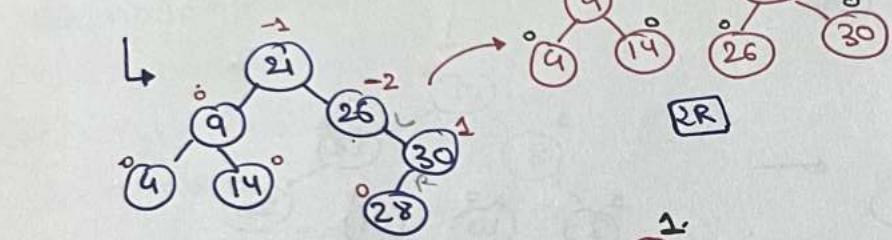
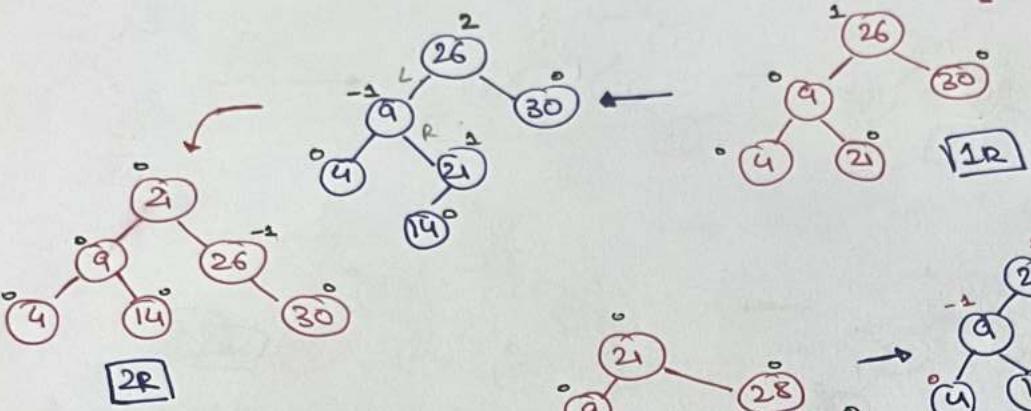
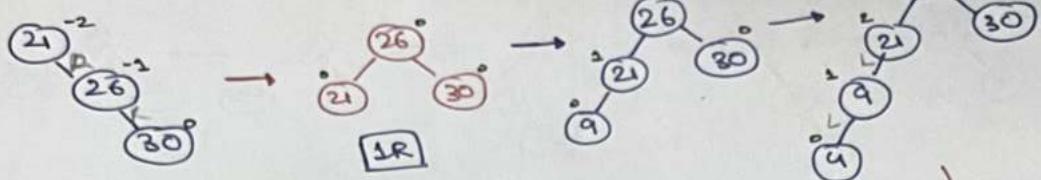
// internally it involve  
double rotation.

- Construct the AVL tree with given node  
→ 12, 8, 7, 14, 18, 10, 20, 2, 3

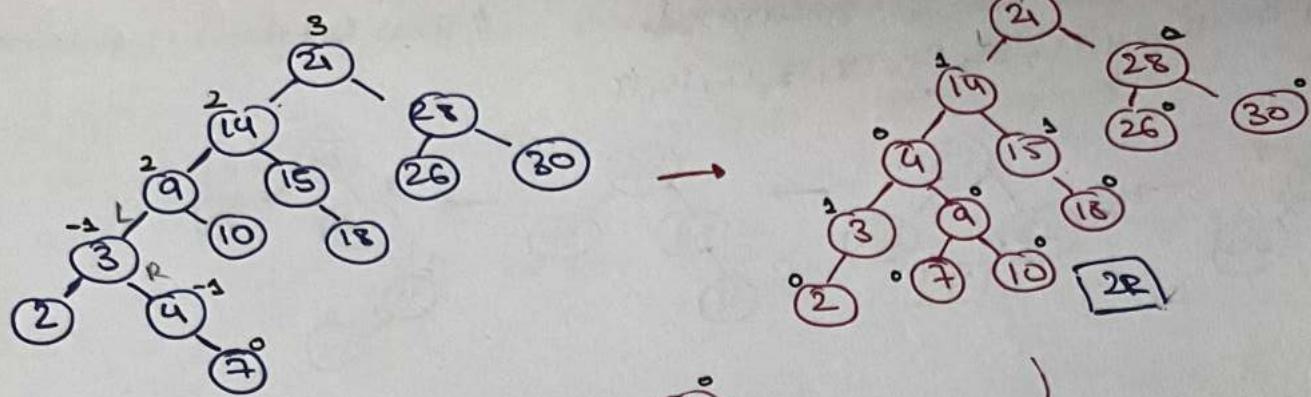


∴ total no. of rotation is 7R.

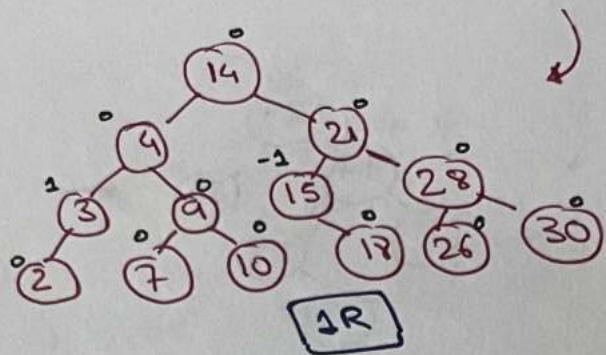
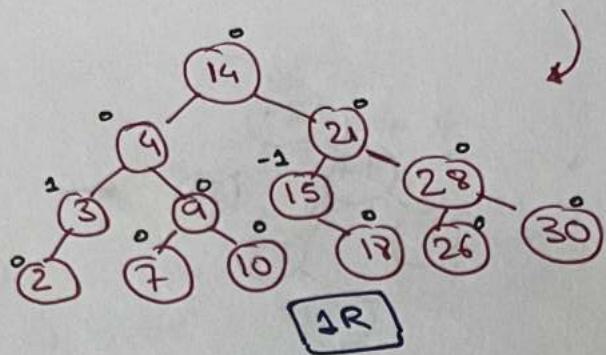
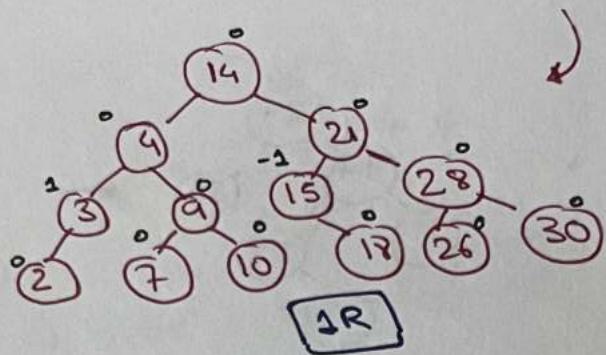
- Construct a AVL tree from given node
- 21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7.



Problem so did LR rotation in last.

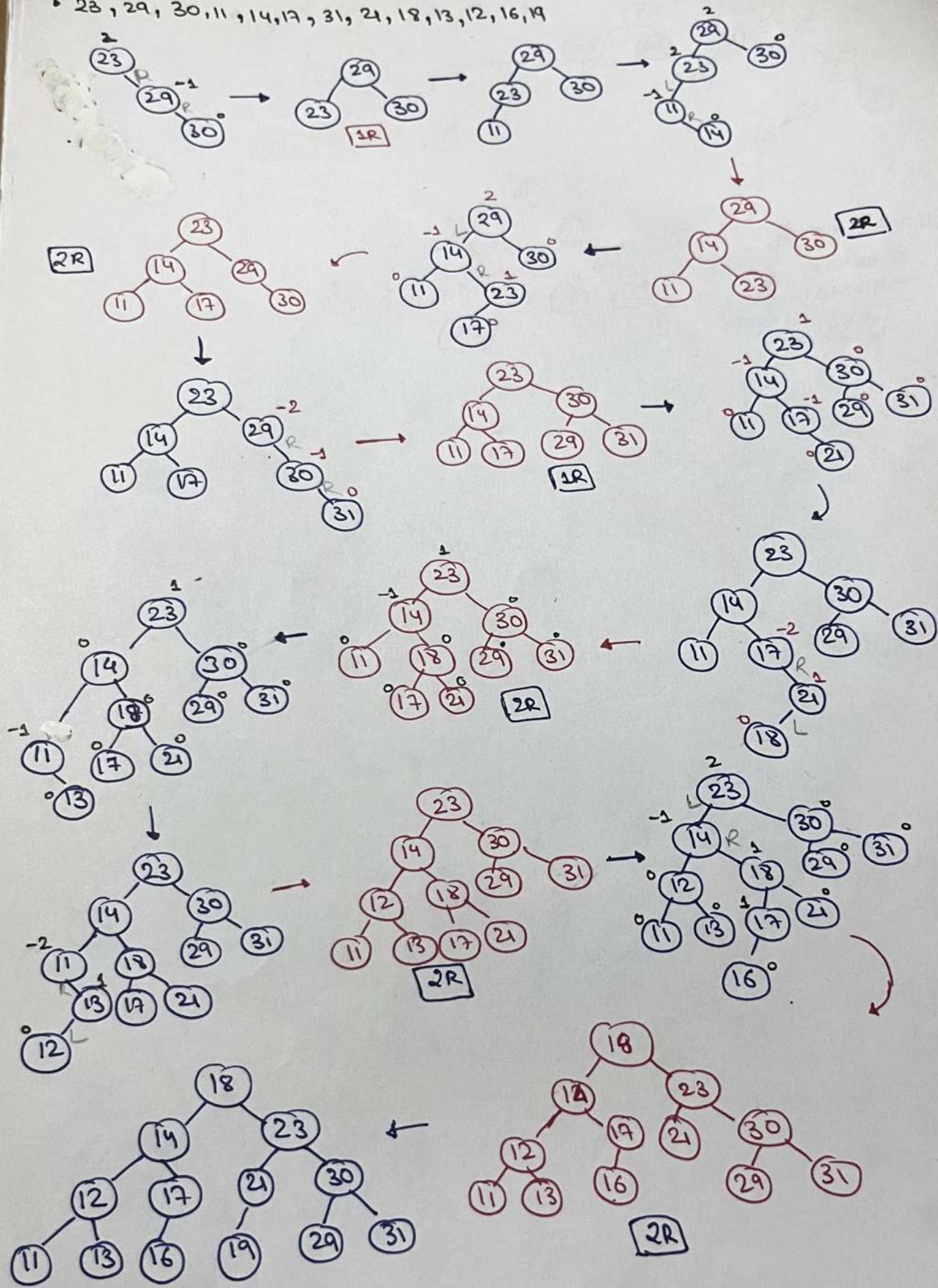


II Total no. of  
rotation  
required:  
15R



- Construct the AVL tree for given nodes
  - 28, 29, 30, 11, 14, 17, 31, 21, 18, 13, 12, 16, 19

// Total Rotation: 12 Rotation.



Note: max. no. of nodes in the binary tree (AVL tree also) with height 'h'

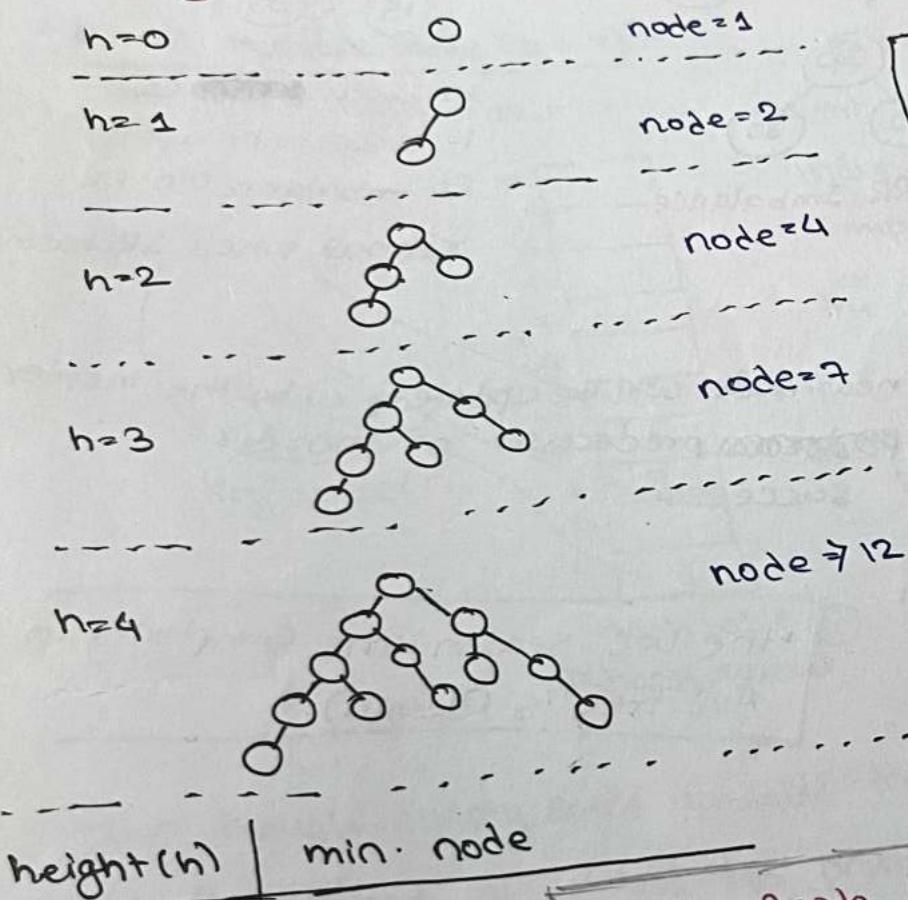
$$2^{h+1} - 1$$

$$h=0 \rightarrow 0 = 2^{0+1} - 1 \Rightarrow 2 - 1 \\ = 1$$

$$h=1 \rightarrow \begin{array}{c} \circ \\ | \\ \circ \end{array} = 2^{1+1} - 1 = 2^2 - 1 \\ = 3$$

$$h=2 \rightarrow \begin{array}{c} \circ \\ | \\ \circ \end{array} \begin{array}{c} \circ \\ | \\ \circ \end{array} = 2^{2+1} - 1 \Rightarrow 2^3 - 1 \\ = 7$$

\* Finding minimum no. of node in AVL tree with height 'h':



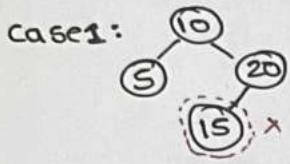
Formula  $\Rightarrow$

$$\min(h) \Rightarrow \min(h-1) + \min(h-2) + 1$$

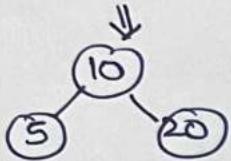
height( $h$ )	min. node
0	1
1	2
2	4
3	7
4	12
5	20
6	33
7	54
8	88
9	143
10	232

min. no. of node  
in AVL tree  
with height( $h$ ) = min. no. of  
node with height( $h-1$ ) + min. no. of  
node with height( $h-2$ ) + 1

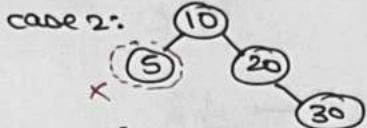
- Deletions in AVL Tree:
- Deletion in the AVL tree is same as BST but after deletion check the balance factor of every node. if any node is having imbalance then apply the proper rotation & make it balance.



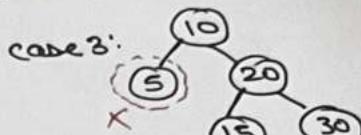
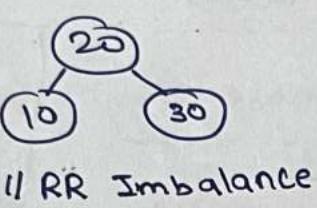
delete 15



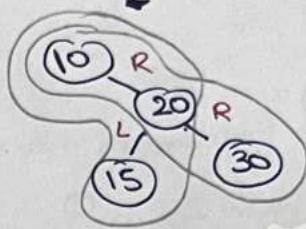
no balancing required



delete 5

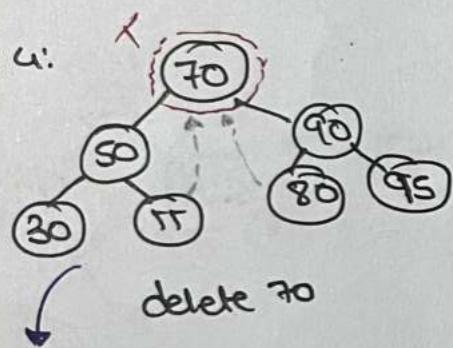


delete 5.

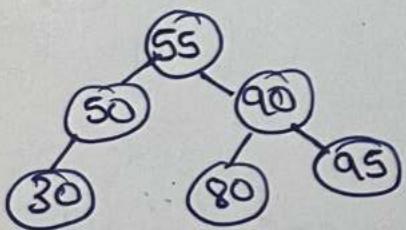


in above case  
it is better to apply  
RR imbalance b/c RR  
internal takes 1 rotation

Case 4:



delete 70



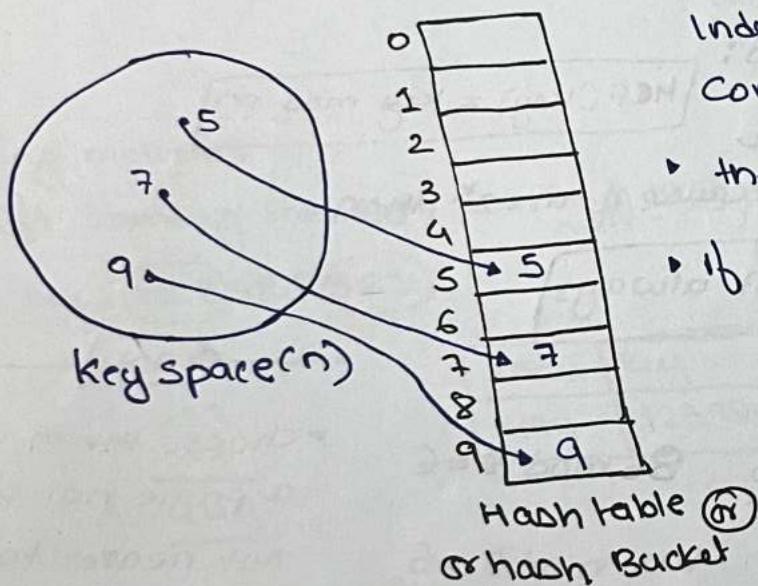
new node will be updated, with the inorder  
successor or predecessor

the WC search time Complexity in  
AVL tree is  $O(\log n)$ .

- Hashing:
  - It's a technique used for searching an element
  - Hashing is mathematical computation over a key.
  - The goal of a hashing is to design the hash function which give min. no. of collisions
  - W.C Time complexity  $O(1)$  {99%}
  - The following component are used in hashing
    - Key Space ( $n$ )
    - Hash functn
    - Hash table ( $m$ )
    - Hash Bucket ( $m$ ).

#### Direct Address table (D.A.T) :-

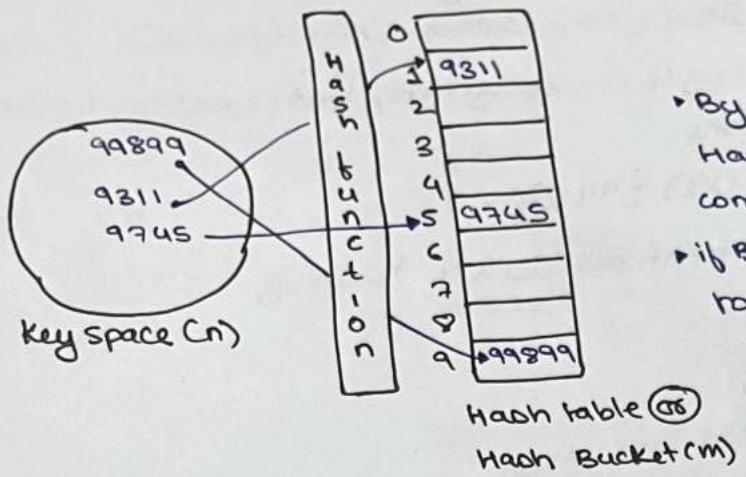
Hash table ( $m$ ) = 10



- In D.A.T Key will be directly stored in the Hash table in the respective index without any mathematical computation.
- The W.C Time complexity is  $O(1)$
- If one key have larger size i.e.  $\text{Key} = 2^{1000}, 9999$   
then we need to maintain the large size Hash table.

- Even though empty space available in the Hash table, it's not efficiently utilized, to avoid this problem., the concept of Hash functn is introduced.

• hash function.



$$H \cdot F(\text{key}) = \text{key mod } m$$

$$m=10(0 \dots 9)$$

- By using the concept of Hash function we can comfortably store large key.
- If both the key are mapped to the same index then it is called as collision.

► To handle collision, various Collision resolution technique are used.

■ Various Hash function:

① Division modulo method:

$$H \cdot F(\text{key}) = \text{key mod } m$$

► Do not choose 'm' value as exactly power of 2, because if  $m=2^k$ , then

$$H \cdot F(\text{key}) = \text{LSB } 'k' \text{ bit always}$$

$$\text{Ex: } m=8=2^3 \neq 2^k, k=3$$

$$1) \text{ Key} = 86 \quad 0\ 1\ 0\ 1\ 0\ \underline{1\ 0\ 0\ 0}$$

$$86 \bmod 8 = 6$$

$$2) \text{ Key} = 102 = 0110\ \underline{0110}$$

$$102 \bmod 8 = 6$$

$$3) \text{ Key} = 118 = 0111\ \underline{0110}$$

$$118 \bmod 8 = 6$$

$$4) \text{ Key} = 126 = 0111\ \underline{1110}$$

$$126 \bmod 8 = 6$$

$$\text{Ex: } m=7 \neq 2^k \neq 2^3$$

$$1) 86 \bmod 7 = 2$$

$$2) 102 \bmod 7 = 4$$

$$3) 118 \bmod 7 = 6$$

$$4) 126 \bmod 7 = 0$$

→ Note

► Choose the m value as a prime no. which is not nearer to power of 2. Then it will give min. no. of collision.

• mid square method :-

$$m = 1000(0 \dots 999)$$

Ex:  $(986)^2 = 9 \boxed{7} 2 \boxed{1} 9 6$

You can choose any 3 digit no.

Index	key
721	986

Hash table (m)

• we choose 3 digit bit only b/c there are large no. of 3 digit bit than one digit & two digit.

- these is a good hash functn

- counter ex<sup>n</sup> is difficult to generate.

- Digit Extraction method.

$$m = 1000(0 \dots 999)$$

$$k = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 9 & 3 & 4 & 6 & 8 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 0 \\ \hline 5 & 4 & 3 & 2 & 1 & 0 \\ \hline \end{array}$$

Index	key
948	934681

Hash table (m)

• hash  
• its a bad functn

• Counter example  
is easy to generate.

• Folding method:

① Shift Boundary method (Fold Boundary method)

key:  $123 \boxed{986} \boxed{321}$

$$\begin{array}{r} 123 \\ 321 \\ + \underline{444} \end{array}$$

Index	key
444	123986321

• its a bad hash functn  
• Counter ex<sup>n</sup> is easy to generate.

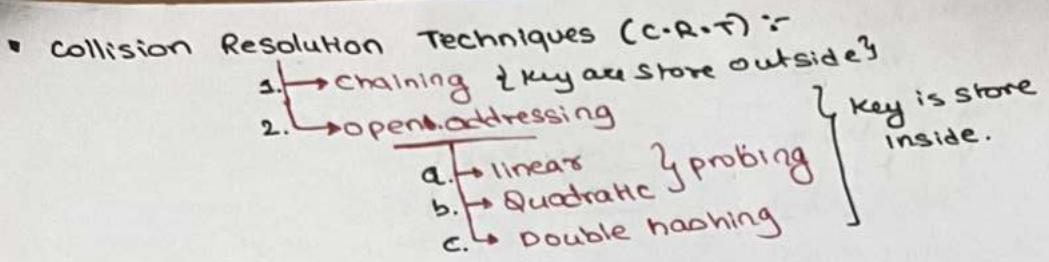
② Fold Shifting method:

key =  $\boxed{321} \boxed{123} \boxed{321}$

$$\begin{array}{r} 321 \\ 123 \\ 321 \\ + \underline{765} \end{array}$$

Index	key
765	321123321

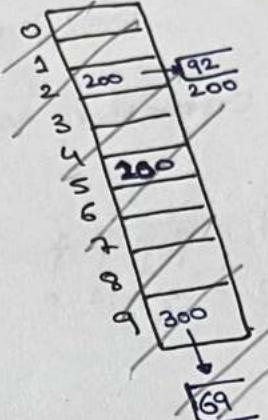
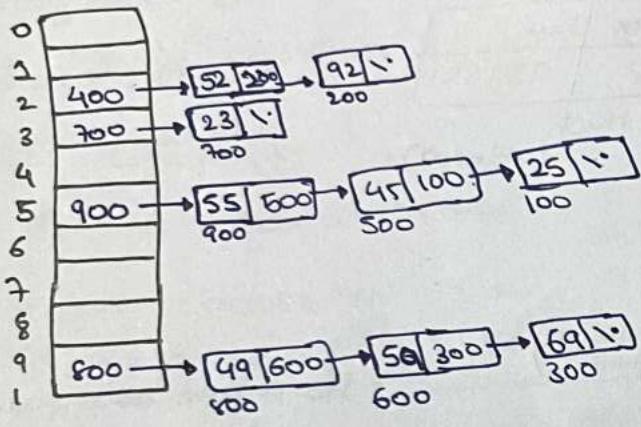
• its a bad hash functn  
• Counter ex<sup>n</sup> is easy to generate.



- Chaining:

- Chaining is implemented using Linked List
- Key will be stored outside the hash table.

key (n) : 25, 92, 69, 52, 45, 59, 23, 49, 75



node & HT [10]

- The main advantage of chaining is that we can handle any no. of collisions until the memory is full.
- Even though empty slots are available in hash table we are using outside space.
- The longest chain possible is n keys.

	B.C	WC
Insertion	O(1)	O(1)
Deletion	O(1)	O(n)
Searching	O(n)	O(n)

• open Addressing:

↳ Linear probing:

$$H.F(\text{key}) = \text{key} \bmod m$$

$m=10(0 \dots 9)$

$$L.P(\text{key}, i) = (H.F(\text{key}) + i) \bmod m$$

$i = 0, 1, 2, \dots, 9(m-1)$

Ex: Keys( $n$ ): 25, 38, 43, 68, 79, 46, 58, 65, 20

1)  $L.P(25, 0) = 5+0=5$

6)  $L.P(46, 0) = 6+0=6$

2)  $L.P(38, 0) = 8+0=8$

7)  $L.P(58, 0) = 8+0=8$

3)  $L.P(43, 0) = 3+0=3$

$8+1=9$

4)  $L.P(68, 0) = 8+0=8$

$8+2=10 \therefore 10 \equiv 0$

$8+3=11 \therefore 11 \equiv 1$

5)  $L.P(79, 0) = 9+0=9$

$9+1=10 \therefore 10 \equiv 0$

8)  $L.P(20, 0) = 0+0=0$

$0+1=1$

$0+2=2$

9)  $L.P(65, 0) = 5+0=5$

$5+1=6$

$5+2=7$

|| Total no. of collision: 9 //.

79	0
58	1
20	2
43	3
	4
25	5
46	6
65	7
38	8
68	9

Hashtable.

• load factor ( $\alpha$ ):-

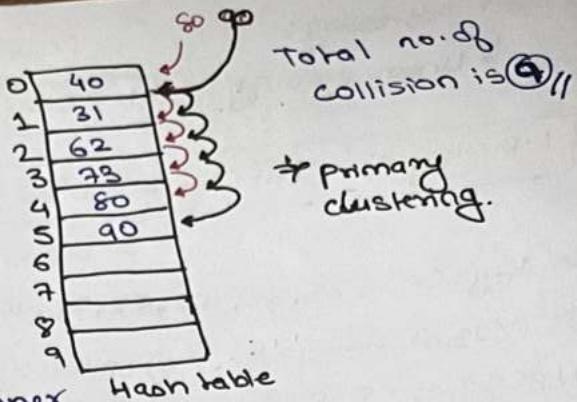
the no. of keys getting stored in one slot is known as load factor.

In "m" slot  $\rightarrow$  "n" keys are stored.

In 1 slot  $\rightarrow$  ?

$$\frac{1}{m} * n = \frac{n}{m} \Rightarrow \boxed{\alpha = \frac{n}{m}}$$

Ex: Keys(n) = 62, 31, 40, 73, 80, 90



- primary clustering.
- If the two keys are mapped on to same starting location in the hash table then both follow the same path in the linear manner & b/c of that search time complexity will increase
- To avoid this problem we will use quadratic probing.

**note**

- deletion will create problem to other keys but we can manage by storing some special symbol like "#"

	B.C	W.C
Insertion	O(1)	O(m)
deletion	O(1)	O(m)
Searching	O(1)	O(m)

- If more no. of deletion occur then perform re-hashing

Quadratic probing:

$$\text{Simple term: } (H \cdot f(\text{key}) + c_1 \cdot i + c_2 \cdot i^2) \bmod m$$

$$Q.P(\text{Key}, i) = (H \cdot f(\text{key}) + c_1 \cdot i + c_2 \cdot i^2) \bmod m$$

$c_1 = 1, c_2 = 1 \quad i = 0, 1, 2, \dots, q(m-1)$

Ex: Keys(n) = 25, 98, 57, 75, 97, 18, 78, 46.

$$1) Q.P(25, 0) \Rightarrow 5 + 0 + 0 = 5$$

$$2) Q.P(98, 0) \Rightarrow 8 + 0 + 0 = 8$$

$$3) Q.P(57, 0) \Rightarrow 7 + 0 + 0 = 7$$

$$4) Q.P(75, 0) \Rightarrow 5 + 0 + 0 = 5$$

$$5 + 1 \cdot 1 + 1 \cdot 1^2 = 5 + 1 + 1 = 7$$

$$5 + 1 \cdot 2 + 1 \cdot 2^2 = 5 + 2 + 4 = 11 \quad \% 10 = 1$$

$$5) Q.P(97, 0) = 7 + 0 + 0 = 7$$

$$= 7 + 1 + 1 = 9$$

0	18
1	75
2	
3	.
4	78
5	25
6	46
7	57
8	98
9	97

Total no. of collision is 6 //

Q) Keys(n) = 62, 81, 43, 80, 90, 10

$$Q.P(90, 0) = 0+0+0=0$$

$$(90, 1) = 0+1+1=2$$

$$(90, 2) = 0+2+4=6$$

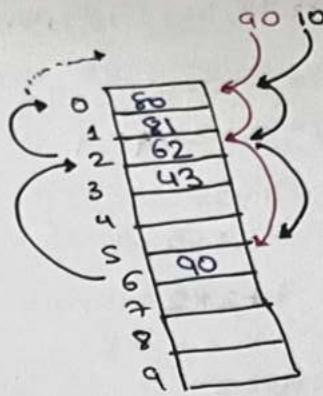
$$Q.P(10, 0) = 0+0+0=0$$

$$(10, 1) = 0+1+1=2$$

$$(10, 2) = 0+2+4=6$$

$$(10, 3) = 0+3+9=12 \mod 10 \\ = 2$$

$$(10, 4) = 0+4+16=20 \mod 10 \\ = 0$$



Secondary clusters.

### Secondary clustering.

- if the two key are mapped on to same starting location in the hash table then they both follow same path in the quadratic manner. b/c of these search time complexity will increase.
- To avoid this problem we will use double hashing.

	B.C	w.c
Insertion	O(1)	O(m)
Deletion	O(1)	O(m)
Searching	O(1)	O(m)

### Double hashing.

$$H \cdot F_1(\text{key}) = k \bmod m$$

$$H \cdot F_2(\text{key}) = (k \bmod m - 2) + 1$$

$$D \cdot H(\text{key}, i) = (H \cdot F_1(\text{key}) + i * H \cdot F_2(\text{key})) \bmod m$$

$$i = 0, 1, 2, \dots, m-1$$

	B.C	w.c
Insertion	O(1)	O(m)
Deletion	O(1)	O(m)
Searching	O(1)	O(m)

$$\left[ \frac{1}{x-1} \dots \frac{1}{x} \right]$$

Q] Keys(n) = 25, 98, 57, 75, 97, 18, 78, 65, 46.

4) DH(75, 0) = 5 + 0 = 5  
 $(75, 1) = 5 + 1 \times 2 = 7$   
 $\Rightarrow DH(97, 0) = 7 + 0 = 7$   
 $(97, 1) = 7 + 1 \times 2 = 9$   
 $(97, 2) = 7 + 2 \times 2 = 11 \text{ } \% \text{ } 10$   
 $= 1$

0	
1	97
2	78
3	65
4	18
5	25
6	46
7	57
8	98
9	75

Hash table.

$\Rightarrow DH(18, 0) = 8 + 0 = 8$   
 $(18, 1) = 8 + 1 \times 3 = 11 \text{ } \% \text{ } 10$   
 $= 1$   
 $(18, 2) = 8 + 2 \times 3 = 14 \text{ } \% \text{ } 10$   
 $= 4$   
 $\Rightarrow DH(78, 0) = 8 + 0 = 8$   
 $(78, 1) = 8 + 1 \times 7 = 15 \text{ } \% \text{ } 10$   
 $= 5$   
 $\Rightarrow DH(78, 2) = 8 + 2 \times 7 = 22 \text{ } \% \text{ } 10$   
 $= 2$

$\Rightarrow DH(65, 0) = 5 + 0 = 5$   
 $(65, 1) = 5 + 1 \times 2 = 7$   
 $(65, 2) = 5 + 2 \times 2 = 9$   
 $(65, 3) = 5 + 3 \times 2 = 11 \text{ } \% \text{ } 10$   
 $= 1$   
 $(65, 4) = 5 + 4 \times 2 = 13 \text{ } \% \text{ } 10$   
 $= 3$   
 $\Rightarrow DH(46, 0) = 6 + 0 = 6$

Total no. of collision: - 10

• **Note**

- if  $n \leq m$ , then by using Perfect hashing we can achieve w.c. search time complexity as " $O(1)$ ". (universal hashing (no collision)).

- The expected no. of probes in an unsuccessful search of open addressing technique is.

$$\frac{1}{1-\alpha}$$

, where ' $\alpha$ ' is load factor.

- The expected no. of probes in successful search of open addressing technique is

$$\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$$

where ' $\alpha$ ' is load factor

$$d = n/m$$

- In a hash-table with chaining & uniform hashing,  
the expected no. of comparisons in an unsuccessful search is  
given by  $\alpha$  ,  $\alpha = \frac{m}{n}$  (record)  
 $\qquad\qquad\qquad$  (slot)