PROGRAM-MING & D.S.

#PROGRAMMING (Dennis, C test your aptitude by Venugopal & N. Chand

1. Basic
2. Storage Class, Scope
3. Pointers, String
4. Array, Recursion
5. Structure, Union
6. Dynamic memory Allocation

DATA STRUCTURE (Sahani)

1. Linked List
2. Stack
3. Queue
4. Tree (BST, AVL)
5. Hashing

BASIC :

OPERATOR	PRECEDANCE	ASSOCIATIVITY
()	1 (High)	L → R
↑ (Power)	2	R → L
* / %	3	L → R
+ −	4	L → R
=	5 (Low)	R → L

ASSOCIATIVITY :

If two or more operator having same precedence expression will be evaluated based on the associativity.

e.g. 2 + 3 * 4 − 3/1

2 + 12 − 3 − 3/1

2 + 12 − 3

14 − 3 = 11

EXPRESSION	RESULTS
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 :

1. If both are integer then O/P will be integer.
2. If anyone is float O/P will be float.

EXPRESSION	ASSIGNED to Int	ASSIGNED to Float
5	5	5.0
5.0	5	5.0
5/2	2	2.0
5.0/2	2	2.5
5.0/2.0	2	2.5
2/5	0	0.0
2.0/5	0	0.4
2.0/5.0	0	0.4

Quest: int a = 2*3/4 + 2.0/5 + 8/5;
pf("%d", a);

O/P → 2

$$\begin{bmatrix} 2*3/4 + 0.4 + 1 \\ 6/4 + 0.4 + 1 \\ 1 + 1 = 2. \end{bmatrix}$$

RELATION & LOGICAL OPERATOR:

1. All the R & L.O. returns 1 or 0.
2. If the expression is true then it return 1 the expressio is FALSE then it return 0.
3. All non-zero is consider as TRUE.
   All zero is consider as FALSE.

TRUE	FALSE
60	0
-1	0.0
0.6	NULL
-10.5	\0
15	
1	

e.g. a = 2>3 → 0

e.g. a = 5>4>3 → 1>3 → 0

e.g. a = (5>4) + (3>2) → 1 + 1 → (2)

MODULAS OPERATOR

1. 15 % 7 = 1
2. -15 % +7 = -1
3. +15 % -7 = +1
4. -15 % -7 = -1
5. -15.5 % +7 = error
6. -5 % +7 = -5
7. +5 % -7 = +5

e.g. 7) 15 (2
-14
1

e.g. -7) 15 (-2
+14
1

e.g -7) 5 (+2
-14
1

Note:

1. Modulous always use numerator sign
2. Modulous does not work on float values it works only on int.
3. If the value is the small without sign then it gives same value as a output.

IF CONDITION

e.g.

```
void main()
{ int a=5;
  if (a=8);
    { pf("Raj");
    }
  else
    { pf("kishore");
    }
  pf("%d", a);
}
```

O/p → Raj
a = 8

NOTE:

* Assignment operator assign the value & returns assinged value.

e.g. Printf();

```
Void main()
 { int a;
   a = pf("Hello");
   pf("%d", a);
 }
```

a = Hello
    01234
    5

O/P → Hello
a=5

NOTE:

Printf always returns integer then i.e. no. of symbols displays on the screen by that particular printf if we want we can correct that value.

e.g.

```
void main()
 { int a
   a = pf(" made %d easy", pf("Delhi"));
     pf("%d", a);
 }
```

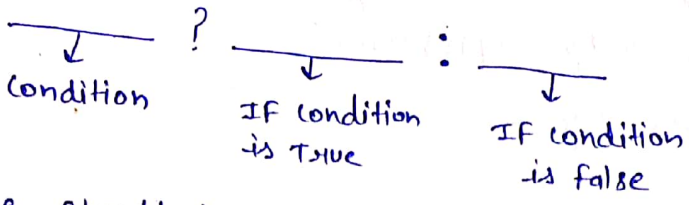
O/P → Delhi made 5 easy 9

Quest: WAP to read two proper integers and print them on the screen otherwise print wrong I/P given.

SCANF: Scanf always returns integer that is no. of proper I/P given according to formate specifies.

```
Void main()
 { int x, y, Sn;
   pf("enter two proper integers");
   Sn = Scanf("%d", &x, &y);
   if (Sn==2)
     { pf("proper I/P given);
       pf("%d %d", x, y);
   else
     {
      pf("wrong input");
     }
 }
```

I/P	Sn
*, #	0
#, 21	1
10, 20	2



1. There should be equal no. of collons & question marks otherwise they gives error.
2. Every collons should match with the just before questionmark.
3. Every quation mark is followed by collon not immediatly but following.

e.g. a = 3>4 ? 10 : 8>7 ? 20 : 30

O/P → 20

e.g. a = 5>4 ? 6>7 ? 10 : 20 : 20

O/P → a = 20

e.g. a = 2>3 ? 5>4 ? 10 : 30 : 8>7 ? 30 : 40

O/P → a = 30

e.g.

a = 6>7 ? 2>3 ? 10 : 6>8 ? -2 : 3 : 0 ? -1 : 6>8 ? 10 : 3>4 ? 10 : 20

O/P → 20

Quest: WAP to find maximum of three no. using ternary operator

```
int a=10, b=20, c=30, ma;
max = (a>b) && (a>c) ? a : b>c ? b : c;
```

Quest: WAP to find minimum of four no.

```
int a=10, b=20, c=30, d=40, min;
min = a<b && a<c && a<d ? a : b<c && b<d ? b : c<d ? c : d;
```

Quest: WAP above two program without using && operator?

GATE

Which combination of the integer variables x, y and z makes variable 'a' get value '4' in the following expression?

```
a = x>y ? x>z ? x : z : y>z ? y : z
```

(a) x=6, y=5, z=3

(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 :

```
a=10;
b=++a;
```

a=11
b=11

```
a=10;
b=a++;
```

a=11
b=10

```
a=10;
b=--a;
```

a=9
b=9

```
a=10;
b=a--;
```

a=9
b=10

GATE

```
main()
{
  int m=10;
  int n, n1;
  n = ++m;
  n1 = m++;
  n--
  n -= n1   // n = n-n/; = 0
  pf("%d",n)
}
```

Quest:

```
a = 50;
b = 50;
a = a++ + ++b;      50 + 51 = 101
b = b++ + ++a;      51 + 103 = 154
```

O/P → a = 103
b = 155

Quest:

```
a = 2;
a = ++a * a++ * a++;   //Bad Prog.
```

Quest:

```
a = 5;
pf(++a, ++a, ++a);   //Bad Prog.
```

8 7 6
6 7 8

Quest:

```
a = 5;
pf(a+1, a+2, a+3);   //Good Prog.
```

6 7 8
6 7 8

Compiler Dependent.

NOTE:

1. Program should work based on the logic not based on the compiler.
2. Incrementing the same variable multiple times in a single line is called as a bad programming.

Quest:

```
main()
{ a = 5;
  pf("%d", (a++)++);   //error
  pf("%d", (a+1)++);   //error
}
```

NOTE:

1. Pre increment and post increment works only on variables but not on a constants not on expression.

LOOPS :

- For
- while
- Do while

Quest: WAP to print 1 to 10 no. using above loop's?

1.
```
void main()
 {
  int i;
  for (i=1; i<=10; i++)      // for(i=1; i<=10; pf("%d", i++));
     {
      pf("%d", i);
     }
 }
```

2.
```
void main()
  {int i;
    while (i <=10)
        {
          pf("%d", i);
          i++;
        }
  }
```

3.
```
do {
    int i
     pf("%d", i);
     i++;
  } while (i<=10);
```

Quest:

```
Void main()
{
  int n=5;
  while (n++<= 7);
  pf ("%d",n);
}
```

O/P → 9

Quest:

```
Void main()
{
  int i=1;
  do
    { while (i++<1);

    } while (i++<=2);
  pf ("%d", i);
}
```

i = 1

i++ = 1 < 1

i = 2

3 2 <= 2

i = 3

i = 4

i = 5

O/P → 5

for (a(); a(); a())

- a() → only once → return value(x)
- a() → min. once → Return value (v)
- a() → min zero; → return value (x)

1. a = 10
   f(a++)
   a.v = 11
   p.v = 10

2. a = 10
   f(++a)
   a.v = 11
   p.v = 11

3. a = 10
   f(a+1)
   a.v = 10
   p.v = 11

H.W

1. input n = 5;

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

2. input n = 6;

```
1..........1 → 10 dots
12........21 → 8 dots
123......321 → 6 dots
1234....4321 → 4 dots
12345..54321 → 2 dots
123456654321 → 0 dots
```

3. Also do minor

```
123456 654321
12345..54321
1          1
 2        2
  3      3
   4    4
    5  5
     66
```

SWITCH CASE :

1. Case and Label should have the space.
2. Case label must end with the collon
3. Case label should be constant and it should not be floating point no.
4. Empty switch cases allowed.
5. Only one default is allowed and it can be placed anywhere in the switch.
6. Default will be executed only when no case is matching.
7. After every case there is a requirement of break. otherwise it will execute all further cases until the occurence of break or till the end of switch.

```
Switch (expression)
  {  case Label 1 :
              body;
              break;
     case label 2 :
              body;
              break;
     default :
              body;
              break;
  }
```

E.g.

```
main()
{ int n;
  sf("%d", &n);
  Switch(n)
   {
     Case 0: n=n+1;
             break;
     default: n=n-1;
     Case 1: n=n+10;
             break;
     Case 2: n=n+100;
  }
  Pf("%d",n)
}
```

I/p	O/p
n=0	1
n=1	11
n=2	103
n=3	12

GATE:

```
main()
{ int()
  for(i=1; i<=25; i++)
    {
      Switch(i)
       {
         Case 0: i+=5;
         Case 1: i+=3;
         Case 2: i+=4;
         default: i+=5;
                  break;
       }
      Pf("%d", i);
    }
}
```

O/p → 13, 19, 25

i = 1

1 ≤ 25 T

i = 1

i = 4

i = 8

i = 13

Pf = 13

---

i = 2

2 ≤ 25 T

i = 13, 19, 25

i = 13

13 ≤ 25 T

i++

i = 14

14 ≤ 25

default

i = 14+5 = 19

i = 19

i = 19

i++

i = 20

i = 20 ≤ 25 T

default

i = 20+5 = 25

i = 13, 19, 25

GATE: Void main()

```
{
  int i, j k=0;
  j = 2*3/4 + 2.0/5 + 8/5;
  k -= --j;
  for (i=0; i<5; i++)
  {
    switch (i+k)
    {
      case 0:
      case 1 : pf("%d", i+k);
      case 2 : pf("%d", i+k);
      default : pf("%d", i+k);
    }
  }
}
```

The number of times pf() statement is executed → 10

$2 \times \frac{3}{4} = \frac{6}{4} = \frac{3}{2}$

1 + 0.4 + 1

2 + 0.4

2.4

j = 2 j = 1

k = k - (--j)

k = 0 - (1)

k = -1

i = 0

0 < 5 T

0 + (-1)

(-1)

(1)

i =

(1)

(3)

(3)

(2)

(1)

(10)

i++ = 3

1

i = ~~-1~~ 0

j = ~~-1~~ 0

k = ~~-1~~ 0

l = 2

i++ = -1

j++ = -1

k++ = -1

SHORT CIRCUIT EVALUATION

a = 0, b = 2, c = 2, d = 1;

a = (b++ > 1 || c++ > 1) && d++ > 1;

(b++ > 1 → F; c++ > 1 → F; (F) || (F) → F; d++ > 1 → Not evaluated)

pf(a, b, c, d);

O/P → (0, 2, 2, 1)

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

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

(i++ → -1; j++ → -1; k++ → 0; (T) || (l++) → N.E.; l++ → 3)

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

If any integer post/pre operation in conditional operator then value will be 1

I/P	O/P i	j	k	l	m
original	0	0	0	2	1
if k=0	0	0	1	3	1
if j=0	1	-1	-1	3	1

FUNCTIONS :

1. The purpose of the function is code reutilization.
2. purpose of the loop is code repeatition
3. purpose of the headersfile is function re utilization.
4. If you want to reuse the function then we need to create headersfile of that function and include that in the required program.
5. The function will have following parameters.
   (i) Prototype / Declaeration
   (ii) Function calling
   (iii) function defination

(i) Prototype contains
   (a) name of the function
   (b) Return type of the function
   (c) No. of Arguments
   (d) Types of Arguments

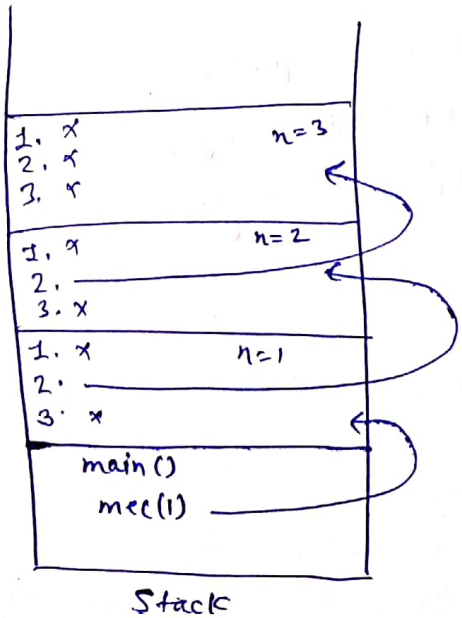
RECUERSION :

1. Function calling itself this called as recursion.
2. Recursion should definetly have termination condition
3.

Ques:

```
void mec(int n)
  {
    Printf("%d", n);
    JF(n<=2)
    mec(n+1);
    PF("%d", n);
  }
void main()
  {
    mec(1);
  }
```

o/p → 123321



NOTE :

1. Recursion internally makes use of a stack.
2. Every time when recursive call is made activation record will be create in the stack.

ACTIVATION RECORD : The stack frame allocated further recursive function call is called as a activation record.

3. The creation and deletion of the activation record depends on actual function calling sequence.

BITWISE OPERATOR'S :

Operators	meaning
&	Bitwise AND
|	Bitwis OR
>>	Right shift
<<	Left shift

→ AND / OR :

e.g.

```
void main()
{
  int n=25, y=12;
  pf("%d", n&y);
  pf("%d", n|y);
}
```

n → [25 → 11001]
y → [12 → 01100]

	16	8	4	2	1	
n =	0	1	0	0	0	AND → 8
y =	0	1	1	0	0	OR → 29

Right Shift (>>)

Right shift operator shift all the bits toward the right by ~~certain~~ certain no. of specified bits.

e.g. 212 = 0000 0000 1101 0100

212>>2 → 0000 0000 0011 0101 → 53

212>>7 → 0000 0000 0000 0001 → 1

Left Shift (<<)

Left shift operator shift all the bits toward the left by certain no. of specified the bits.

e.g. 212 = 0000 0000 1101 0100

212<<1 → 0000 0001 1010 1000 → 424

212<<4 → 0000 1101 0100 0000 → 3392

Quest:

```
main()
{
  int i, num = 212;
  for(i=0; i<=2; i++)
  pf("%d", num>>i);
  for(i=0; i<=2; i++)
  pf("%d", num<<i);
}
```

>>	<<
i=0, 212	i=0, 212
i=1, 106	i=1, 424
i=2, 53	i=2, 848

H.W.

16 Bit

int 6B

2B

Date 2B → 1 to 31 → 5 bits

month 2B → 1 to 12 → 4 bits

year 2B → 128 year → 7 bits (−1900, +1900)

HW: WAP to store date of birth in a two bytes?

Quest:
```
Void main()
{ int n = 4096;
  while (n) //4096, 2048, 1024
    { if (n & 1)
        Pf("made Easy");
      n >>= 1;
    }
}
```

$4096 = (1000\ 0000\ 0000)_2$

1
0

n = 1000 0000 0000

1
0

n = 1000 0000 000

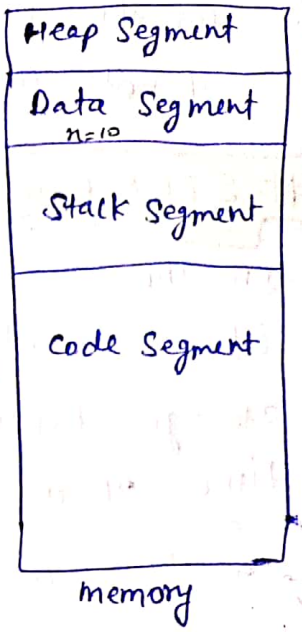
How many time string made easy printed? O/P → 1

PRE PROCESSOR SUBSTITUTION:

1. pre processor will make only substitution and it will not make any calculation.
2. It is recognize those statement by #
3. It will be done before compilation of the program.

```
#define cube(x) x*x*x
Void main()
{ int x=3, y;
  y = cube(x);
  Pf("%d", y);
}
```

MEMORY SEGMENTS :



Code Segment : In the code segment of the memory all the general instructions which will not change through out the program execution will be stored.
There called as read only and protected.
e.g. – Notepad, wordpad

Stack Segment : In the stack segment all the recursive function call's, local variables and formal parameters will be stored.

Data Segment : In the data segment all the global variables and static variables will be stored.

Heap Segment : In the heap segment all the dynamic memory allocation will take place.
The memory allocation by using standard functions like
malloc();
calloc();
realloc();

e.g.
```
int n=10;
void main()
 {
  int y=20;
   {
    int n=30;
     pf("%d",n);
   }
  pf("%d,%d",n,y);
}
```

NOTE : 1. local variable having high precendene comape to globle
2. local variable memory allocated at run time. if it is not initilize it will contain garbage value

3. for the global var. mem. allocated at Compiled time if not initilize default contain zero

STORAGE CLASSES :

In the C lang. every varaible is having some chasacter like - datatype, Scope, Default value and life time.

Storage Classes mainly deals with the scope and the life time of a variable.

SCOPE : The region in which ~~whenever~~ variable is available and accesesable is called as the scope of variable.

Type	Scope	Life Time	Default Value	Storage	memory Segment
Auto	Body	Body	Garbage Value	RAM	Stack Segment
Register	Body	Body	Garbage Value	Registers	CPU Segment
** Static	Body	Program	0	RAM	DATA Segment
Extern	Program	Program	0	RAM	DATA Segment

AUTO :

Syntax : auto datatype Variable Name;

e.g auto int n;

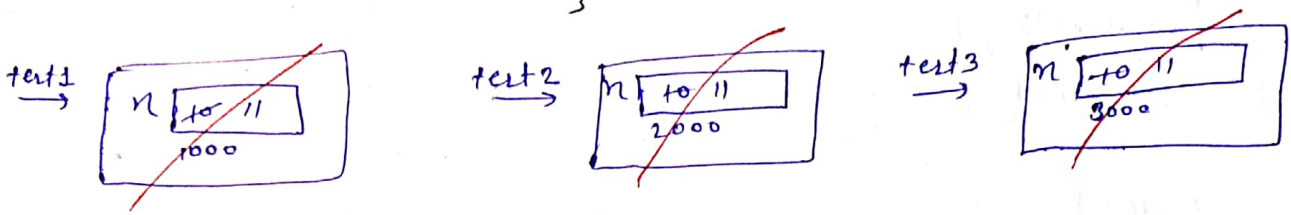
1. By default all the local variables are auto variables.
2. for every function call auto variable is recreated and reinitilize.
3. It has local Scope and life time is with in the body.
4. Default value is garbage value.

e.g: int n;

```
void main()
{
  test();
  test();
  test();
}
```

```
void test()
{
  auto int n=10;
  pf("%d", n);
  n++;
}
```

O/p → 10, 10, 10



REGISTER :

Syntax: register data_type variable_Name;

1. The keyword register is just request to the compiler to allocate the memory in the registers instead of RAM.
2. If the free registers are available then it will allocate in the registers otherwise it will allocated in the RAM only.
3. Register variable are faster Compare to auto variable hence they are used in the loops & functions.
4. By default register variables behave likes autoVarible only.

Imp STATIC :

Syntax: Static data_type variable_name;

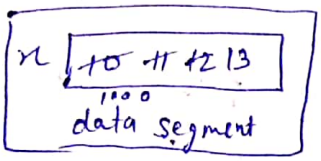
1. The variable ~~for~~ persisting previous state value even though the distraction of several function call is called as Static variable.
2. Static variable memory allocated only once it compile time in the data segment.
3. It has local scope but the life time is entire program.
4. If it is not initilize default contain zero.
5. Static variable can be local variable but the memory allocated at compile time.

e.g.

```
void main()
{
   test();
   test();
   test();
}
```

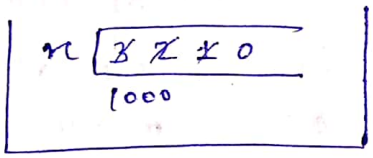
```
void test()
{
  static int n=10;
   pf("%d", n);
   n++;
}
```

O/P → 10, 11, 12

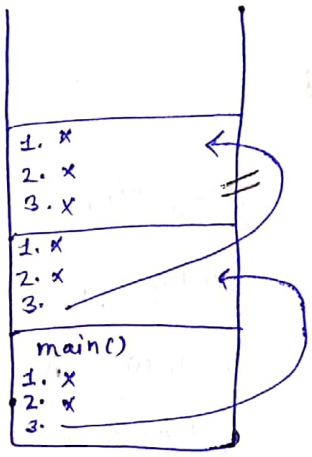


e.g.

```
main()
{
1-static int n=3;
2-pf("%d", n--);
3-if(n)
    main();
}
```



data segment

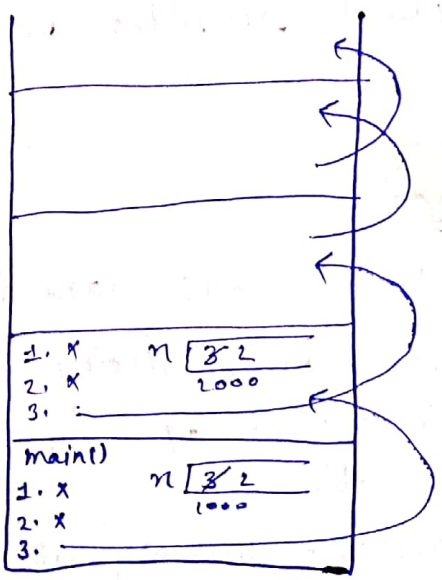


O/P → 3, 2, 1

e.g. without static

```
main()
{  int n=3;
   pf("%d", n--);
   if(n)
      main();
}
```

O/P → 3, 3, 3 ----



* This is called termination with stack overflow.

e.g. void main()

```
{
  while(1)
  {
    pf("Hi");
  }
}
```

Infinite loop

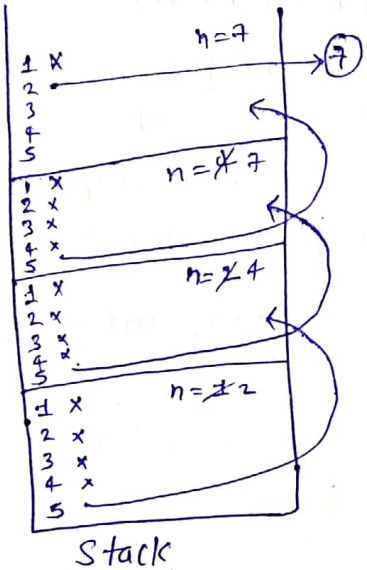
o/p → Hi Hi - - -

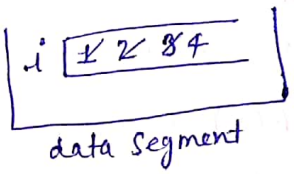
GATE :→

```
int f(int n)
{
1 static int i=1;
2 if (n>=5)
    return n;
3 n = n+i;
4 i++;
5 return f(n);
}
```

i=1
n=1
1>=5 F
n=1
n=1+1=2
n=2, i=2
2>=5 F
n=2
n=2+2=4
n=4
i=3
4>=5
n=4
n=4+3=7
i=4
n=7

What is the return value of f(1) 7 ?

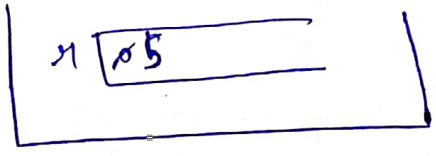


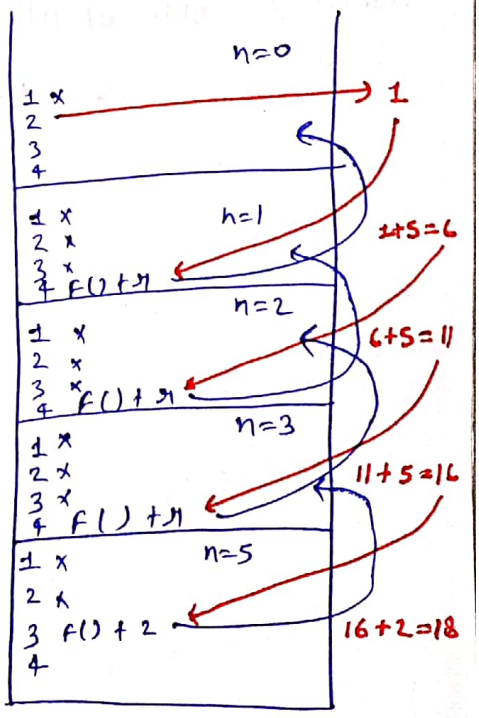


GATE :→

```
int f(int n)
{
1 static int x;
2 if (n<=0) return 1;
3 if (n>3)
  {
    x=n
    return f(n-2)+2;
  }
4 return f(n-1)+x;
```

what is the return value to f(5) 18 ?





GATE: ⟶

```
int incr (int i)
{
  static int count;
  count = count + i;
  return count;
}
void main()
{
  int i,j;
  for (i=0; i<=4; i++)
    j = incr(i);
    pf("%d", j)
}
```

i	j	count
~~0~~	~~0~~	~~0~~
~~1~~	~~1~~	~~1~~
~~2~~	~~3~~	~~3~~
~~3~~	~~6~~	~~6~~
~~4~~	(10)	10
5		

what is O/P ——— ? → 10

GATE : ⟶

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

```
int f.(int n)
{ int y;
  y = g(n);
  return (n+y);
}
int g(int n)
{ static int y = 5;
  y = y + 7;
  return (n+y);
}
```

what is sum of all values printed by above program 186

56 + 130 = 186

GATE :→

```
int f()
{
  static int num=7;
  return num--;
}
void main()
{
  for (f(); f(); f())
  printf("%d", f());
}
```

num | 7 6 5 4 3 2 1 0 -1

what is o/p printed?

(a) 41
(b) 630
(c) 63
(d) 52

(41)

```
#include <stdio.h>
int total(int v)
{ static int count=0;
  while (v)
  { count += v&1;
    v>>=1;
  }
  return count;
}
void main()
{ static int n=0;
  int i=5;
  for(; i>0; i--) {
    n = n + total(i);
  }
  printf("%d\n",n);
}
```

GATE? →

```
void count (int n)
{ static int d=1;
  pf ("%d", n);
  pf ("%d", d);
  d++;
  if (n>1)
  count (n-1);
  pf ("%d", d);
}
void main()
{ count(3)
}
```

count (3)
n=3, d=1
pf(n) → 3
pf(d) → 1
n>1
3>1 T
count (3-1)
count (2)
n=2, d=2
pf(n) → 2
pf(d) → 2
d++
d=3
2>1 T
count (n-1)
count (1)
n=1, d=3

(a) 3122134 4 4
(b) 3121112 2 2
(c) 312213 4
(d) 312112

EXTERN :

Syntax :→ extern data_type variable_name ;

i.e extern int n;

1. No memory will be allocated for the extern variable they make use of memory of a outside variable (globle variable)

e.g.

```
int a=8;
main()
{ extern int a;
  pf(a);
  a=20;
  pf(a);
}
```

o/p → a→8
a→20

e.g.

```
main()
{ extern int a;
  pf("Hello");
}
```

o/p → Hello

e.g.
```
main()
{
    extern int a;
    pf(a);
}
```

o/p → error

e.g.
```
int a=5;
main()
{
    extern int a;
    extern int a;
    extern int a;
    pf(a);
}
```

o/p → a=5

* If we remove all extern in this program error will be occur.
* multiple declaration is allowed only with extern class

e.g.
```
main()
{
    extern int a;
    pf(a);
}
int a=10;
```

o/p → 10

e.g.
```
main()
{
    extern int a=10;
    pf(a);
}
```

o/p → error

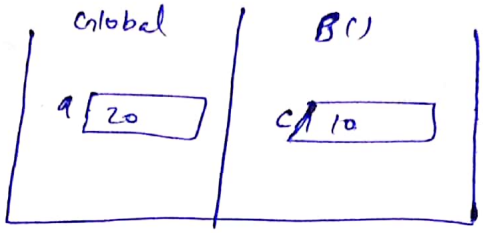
* extern variable initilzation is not allowed only assignment is allowed

Declaration — D
Initialization — I
Assignment — A

e.g. ① `int a=20;` → D,I

```
main()
{
  int a=10;   → D,I
  int b;      → D
  b=20;       → A
  B();
}
```

```
B()
{
  static int c=10;   (D,I)
  a=10;              (A)
}
```

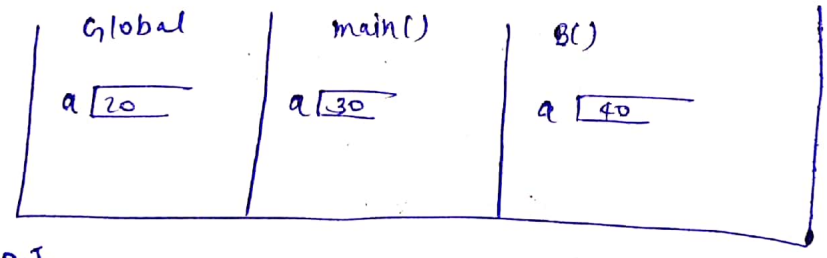


e.g ② `int a=20` → D,I

```
void main()
{
  static int a=30;   → D,I
  B();
}
```

```
B()
{
  static int a=40;   → D,I
  C();
}
```

```
C()
{
  extern int a;   → D
}
```



Imp
e.g. ③ `a=10;` → D,I

```
main()
{
  a;      → D
  a=20;   → A
  B();
}
```

```
B()
{
  a=20;   → D,I
  a=10;   → A
}
```

SCOPE

- Static Scoping (Compile Time)
- Dynamic Scoping (Run Time)

NOTE:

Scoping rules are aplicable only to the "free Variable"

Free Variable: The variable which is not declare in the function but used in the function is called as the free variable

1. STATIC SCOPING:

In the static scoping free variable are resolved from Global Variable.

2. DYNAMIC SCOPING:

In This scoping free variable are resolved from previous Calling function.

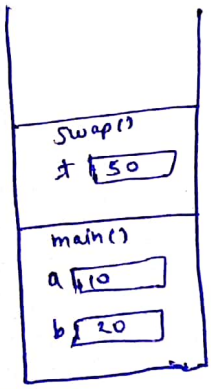
NOTE:

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.
2. C lang. by default follows static scoping.

e.g ①

```
int a=50, b=60;
main()
{ int a,b;
  a=10, b=20;
  swap();
  pf(a,b);
}
```

```
swap()
{ int t;
  t=a;
  a=b;
  b=t;
  pf(a,b);
}
```



3

3

Static	Dynamic
a 50 60 b 60 50	swap() t 10 main() a 10 20 b 20 10
o/p 60, 50 10, 20	a 50 b 60 o/p → 20, 10 20, 10

e.g.

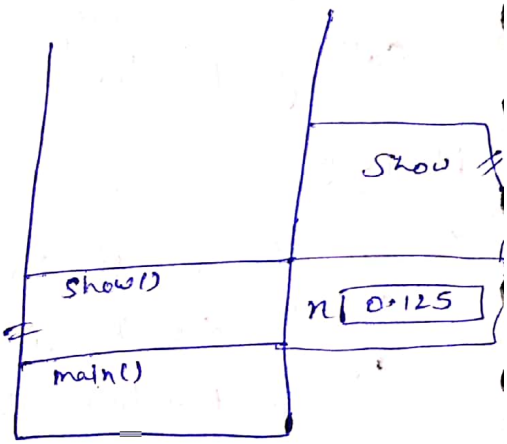
```
int a=5;
main()
{
    int a=10;
    B()
}

B()
{
    int a=20;
    C();
}

C()
{
    int a=30
    D();
}

D()
{
    int a=40;
    Pf(a);
}
```

* Scoping rule are available only for free variable

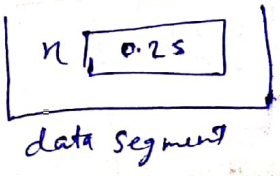
Static	Dynamic
5	30
	20
	10
	5

GATE :→

```
Variable n : real
Procedure show()
  begin
      print(x)
  end
Procedure small()
  begin
      Variable n : real
          n = 0.125;
          show()
  end
main → begin
          n=0.25;
          show();
          small();
       end
```



STATIC	Dynamic
0.25	0.25
0.25	0.125



GATE :⟶

```
Variable x,y : integers
Procedure P(variable n : integer)
    begin
        n = (n+2) * (n-3);
    end
Procedure Q()
    begin
        variable x,y : integer
            x = 3;
            y = 4;
            P(y);
            PF(x);
        end
main ⟶ begin
            x=7;
            y=8;
            Q();
            PF(x);
       end
```

x = 7
y = 8

P(4)
n = 4
n = (4+2) * (4-3)
n = 6 * 1
n = 6

Static	dynamic
3, 6	6, 7

GATE :

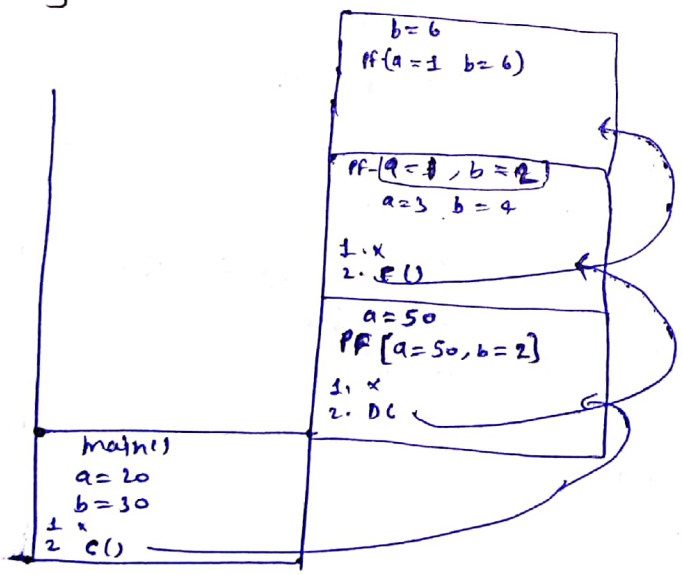
```
int a=1, b=2;
main()
{ int a = 10, b = 30;
1 pf(a,b);
2  C()
  P(a,b);
  D();
3
```

```
C()
{ int a=50;
  pf (a,b);
   D()
  pf (a,b);
3
```

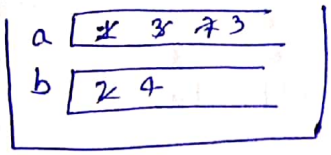
```
D()
{ pf(a,b);
  a=3;
  b=4;
  pf=(a,b);
  E();
3
```

```
E()
{
 int b=6
   pf(a,b);
     a=7;
     b=8;
3
```

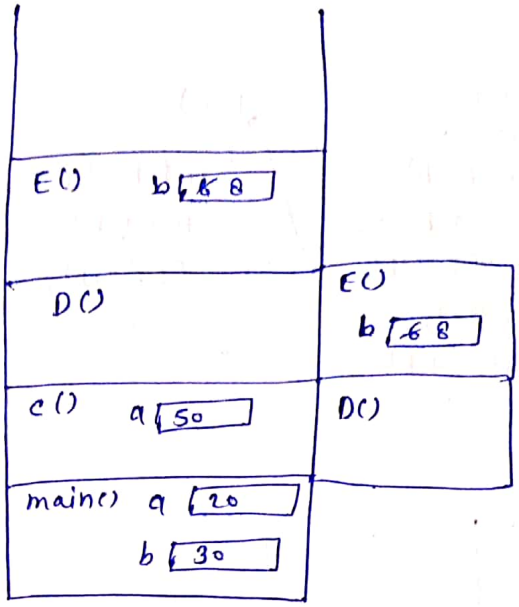
Static	Dynamic
20, 30	20, 30
50, 2	50, 30
1, 2	50, 30
3, 4	3, 4
3, 6	3, 6
50, 4	7, 4
20, 30	20, 4
7, 4	20, 4
3, 4	20, 4
3, 6	3, 4
	3, 6



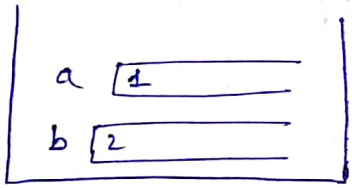
20 30 50 2 1 2 3 4



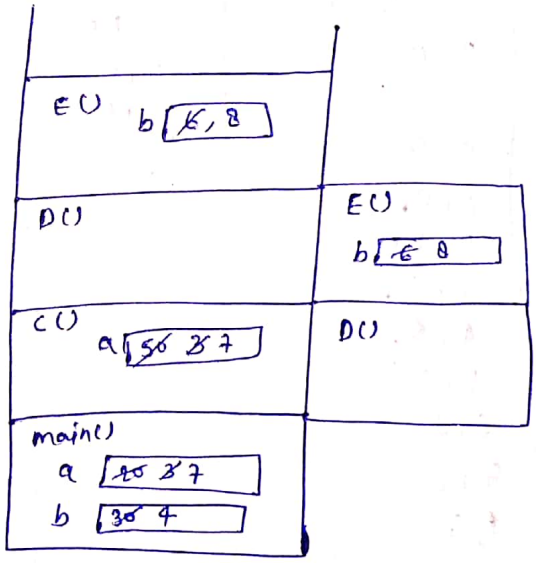
data Segment



Static



data Segment



Dynamic

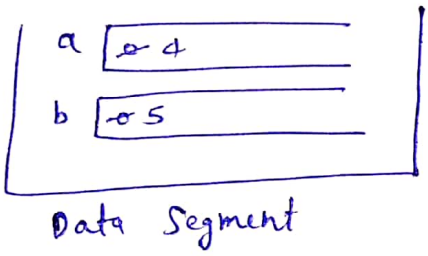
GATE →

```
int a,b;
main()
{
 int a=3;
 1 Pf(a,b);
 2 C();
 3 Pf(a,b);
 4 D();
}
```

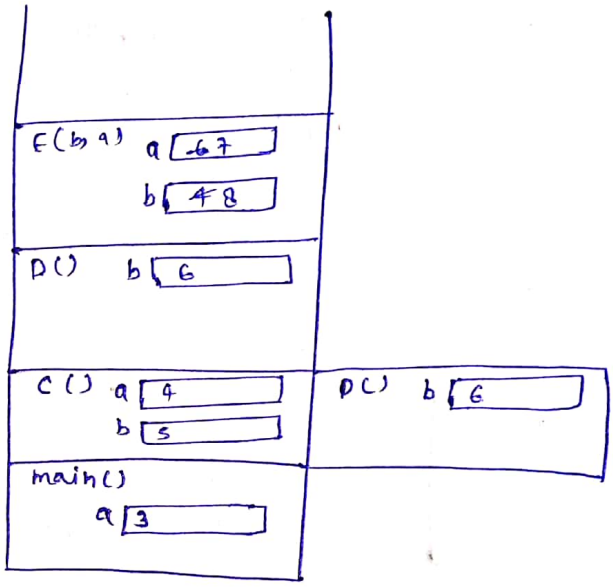
```
C()
{
1 Pf(a,b);
 a=4, b=5;
2 Pf(a,b);
3 D();
4 Pf(a,b);
}
```

```
D()
{
 int b=6;
 E(a,b);
 Pf(a,b);
}
```

```
E(int b, int a)
{
 Pf(a,b);
 a = b+a;
 b = a-b;
 a = 7;
 b = 8;
Pf(a,b);
}
```



Data Segment



Static

3, 0	3, 0
0, 0	0, 0
4, 5	4, 5
6, 4	6, 4
7, 8	7, 8
4, 5	4, 5
4, 6	3, 5
7, 8	6, 4
	7, 8
	4, 6

Dynamic

3, 0
3, 0
4, 5
6, 4
7, 8
4, 6
4, 5
4, 5
6, 4
7, 8
4, 6

$a = b + a = 10$
$b = a - b = 2$

GATE →

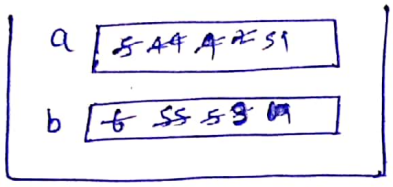
```
int a=5, b=6;
main()
{ auto int a=1;
  pf(a,b);
  C();
  a=2, b=3;
  E();
  pf(a,b);
}
```

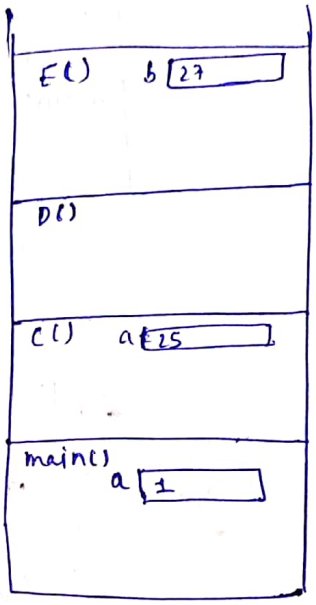
```
C()
{ register int a=25;
  static int b;
  D();
  pf(a,b);
  a=4, b=5;
}
```

```
D()
{
  extern int a;
  pf(a,b);
  E();
  a=44, b=55;
}
```

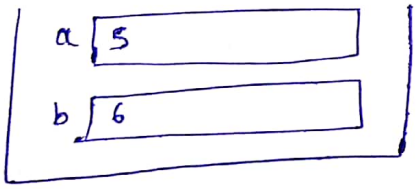
```
E()
{
  static int b=27;
  pf(a,b);
  a=59, b=69;
}
```

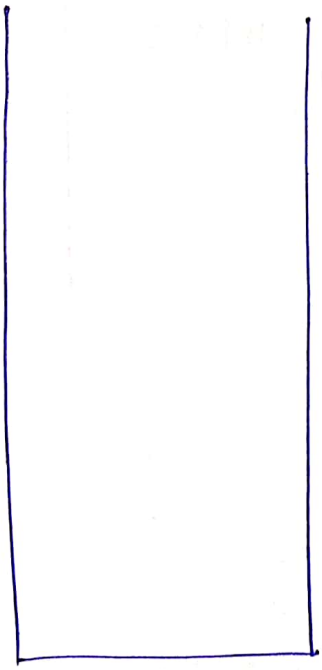
when static is occur then and value initilize the no value change in global like here a & b



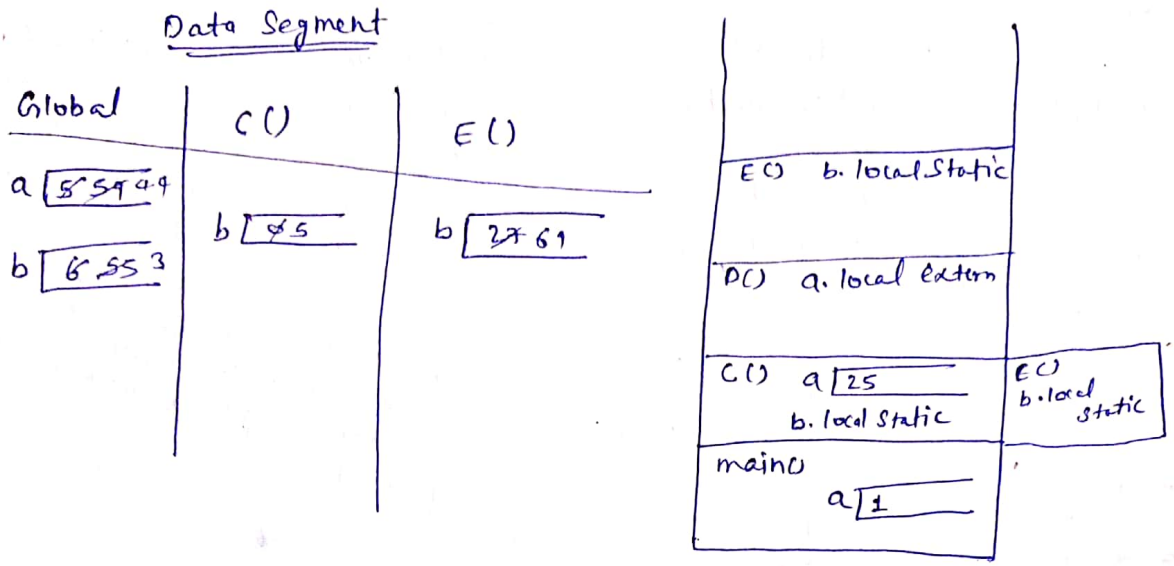


Static	Dynamic
1, 6	
5, 6	
5, 27	
25, 0	
2, 27	
1, 3	





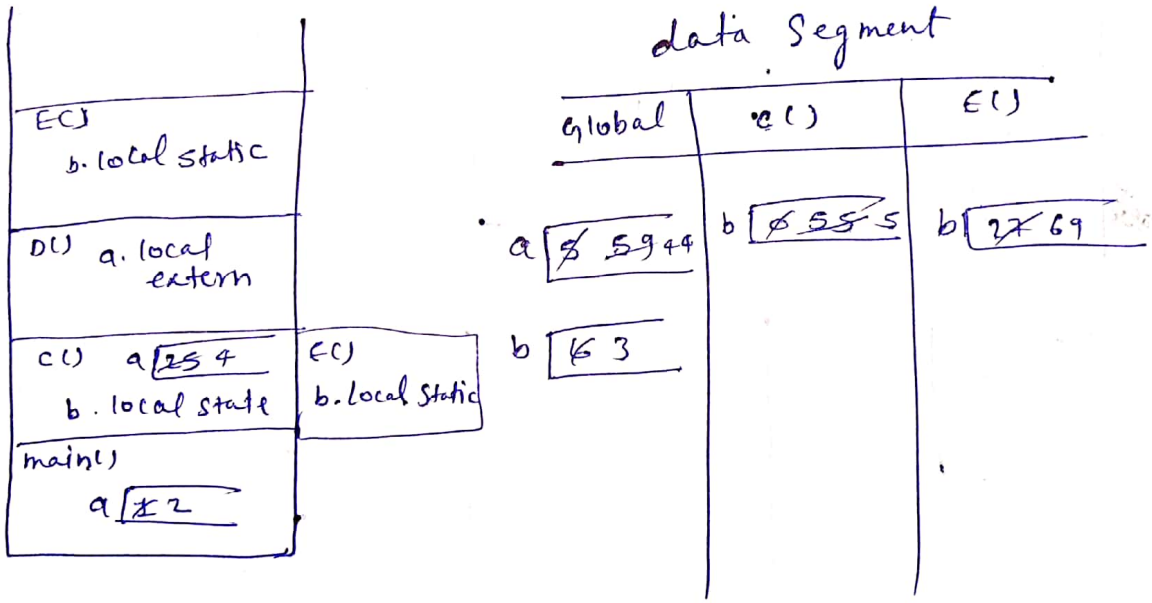
Static



O/P

1, 6
5, 6
5, 27
25, 0
4, 69
2, 3

Dynamic



O/P

1, 6
5, 0
5, 27
25, 55
2, 69
59, 3

POINTERS

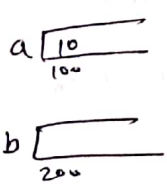
1. Pointer is a variable which is used to store the address.
2. The two different operators used in the pointers.
   (a) & → address of operator
   (b) * → indirection operator / value at location / dereference operator / object at location
3. Pointer variable takes same bytes of memory irrespective of its data type. (it can be int pointer, char pointer, float pointer)
4. Pointer variable can dereference (Access) no. of bytes depending on its data type. (int. pointer can dereference 2 bytes, char pointer can dereference 1 byte, float pointer can access 4 bytes)
5. Pointer variable is declared by using "*"
6. Address can not be negative.
7. Address size will change from system to system depending on the operating system

CONTENTS

- Basic
- Pointer to pointer
- pointer to array
- multi dimensional Array
- Pointer to String
- Array of pointer
- Pointer to structures, functions.

#BASIC

e.g.
```
int a=10;
int b;
b= &a;
```

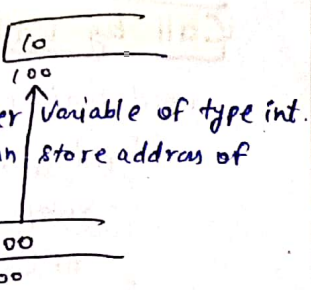


* This is not allowed because there is no pointer.

e.g.
```
int a=10;
int *p; // p is a pointer variable of type int.
p=&a;
```

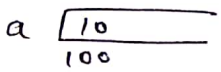
D ← int *p; // p is a pointer variable of type int.
A ← p=&a; where we can store address of any integers.

int *p=&a;
D, I




```
Pf(a); // 10
Pf(p); // 100
Pf(&a) // 100
Pf(&p) // 200
Pf(*p) // value at the location (10)
```

e.g. int a=10;

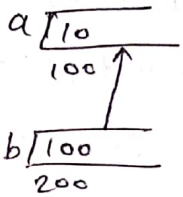


pf(a); = 10
pf(&a); = 100
pf(*&a); = 10

a = *&a

e.g. int a=10

int *b = &a;



pf(a) = 10;
pf(b) = 100;
pf(*b) = 10
pf(a *b) // error
10 10

If *b initilize with datatype then it is called pointer

pf(a * *b); 100
10 * 10

pf(* *b) //error
* (100)
(10)

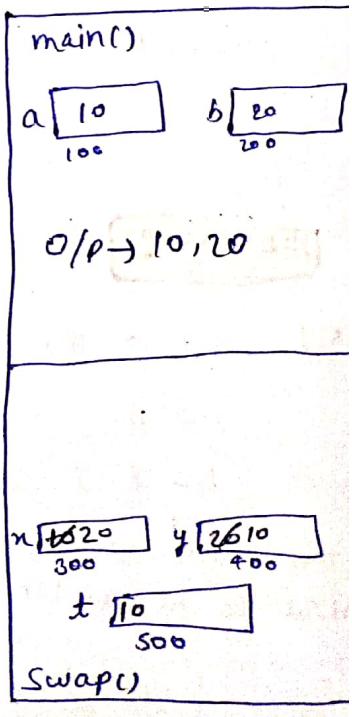
PARAMETER PASSING TECHNQUE :

- Call by value
- Call by reference or
- Call by address

Call by Value :

```
void main()
{ int a=10, b=20;
  swap (a,b);   -> actual parameter
  pf("%d", a,b);
}
```

```
void swap (int x, int y)   <- formal parameter
{ int t;
  t=x;
  x=y;
  y=t;
}
```

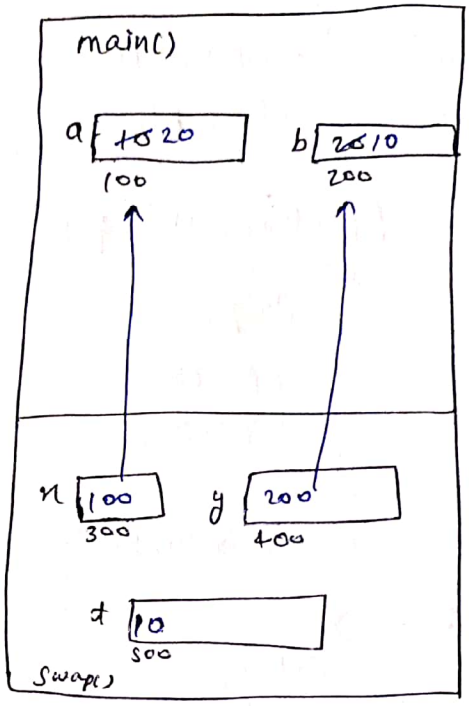


→ Values change in the formal parameters will not reflect in the actual parameters.

Call by address / reference :

```
void main ()
{ int a=10, b=20;
   swap (&a, &b);
   Pf("%d%d", a,b);
}
void swap ( int *x , int *y)
    { int t;
      t= *x;
      *x= *y;
      *y=t;
    }
```

o/p → 20,10



→ In the above program the actual parameters are getting changed because of call by address.

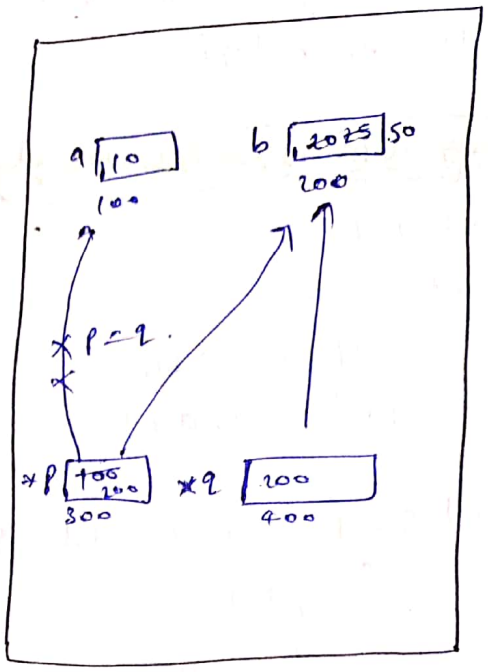
NOTE:

1. When the address is being passed reciver should be pointer variable.
2. C lang. supports only call by value but we can achive call by reference by ~~forcing~~ passing the address.
3. In the question if the callby reference is given then whenever parameter is passed in the function we need to understand implicitle address is being passed and reciver is pointer variable and actual parameter will be changed.

GATE →

```
int a=10, b=20
Void main()
  {
    f(&a, &b)
    pf(a,b);
  }
f(int *p, int *q)
  {
    p=q;
    *p=25;
    *q=50;
  }
```

o/p → 10, 50



GATE : →

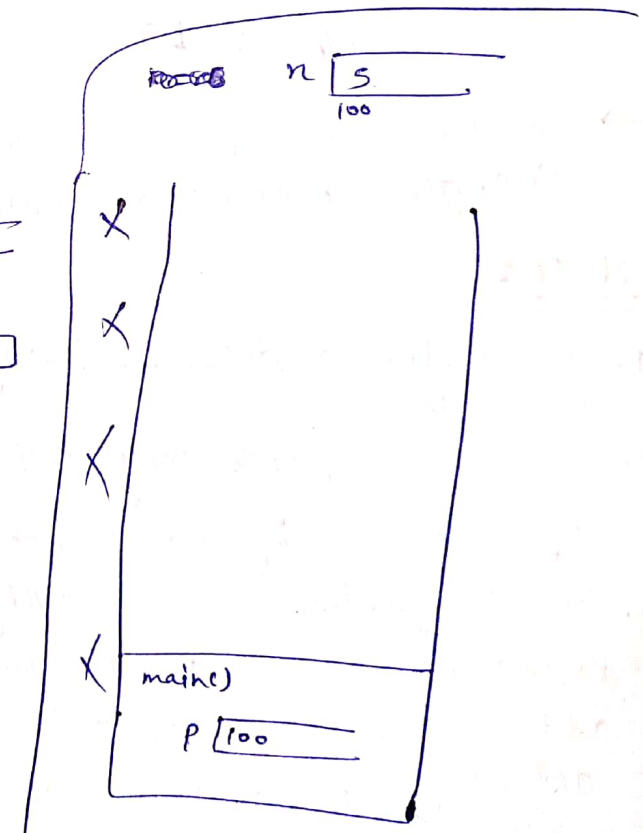
```
int n;
void Q(int z)
  {
    z = z+n;
    Print (z);
  }
void p(int *y)
  {
    int n = *y + 2;
    Q(n);
    *y = n-1;
    pf(n);
  }
main()
  {
    n=5;
    p(&n);
    pf(n);
  }
```

z = 12

n 5 6
100

y 100
200

n 7



*y = 100

n = 100 + 2 = 102

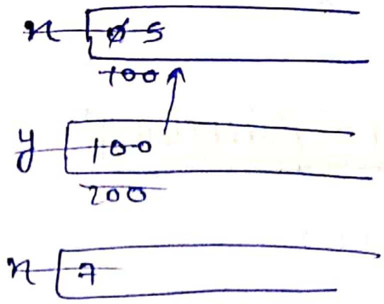
Q(n)

z = 5

z = 5 + 5 = 10

*y = 5 - 1 = 4

o/p → 12, 7, 6

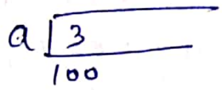


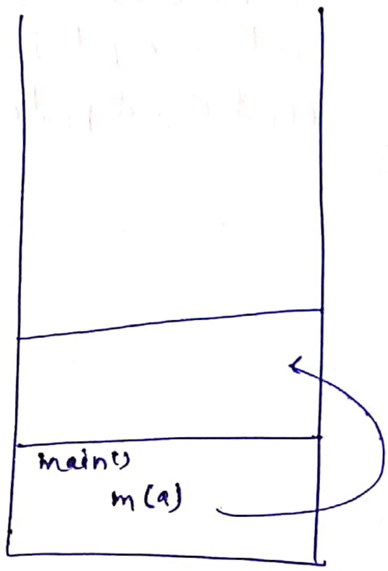
~~x = *y+2~~

~~x = 5+2=7~~

GATE →

```
a = 3   → 0,5
void n(x) // *x
{
    n = n*a; // *x = *x * a
                     2 * 2 = 4
    pf (n);  // *x
}
void m(y) // *y
{
    a = 1;        a [1 2 4]
    a = y - a; // a = *y - a;
                     3 - 1 = 2
    n(a);
    pf (a);
}
void main
{ m (a);

}
```





What will be the O/P of above program, when parameter are passed by reference and dynamic Scoping is assumed?

(a) 6, 2

(b) 6, 6

(c) 4, 2

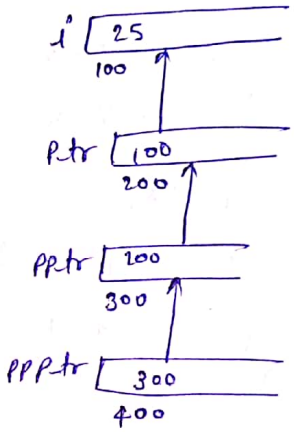
(d) ✓ 4, 4

Scanned with CamScanner

GATE ⟶

#POINTER TO POINTER

```
void main()
{
  int i;
  int *ptr; // It is called Single pointer it can dereference (access) one level
  int **pptr; // It is called as a double pointer or pointer to point it can access of two level
  int ***ppptr; // Triple pointer or pointer to "pointer to pointer" access upto 3
  i=25;
  ptr = &i;
  pptr = &ptr;
  ppptr = &pptr;
}
```



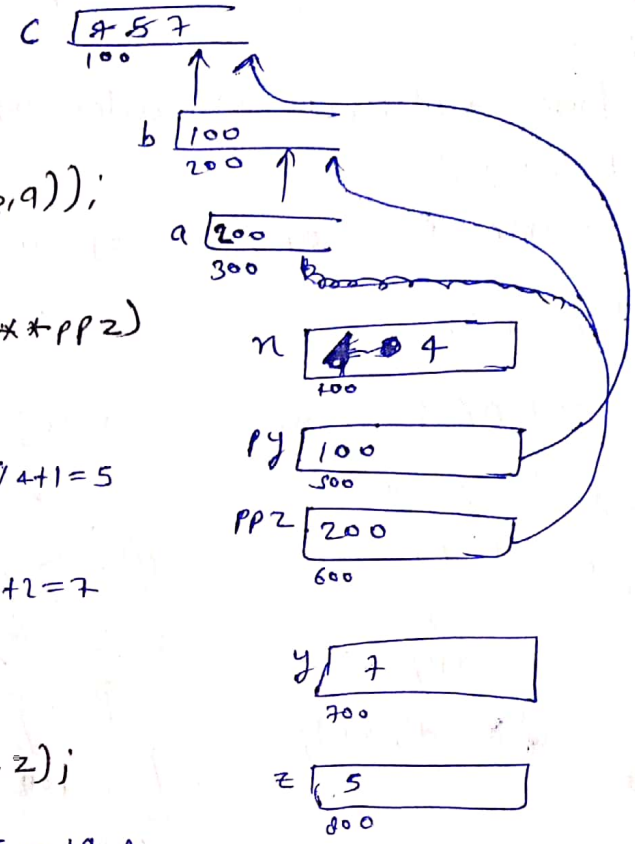
i — 25
ptr — 100
&i — 100
*ptr — 100
&ptr — 200
pptr → 200
*pptr → 100
**pptr → 25
ppptr → 300
&ppptr → 200
*ppptr → 200
**ppptr → 100
***ppptr → 25
****ppptr → error
*****&ppptr → 25

↓
value at the location

e.g. Void main()

```
{ int c, *b, **a;
  c = 4;
  b = &c;
  a = &b;
  Pf("%d", f(c,b,a));
}
int f(int n, int *py, int **ppz)
{ int y, z;
  **ppz += 1 // 4+1=5
  z = **ppz;
  *py += 2; // 5+2=7
  y = *py
  n = n+3;
  return (n+y+z);
}
```

7+7+5 = 19 Ans



e.g. Void main()

```
{ int a;
  int *b = &a;
  Scanf("%d", b);
  Pf("%d", *b + 10);
}
```

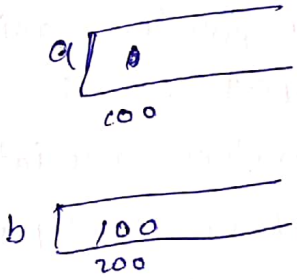
what is the printed?

(a) addres of a + 10;

(b) value entered + 10;

(c) addres of b + 10;

(d) error



Note:-

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

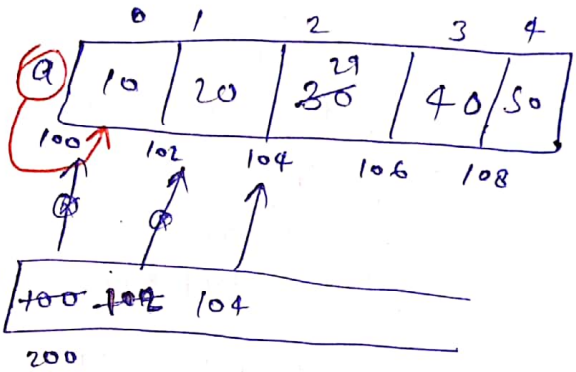
1. Array name without subscript contain base address of array
2. This base address is consider as constant and can not be modified.
3. Array name is consider as a constant point.

i.e.

```
void main()
{
    int a[5] = {10, 20, 30, 40, 50}
    int *p = a;
    ++p;
    ++p;
    --*p;
    pf("%d", *p);
}
```

int *p
p = a
or
int *p = &a[0]

same



++p ✓
++a ✗

{pf(p) = 100
pf(a) = 100}

o/p → 29

NOTE:

1. when the pointer variable is incremeted it will increment with respect to size of the element to which it is pointing.
2. Unary operators associativity is right to left.
3. In the above example 'p' and 'a' both are pointers to the array but 'p' is a pointer variable and 'a' is a pointer constant.

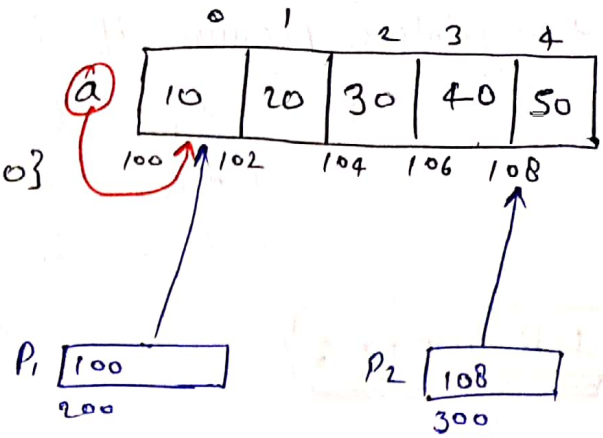
i.e ++p → allowed
--p → allowed
++a → Not allow
--a → Not allow

4. Array name is given as constant point because that is the only source in the memory how the array contains can be accessed.

POINTER ARITHMATIC :

```
void main()
{
  int a[5] = {10, 20, 30, 40, 50}
  int *P1 = a;
  int *P2 = a+4;
  Pf("%d", P2-P1);
}
```

O/P → 108 − 100
= 8
= $\frac{8}{2}$ = 4 ✓



Pf(P1) = 100
Pf(a) = 100
Pf(P1+1) = 102
Pf(a+1) = 102

{ Variable name → inside value
Array name → Base Address }

1. we can add integer constant to the pointer it means skiping the elements in the forward direction

e.g. P1 + 1 → allow
a + 1 → allow

2. we can subtract integer constant to the pointer it means skiping the elements in the backword direction.

e.g. P2 − 1 → allow
108 − 1 = 106

3. we can perform subtraction blw two pointers. this is allowed only when both are pointing to same anay. and P2 >= P1. This operation gives no. of elements from P2 before P1

e.g. P2 − P1
108 − 100 = 8
$\frac{8}{2}$ = 4
(2 ← Size of elem.)

4. we cannot add floating point no. to the pointer because it has no meaning.

e.g. P1 + 2.5 ✗

5. we cannot perform addition, multiplication and division between two pointers. because it is no meaning.

e.g: $P_1 + P_2 \rightarrow$ ✗

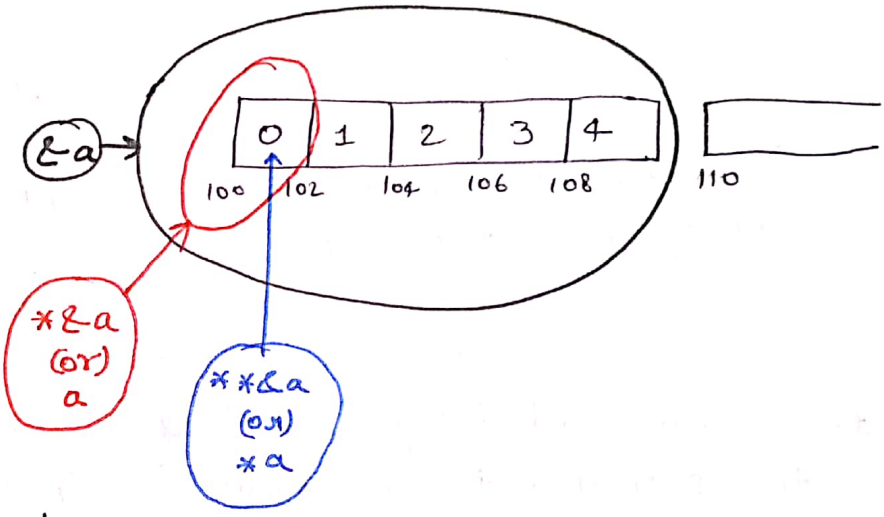
$P_1 * P_2 \rightarrow$ ✗

$P_1 / P_2 \rightarrow$ ✗

1-D ARRAY :

```
int a[5] = {0,1,2,3,4}
```

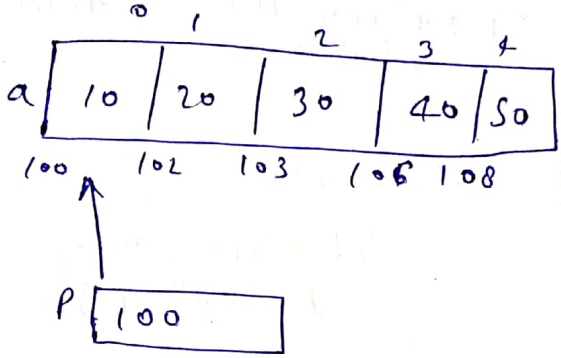
a[3]	a[5]
*(a+3)	*(a+3)
*(3+a)	*(100+3)
3[a]	*(106)
	3



EXPRESSION	PRINT VALUE	TYPE	MEANING
&a	100	int *[]	Base Addres of Entire 1D Array
*&a	100	int *	Base Addres of zero th elemenent
a	100	int *	" " " " " "
*a	0	int	~~BA~~. Value of the 0th element
**a	error	?	Out of the array memory.
&a+1	110	int *[]	BA of the next continous one-D array
a+1	102	int *	BA of the first element of array
*(a+3)	3	int	BA of the 3rd element of array
**&a	0	Int	BA of the 0st element of array

1. In the C lang. if we want to pass an array elements to the function then it is not a good idea to pass all the array elements instead by simply passing ~~all~~ ~~the array elements~~ Base address of array we can access all the array elements.

```
Void main()
  { int a[] = {10,20,30,40,50}
    display(a,5);
  }
Void display(int *p, int n)
  { int i;
    for(i=0; i<n; i++)
      pf("%d", p[i]);
  }
```



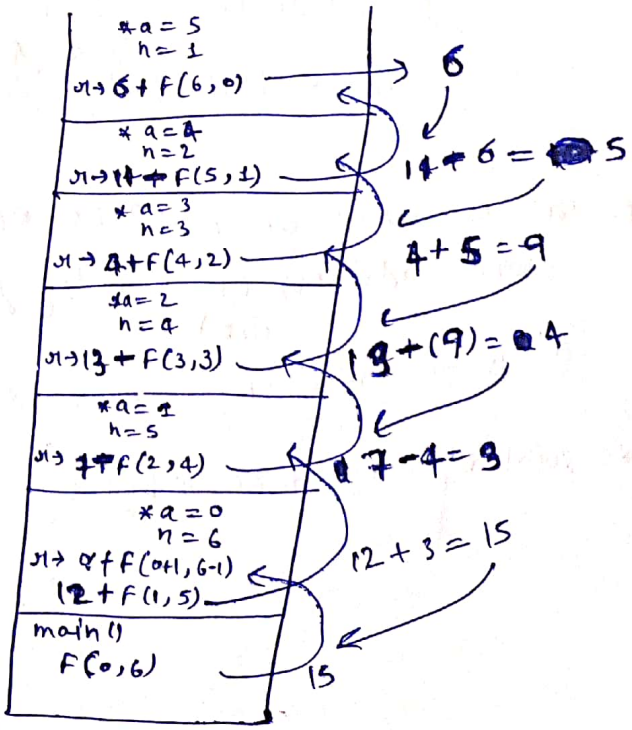
o/p → 10,20,30,40,50

p[0]	p[1]
*(p+0)	*(p+1)
*(100+0)	*(100+1)
*(100) = 10	*(102)
	20

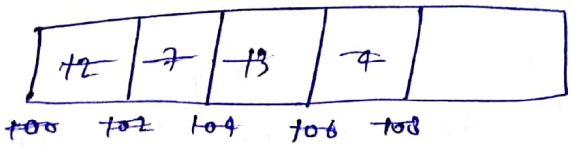
GATE

```
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);
  }
Void main
  { int a[6] = {12,7,13,4,11,6}
    pf("%d", f(a,6));
  }
```

o/p → 15



f(*a, n)



NOTE :

1. If array is initilized then remaining all will be stored as zero if it is not initilized then it contains garbage value

e.g ① void main()
{
int a[5];
o/p → garbage value
pf("%d", a[2]);
3

② void main()
{
int a[5] = {};
pf("%d", a[2]);
3
o/p → 0

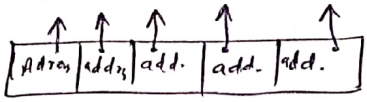
③ void main()
{
int a[5] = {10,20}
pf("%d", a[3]);
3
o/p → 0

ARRAY OF POINTER'S

just like array of integers we can also have array of pointers

e.g int a[5];

e.g. int *a[5];



1. 'a' is array of pointers of type of integer where we can store address of 5 integers.

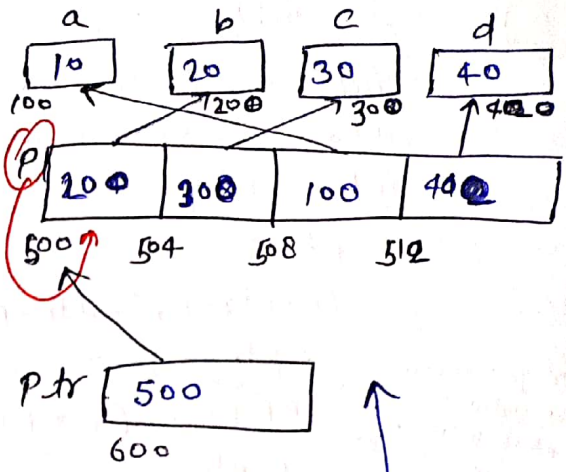
(OR)

'a' is a array of 5 element where every element is integer pointer

* pointer array take 4 bytes

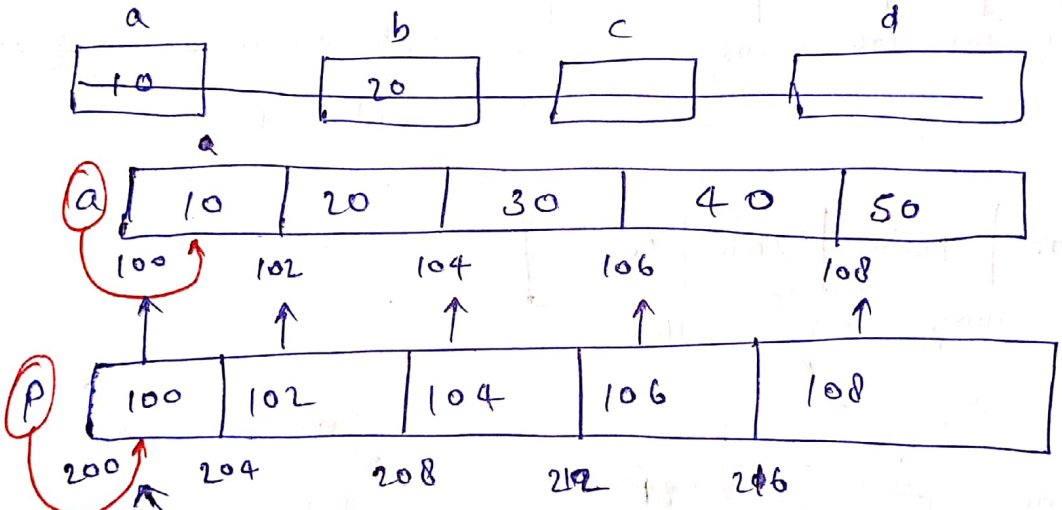
e.g① main()

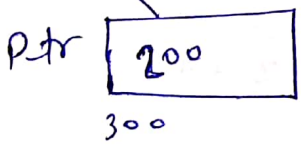
```
{ int a=10, b=20, c=30, d=40;
  int *p[4] = {&b, &c, &a, &d};
  int **ptr = p;
}
```



e.g② main()

```
{
  int a[5] = {10, 20, 30, 40, 50}
  int *p[5] = {a, a+1, a+2, a+3, a+4}
  int **ptr;
  ptr = p
```





pf(a) = 100
pf(p) = 200
pf(ptr) = 200
pf(*ptr) = 100
pf(ptr-p) = 200-200 = 0/4 = 0
pf(*p) = 100
pf(*p-a) = 100-100 = 0/2 = 0

pf(a) = 10
pf(p) = 500
pf(ptr) = 500
pf(p+1) = 504
pf(ptr+1) = 504
pf(*ptr) = 200

e.g.

```
void main()
{
  int a[5] = {0,1,2,3,4}
  int *p[5] = {a, a+1, a+2, a+3, a+4}
  int **ptr = p
    ptr++;
    pf(ptr-p, *ptr-p, **ptr);
    *ptr++;
    pf(ptr-p, *ptr-a, **ptr);
    *++ptr;
    pf(ptr-p, *ptr-a, **ptr);
    ++*ptr;
    pf(ptr-p, *ptr-a, **ptr);
}
```

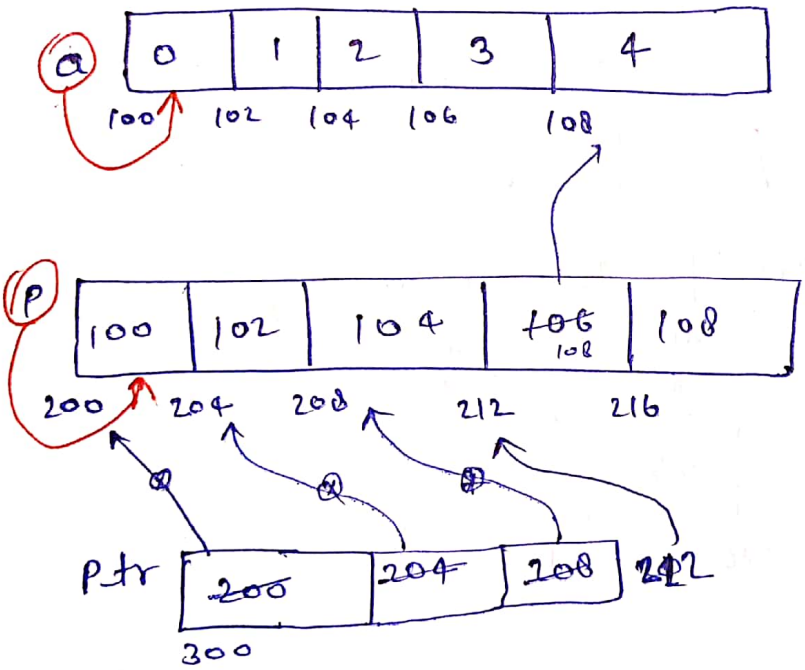
If you assign the value then *ptr++ value initilize otherwise * is useless

ptr - p = 204 - 200 = 4/4 = 1
*ptr - a = 102 - 100 = 2/2 = 1
**ptr = 1
} 1 PF

ptr - p = 208 - 200 = 8/4 = 2
*ptr - a = 104 - 100 = 4/2 = 2
**ptr = 2
} 2 PF

ptr - p = 212 - 200 = 12/4 = 3
*ptr - a = 106 - 100 = 6/2 = 3
**ptr = 3
} 3 PF

ptr - p = 212 - 200 = 12/4 = 3
*ptr - a = 108 - 100 = 8/2 = 4
**ptr = 4
} 4 PF

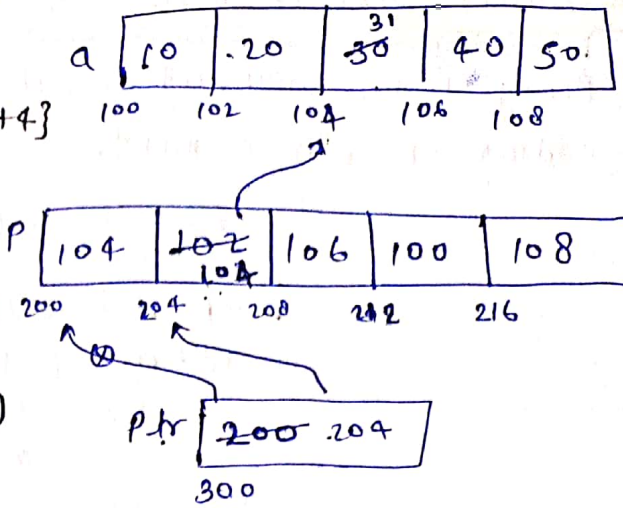


O/P

1 1 1
2 2 2
3 3 3
3 4 4

e.g. Void main()

```
{ int a[5] = {10,20,30,40,50}
int *p[5] = {a+2, a+1, a+3, a+4}
int **ptr = p;
**ptr++;
pf(ptr-p, *ptr-a, **ptr)
*++*ptr;
pf(ptr-p, *ptr-a, **ptr)
++**ptr;
pf(ptr-p, *ptr-a, **ptr);
}
```



ptr - p = 204 - 200 = 4/4 = 1
*ptr - a = 102 - 100 = 2/2 = 1
**ptr = 20

ptr - p = 204 - 200 = 4/4 = 1
*ptr - a = 104 - 100 = 4/2 = 2
**ptr = 30

ptr - p = 204 - 200 = 4/4 = 1
*ptr - a = 104 - 100 = 4/2 = 2
**ptr = 31

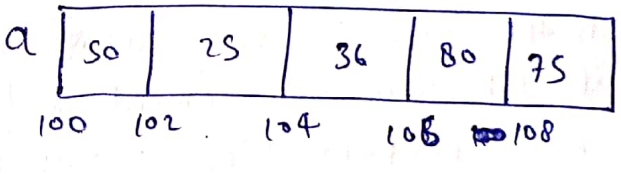
O/P

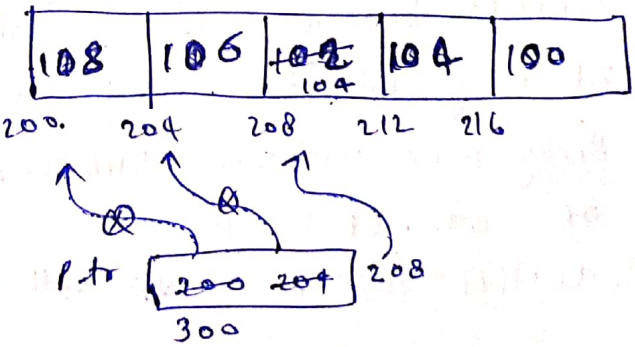
1, 1, 20
1, 2, 30
1, 2, 31

---

e.g. Void main()

```
int a[5] = {50, 25, 36, 80, 75}
int *p[5] = {a+4, a+3, a+1, a+2, a}
int **ptr = p;
**ptr++;
pf(ptr-p, *ptr-a, **ptr);
*++ptr;
pf(ptr-p, *ptr-a, **ptr);
++*ptr;
pf(ptr-p, *ptr-a, **ptr);
++**ptr;
pf(ptr-p, *ptr-a, **ptr);
}
```





208 - 200 = 8/4 = 2
= 2
= 36

= 2
= 2
= 37

204 - 200 = 4/4 = 1
106 - 100 = 6/2 = 3
= 80

208 - 200 = 8/4 = 2
102 - 100 = 2/2 = 1
= 25

MULTI-DIMENSIONAL ARRAYS

1. Array name without subscript always gives constant base address of 0th element.

0th element

→ If it is a 1D array then 0th element means base address of the normal element.

e.g. int a[5];

a → int *;

→ If it is 2D array then 0th element means base address of 0th 1D Array.

e.g. int a[5][6];

a → int *[];

→ If it is 3D array then 0th element means base address of the 0th 2D array.

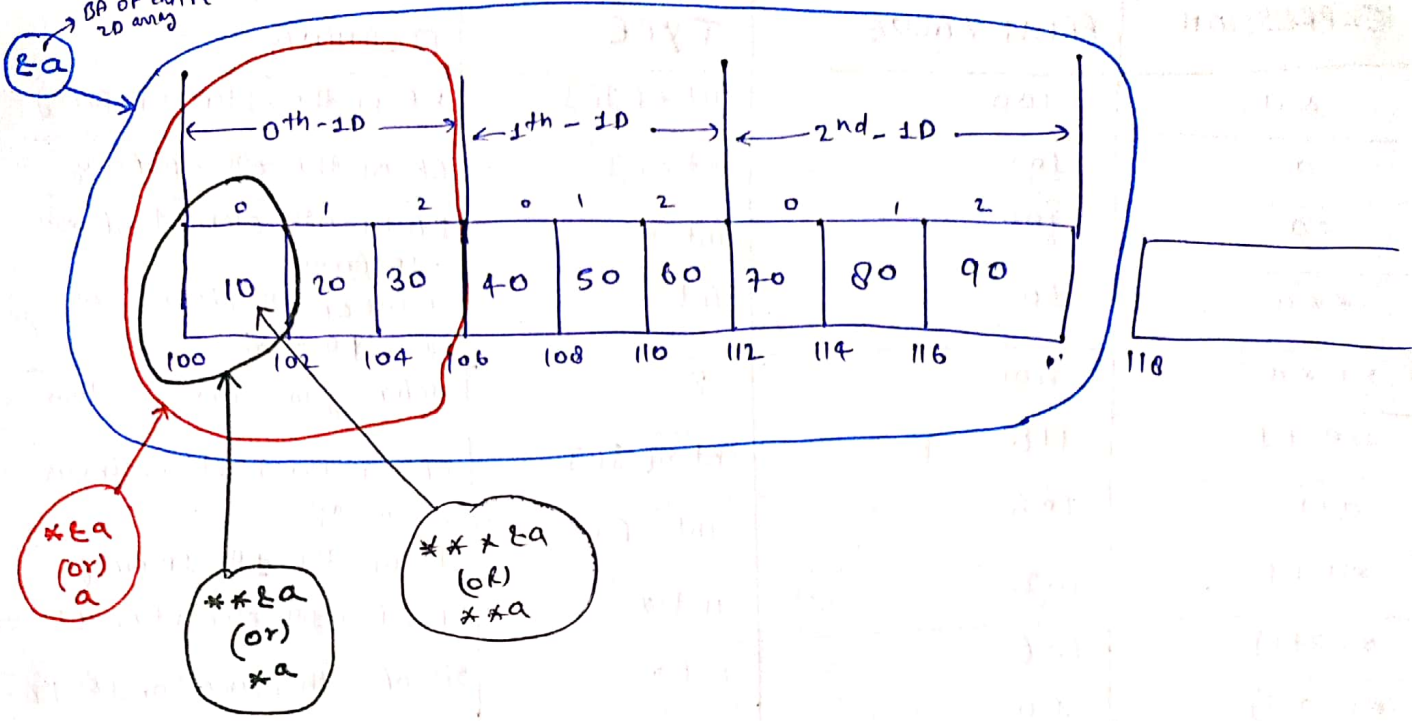
e.g. int a[4][6][5];

a → int *[][];

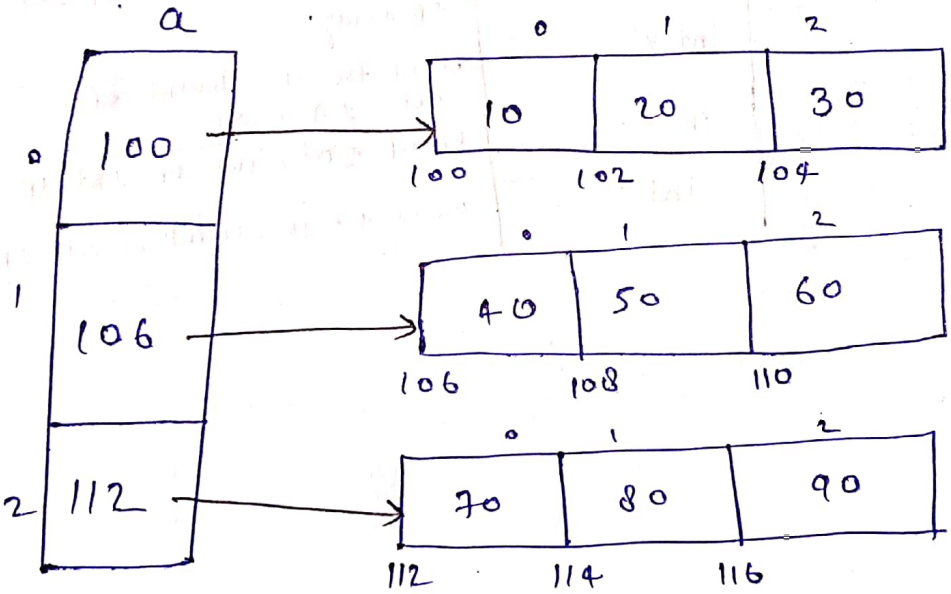
2-D ARRAY :

int a[3][3] = {1,2 --- 9};

1. A good example of 2D array is pointer to pointer.
2. In the above example it represent 3 rows and 3 columns.
3. * In the two-D array a[i] represence constant base addres of ith row, 0th element and it can not be changed.
4. * If we perform a[i] = 2000; it gives error bcoz we can not modify constant base address of ith row, 0th element.
5. 2D array will be stored in the form of multiple 1D arrays
6. Array name without subscript can gives constant base address of 0th 1-D array.
7. a[i][j] = 20; it means ith row, jth column will replace with 20.



PHYSICAL VIEW



LOGICAL VIEW

EXPRESSION	PRINT VALUE	TYPE	MEANING
&a	100	int *[][]	BA of the entire 2D Array
a	100	int *[]	BA of the 0th 1D Array
*a	100	int *	BA of 0th element of 0th 1D Array
**a	10	int	Value of 0th element of 0th 1-D array
***a	error	?	Refering unknown location
&a+1	118	int *[][]	BA of the next continous 2D array
a+1	106	int *[]	BA of the 1th 1D Array
*a+1	102	int *	BA of 1th element of 0th 1
*(a+1)	106	int *	BA of 0th element of 1st 1D-
**(a+1)	40	int	Value of 0th element of 1st 1D-
*(a+2)+1	114	Int *	BA of the 1st element of the 2nd 1D array
a[0] = *(a+0) *a	100	int *	BA of the 0th element of 0th 1D array
*(a+1)+2	110	int *	BA of the 2nd element of 1st 1-D array
*(a+0)+2 or *a+2	104	int *	BA of 1st elem of 1st 1D
***&a	10	int	Value of 1st element of 0th 1D

a[1][1]

= *(*(a+1)+1)

= *(*(100+1)+1) → 0th 1D

= *(*(106)+1) → 1st 1D

= *(106+1) → 0th element, 1st 1D

= *(108) → 1st element of 1D

= 50

GATE

```
void main()
{
int a[5][3] = {10, 20, 30 ---150};
Pf("%d", ((a == a[0]) && (a[0] == *a)));
}
```

what is the o/p ?

(a) 1 (b) 0 (c) compiler error (d) Run time error

GATE

```
Void main()
{
int a[4][5] = {{1,2,3,4,5},0
               {6,7,8,9,10},1
               {11,12,13,14,15} 2
               {16,17,18,19,20}} 3
Pf("%d", *(*(a + **a + 2) + 3));
}
```

1

o/p → 19

((a + 3) + 3)

100 + 3 = 130 + 3 = 136

*(a + 1 + 2)

*(a + 2)

[(*106) + 3]

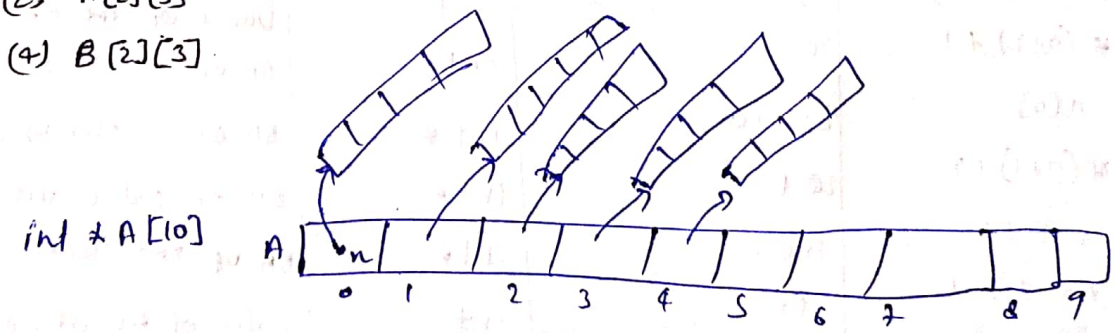
(4 + 3)

106 + 3

(6) int * A[10], B[10][10]

(1) A[2] (2) A[2][3]

(3) B[1] (4) B[2][3]



A[2][3] int n(5) = {1,2,3,4,5}

n = int *

n = 40

HW ① int [4][5] = {0, 1, 2, ---19}, B.A. = 2000, size of element = 2B

print all expressions.

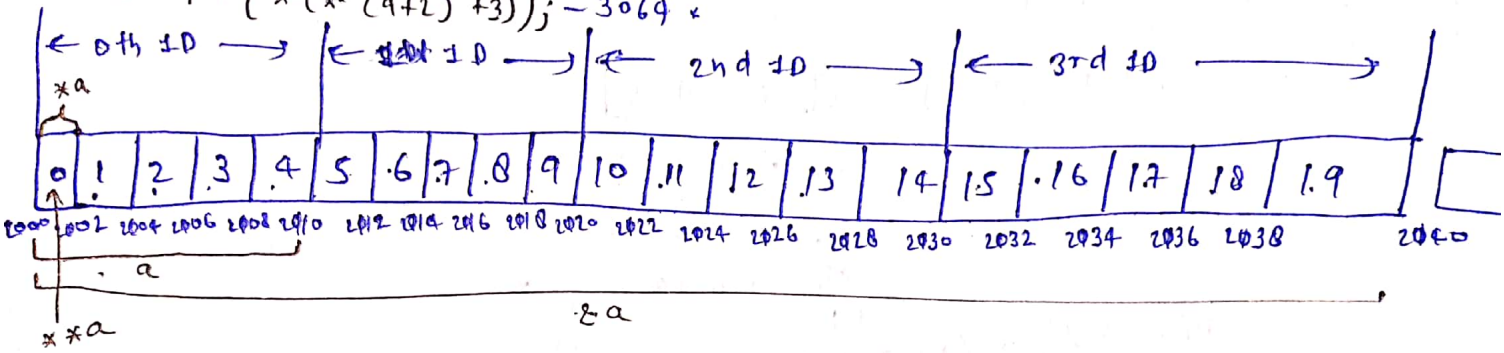
② int [5][6] = {0, 1, 2, ---29}, BA = 3000, Size of element = 4B

(a) pf (*(a+4)); — 3106 *

(b) pf (*(a+3)+2); — 3088

(c) expand a[4][5]; — 3126 *

(d) pf (*(*(a+2)+3)); — 3069 *

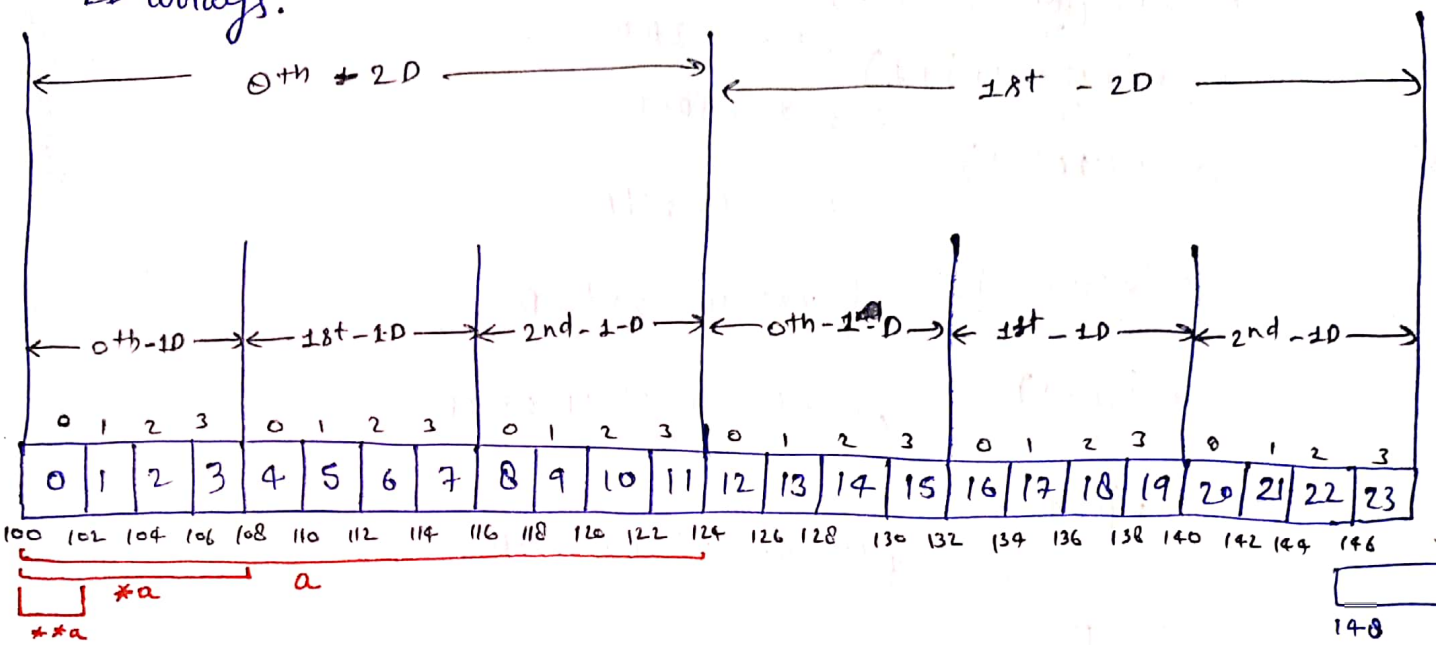


EXPRESSION	PRINT-VALUE	TYPE	MEANING
&a	2000	int*[][]	BA of the whole array
a	2000	int*[]	BA of the 0th 1D A
*a	2000	int*	BA of 0th elment of 0th 1D A
**a	0	int	Value of the 0th element of 0th 1D A
***a	error	?	refering unknown location
&a+1	2040	a|int*[][]	BA of the next continous 2D Arr.
a+1	2010	int*[]	BA of 0th element of 1st 1D A
*a+1	2002	int*	BA of 1st elemet of 0th 1D A
*(a+1)	2010	int*	BA of 0th elemet of 1st 1D A
**(a+1)	5	int	Value of the 0th element of 1st 1D A
*(a+2)+1	2022	int*	BA of the 1st element of 2nd 1D A
a[0]	2000	int*	BA of 0th element of 0th 1D A
*(a+1)+2	2014	int*	BA of 1st elemet of 1st 1D A
*a+2	2004	int*	BA of 2nd elemet of 0th 1D A
***&a	0	int	value of the 0th elment of 0th 1D A

3-D ARRAY

int a[2][3][4] = {0,1,---23};

1. 3D array will be stored in the memory in the form of multiple 2D arrays.



EXPRESSION	PRINT VALUE	TYPE	MEANING
&a	100	int *[][][]	BA of entire 3D array
a	100	int *[][]	BA of 0th 2D array
*a	100	int *[]	BA of 0th 1D array of 0th 2D array
**a	100	int *	BA of 0th element of 0th 1D of 0th 2D ar
***a	0	int	Element of value of 0th 1D of 0th 2D
****a	error	?	Refer Unknow location
&a+1	148	int *[][][]	BA of next continous 3D array
a+1	124	int *[][]	BA of 1st 2D array
*a+1	108	int *[]	BA of 1st 1D array of 0th 2D array
**a+1	102	int *	BA of 0th of 1D arr of 0th 2D ar 1st element
*(a+1)	124	int *[]	BA of 0th 1D of 1 2D array
((a+1)+1)+2	136	int *	BA of the 2nd element of 1D array of 1st 2D array
a[1]+2	140	int *[]	BA of the 0th 2nd 1D array of 1st 2D ar
**a+3	106	int *	4th element, 0th 1D 0th 2D
*(*a+2)+2	120	int *	BA of 3rd
****&a	0	int	

a[1][1][1]

((*(a+1)+1)+1)

((*(100+1)+1)+1) → 0th 2D BA

((*(124)+1)+1) → 1st 2D BA

((124)+1) → 0th 1D, 1st 2D

((132)+1) → 1st 1D, 2nd 2D

*(132+1) → 0th element of 1D of 2D

*(134) → 1st elem, of 1D, 1st 2D

17

a[1][2][3]

- 3rd element
- 2nd 1D
- 1st 2D

H.W (1) int a[3][4][5] = {0, --- 59}; BA = 3000; size of element = 2B
Print all expression (16)

(2) int a[4][5][6] = {0, 1, --- 119}; BA = 4000; size of element = 4B

(a) pf(*(a+2)+3) = ?

(b) pf(*(*(a+3)+2)+1 = ?

(c) pf(*(*(a+1)+3)+4) = ?

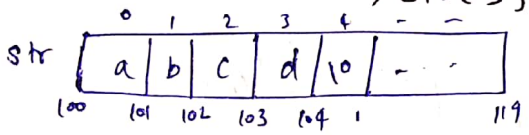
(d) expand a[3][2][3]

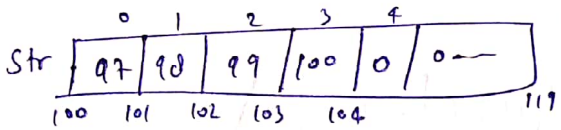
STRINGS :

1. Character array is called as a string. String always ends with the null(\0)character
2. null (\0) character ASCII value is the '0'
3. String name always gives base address.
4. Every character of the string will be stored in the memory. sequentially in the form of their ASCII values.

e.g: void main()
{ char str[20] = {'a', 'b', 'c', 'd', '\0'};
pf("%c %c %c %c %c", str[0], str[1], str[2], str[3]);

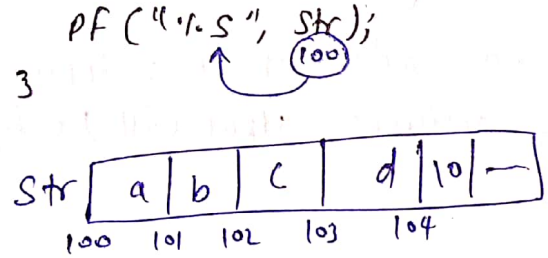
A - 65
a - 97





① %c → str[0] = a
② %d → str[0] = 97
③ If we are reading str[0] then we need to consider ASCII value of the respective character.

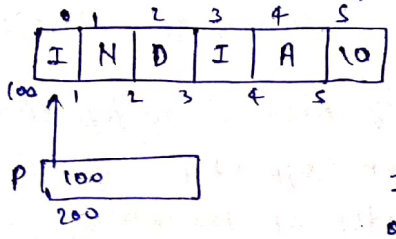
e.g: Void main()
{ char str[20] = "abcd";
pf("%s", str);
}



① we need to provide base address to the %s
② From the base address provided it will print all the characters until the null character

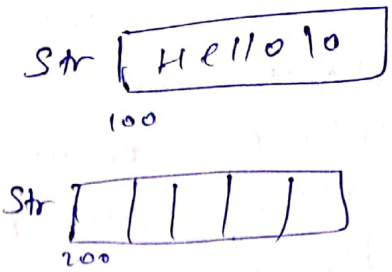
INITIALIZATION OF STRING :

1. char S[6] = "INDIA"; → Explicitly we need to provide the size for null character other wise it may print some garbage value.
2. char S[] = "INDIA"; → Implicitly it will calculate the size and allocate the memory accordingly
3. char *p = "INDIA"; → The meaning of the 3rd statement is we are requesting the compiler to allocate the memory for that String INDIA and the Base address of that String will be stored in the pointer variable P. It is consider as string constant and content of the string constant cannot be modified.

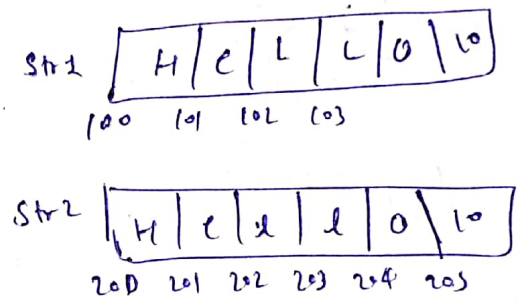


* P[1] = 'A' → this is not allowed in point 3rd

```
{ char str1[10] = "Hello";
  char str2[10];
  char *s = "INDIA";
  char *q;
  str2 = str1;   x
  q = s;
}
```



②
```
main()
{ char str1[10] = "Hello";
  char str2[10] = "Hello";
  if (str1 == str2)
    pf("Hi");
  else
    pf("Bye")
}
```

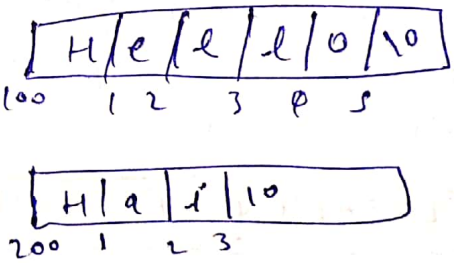


o/p → bye

* we are comparing the base addresses of 2 different strings. because they are different output will be bye

③
```
main()
{
  if(*"Hello" == *"Hai")
    pf("Hi");
  else
    pf("Bye")
}
```

o/p → Hi



* we are comparing only 1st character of string with base address with the help of ASCII

* If the string is given in double coutes "" it means base address is given
* In above example we are comparing only first character of bothe the string beccoze is this same then o/p → Hi
* we are not comparing remaining character of the string.

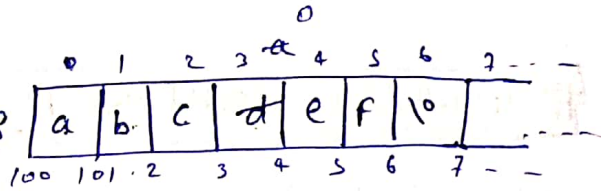
NOTE:

General functions available in the <string.h>

1. Strcy(T,S) → copy source string to target string
2. Strrev(S); → Reverse given string.
3. Strupr(s); → Convert all the character to upper case.
4. Strlwr(s); → Convert all the character to lower case.
5. Strlen(s). → String length. (actual length of the string)

e.g. char *S = "INDIA";
Strlen(S) = 5

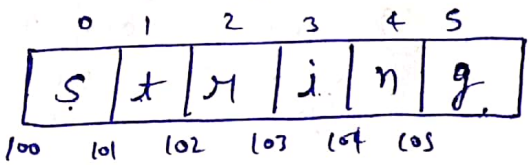
e.g. void main()
{ char S[20] = "abcdef";



Pf("%s", S); // abcdef
S[3] = 97;
Pf("%s", S); // abcaef
S[3] = '0';
Pf("%s", S); // abc
S[3] = '0';
Pf("%s", &S); // abc0ef
Pf("%s", *S); // error → we provide %s to base address
Pf("%c", S+3); // 103 ← 100+3, error → you should provid ASCII value to %c
PF("%s", S+3); // 0ef
Pf("%c", *(S+3)); *(100+3) = *103 = 0
}
↳ for ASCII value ↳ we also give ASCII value

A - 65
a - 97

GATE char P[20];
char *S = "String";
int length = Strlen(S);
For(i=0; i<=length; i++)
P[i] = S[length-i];
Pf("%s", P);
what is o/p ?

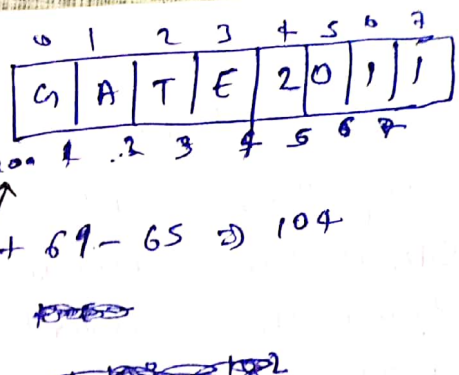


P[0] = S[6]
P[1] = S[5]
P[2] = S[4]
P[3] = S[3]
P[4] = S[2]
P[5] = S[1]
P[6] = S[0]

(a) gniets
(b) niets
(c) string
(d) No output

GATE

```
char *p = "GATE 2011";
Pf("%s", p + p[3] - p[-1]);
```



(a) GATE 2011
(b) E 2011
(c) 2011
(d) 011

A = 1
B = 2
C = 3
D = 4
E = 5

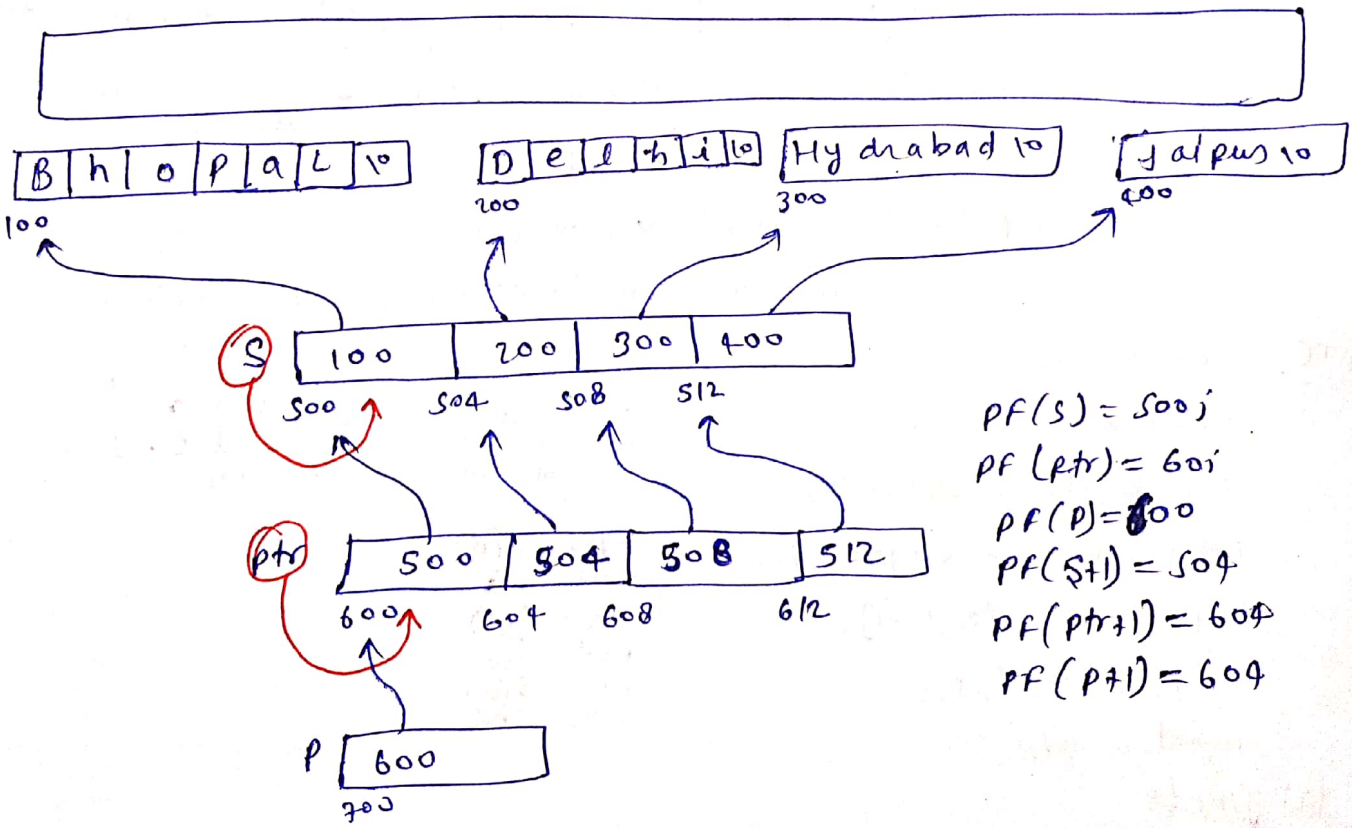
100 + 5 - 1 = 104

ARRAY OF POINTERS TO STRING:

e.g.

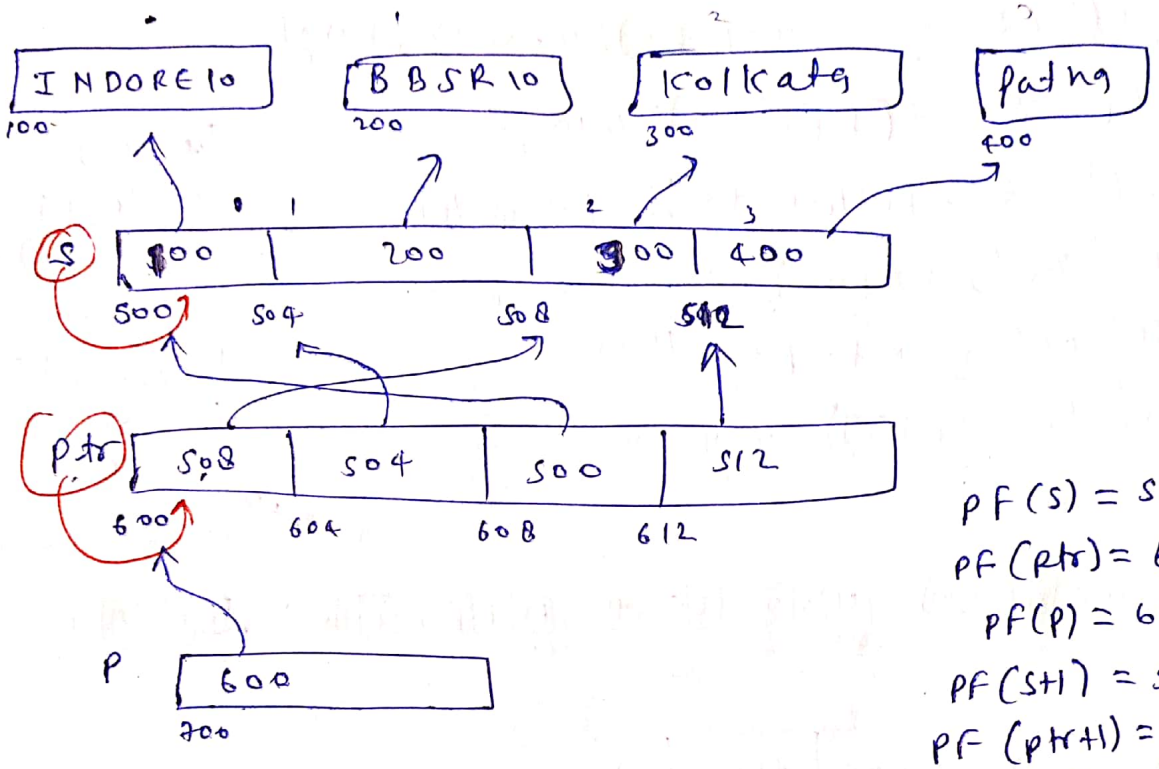
```
main()
{
  char *S[4] = {"Bhopal", "Delhi", "Hydrabad", "jaipur"};
  char **ptr[4] = {S, S+1, S+2, S+3};
  char ***P = ptr;
}
```

If we remove * then this give error becoz you need pointer to store address



Pf(S) = 500;
Pf(ptr) = 600
Pf(P) = 700
Pf(S+1) = 504
Pf(ptr+1) = 604
Pf(P+1) = 604

e.g.
```
main()
{ char *S[4] = {"Indore!", "BBSR", "Kolkata", patna"};
  char **ptr[4] = {S+2, S+1, S, S+3};
  char ***P;
    P = ptr;
}
```



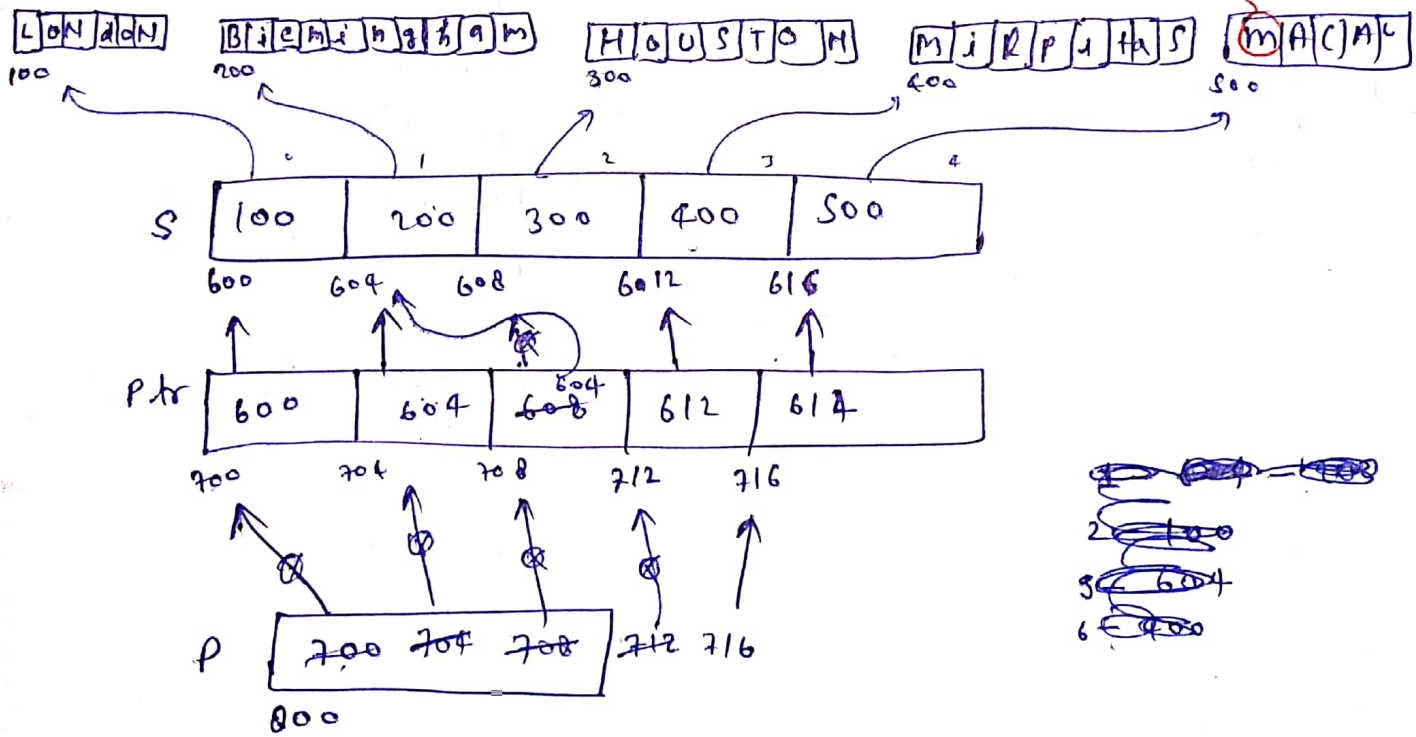
pF(S) = 500
pf(ptr) = 600
pf(P) = 600
pf(S+1) = 504
pf(ptr+1) = 512
pf(P+1) = 604
pf(*P) = 508
pf(*ptr) = 508

GATE

```
Void main()
{
  char *s[5] = {"London", "Biemingham", "Houston", "milpitas", "macaw"};
  char **ptr[5] = {s, s+1, s+2, s+3, s+4};
  char ***p = ptr;
1 pf("%s", **++p);      // 200, Biemingham
2 pf("%s", *--*++p+3);  // 200+3, mingham
3 pf("%s", *[p[-2]+3]); // 103, don
4 pf("%s", p[-1][-1]+1); // 101, ondon
5 pf("%s", **p++);      // 200, Biemingham
6 pf("%c", ***++p+2);   // 0
7 pf("%c", s[1][2]);    // m, 202
}
```

```
*[p[-2]+3]
*(*(p-2)+3
*(*(708-2))+3
*(*(700)+3
*(600)+3
100+3
=103
```

m+2=0



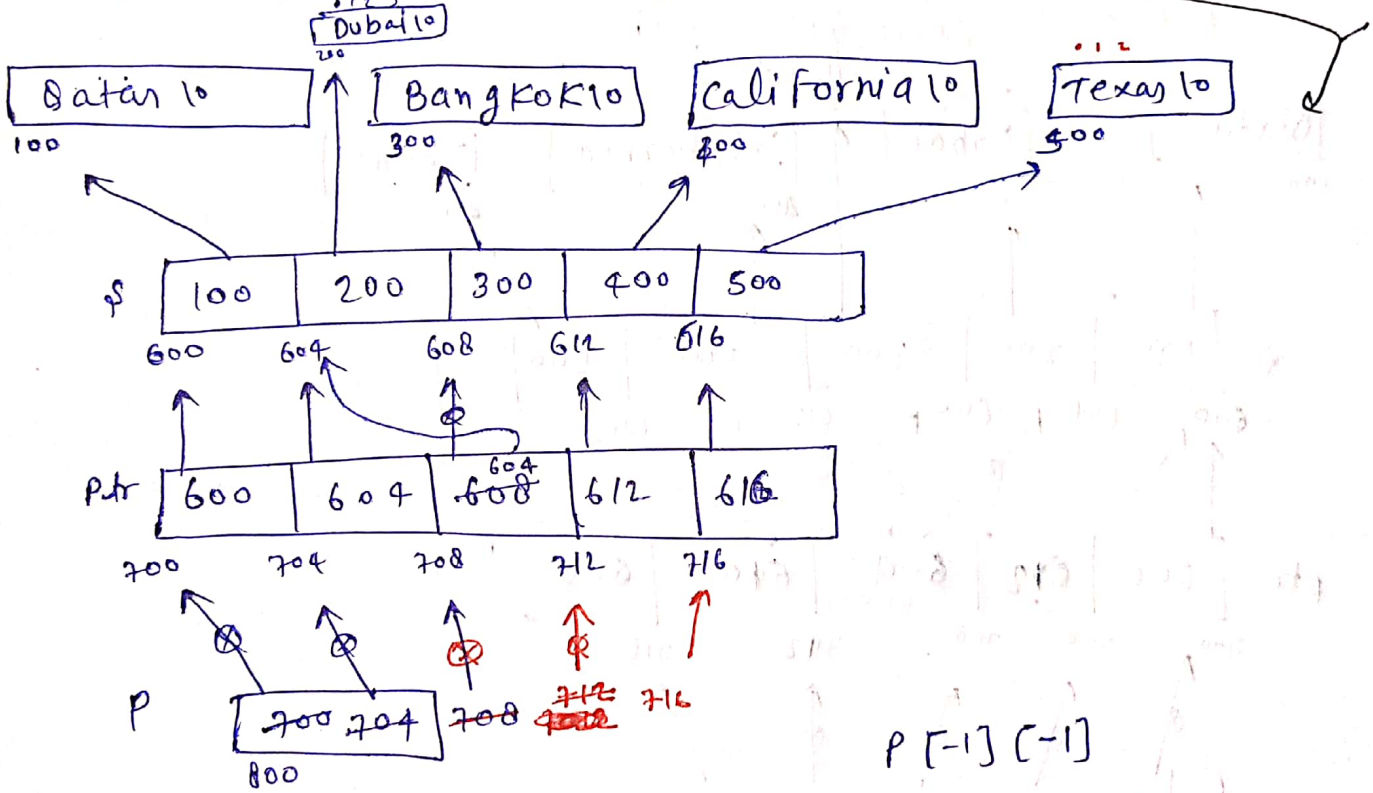
Quest

```
void main()
char *$[5] = {"Qatar", "Bangkok", "California", "Texas"};
char **ptr[5] = {$, $+3, $+1, $+4, $+2};  = {$, $+1, $+2, $+3, $+4}
char ***p = ptr;
```

1. pf("%s", **p++) // 100, Qatar
2. pf("%s", *--*++p+3); //203, ai
3. pf("%s", **++p); 400, California
4. pf("%s", p[-1][-1]+1); //102, atar
5. pf("%s", *(p[-2])+3); //103, an
6. pf("%c", **++p+2); 502, ✗
7. pf("%c", $[1][2]); //202, bai

Do this one in Next page

This is the actual



$x(x(s+1)+2)$

$(x(x(600+1)+2)$

$(x(*604+2)$

$x(200+2)$

$x(202) =$

p[-1][-1]

$*(*(p-1)-1)$

$*(*(712-1)-1)$

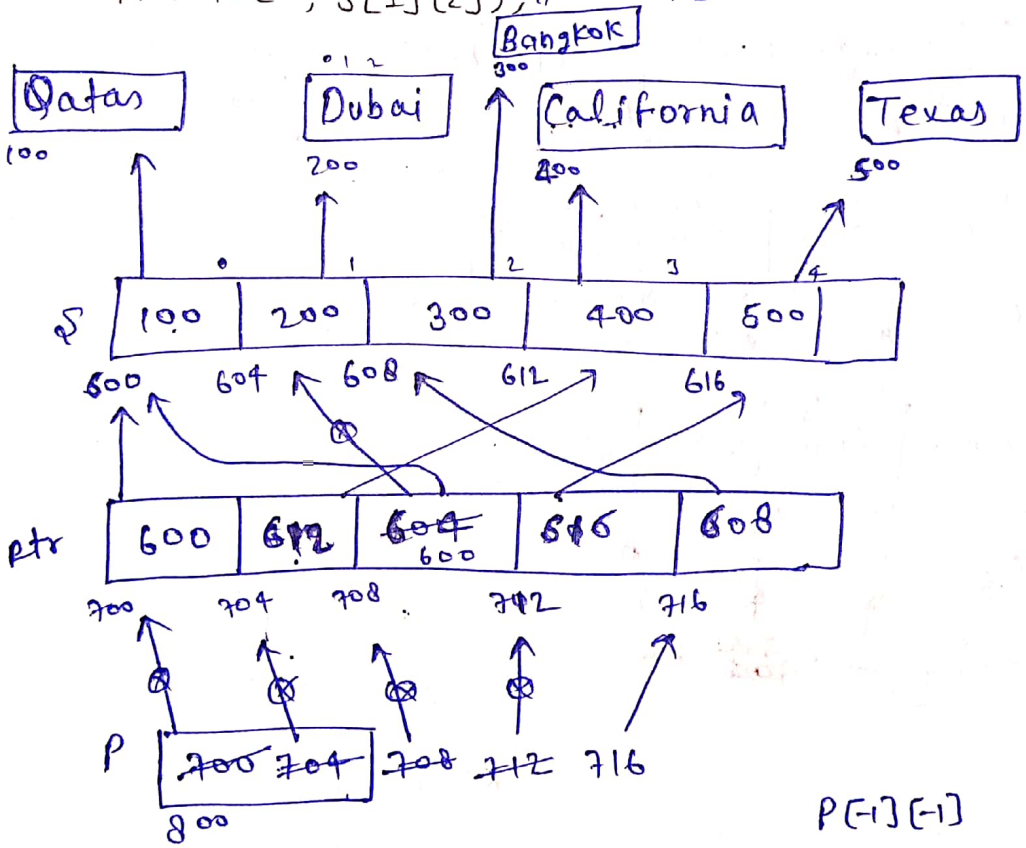
$*(*(708)-1)$

$*(604-1)$

$*600 \Rightarrow$ [illegible] $= 100 + 1 = 102$

Ques: void main()

```
char *S[5] = {"Qatar", "Dubai", "Bangkok", "California", "Texas"};
char **ptr[5] = {S+3, S+1, S+4, S+2}
char ***P = ptr;
PF("%s", **P++);          // 100, Qatar
PF("%s", *--*++P+3);      // 100, Qatar
PF("%s", **++P);          // 500, Texas
PF("%s", P[-1][-1]+1);    // G.V
PF("%s", *(P[-2])+3);     // 403, ifornia.
PF("%c", ***++P+2)        // 302, n
PF("%c", S[1][2]);        // 102, b
```



S[1][2]

((S+1)+2)

((600+1)+2)

*((*604)+2)

*(200+2)

*202

P[-1][-1]

((P-1)-1)

((712-1)-1)

*((*708)-1)

*(600-1)

(*596) → G.V

P[-2]

*(P-2)

*(712-2)

*(708)

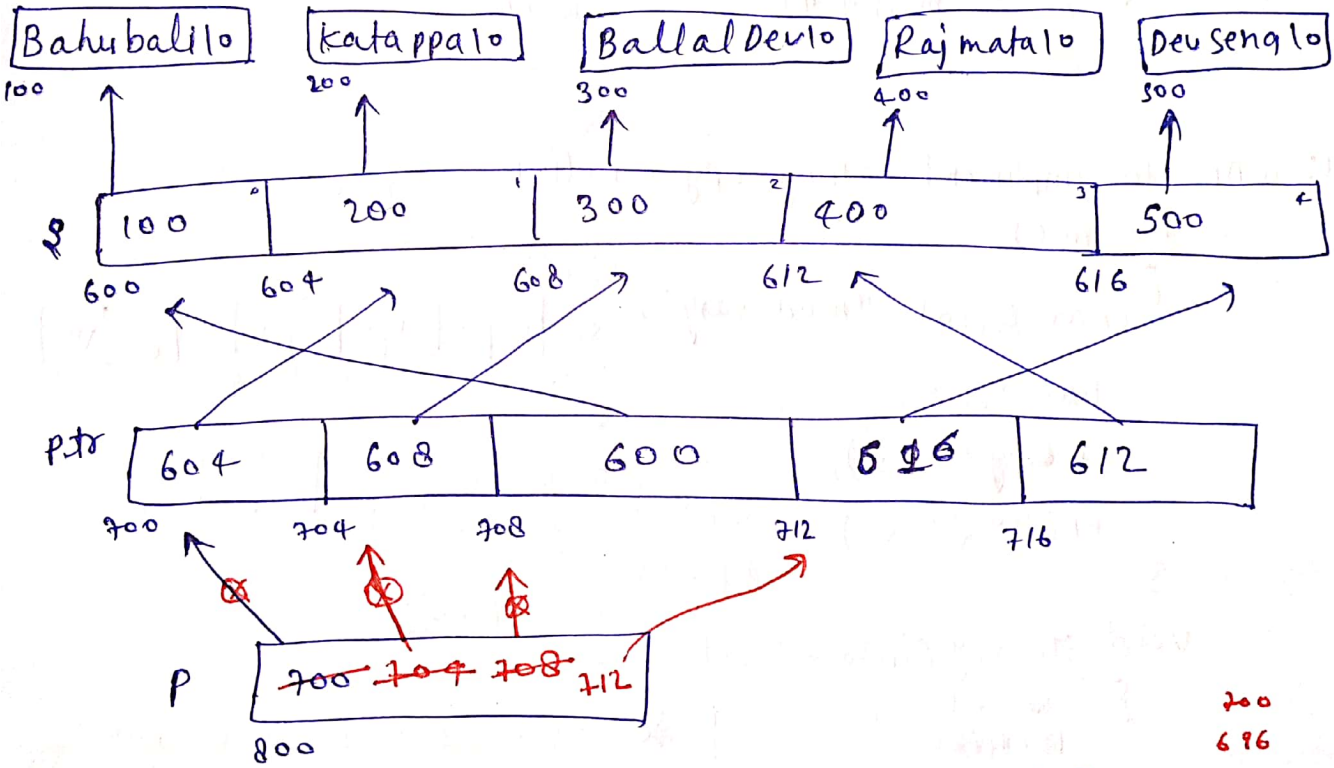
(*612)

400+3

403

Ques: Void main()

```
{ char *s[5]={"Bahubali","katappa","BallalDev","Rajmata","Devsena"
  char **ptr[5]={s+1, s+2, s, s+4, s+3}
  char ***p = ptr;
1 pf("%s", **p++);          200, katappa
2 pf("%s", *(p[-2])+3);     G.V
3 pf("%s", **++p);          100, Bahubali
4 pf("%s", *--*++p+3);      403, mata
5 pf("%s", p[-1][-1]+1);    G.V
6 pf("%c", s[1][2]);        t
7 pf("%c", ***++p+2);
}
```



200
696

P[-2]
*(P-2)
*(704-2)
*(696) → G.V

((P-1)-1)
((712-1)-1)
*(*708)-1)
*(600)-1

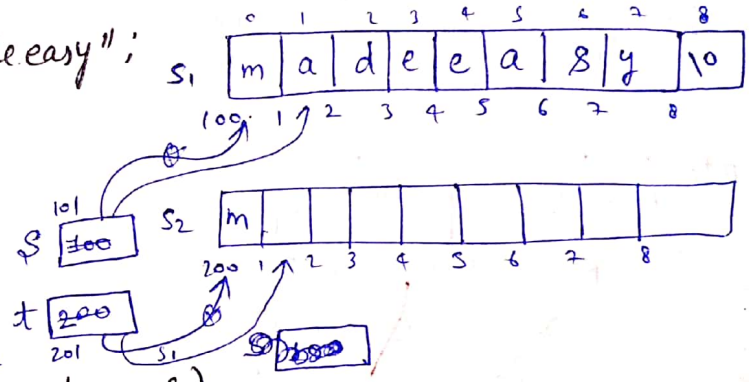
SC
((S+1)+2)
(600+1)
((604)+2)
*(200+2) =
*(202) =

MULTI-DIMENSIONAL ARRAY ON STRING

```
Void main()
{
  char mess[6][30] = { "koi bhi Lakshya",
                       "Bada nahi",
                       "Jeeta wahi",
                       "jo darq nhi",
                       "hara wahi",
                       "Jo lada nahi" };

  Pf("%s", mess+3);   // jo darq nhi
  Pf("%c", *(*(mess+2)+3)); // T
  Pf("%s", *(mess+4)); // hara wahi
}
```

Ques: WAP to impliment string copy function

```
void main()
{
  char s1[20] = "madeeasy";
  char s2[20];
  strcpy(s2, s1);
  Pf("%s", s2);
}
```




```
void strcopy(char *t, char *s)
{
  // int i=0
  // while(i>0)
  // {
  //   s[i] = t[i];
  //   i++;
  // }
  while(*t++ = *s++)
}
```

* when *t++ is on 10 (null) the assign value will be '0' and in while loop '0' is false condition

}

TYPE CASTING :

→ In the C lang in order to perform any operation b/w two variables the type of both should be same otherwise we need to perform the type casting.

e.g.

Implicit type Casting or Coersion :

```
int i; ⇒ 2B
Char c; ⇒ 1B
float f; ⇒ 4B

i=c; //No error
```

[2B] [1B]

```
i=(int) c;
```

- → type casting to integer
- → compiler will perform this
- → Implicit type casting or coersion
- → No data loss

Explicit Type Casting :

```
i= f; // error
```

2B 4B

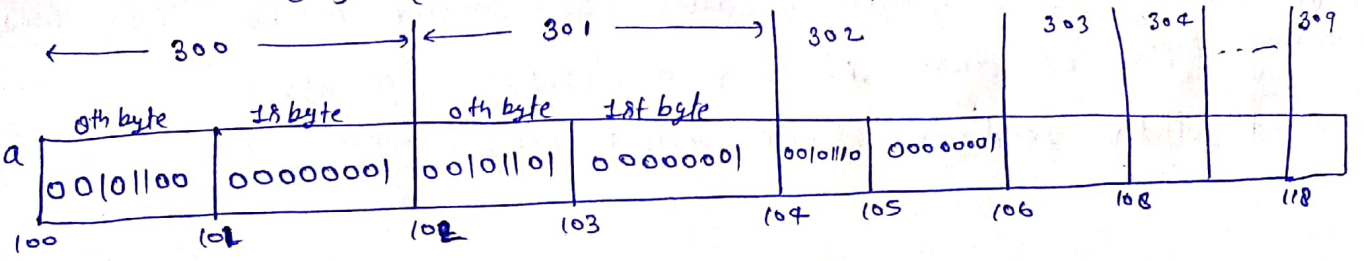
```
i= (int)f; // No error
```

- → Type casting to integer
- → explicit type casting
- → data loss
- → user

TYPE CASTING IN POINTERS :

00000001 00101100 → 300
H.V. L.V.

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



int *b = a;	char *c = a;	float *f = a;	void *e = a;
	4B 4B	4B 4B	4B 4B
No Type Casting	Type Casting require	Type Casting require	No type casting
pf(*b) = 2B	char *c = (char*) a;	float *f = (float*) a;	pf(*e); //error
b+1 = 102	↳ type casting Character pointer	↳ type casting to float pointer	e+1 = error
	pf(*c) = 4B	pf(*f) = 4B	
	c+1 = 101	f+1 = 104	

Quest:

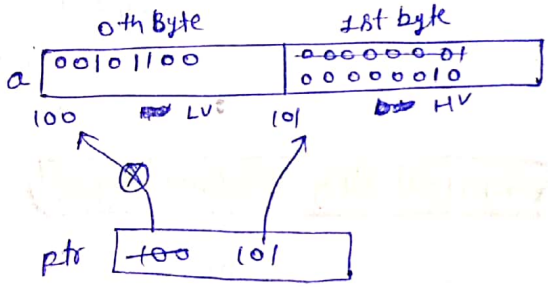
```
main()
{
    int a = 300;
    char *ptr = (char*) &a;
    ptr++;
    *ptr = 2;
    pf("%d", a);
}
```

a = 300 (00000001 00101100; HV, LV) — 100

ptr = 100 — 200

O/P → 556

00000010 00101100 → 556



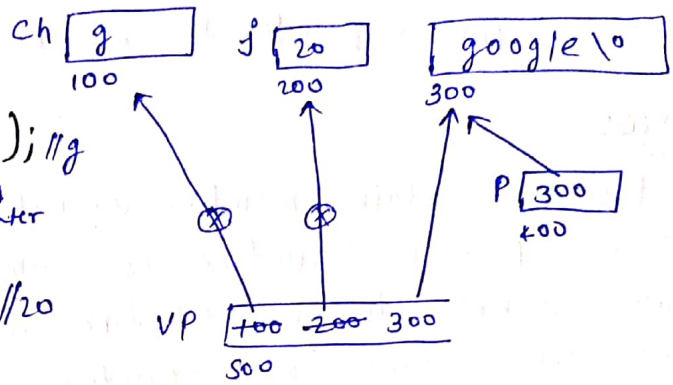
VOID POINTER :

1. Just like we had integer and character pointer similarly we can also have void pointer
2. In the void pointer we can store address of any type of variable. while accessing the data we need to perform respective type casting accordingly.
3. Type casting is required because void pointer doesnot know how many bytes of data it has to dereference/access.
4. Void pointer used in device driver programming.

e.g.
```
main()
{
  void *vp;
  char ch = 'g';
  int j = 20;
  char *p = "google";
    vp = &ch;
  pf("%c", *(char*)vp); //g
  vp = &j;
  pf("%d", *(int*)vp); //20
  vp = p;
  pf("%s", (char*)vp+3); //gle
}
```

vp is pointer variable of type void where we can store address of any type of variable

type casting to char. pointer



POINTER TO FUNCTION :

```
Void test();
  main()
  {
  ① void (*p)();
  ② p = test;
  ③ (p)();
  }
```

```
void test()
{ pf("Hello");
}
```

o/p → Hello

① → Declaring the function pointer.
'p' is a pointer to a function which doesnot take any arguments and returns void.

② → Assigning the base address of the base function to the pointer variable 'p'.

③ → Calling the function through pointer variable 'p'.

STRUCTURES:

1. Structure is used to create user defined data type. In the structure the two different operators are used.
   (a) . Structre to member
   (b) → pointer to member

NOTE:

1. Structure contains member variables.
2. Size of the structure is some of all member variables.
3. we can not define structure without any member variables
4. we can define the structure inside the function.
5. we can not define the function inside the structure.
6. we can create existing structure as member to the other structure.
7. we can not create same structure variable as a member to the structure because logically size is undefined
8. we can create same structure pointer as member to the structure. and this is called as self-referensial structure. (this is used in the linked list)

```
struct emp        // keyword → struct, Name of structure → emp
{
    int id;       // 2B
    char name[20]; // 20B
    int sal;      // 2B
};                // member variables
```

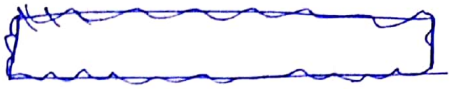
The memory allocation is zero for struct.

```
typedef struct emp EMP;
void main()
{
    ✗ emp e1;   // Struct keyword is missing that is why this is not allowed.
    ✓ struct emp e1;   // this is called as structure variable and the memory allocate for this is 24B
    ✓ EMP e2, e3, e4;  // ✓ Structure Variable
    ✓ e1.Sal = 2000;   // Structuretomember to access Sal.
    ✓ e1.id = 10; // ✓
    ✓ EMP *ptr;  // ptr is pointer variable of type Structure EMP where we can store address of any structure variable.
```

`e1.name = "Ramesh";` // ✗ not allowed because we are trying to modify constant base address.

`e1.name = Ramesh;` // ✗ we can not modify constant base address

`Strcpy(e1.name, "Ramesh");` // ✓ (e1.name → 104 Target; "Ramesh" → 204 Source)



`ptr = &e1` // ✓ Assigning the address of a structure variable e1 to pointer variable ptr

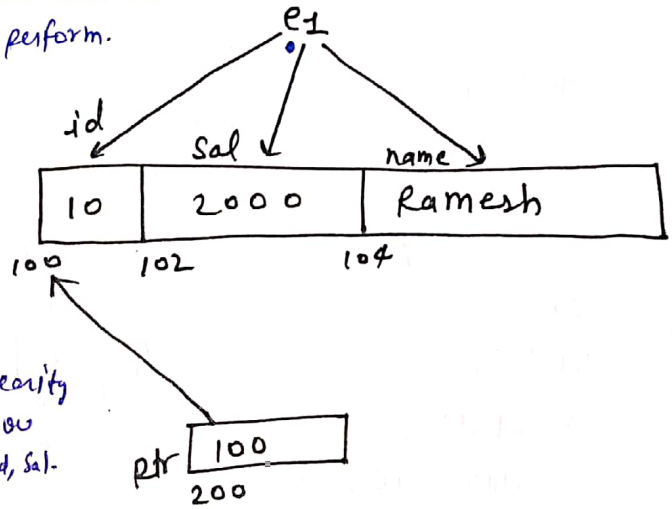
`Pf(ptr) = 100;`

`Pf(ptr+1) = 124;` // structure size is 24B then this is jump 124B

`e2 = e1;` // ✓ This is allowed but not suggestable to perform.

`e2.Sal = e1.Sal;` // ✓

```
e2 = e1 + e3;
e2 = e1 * e3;
e2 = e1 / e3;
```
✗ It has no meaning



`Pf(e1);` // ✗ you are not giving clearity to compiler which you want to print e.g. – id, Sal.

Same ↓

`Pf(e1.Sal);` // 2000

`Pf(*ptr);` // ✗

`Pf((*ptr).Sal);` // 2000

Same ↓

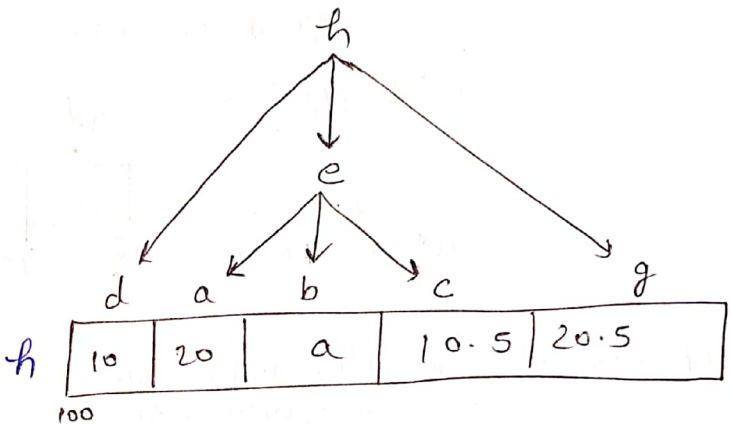
`Pf(ptr→Sal);` // 2000

e.g.
```
struct node
{ int data;
  struct node n1; // ✗
};
```
* This is checking the memory allocation or data type you can not create structure inside struct node.

e.g.
```
struct node
{ int data;
  struct node *n1; // ✓
};
```
* If pointer structure then 4B is allocate refer to point8

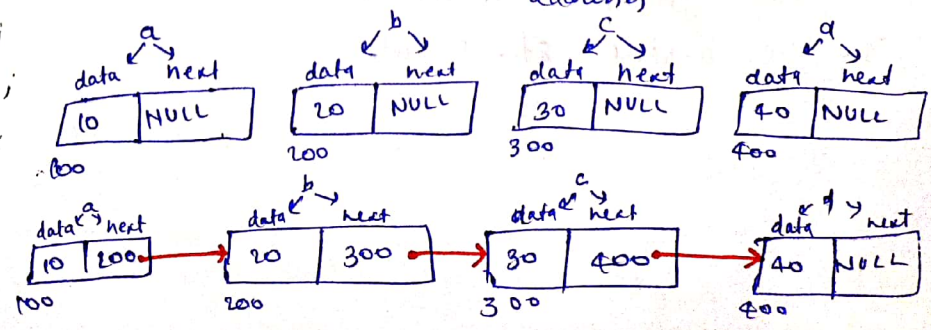
e.g.
```
void main()
{
  struct emp
    {
      int id;
      float f;
    };
  mec();
}
void mec()
{
  struct emp e1, e2; // ⊗ out of scop in static scoping
}
```
struct is inside the main function.

e.g.
```
struct S1
{ int a;
  char b;
  float c;
};
struct S2
{ int d;
  struct S1 e;
  float g;
};
```

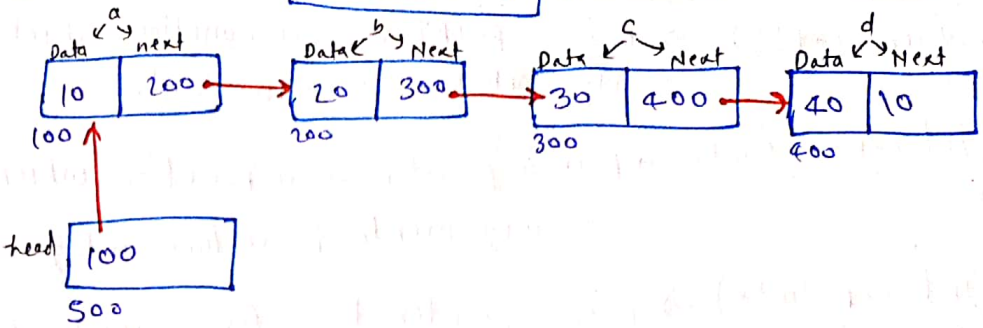



```
struct S2 h = {10, {20, 'a', 10.5}, 20.5}
Pf(h.d) = 10
Pf(h.e); // error
Pf(h.e.a); = 20
```

e.g.
```
struct node
{ int data;
  struct node *next;
};
typedef struct node node; // a,b,c,d is structur variable. // there is no relationship in addresses
node a = {10, NULL};
node b = {20, NULL};
node c = {30, NULL};
node d = {40, NULL};
a.next = &b;
b.next = &c;
c.next = &d;
```



node *head = NULL; → node *head; → D
head = &a; head = NULL; → A



PF(head) = 100;

PF(head → data) = 10;

PF(head → next) = 200;

PF(head → next → next → data) = 30;
(200 → 300 → 30)

* If (head == NULL) → If the above condition is true it means linked list is empty.

* If (head → next = null) → Linked list having one node.

UNION'S:

1. The size of the union will be max size of the member variable.
2. In the structure we can acess all the members at any time. But in one of union, we can access only 1 member variable & all the remaining members will contain garbage value.
3. Same memory space will be shared by all the members in the union.

e.g. keyword union data → Name of union

```
{
    int i;   2B
    Float f; 4B
    char str[20]; 20B
} 20B  member variable
};
main()
{ union data d; // union variable
  d.i = 20;
  d.f = 300.5;
  Strcpy (d.str, "made easy");
  PF("%d", d.i); // G.V.
  PF("%d", d.f); // G.V.
  PF("%s", d.str); // made easy.
}
```

d ⇒ 20B

~~20~~	~~300.5~~	made easy

TERMINALOGY OF VARIOUS EXPRESSION:

1. void (*p)(); → P is a pointer to a function which does not take arguments & return void.

2. int (*p)(int,int); → P is a pointer to a function which take two integers as arguments & return integer.

3. int (*p)(int*) → P is a pointer to a func. which takes integer pointer as argument & returns integer.

4. int a[5]; → a is a array of 5 elements where every element of integer

5. int *a[5]; → a is array of ~~pointer~~ 5 elements every element is integer pointer

6. int (*a)[5] → a is a pointer to array of 5 element whose every element is integer

7. int (*a)() → a is a pointer to function which does not take any argument and return integer.

8. int *a(); → a is a function which does not take any argument and return integer pointer.

9. char **(*a)(*,*); → a is a pointer to a function which takes two pointers as argument and return character double pointer.

10. int (*a[5])(); → a is array of 5 ~~element~~ pointer to function which does not contain any argument & return integer.
where every pointer points

11. char (*(*a())[30])(); → a is a function which does not take any argument returns pointer. This pointer is pointing to array of 30 element where every element is a pointer to a function which does not take a function & return character.

12. char (*(*a[3]())[5]; → a is a array of 3 elements pointer of every one is pointer. to a function which does not take any argument & return pointer. That pointer points to 5 element of character

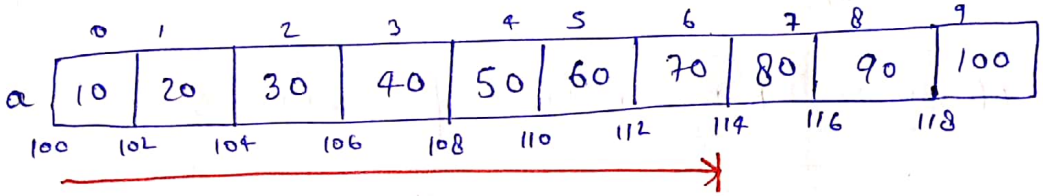
13. void (*b (int, void (*f)(int)))(int); → b is a function which is taking two arguments it is return pointer. pointer pointing to function which is return void take integer as argument & return void.
2 argument → integer
↳ f is a pointer to a function which takes int as a argument ret

14. void (*ptr) (int (*)[2], int (*)(void)); → ptr is a pointer to function which take two arguments & return void. 2 arguments.
- → pointer to array of 2 element (over element is int)
- → point to function which take void argument & return integers.

ARRAY'S :

1. Array is a collection of similary type data/elements.
2. Array elements will be stored in continous location.
3. Array supports random access.

e.g. int a[10] (=(B-o+1)) = a{0 (lb) .... 9 (UB)} = {10, ... 100};



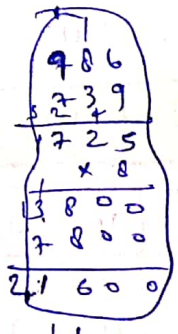
* Loc(a[7]) = 100 + (7-0)*2 → size of element
  - 100 → BA
  - 7 → i
  - 0 → lb

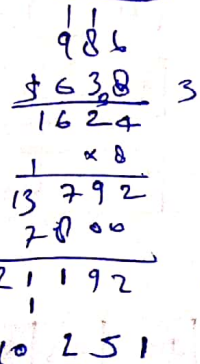
  = 100 + 14 ⇒ 114

* Loc(a[9]) = 100 + (9-0)*2
  = 100 + 18
  = 118

Quest: a[-986 (lb) ... +739 (ub)], BA = 7800, SOE = 8

Loc(a[638]) = 7800 + (638 - (-986)) × 8
= 7800 + 1392 = 21192





Quest: a[-356 ... +981] BA = 3500, SOE = 9

Loc(a[783]) = 3500 + (783 + 356) × 9
= 3500 + (1139) 9
= 3500 + 10251 =

NOTE:

1. $a[lb \ldots ub]$, B.A. S.O.E = s

$$Loc(a[i]) = B.A. + (i - lb)s$$

start with the

2. Array index always start with zero other wise we need to perform one additional subtraction operation.

2-D ARRAY :

$a[1 \ldots 4][1 \ldots 5]$

lb_1 ub_1 lb_2 ub_2

lb_1 — rows, ub_1 — rows, lb_2 — columns, ub_2 — columns

$$\text{No. of rows } (n.r) = ub_1 - lb_1 + 1 \quad = 4 - 1 + 1 = 5$$

$$\text{No. of columns } (n.c) = ub_2 - lb_2 + 1 \quad = 5 - 1 + 1 = 5$$

1. Two D array will be stored in the memory in the form of multiple 1D arrays.
2. It will be stored in the two different ways.
   (a) Row major order (R.M.O.)
   (b) Column major Order (C.M.O.)
3. C lang. by default follows row major order.

(a) 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}

4×5

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}	
100	102	104	106	110	112	114	116	118	120	122	124	126	128	130	132	134	136	138	140	

$$Loc(a[4][3]) = 100 + \left[(4-1)*5 + (3-1)\right]*2$$

lb_1 n lb_2

$$= 100 + [15+2]*2$$
$$= 100 + 34 \Rightarrow 134$$

$$loc(a[3][4]) = 100 + \left[(3-1)*5 + (4-1)\right]*2$$
$$= 100 + [10+3]2$$
$$= 100 + [13]2 \Rightarrow 126 \text{ B}$$

Ques: $a[-358 \ldots +736, -129 \ldots +398]$ BA = 4800 SOE = 8 B

lb_1 ub_1 lb_2 ub_2

$$loc(a[693][358]) = 4800 + \left[(693 - -358)*528 + (358 - -129)\right]*8$$
$$= 4800 + \left[(1051)*528 + (487)\right]*8$$
$$= 4448120$$

Ques: $a[-539 \ldots +376][-417 \ldots +918]$ BA = 5600 SOE = 9

$$loc(a[367][895]) =$$
$$4800 + \left[(367 - -539)*(918 - -417+1) + (895 - -417)\right]*9$$
$$\Rightarrow 4800 + \left[914*1336 + 1312\right]9$$
$$\Rightarrow 4800 + 10905552 = 10911152$$

NOTE:

1. $a[lb_1 \cdots ub_1][lb_2 \cdots ub_2]$, B.A., SOE, RMO

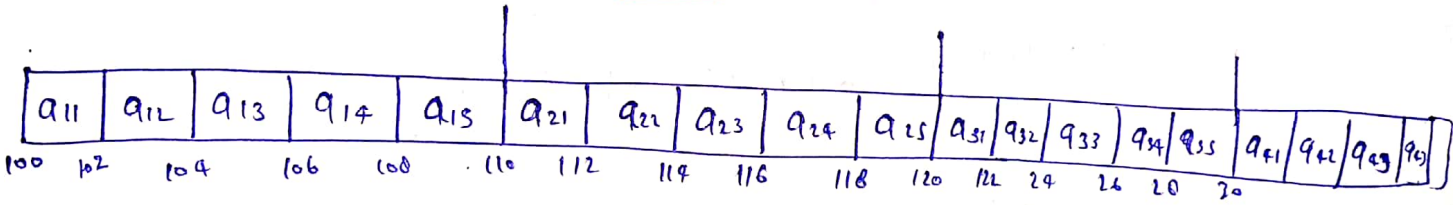
$$Loc(a[i][j]) = BA + \left[(i - lb_1) * nC + (j - lb_2)\right] * S$$

$$nC = ub_2 - lb_2 + 1$$

(b) Column Major Order:

$a[1 \cdots 4][1 \cdots 5]$

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



$$Loc(a[3][4]) = 100 + \left[(4-1) * 4 + (3-1)\right] * 2 = 128$$

(lb_2, n.r, lb_1)

$$Loc(a[2][3]) = 100 + \left[(3-1) * 4 + (2-1)\right] * 2 = (8+1) * 2 = 118$$

NOTE: $a[lb_1 \cdots ub_1][lb_2 \cdots ub_2]$, BA, SOE = S, CMO

$$Loc(a[i][j]) = BA + \left[(j - lb_2) * nR + (i - lb_1)\right] * S$$

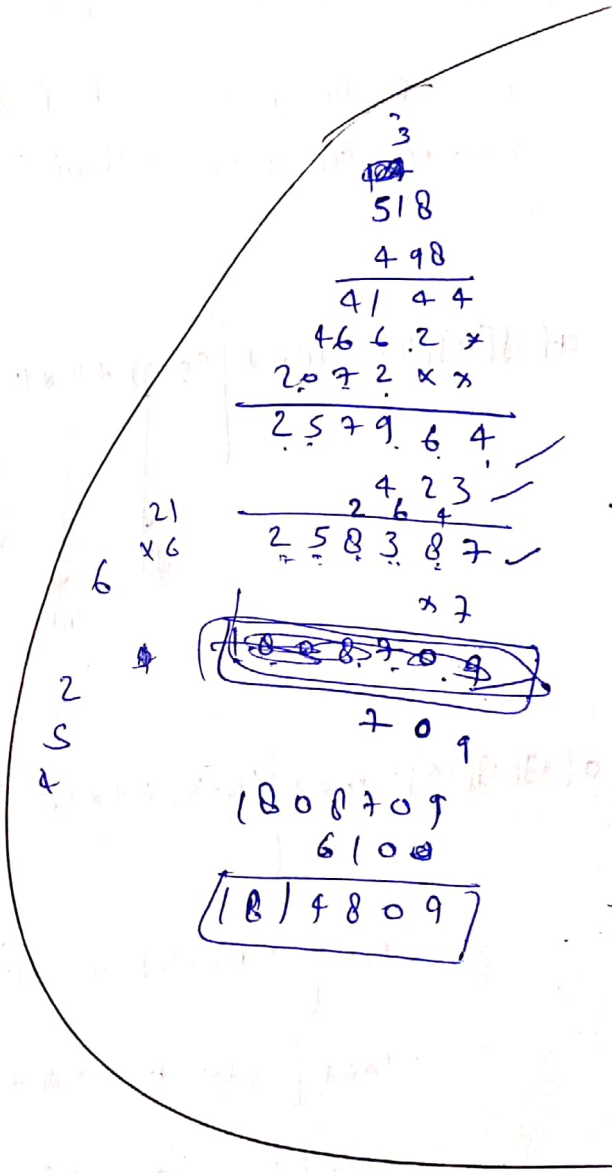
$$nR = ub_1 - lb_1 + 2$$

Quest: $a[-126 \text{ to } +371][-416 \text{ to } +109]$ BA = 6100, SOE = 7

$$loc(a[297][102]) = 6100 + \left[(102+416)(371+126+1) + (297+126)\right] \times 7$$

$$= 6100 + \left[(518 * 498) + 423\right] \times 7$$

$$= 1814809$$



3D ARRAY'S :

$a[lb_1 \dots ub_1][lb_2 \dots ub_2][lb_3 \dots ub_3]$

(rows) (columns)

No. of rows $(n_R) = ub_2 - lb_2 + 1$

No. of columns $(n_C) = ub_3 - lb_3 + 1$

$a[3 \overset{\Rightarrow 7}{\dots} 9][4 \overset{\Rightarrow 9}{\dots} 12][5 \overset{\Rightarrow 12}{\dots} 16]$

$7 \Rightarrow$ 2D - Arrays $\Rightarrow$ each of size $9 * 12$

$9 \Rightarrow$ 1D - Arrays $\Rightarrow$ each of size 12

3 = 9*12
4 = 9*12
5 = 9*17
6 =
7 =
8 =
9 =

5 6 7 8 ... 12
4
5
6
7
8
.
.
.
12

$a[5][8][7] = 100 + \left[(5-3) * 9 * 12 + (8-4) * 12 + (7-5)\right] * 2 = 632$

(lb_1, n_R, n_C, lb_2, n_C, lb_3)

$a[8][11][14] = 100 + \left[(8-3) * 9 * 12 + (11-4) * 12 + (14-5)\right] * 2$

$= 100 + \left[(5 * 9 * 12) + (7 * 12) + 9\right] * 2$

$= 100 + \left[540 + 84 + 9\right] * 2$

$= 100 + \left[633\right] * 2 = 1366$

$= 100 + 1266 \Rightarrow 1366$

NOTE: Row Major order :

$a[lb_1 \dots ub_1][lb_2 \dots ub_2][lb_3 \dots ub_3]$, BA, S

$$loc(a[i][j][k]) = BA + \left[(i - lb_1) * n_R * n_C + (j - lb_2) * n_C + (k - lb_3)\right] * S$$

Column major order:

$$loc(a[i][j][k]) = BA + \left[(i - lb_1) * nR * nC + (k - lb_3) * nR + (j - lb_2)\right] * S$$

NOTE: If the structure variable is passed in the function then receiver should also be structure variable of same type.

Lower Triangle Matrix: (L.T.M.)

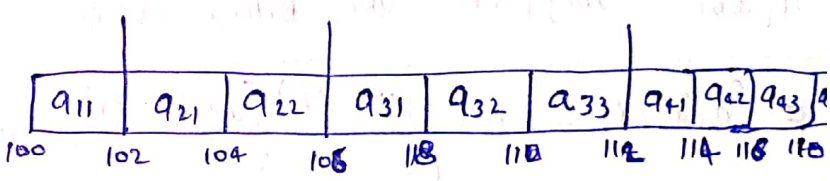
1. L.T.M. is possible for square matrix.

$a[1 \text{---} 4][1 \text{---} 4]$

lb_1 ub_1 lb_2 ub_2

R.M.O:

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



$i < j \Rightarrow$ zero

$i \geq j \Rightarrow$ non-zero (stored)

non-zero elements

$1 + 2 + 3 + 4 + \text{---} + n$

$\Rightarrow \frac{n * (n+1)}{2}$

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

(lb_1 under $(4-1)$; lb_2 under $(3-1)$)

$$= 100 + \left[\frac{3 \times 4}{2} + 2\right] * 2 = 100 + (8)2 = 116$$

$$loc(a[3][2]) = 100 + \left[\frac{(3-1)(3-1+1)}{2} + (2-1)\right] * 2 = 100 + 8 = 108$$

(lb1 over (3-1); lb1 over (2-1))

Check, $i \geq j$

$3 \geq 2$ T

Quest: $a[-128 \text{ ---} +256][-128 + \text{ --- } 256]$, BA = 4800, SOE = 6

lb_1 ub_1 lb_2 : ub_2

$$loc(a[219][197]) = 4800 + \left[\frac{(219+128)*(219+128+1)}{2} + (197+128)\right] * 6$$

$$= 4800 + \left[\frac{347 * 348}{2} + 325\right] * 6$$

(174 over 348)

$i \geq j$

$219 \geq 197$ T $= 369018$

NOTE: R.M.O.

$a[lb_1 \text{ ---} ub_1][lb_2 \text{ --} ub_2]$, B.A., SOE = $

$$loc(a[i][j]) = BA + \left[\frac{(i-lb_1)(j-lb_1+1)}{2} + (j-lb_2)\right] * S$$

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

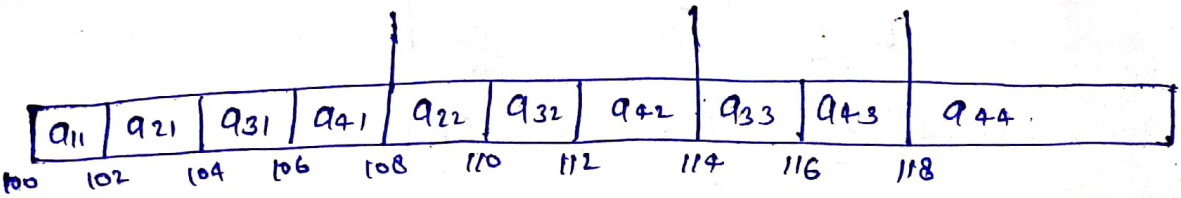
 # C.M.O.:

$a[1 \text{ ---} 4][1 \text{ --} 4]$

lb_1 ub_1 lb_2 ub_2

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

Non-zero element



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

(i, i, j, j; i, j)

$$= 100 + [10 - 3 + 1] * 2 = 116$$

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

$$= 100 + [10 - 6 + 1] * 2 = 110$$

Quest: $a[67 \text{---} +379][67 \text{---} +379]$ BA = 7100, SOE = 7

$$loc(a[369][293]) = 7100 + \left[\frac{(379-67+1)*(379-67+1+1)}{2} - \frac{(379-293+1)*(379-293+1+1)}{2} + (369-293) \right] * 7$$

$$= 7100 + \left[\frac{379 * 380}{2} - \frac{87 * 88}{2} + 76 \right] * 7$$

(190, 44)

$$loc(a[369][293]) = 7100 + \left[\frac{(379-67+1)*(379-67+1+1)}{2} - \frac{(379-293+1)*(379-293+1+1)}{2} + (369-293) \right] * 7 \Rightarrow 32483$$

Quest: $a[-23 \text{---} +259][-23 \text{---} +259]$ BA = 4850, SOE = 9

$$loc(a[245][236]) = 4850 + \left[\frac{(259+23+1)*(259+23+1+1)}{2} - \frac{(259-236+1)*(259-236+1+1)}{2} + (245-236) \right] * 9$$

(142, 12)

$$= 4850 + \left[\frac{283 * 284}{2} - \frac{24 * 25}{2} + 9 \right] * 9$$

$$= 365905$$

NOTE: $a[lb_1 \ldots ub_1][lb_2 \ldots ub_2]$ B.A. $\in$ = ?, CMO, LTM.

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

$i \geq j$

RECURSION:

Recursion is catogaries into 4 types.

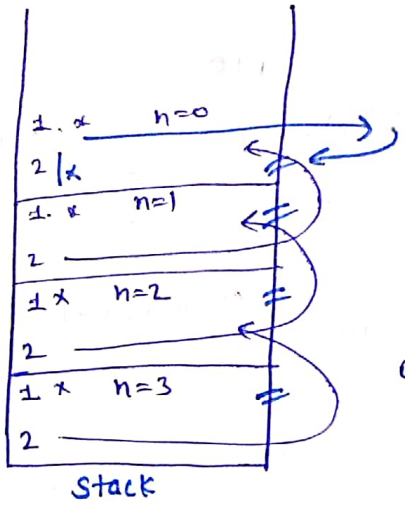
- Tail Recursion
- Non Tail Recursion
- Indirect Recursion
- Nested Recursion

1. Tail Recursion:

In the program very last statement is recursive call and there is no other statement after that that is called tail re

e.g.
```
TR(int n)
{
1. if(n==0) return;
   else {
2.   PF(n);
     return TR(n-1);
   }
}
```

what is o/p of TR(3) ?



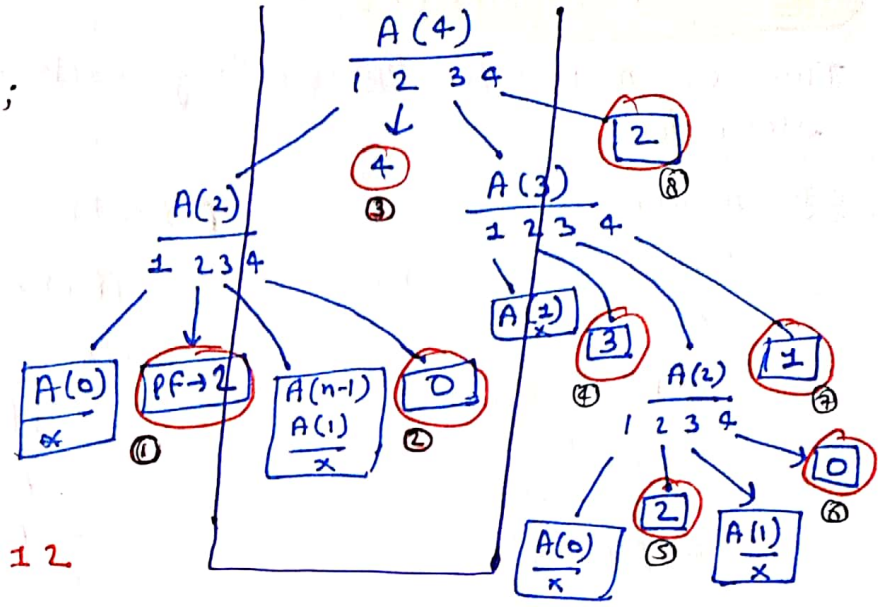
o/p → 3 2 1

1. It is very easy to write equivalent non recursive program
2. In the TR we are wasting lot of stack space.

2. Non-Tail Recursion:

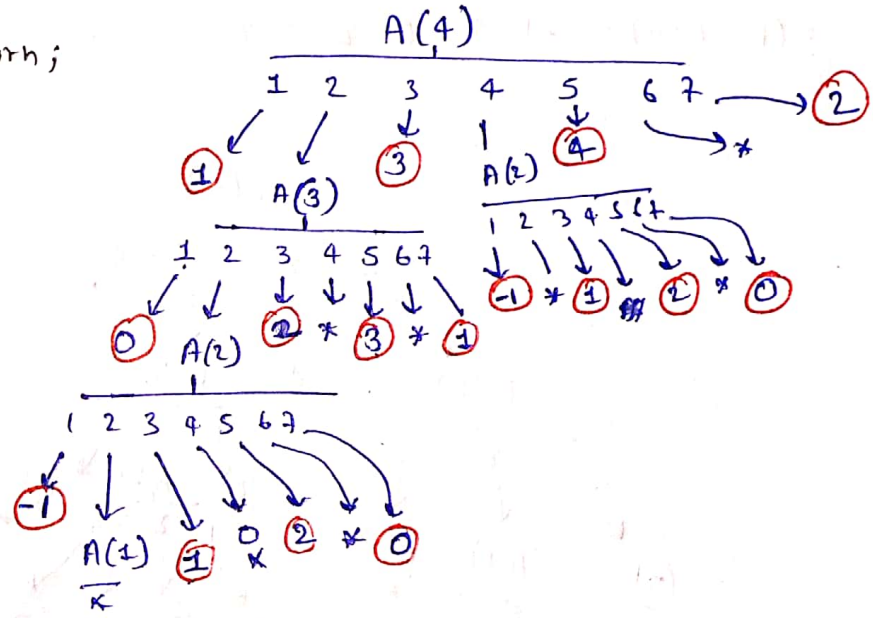
In the program after recursive call ther are some statement to execute then is called as non-tail recursion.

e.g.
```
A(int n)
{
IF (n<=1) return;
else
 {
 1 A(n-2);
 2 Pf(n);
 3 A(n-1);
 4 Pf(n-2);
 }
}
```



O/P of A(4) = ? 2 0 4 3 2 0 1 2

Quest:
```
A(int n)
{ if (n<=1) return;
 else
  {
 1 Pf(n-3);
 2 A(n-1);
 3 Pf(n-1);
 4 A(n-2);
 5 Pf(n);
 6 A(n-3);
 7 Pf(n-2);
 }
}
```



A(4) = 10 -1 1 2 0 2 3 1 3 -1 1 2 0 4 2

1. It is very difficult to write equivalent non-recursive program
2. We are not misusing the recursion.

3. Indirect Recursion:

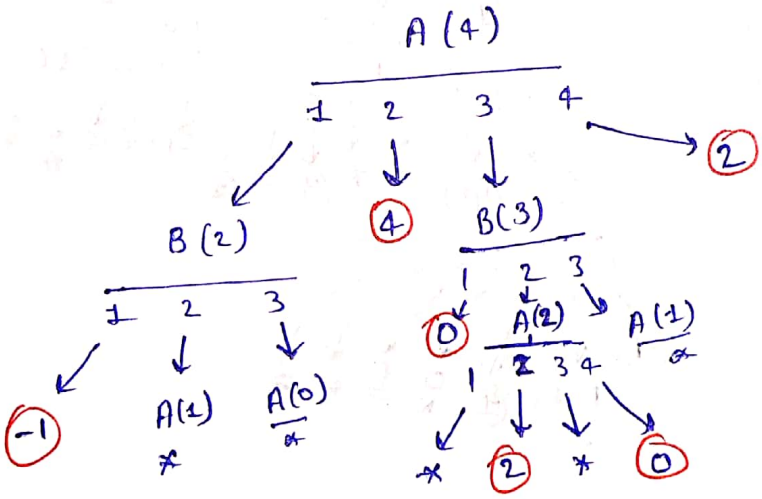
Two or more functions calling each other is called as indirect recursion.

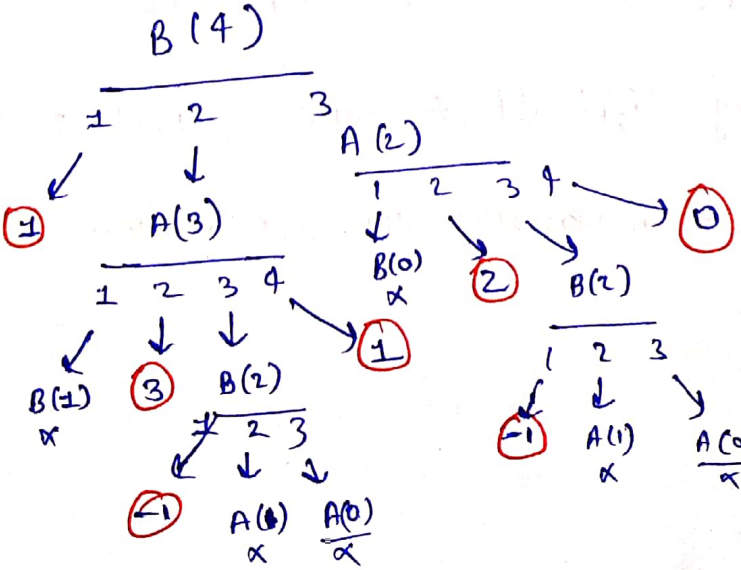
e.g.

```
A(int n)
{ if (n<=1) return;
  else
  {1. B(n-2);
   2. pf(n);
   3. B(n-1);
   4. pf(n-2);
  }
}
```

```
B(int n)
{ if(n<=1) return;
  else
  { 1. pf(n-3);
    2. A(n-1);
    3. A(n-2);
  }
}
```

O/P A(4) = -1 4 0 2 0 2





Nested Recursion

A recursive function which is passing itself as a parameter to a recursive call is called as a nested recursion.

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

H.W

1. $A(3,2) = 29$
2. $A(3,3) = 61$

$A(2,3) \Rightarrow 9$

$A(1, A(2,2))$

$A(1, A(2,1))$

$A(1, A(2,0))$

$A(1,1)$

$A(0, A(1,0))$

$A(0,1)$

$A(1,2)$

$A(0, A(1,1))$

$A(0, A(1,0))$

$A(0, A(0,1))$

$A(1,3)$

$A(0, A(1,2))$

$A(1,4)$

$A(0, A(1,3))$

$A(1,5)$

$A(0, A(1,4))$

$A(1,6)$

$A(0, A(1,5))$

$A(1,7)$

$A(0, A(1,6))$

o/p → $A(1,5) = 7$

DYNAMIC MEMORY ALLOCATION (DMA):

#include <stdlib.h>

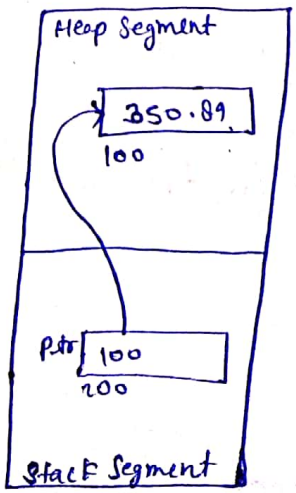
1. Array follows static memory allocation. we should know the requirement of the size prior to the declaration.
2. If the requirement is dynmically adding the elements or dynamically increase the size then we need to use concept of DMA.
3. By using the concept of DMA we can dynamically increase the size at run time.
4. whatever the memory allocated the using DMA it will be store in the heap segment.
5. The lifetime of dynamically memory allocated is entire program.
6. Header file use for the DMA is #include<stdlib.h>
7. The following functions are used in the DMA.
   - * malloc();
   - calloc();
   - realloc();
   - * free();

1. malloc();

Syntax :→ pointer_variable = (type_cast *) malloc (size of (size))

eg: Float *ptr;
ptr = (Float*) malloc (size of (Float));
*ptr = 350.89;
pf("%f", *ptr);
}

o/p → 350.89

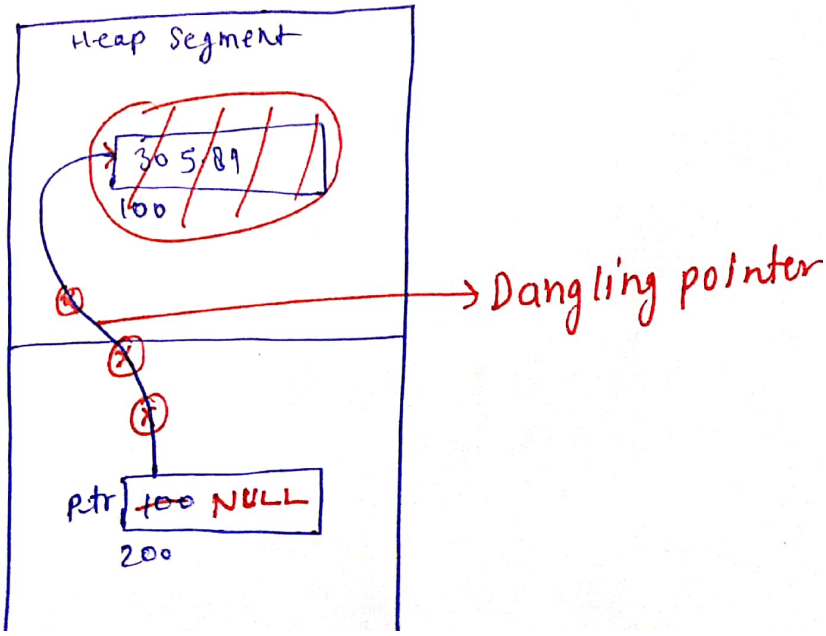


1. malloc dynamically allocates the memory in the heap segment and it takes one argument that is in bytes.
2. (Size of) is an operator it is calculate size in bytes.
3. malloc allocates the memory in the heap segment and return it's base addres.
4. This base address is integer pointer (old Turbo c is a void pointer)
5. Because it is returning base address reciver should be pointer variable.
6. Because it is returning integer pointer, then respective type casting is required to access the allocated memory.
7. malloc does not know to whom it is allocating the memory
8. whatever the memory allocated to the stack segment. can be accesed by using variable name and also through pointers but whatever the memory allocated to heap segment can be access only through pointers.
9. malloc returns NULL if it is not able to allocate the memory when the memory is full.

2. free :

1. free function deallocates the memory in the heap segment and it take one argument that is ~~in~~ ~~bytes~~ base addres

e.g free (ptr)
ptr = NULL;



Dangling pointer: A pointer pointing to unknown location (deallocated memory). is called as a dangling pointer.

NOTE:

1. After using the dynamically allocated memory in the heap segment it must be deallocated other wise it is called as "memory leakage".

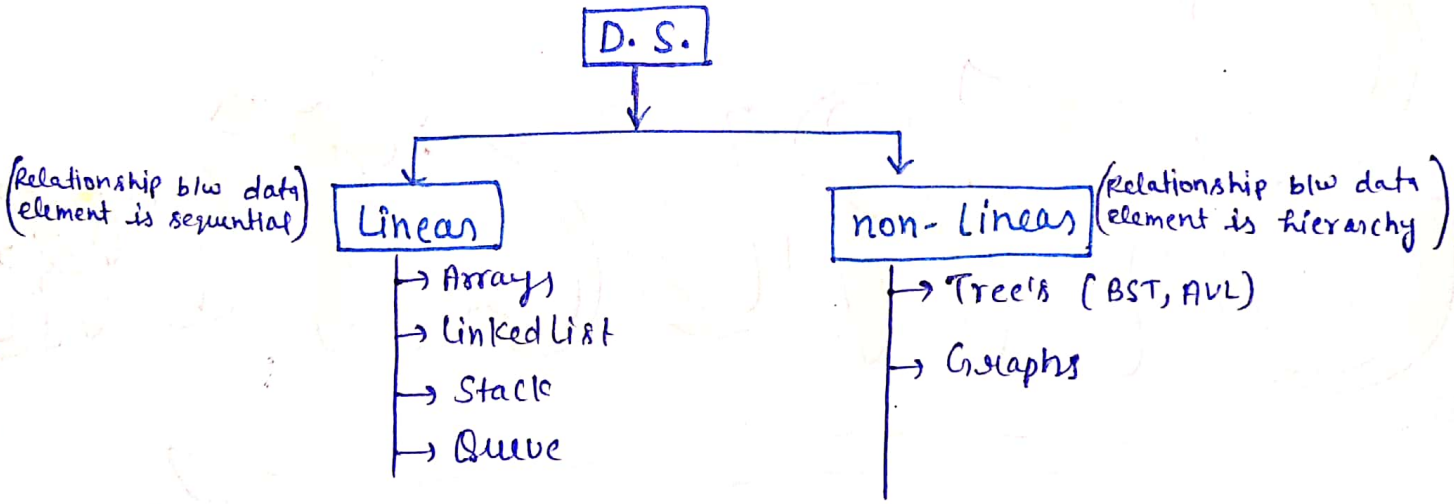
DATA STRUCTURE

DATA STRUCTURE:

1. Data Structure is a way of ~~again~~ organising the data elements by considers its relation ship.
2. D.S. mainly deals with the:
   - → How efficently data can be stored and organization in computer memory.
   - → How Efficiently data can retrived and manipulated.

Program = Algorithm + Data Structure

Data Structure = Organized data + Operations



Abstract Data Type (ADT)

1. In Order to simplyfy the proces of solving problems the data structures combined along with their operations is called as ADT.
2. ADT mainly deals with the:
   - → Declaration of data
   - → Declaration of operator

LINKED LIST :

1. Linked list is a D.S. used to store the data element. The succesive element will be connected by the pointer the last element will have NULL.
2. We can grow the size of the linked list dynamically Untill the memory is full.
3. IF the requirement is frequently inserting / deleting at various position then the linked list is best suitable D.S.
4. The main drawback is l.l. support only sequential acces.

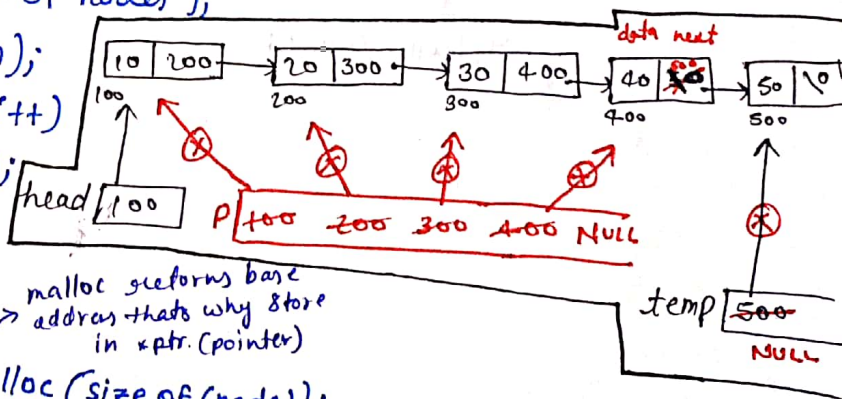
Ques: WAP to create a Linked List with 'n' nodes?

```
#include <stdio.h>
#include <stdlib.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();
}
Void createnode()
{
    node *temp;
    temp = (node*) malloc (size of (node));
    if(temp == NULL) exit(); //if we are not write this line then
    pf("enter data elment");
    sf("%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;
    }
    temp = NULL;
}
```

malloc performs base address thats why store in *ptr. (pointer)

This gives you segmentation error

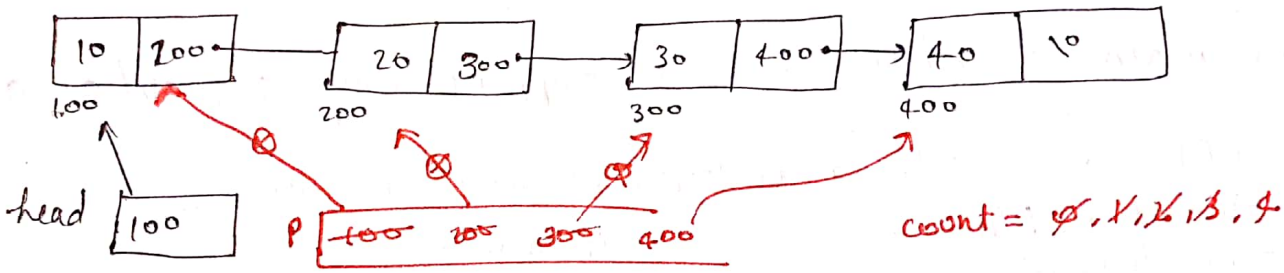
$T(n) = O(n^2)$



NOTE:

1. { } ⇒ Break;
2. Function ⇒ return
3. exit(0) ⇒ Succesful exit
4. exit(1) ⇒ unsuccesful exit
5. exit() ⇒ Not informing anything

Quest: WAP to count no. of nodes in given linked list?



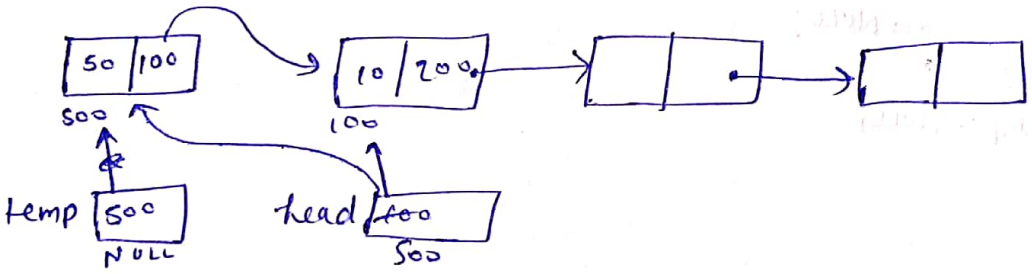

```
int listlength(node *head)
  { int count=0;
    if(head == NULL)
     return count;
    else
      { count=1;
        node *p=head;
        while(p->Next != NULL)
            { p=p->Next
              count++;
            }
        return count;
      }
  }
```

T.C = O(n)

Ques: WAP to insert a new node at the begging of the L.L.
insert at begin?

* Firstly connet the link after that break.

```
insert At Begin (node *head)
{  node *temp;
   temp = (node*) malloc (sizeof (node));
   if(temp == NULL) exit();
   Pf("Enter the node");
   Sf("%d", &tem→data);
     temp→next = NULL;
     if (head == NULL)
       head = temp;
     else
     {  temp→next = head;
        head = temp;
     }
   temp = NULL;
}
```



Ques: WAP to insert new node in the end of the linked list?

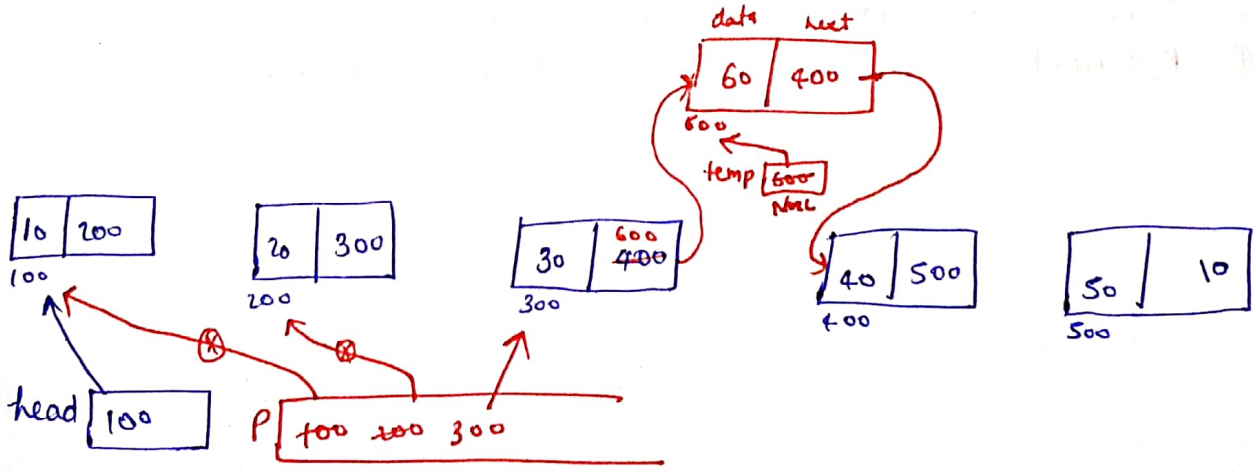
```
insert AtEnd (node *head)
{
    node *temp;
    temp = (node*) malloc (sizeof (node));
    pf ("enter the value");
    sf("%d", &temp->data);
      temp -> next= NULL;
      if (head == NULL)
         head = temp;
      else
      {  node *p = head
         while (p -> next != NULL)
              {  p= p-> next;
              }
         p -> next = temp;
         p= NULL;
      }
      temp= NULL;
}
```

T.C = O(n)

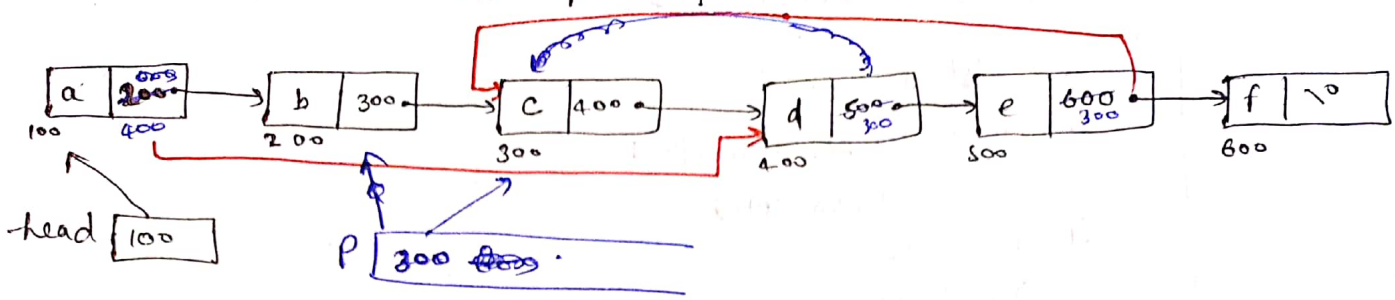
Ques: WAP to insert the new node at the given position ?

```
insert Atpos (node *head , int pos)
{
   L = list length (node);
   if (pos ≤ 0 || pos > length+1)
       Invalid position ;
   if (pos == 1)
       insert Atbegin (node);
   if (pos == length+1)
      Insert Atend (node);
   else
   { int k=1;
      node *temp , *p = head;
      temp= (node*) malloc (sizeof (node));
      if (temp == NULL) exit ();
      pf ("Enter the value");
      sf("%d", &temp→data);
      while (k<pos-1)
        { p = p→next;
          k++;
        }
      temp→next = p→next ;
      p→next = temp;
      p= NULL ;
      temp= NULL;
   }
}
```

Pos=4



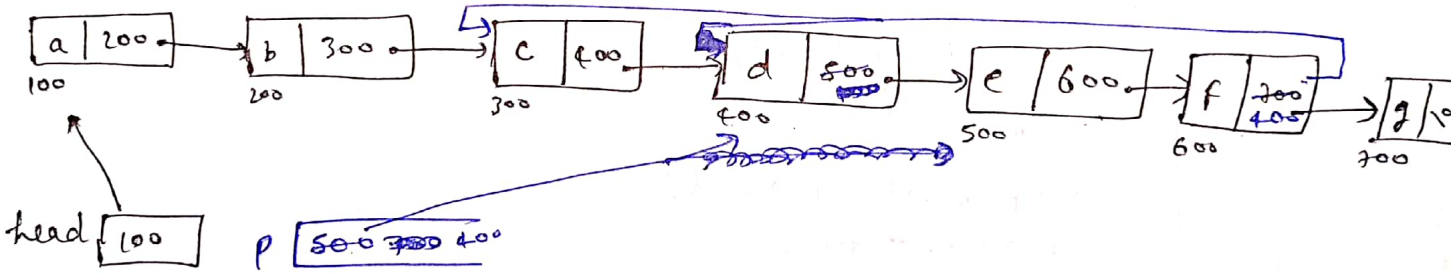
Quest: Consider the following linked list with the given statement what is the o/p printed?




```
Struct node *p;
p= head → next → next;
p→ next → next → next = p;(300)
head → next = p → next
pf("%d", head→ next→next→next→next→Data);
```

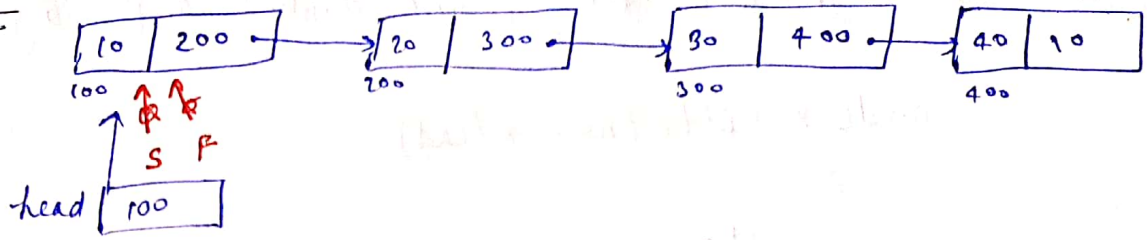
o/p → d

Quest: Consides the following linked list with the given statement what is the o/p printed!

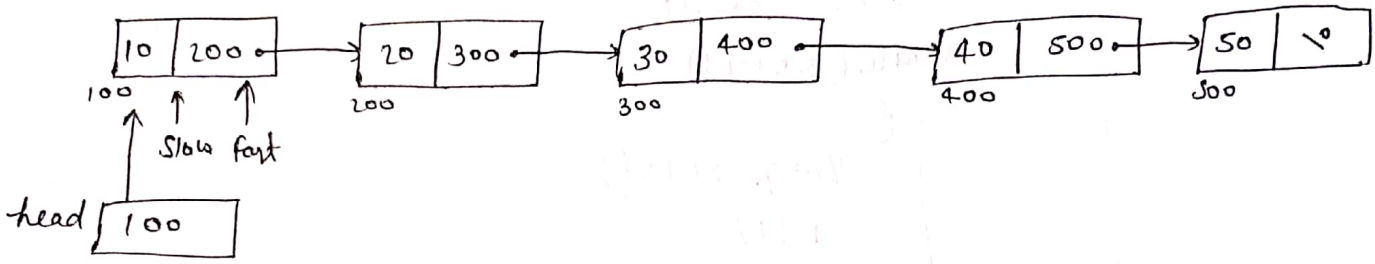



```
Struct node *p;
p= head → next → next → next → next;
head → next → next → next = p → next → next
pf("%d", p→ next → next→ next → next→ next → next→ data);
```

Workbook-8



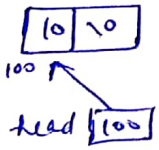
Ques: WAP to find middle of linked list without using length func.?




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

① head == NULL

②



Quest: WAP to find middle of the linked list using list length?

```
node * middle (node * head)
{
    int l = listlength (head);
    int k=0 , mid = l/2;
    node *q = head;
    while ( k<mid)
    {
        q= q -> next;
        k++;
    }
    return q;
}
```

GATE

Quest: Consider the function 'f' defined below.

```
struct item
  {
    int data;
    struct item *next;
  };
int f (struct item*p)
  {
return((p==NULL) || (P->Next==NULL) || ((p->data <= p->next->data && f(p->next))));
  }
```

for a given l.l. p the function 'f' returns 1 if and only if

(a) the list is empty and has exactly one element.

(b) The element in the are sorted in the non decreasing order of data

(c) " " " " " " non increasing

(d) not all elements in the list have the same data.

NOTE :

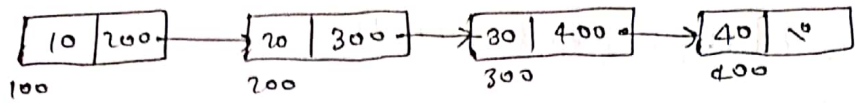
increasing

1,2,3,4,5

non-decreasing

1,2,2,3,4,4,5,6,7

Quest: WAP to delete to last node of given linked list?




```
delete AtEnd (node *head)
{
  if (head == NULL)
     return;
  if(head->next == NULL)
  {
     free(head);
     head = NULL;
  }
  else
  {
     node *p, *q = head;
     while (q->next != NULL)
     {
        p = q;
        q = q->next;
     }
     p->next = NULL;
     free(q);
     q = NULL;
     p = NULL;
  }
}
```

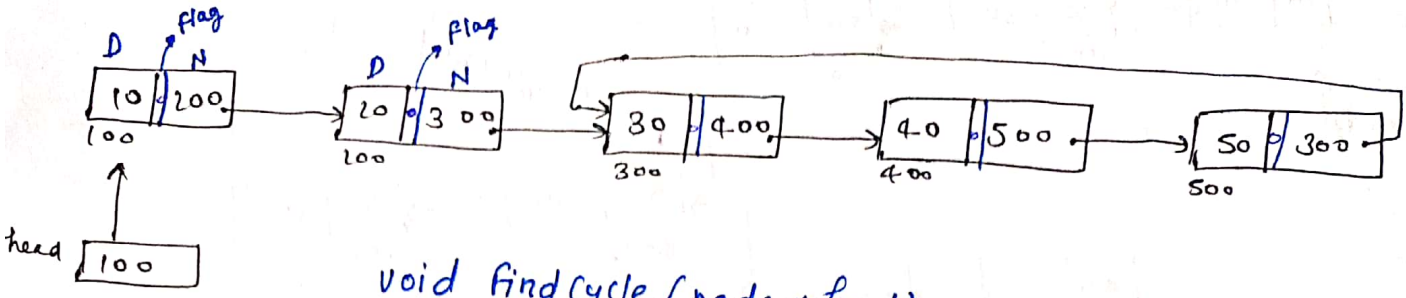
Quest: WAP to delete first node of the given ll.?

```
delete Atbegin (node *head)
  {
    if (head == NULL)
       return ;
    if (head -> next == NULL)
       free (head)
       head = NULL.
    }
  else
     { node *p = head;
       head = head -> next;
        free (p);
        p = NULL;
     }
}
```

Question: WAP to delete the node at given position?

```
delete AtPos (node * head, int pos)
  { L = listlength (node)
    if (pos <= 0 || pos > length)
       pf(" invalid");
    if (pos == 1)
      delete Atbegin (head);
    if (pos == length)
      delete AtEnd(node)
    else
      { node *p, *q = head;
        p=q;
        q = q -> next;
        k++;
      }
    p -> Next = q -> next;
      free(q);
      q = NULL;
      p = NULL;
}  }
```

Ques: WAP to find whether cycle exist or does not exist in the given linked list?

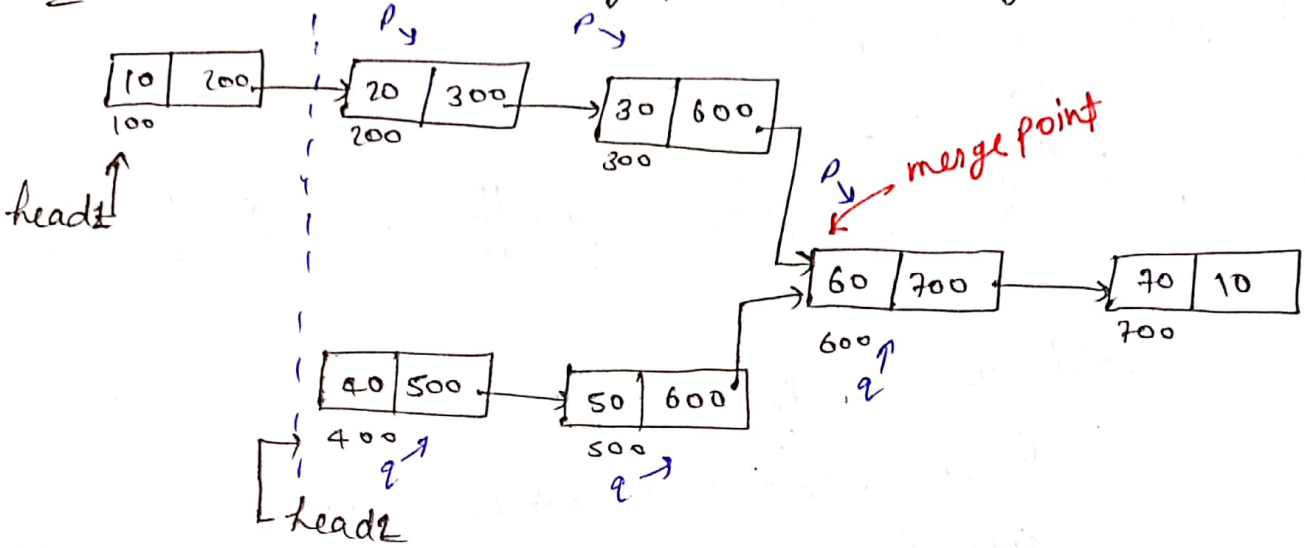



```
void findcycle(node *head)
{
  node *slow=head, *fast=head;
while (fast!=NULL && fast->next!=NULL && fast->next->next!=NULL)
  {
    slow = slow->next;
    fast= fast->next->next;
     if (slow==fast)
       {pf("cycle exist");
        exit();
       }
  }
PF("No cycle exists");
}
```

Sol2

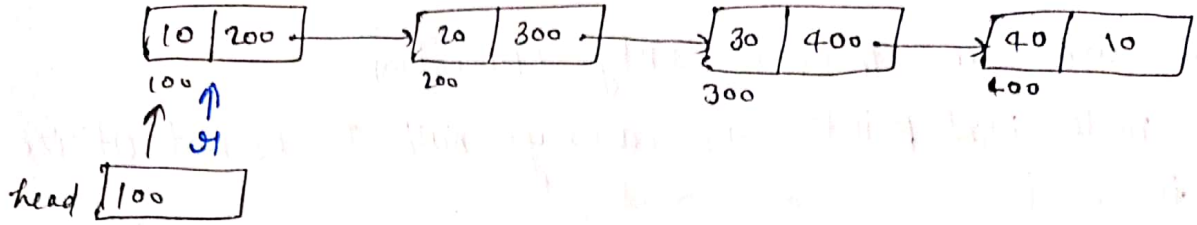
```
Struct node
  { int data;
    int flag;
    struct node *next;
  };
```

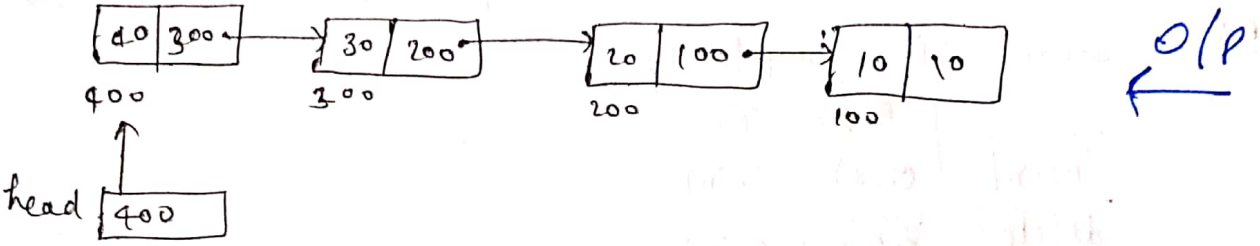
Ques: WAP to find merge point in the given linked list?




```
L1=5
L2=4
if (L1>L2)
diff = L1-L2;
else
 diff = L2-L1
```

Ques: WAP to reverse a given linked list?






```
node *reverse (node *head)
{
    node *p=NULL, *q=NULL, *r=head;
    while(r)
    {
        q=r->next;
        r->next=p;
        p=r;
        r=q;
    }
    return p;
}
```

Drawbacks of Singly Liked List:

1. In S.L.L. we can traverse only 1 Direction.
2. The last node next pointer is always null. It is not utilize properly, to avoi
3. To avoid this drawbacks we will implement circulas S.L.L.

NOTE:

① S.L.L. Time complexity

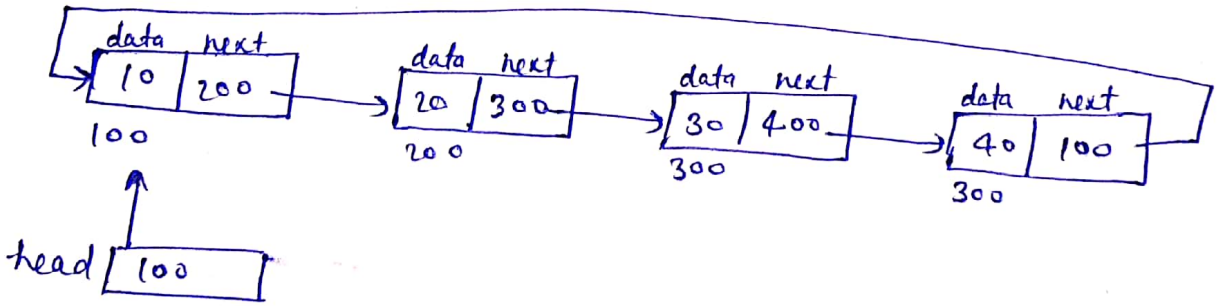
	Begin	End
insert	O(1)	O(n)
delete	O(1)	O(n)

② In the S.L.L. pointer p is given pointing to some node then delete that node what is the time complexity.

T.C. = O(n)

CIRCULAR LINKED LIST : [C.S.L.L.) or (S.C.L.L)]

In the C.S.L.L. the last node next pointer will always point to head node



Quest WAP to find length of C.S.L.L ?

```
int listlength CSLL(node* head)
{   int count=0;
    if(head==NULL)
      return count;
    else
    { node *p=head
      count=1;
      while(p->Next!=Read)
        {
          p=p->next;
          count++;
        }
      return count;
    }
```

Quest: WAP to insert a new node at the end of the C.S.LL.

```
insert AtEnd(SLL(node *head)
{ node *temp
  temp=(node*) malloc(sizeof(node));
   if(temp==NULL) exit();
  pf("enter the value");
  sf("%d", &temp->data);
   temp->next=head;
  if(head==NULL)
    { head=temp;
      head->next=head;
    }
  else
    { node *p=head;
      while(p->next!=head)
        { p=p->next;
        }
      temp->next=head;
      p->next=temp;
  } } temp=NULL; p=NULL;
```

H.W. WAP to insert the new node at the begning of the C.S.L.L ?

```
Insert Atbegin CSLL (node *head)
{ node *temp;
  temp = (node*) malloc (sizeof (node))
  if (temp == NULL) exit();
  Pf("\n enter the node");
  Sf("%d", &temp -> data);
   if( head == NULL)
     { head = temp;
       temp -> next = head;
     }
  else
     { node *p = head;
       while (p -> next != head)
          { p = p -> next
          }
       temp -> next = head;
         p -> next = temp;
       head = temp;
     }
}
```

Quest: WAP to delete the first node of the C.S.L.L ?

```
delete Atbegin (.S.L.L. ( node *head)
{  if ( head == NULL)
     return;
   if (head -> next == NULL)
       free (head);
       head = NULL;
  else
     { node *p = head , *ptr = head;
       while (p -> next != head)
          { p = p -> next;
          }
       p -> next = head -> next;
       free(head);
       head = head -> next;
         free (ptr);
         ptr = NULL;
         p = NULL;
}
```

H.W WAP to delete the last node of the C.S.L.L.?

NOTE :

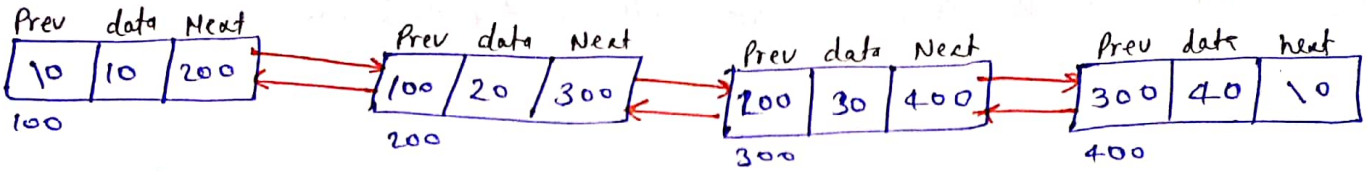
T.C. → C.S.L.L

	Begin	End
insert	O(n)	O(n)
delete	O(n)	O(n)

DOUBLE LINKED LIST (D.L.L.) :

1. The advanvdage of the D.L.L. is that traverse is both the directions.
2. In the D.L.L. every node contains two pointers points to previous node and pointing to next node
3. In the D.L.L. Insertition and deletion will talce more amoont of time because of more pointer updation.

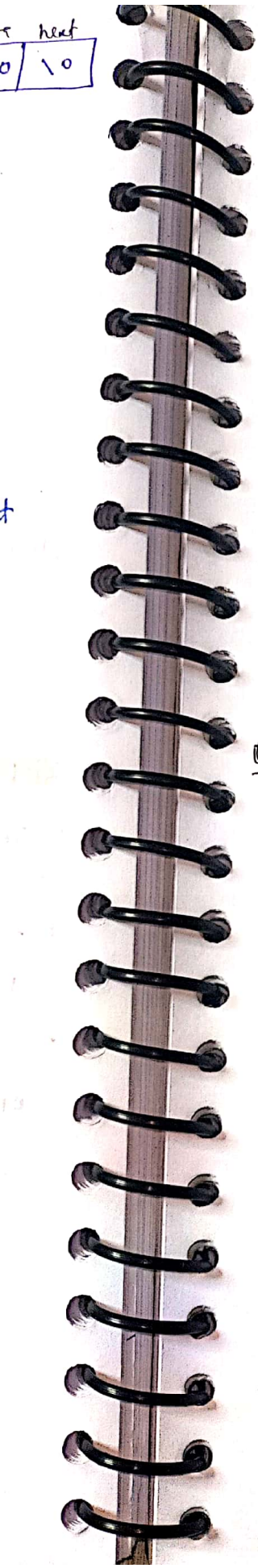
```
Struct DLLnode
  {
     Struct DLLnode *prev;
     Int data;
     Struct DLLnode *next
  }
typedef Struct DLLnode node;
```



Ques: WAP to insert a new node at the begining of D.L.L.?

```
insertAtBeginDLL(node *head)
{
    node *temp;
    temp = (node*)malloc(sizeof(node));
    if(temp==NULL)
        exit()
    PF("Enter the node");
    SF("%d", &temp->data);
    temp->prev = NULL;
    temp->Next = NULL;
    if(head==NULL)
    {
        head = temp;
    }
    else
    {
        temp->next = head;
        head->prev = temp;
        head = temp;
        temp = NULL;
    }
}
```

T.C. = Constant



Ques: WAP to insert the new node at the End of the D.L.L?

Ques: WAP to delete the last node of D.L.L?

```
deleteAtEnd DLL (node * head)
{ if (head == NULL)
    return
  if (head->next == NULL)
    { free(head);
      head = NULL;
    }
  else
    { node *p = head;
      while (p->next != NULL)
        { p = p->next;
        }
      p->prev->next = NULL
      free(p);
      p = NULL;
    }
}
```

T.C.= O(n)

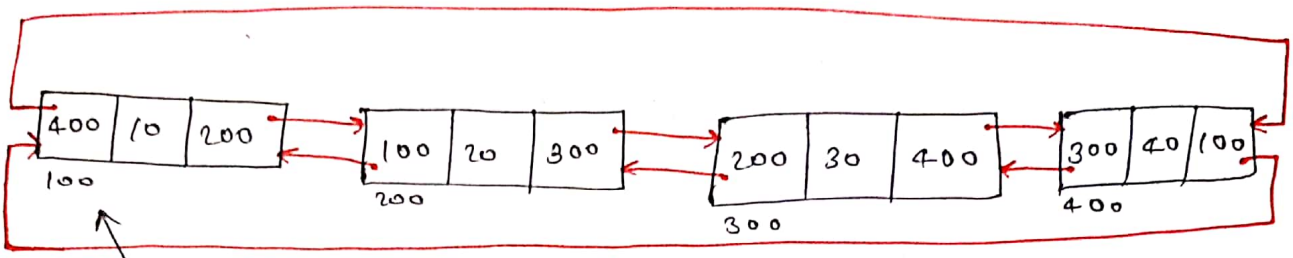
H.W WAP to delete of the firstnode of the D.L.L. ?

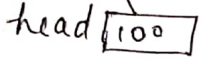
Note : T.C. → D.LL

	Begin	End
Insert	O(1)	O(n)
delete	O(1)	O(n)

Circular D.L.L :

In the CDLL first node previous will point to last node and the last node next will point to Read node.





T.C → CDLL

	Begin	End
insert	O(1)	O(1)
delete	O(1)	O(1)

H.W

1. Insert At Begin
2. Insert At End
3. delete At Begin
4. delete At End

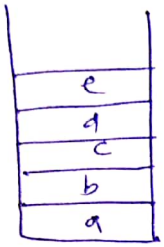
NOTE:

* After the L.L. the next level data structure is "Skip List". Skip List treat as advance data structure

STACK :

1. Stack is a d.s. used to store the data elements with the restriction of Last in first out / first in last Out.
2. The elements are inserted and deleted from the same end called "Top".

e.g.



element : a, b, c, d, e
insertion : a, b, c, d, e
deletion : e, d, c, b, a

Application :

1. Recursive function calls
2. Balancing paranthesis
3. HTML & XML tag checking
4. Page back (or visited) in browser
5. Redo and Undo
6. Infix to postfix conversation
7. Post fix evaluation
8. Fibbonacci Series
9. Tower of Tanoi

STACK ADT :

1. Declaration of Data :

(a) Space to store data elements.
(b) A variable Top pointing to top most element.

2. Declaration of Operation :

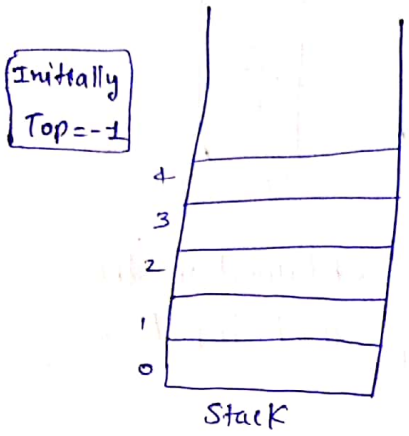
(a) push (x); → element to insert into the stack with the help of Top
(b) pop (); → deletes an element from the stack with the help of top
(c) isEmpty() → Returns True when the stack is empty.
(d) isFull() → Returns True when the stack is full

Implementation Stack :

Using Array :-

int s[N] N=5

underflow condition : Top == -1



Overflow condition :- Top == N-1

Qust: WAP to perform push operation in Stack ?

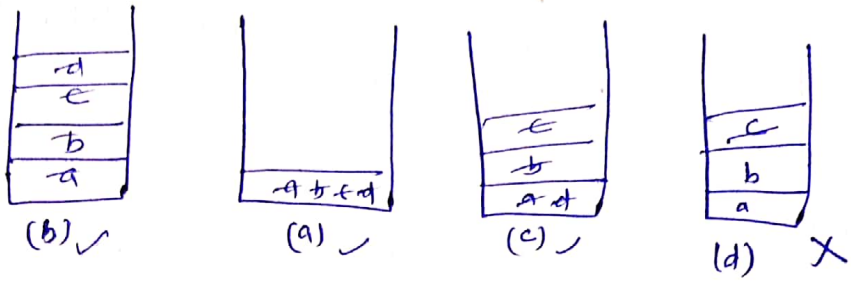
```
Push (S, N, Top, n)   // S→Stack, N→size, Top→Top, n→element to insert
{
  if (Top == N-1)
    { pf("over flow");
      exit();
  else
    {
      Top++;
      S[Top] = n; // S[++Top] = n;
    }
```

Qust: WAP to perform POP operation in the stack. ?

```
int pop(S, N, Top)
  { if (Top == -1)
      { pf ("underflow");
        exit();
      }
    else
      { int n;
        n = S[Top];
        Top--;
        return n;
  } }
```

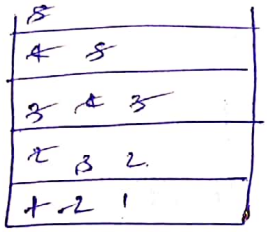
GATE A program attempts to generate as many strings as possible of the string "abcd" by pushing the characters a, b, c, d in the same order onto a stack, but it may pop off the top character at any time which one of the following.

(a) a b c d
(b) d c b a
(c) c b a d
(d) c a b d



GATE which of the following permutations can be obtained in the output (in the same order) using a stack assuming that input is the sequence 1, 2, 3, 4, 5, is order?

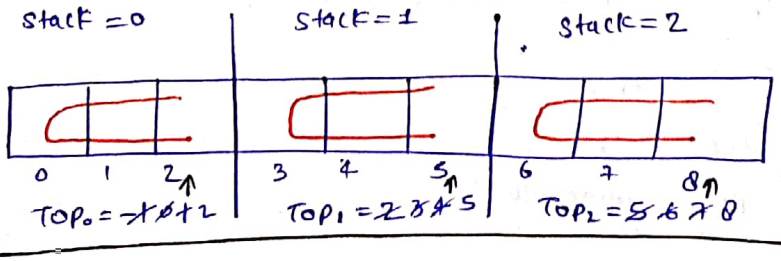
(a) 3, 4, 5, 1, 2
(b) 5, 4, 3, 1, 2
(c) 1, 5, 2, 3, 4
(d) 3, 4, 5, 2, 1



Implementing Multiple Stack in a Single Array :-

1. In Order to execute one recursive program the require one Stack.
2. If we want to execute multiple recursive programs we need multiple stacks in the memory. This is similar to implementing multiple stacks 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 (i)th Stack = $(i) * \frac{N}{M} - 1$

Initial Top of stack (0) = $0 * \frac{9}{3} - 1 \Rightarrow 1$

Top = -1

Initial Top of stack (1) = $1 * \frac{9}{3} - 1 \Rightarrow 2$

$Top_1 = 2$

Initial Top of stack (2) = $2 * \frac{9}{3} - 1 = 5$

$Top_2 = 5$

NOTE:

* ith Stack overflow condition arises when the top i initial Top of the (i+1) stack.

Ques: WAP to perform push operation in the ith stack?

```
int Push(S, N, M, i, Top_i, n)        // S: stack, N: size, M: no. of stacks, i: stack #, n: element to insert
{ if (Top_i == (i+1) * N/M - 1)
    { pf("overflow");
      exit(1);
    }
  else
    { Top++i
      S[Top] = n;  // S[Top++] = n
    }
}
```

Ques: WAP to perform pop operation in the ith stack?

```
int Pop(S, N, M, i, Top_i, n)
{ if (Top_i == i * N/M - 1)
    { pf("underflow");
      exit(1);
    }
  else
    { int n;
      n = S[Top];
      Top_i--;
      return n;
    }
}
```

NOTE:

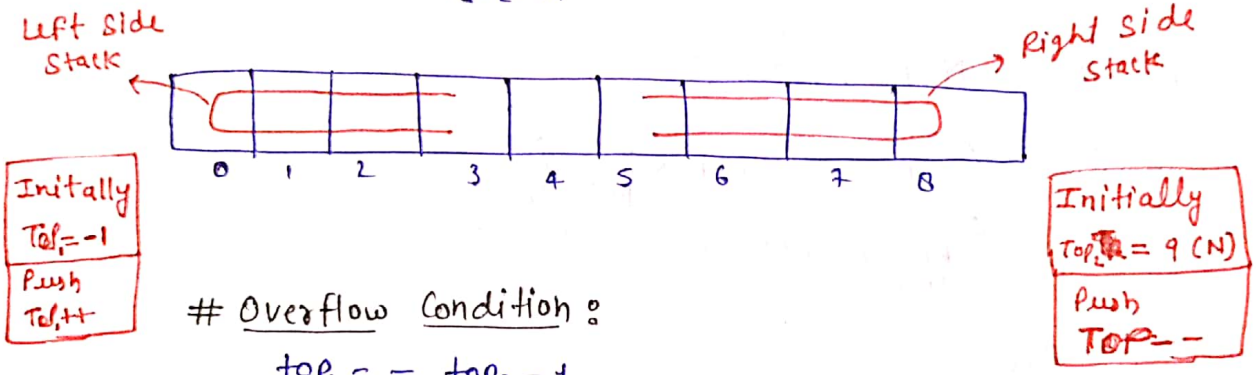
→ In the above implementation if one stack is full and all the other stack are empty then we can not make use of empty stack available.

→ Even though empty slot available in the memory they are not efficiently utilize to avoid this problem we will use another approch

Implementing Multiple Stack in a Single Array efficient way :

In this implementation we use two stack in a single array growing in opposite direction.

array size (N)=9 int S[N];



Overflow Condition :

$top_1 == top_2 - 1$

$top_2 == top_1 + 1$

$top_2 - top_1 = 1$

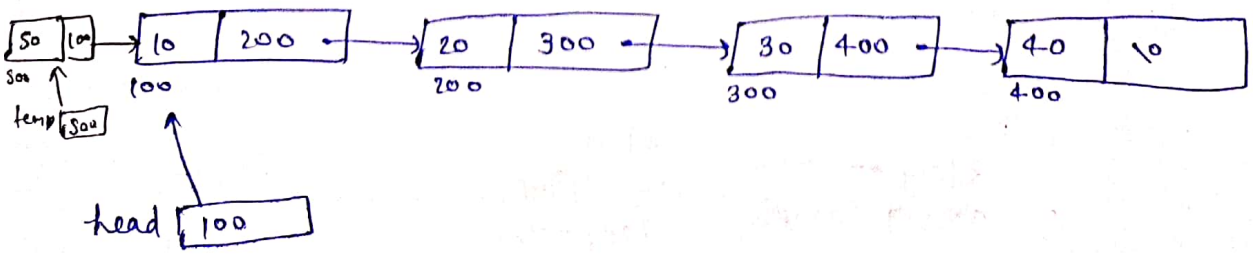
Underflow Condition :

$top_1 == -1$

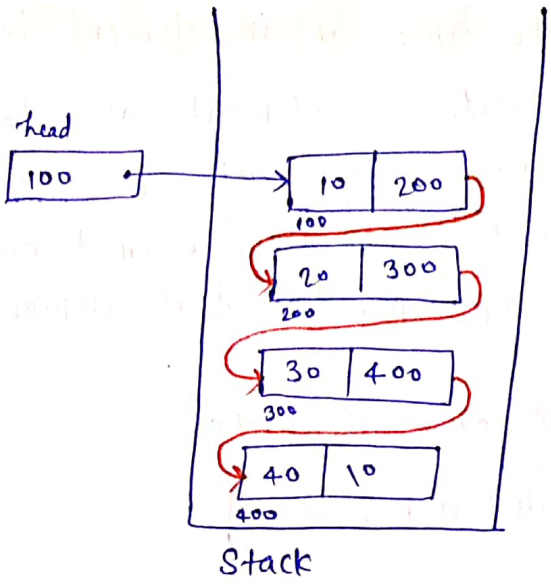
$top_2 == N$

Implementing Stack using linked list :

1. In this implementation using linked list insertion and deletion happens from the head node only.
2. Always new node will be inserted at the first and always deletion happen at the first node.



# Underflow :-
head = = NULL



Quest: WAP to perform push operation in stack using Linked List?

```
push (node * head , int n)        // n → data element
{ node * temp
temp = (node*) malloc (sizeof (node))
if (temp == NULL)
{ exit();  → pf("overflow");
}
temp → value = n;
temp → next = head;
head = temp;
temp = null;
}
```

T.C = O(1)

Quest: WAP to perform pop operation in the stack implemented using L.L.?

```
int pop(node * head)
{ if (head == NULL)
    pf("underflow");
    exit();
  else
  { int n;
    node * p = head;
    n = head → data;
    head = head → next;
    free (p);
    p = NULL;
    return n;
}
```

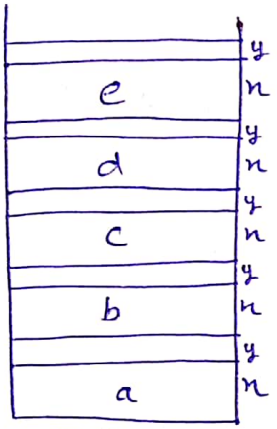
Finding average life Time of an element in the Stack :

→ The time for insert/Delete an element takes 'n', and there is a delay of 'y' between every such operation.

→ The life time is Consider as time elapsed elapsed from end of push to start of pop operation that remove an element from the stack.

→ Find average life time of a element :

elements : a, b, c, d, e



element	Life Time
e	y
d	$y+n+y+n+y = 2n+3y$
c	$4n+5y$
b	$6n+7y$
a	$8n+9y$

Quest 38 P.No. 144 } take above example ↑

$$y + 2n + 3y + 4n + 5y + 6n + 7y + 8n + 9y$$

$$2n + 4n + 6n + y + 3y + 5y + 7y + 9y$$

$$2n(1+2+3+\cdots+n-1) + y(1+3+5+7+\cdots n)$$

$$\not{2}n\left(\frac{(n-1) * ((n-1)+1)}{\not{2}}\right) + y * n^2$$

$$n(n-1) * n + y * n^2$$

For avg $\quad \dfrac{n(n-1) * n + y * n^2}{n}$

$$n(n-1) + y * n$$

$$n * n - n + n * y$$

$$\boxed{n(n+y) - n}$$

Infix, Postfix, Prefix : Postfix → Reverse Polish Notation.

Infix :→ Binary operation b/w two operands.

e.g. a + b
↙ ↓ → operand
Operand operator

Prefix :→ Binary operation before two operands.

e.g. + a b
↙ ↘ operands
operator operand

Postfix :→ Binarary operation after two operands

e.g. a b +
↙ ↓ ↘ operator
Op1 Op2

Ques: Infix :- A + B * (C + D) / F + D * E

Prefix :- A + B * + CD / F + D * E

A + * B + CD / F + D * E

A + / * B + CD F + D * E

A + / * B + CDF + * DE

+ + A / * B + CDF + * DE

++A/*B+CDF*DE Ans

Post fix: ~~A + B * C + D~~ A + B * CD + / F + D * E

A + BCD * / F + D * E

A + BCD * F / + D * E

A + BCD * F / + DE *

A B CD * F / + + DE *

A B CD * F / + DE * + Ans

Prefix to postfix Conversion :

To convert from prefix to post-fix we need

① operator followed by

② two immediate operands.

then convert it into postfix.

e.g. Prefix :- * + cd - ba

Postfix : * cd+ - ba

* cd + - ba

* cd+ ba -

cd+ba - *

Infix to prefix Infix to postfix	According to Precedance table
Prefix to postfix Prefix to Infix	+ cd , operator after two operands.

Ques: Prefix

↳ * ↑ / + cae * db ↑ fg

* ↑ / ca+e * db ↑ fg

* ↑ / ca+e db* ↑ fg

* ↑ / ca+e db* fg↑

* ↑ ca+e/ db* fg↑

* ca+e/db* ↑ fg↑

Post → ca+e/db*↑fg↑*

Ques: match the following :

Prefix	Postfix
a) * + ab - cd	① ab* cd+- d
b) + - ab * cd	② ab* cd - + e
3) + * ab - cd	③ ab+cd - * a
4) - * ab + cd	④ ab - cd* + b

Prefix to Infix Conversion:

e.g Prefix $*+ad-cd$

$*(a+d)-cd$

$*(a+d)(c-d)$

Infix :→ $(a+d)*(c-d)$

Postfix to Infix Conversion:

Two operands followed by immediate operator that convert it into infix.

Eg

Postfix → $ab*cd/-$

$(a*b)cd/-$

$(a*b)(c/d)-$

Infix → $(a*b)-(c/d)$

NOTE:

1. In order to evaluate infix expression we need to scan infix string multiple times jumping from one place to other place according to the precedence of the operator.
2. The procedure must be sequential and it requires the good solution the best solution is converting infix to postfix expression in a single scan with the help of stack and evaluating the postfix expression in a single scan with the help of stack.

Infix to postfix conversion : using operator Stack

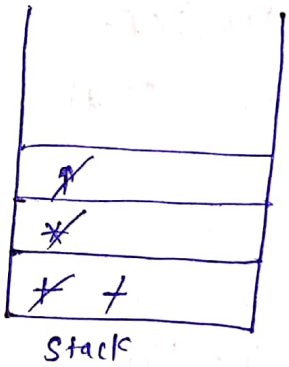
Top of Stack	Next Operator	operation
Low	High	Push
High	Low	pop
Same	Same (L→R)	pop
Same	Same (R→L)	push

* If any open braces comes from there treat it as new start until the closing braces.

e.g. ① infix :- a+b*c↑d-e

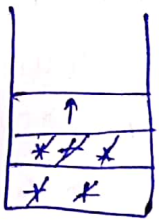
Postfix :- abcd↑*+e-

Max. Stack Size Used :- 3



e.g ② infix :- a+b*c-d/e↑f*g+h

Postfix : abc*+def↑/g*-h+



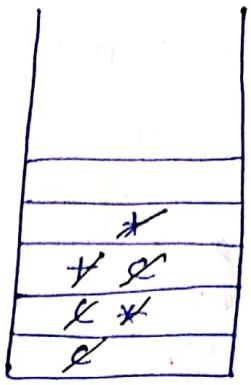
e.g ③ when we will Search Symbol '↑' while scanning during conversion from infix to post fix what will be the content of stack Infix : a*b/-c/e*f+g

a) -, *
b) /, *
c) *, +
d) *, -

*** Infix: ((P+Q) * (R+S))

Postfix: PQ+RS+*

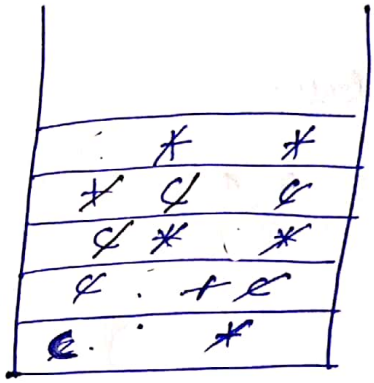
M.S.U. = 4



e.g. Infix :- (((P+Q) * (R+S))/T) + (A * (B+C))

Postfix :- PQ+RS+ *T/ ABC+*+

M.S.U. = 5



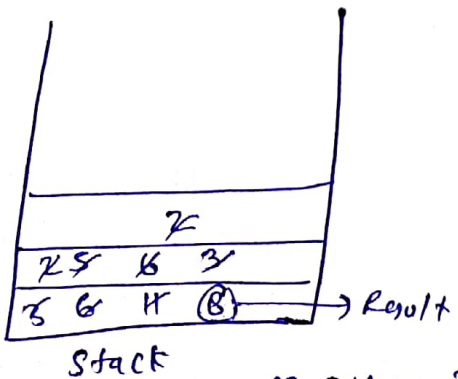
Postfix Evaluation : using operand stack

NOTE : ① If (Symbol == operand)
Push (symbol);

② If (Symbol == operators)
(1) OP_2 = pop();
(2) OP_1 = pop();
(3) push (OP_1 < operator > OP_2);

③ At the end result will be in the stack.

e.g. Postfix : 32 * 5 + 62 / -

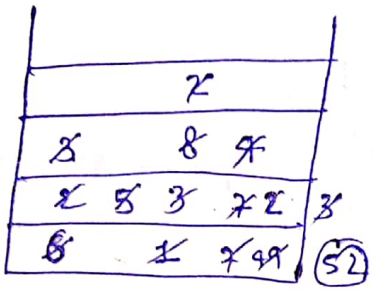


M.S.U. = 3

OP_1	operator	OP_2	Result
3	*	2	6
6	+	5	11
6	/	2	3
11	-	3	8

e.g. Postfix :

6 2 3 + − 3 8 2 / + * 2 ↑ 3 +



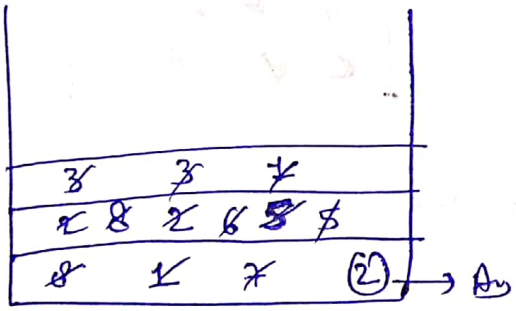
M.S.U = 4

Op1	Operator	Op2	Result
2	+	3	5
6	−	5	1
8	/	2	4
3	+	4	7
1	*	7	7
7	↑	2	49
49	+	3	52

52 Ans

e.g. Postfix :

8 2 3 ↑ / 2 3 * + 5 1 * −



M.S.U = 3

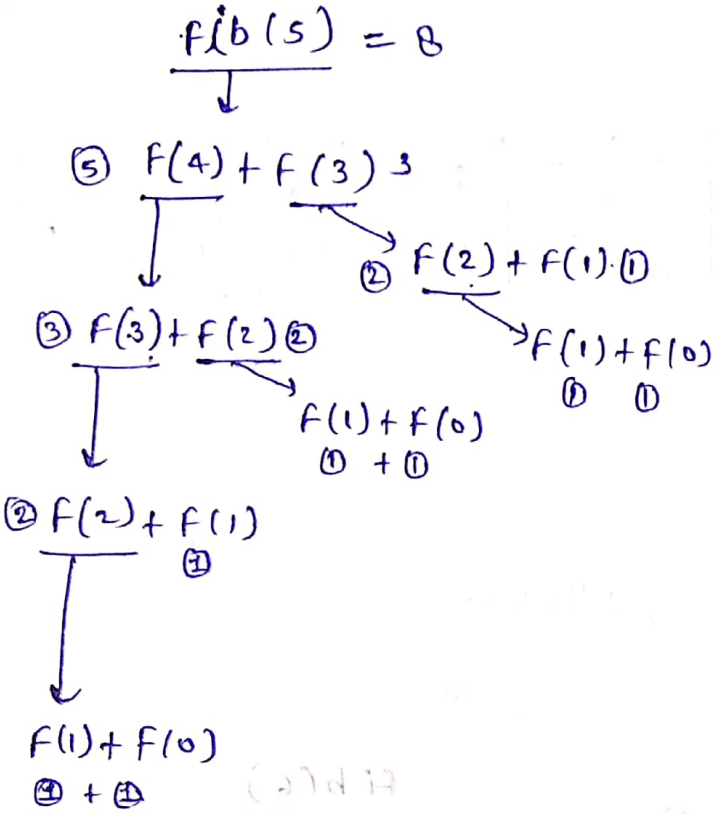
Op1	Operator	Op2	Result
2	↑	3	8
8	/	8	1
2	*	3	6
1	+	6	7
5	/	1	5
7	−	5	2

Fibonacci Series:

n	0	1	2	3	4	5	6	7	8	9	10	11
fib(n)	0	1	1	2	3	5	8	13	21	34	55	89

$$fib(n) = \begin{cases} n, & \text{if } (n==0) \text{ or } n==1 \\ fib(n-1) + fib(n-2), & \text{otherwise} \end{cases}$$

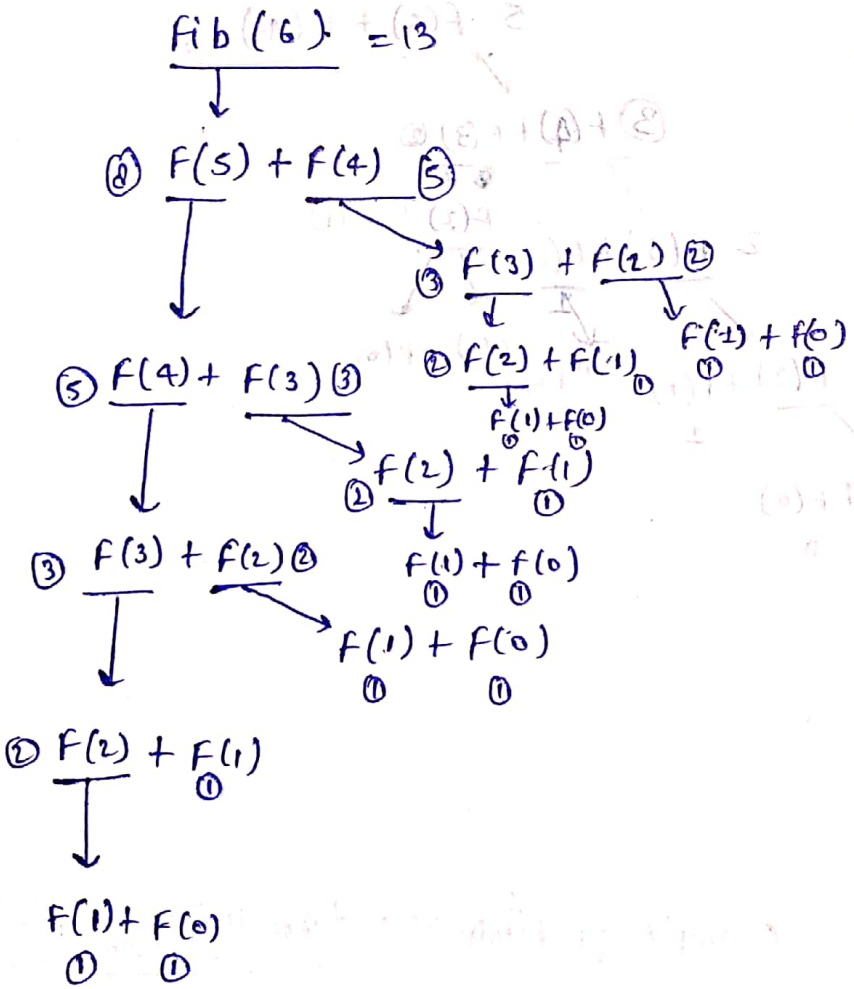
Example of fibonacci turn this page →



No. of addition = 7

No. of function calls = 15

fib(5) = 8



No. of addition = 12

No. of function calls = 25

fib(6) = 13

fib(n)	No. of additions
0	0
1	0
2	1
3	2
4	4
5	7
6	12
7	20
8	33
9	54
10	88

No. of additions in fibb. = No. of additions in fib(n-1) + No. of additions in fib(n-2) + 1

fib(n)	No. of fun. calls
0	1
1	1
2	3
3	5
4	9
5	15
6	25
7	41
8	67
9	109
10	177

T.C = $O(2^n)$

No. of function call in fibb. = No. of function calls in fib(n-1) + No. of function calls in fib(n-2) + 1

QUEUE :

1. Queue is the D.S. use to store the data elements with the restriction of First in first Out / LAST IN LAST OUT.
2. The elements are inserted from one end and deleted from another end.
3. It requires two pointers
   (1) front ⇒ used for deletion of an element
   (2) Rear ⇒ used for insertion of an element.

Application :

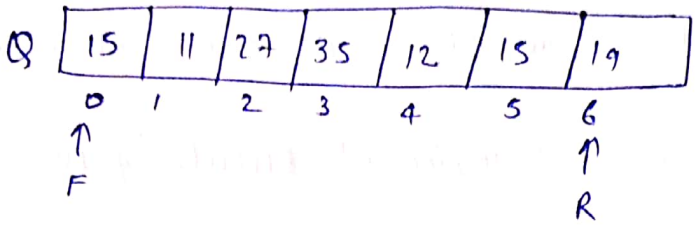
1. Ticket Reservation
2. Job Scheduling
3. Printer Spooler Daemon

Queue ADT :

1. Declaration of data :-
   → Space to store queue elments.
   → Two pointers front and rear
2. Declaration of operations :-
   → enqueue (x);
   ↳ element to insert
   Insert the element in the queue with the help of rear pointe
   → dequeue (x) ;
   ↳ release the element from the queue with the help of front pointer.
   → isEmpty(); :- return queue if the queue is empty
   → isFull(); :- return to queue if the queue is full.

Implementation of Queue using Array :-

int Q[N] , N=7



Initially
front = -1
Rear = -1

Overflow Condition :
(R == N-1)

underflow condition :
(F == -1)

Quest: WAP to perform ENQUEUE Operation in the queue ?

```
enqueue (Q, N, F, R, x)        // Q: queue, N: size, F: front, R: rear, x: element
{
    if (R == N-1)
    {
        pf("overflow");
        exit(1);
    }
    if (R == -1)
    {
        F++;
        R++;
    }                          // first insertion
    else
    {
        R++;
    }
    Q[R] = x;
}
```

Quest: WAP to perform DEQUEUE operation in the Queue ?

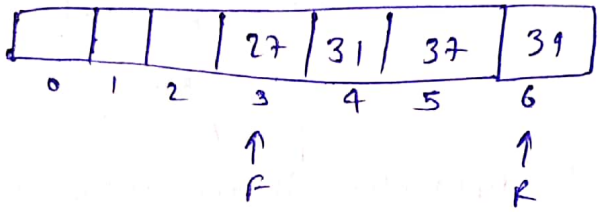
```
dequeue (Q, N, F, R)
{
    int x;
    if (F == -1)
    {
        pf("underflow");
        exit();
    }
    x = Q[F];
    if (F == R)
    {
        F = -1;
        R = -1;
    }                          // last delete
    else
    {
        F++;
    }
    return x;
}
```

F = 0, 1, 2, 3, 4, 5, 6, -1
R = 6, -1

* Initially check underflow then simple exit().
* if it is not underflow than delete the element by using front pointer
* if front & Rear both are same it is the last deletion of queue than reinilize both the pointer to -1 otherwise increament only front pointer then return deleted element

NOTE: In the above implimentation when the rear pointer reach us to right most point and front pointer is somewhere in the middle then still it will show overflow condition.

2. Even though empty slots are available in the queue they are not properly utilized.
3. To avoid this problem we will implement circular queue.

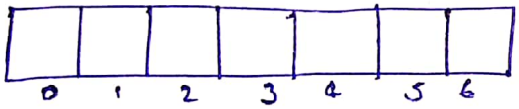


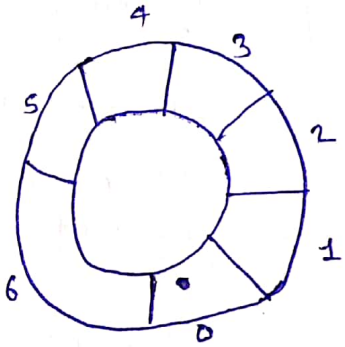
F = -1, 0, 1, 2, 3

R = -1, 0, 1, 2, 3, 4, 5, 6

Circular Queue:

int Q[N], N=7





Initially
F=-1
R=-1

#overflow:
(R+1) % N == F

#underflow:
F==-1

R= (R+1) % N;
F= (F+1) % N;

Quest: WAP to Perform ENQUEUE Operation in Circular Queue?

```
Circular Enqueue (Q, N, F, R, x)
{ if ((R+1) % N == F)
  {
    pf("Overflow");
    exit(1);
  }
  if (R == -1)
  {
    R++;
    F++;
  }
  else
  {
    R = (R+1) % N;
  }
  Q[R] = x;
}
```

Quest: WAP to perform DEQUEUE Operation in Circular Queue?

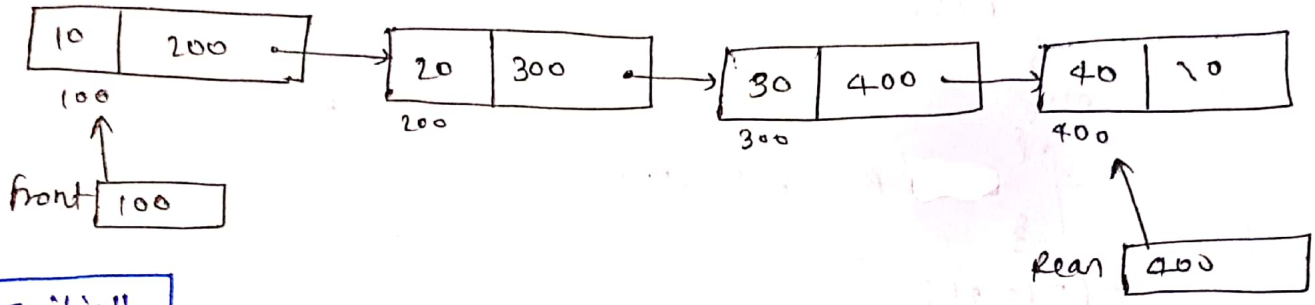
```
int Circular Dequeue (Q, N, F, R)
{ int x;
  if (F == -1)
  {
    pf("underflow");
    exit(1);
  }
  x = Q[F];
  if (F == R)
  {
    F = -1;
    R = -1;
  }
  else
  {
    F = ((F+1) % N) ;
  }
  return x;
}
```

last deletion

T.C. = O(1)

Implimenting Queue Using Linked List:

1. To impliment the queue using L.L. we need two pointers front and rear initially both will be assign to null
2. when the first insertion happens then both front and rear will point to first node.
3. front pointer used for deletion and rear is used for insertion



Initially
F = NULL
R = NULL

Ques: WAP to perform operation Enqueue implimenting using Linked List?

```
Enqueue LL(node* front, node *rear, int n)
{ node * temp;
  temp= (node*)malloc(sizeof(node))
  if(temp==NULL)          PF("overflow") exit();

  temp→data= n
  temp→Next = NULL;
  if(Rear==NULL)
  {  front = temp;
     Rear = temp;
  }
else
  {
    Rear→next= temp;
     Rear=temp;
  }
}
```

T.C. = O(1)

Quest: WAP to perform Dequeue operation in queue implemented using L.L. ?

```
int dequeue LL ( node * front , node * rear)
{
  if( front == NULL)
    { pf("Underflow ");
      exit();
    }
  n = front -> data;
  If ( front == Rear)
    {
      front = NULL;
      Rear = NULL;
    }
  3
else
  { node *p = front

    front = front -> next;
  3
free (p)
p = NULL
return n;
3
```

} last deletion in the queue

Quest: Consider the following function which takes 'Q' element as a input and performs some functionality

```
Q = 5, 6, 10, 15
ABC (Queue Q)
 {
   if (!isEmpty
     {
       n = deque (Q);
         ABC (Q);
       enqueue (Q, n);
     }
 }
```

what will the values in Q after completion of the above program?

O/p → 15, 10, 6, 5

Quest: Consider the following program which uses '3' data structure Queue$_1$ (Q_1), Queue$_2$ (Q_2), and Stack (st) the program takes 'Q_1' values as input and perform some functionality.

Q_1 = -31, 17, 8, 23, 11, 0, -16, 4, 18, 25, 36, 0, -18, 21

```
xyz (Queue Q1)
 { int s=0;
   while (!Empty (Q1))
     {
       n = dequeue (Q1);
       if (n==0)
         { s = s + pop (st)
           enqueue (Q2, s);
           s=0;
         }
       Push (st, n)
     }
}
```

Q1. what is sum of all values of stack after completion of the Program 51 ?

Q2. what is the max stack size used 12 ?

Q3. what is sum of all values of Q_2 after completion of the program 48

Q4. what is max size of Q_2 used 2 ?

Ques: Consider the following

Q_1 = 13, 7, 16, -20, -19, 0, 41, 15, 12, -8, -6, 0, -18, 31

```
ABC (Queue Q1)
{ while (!isEmpty(Q1))
  { n = dequeue(Q1);
    if (n==0)
    { pop(st);
      pop(st);
      while (!isEmpty(st));   // st: stack
      enqueue(Q2, pop(st));
    }
    Push(st, n);
  }
}
```

Q1 – 13
Q2 – 6
Q3 – 104
Q4 – 7

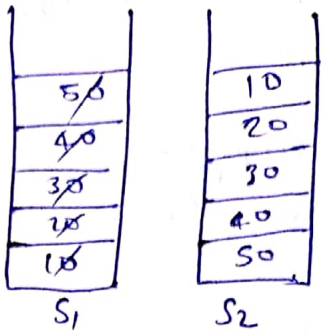
Ques: Q_1 = 13, -17, 6, 19, 21, 0, -14, 15, 12, 8, 6, 0, -18, 31

```
ABC (Queue Q1)
{
  static int s;
  while (!isEmpty(Q1))
  { n = dequeue(Q1);
    if(n==0)
    { while (!isEmpty(st))
        s = s + pop(st);
        s = s + s;
      } enque(Q2, s);
    Push(st, n)
  }
```

Q1 – 13
Q2 – 6
Q3 – 306
Q4 – 2

Implimenting Queue using Stack:

Q1 | 10 | 20 | 30 | 40 | 50 |



write the pseudo code for enque and deque operation?

```
enqueue(S1, n)
{
  push(S1, n)
}
```

```
int dequeue(S1, S2)
{
  if (S2 is not Empty)
    return pop(S2);
  if (S1 is empty)
  {
    pf("Queue is empty");
    exit(1);
  }
  else
  { while (S1 is not Empty)
    { n = pop(S1); // pop all element from S1
      push (S2, n); // push them into S2
    }
    return pop(S2);
  }
}
```

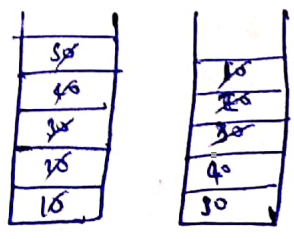
GATE Consider the above implimentation of Queue using two stack let 'n' insert and m (≤n) delete operations be performed in an arbitary order on an empty queue Q. Let n and y be the number of push and pop operations performed respectively in the process which one of the followin is TRUE for all 'm' and 'n'?

(a) $n+m \le n \le 2n$ and $2m \le y \le n+m$

(b) $n+m \le n < 2n$ and $2m \le y \le 2n$ n=5 m=3

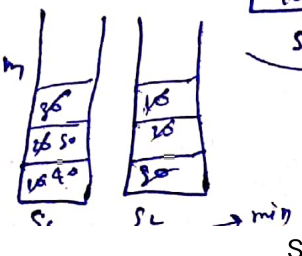
(c) $2m \le n < 2n$ and $2m \le y \le n+m$

(d) $2m \le n < 2n$ and $2m \le y \le 2n$



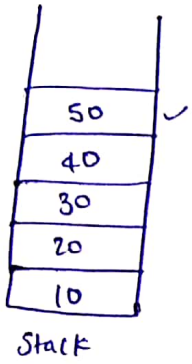
max → Push = 5+5 = 2n
pop = 5+3 = 8 = n+m

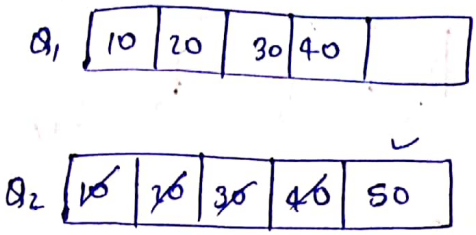
min → push = 5+3 = 8 = n+m
pop = 3+3 = 6 = 2m



Implimenting stack using stack:

elements = 10, 20, 30, 40, 50



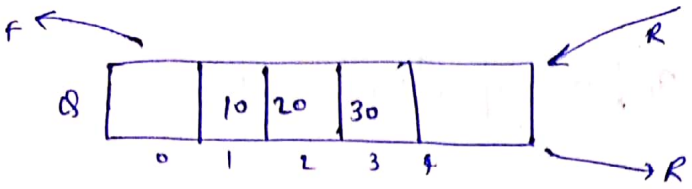


Quest: write the pseudo code for push and pop operation?

```
Push (Q1, Q2, n)
{
  if (Q1 is Empty)
     Enqueue (Q2, n);
  else
     Enqueue (Q1, n);
}
```

```
int pop (Q1, Q2)
{ if (Q1 is Empty)
  { if (Q2 is Empty)
    {
      pf("Stack is empty");
      exit(1);
    }
    else
    { while (Q2 doesnot contain one elem.)
      {
        n = dequeue (Q2);
        enqueue (Q1, n);
      }
      return dequeue (Q2);
    }
  }
  else
  { while (Q1 doesnot contain one elem.)
    {
      n = dequeue (Q1);
      enqueue (Q2, n);
    }
    return dequeue (Q1);
  }
}
```

Input Restricted Queue: enqueue operation is restricted but dequeue can be done from both front and rear.



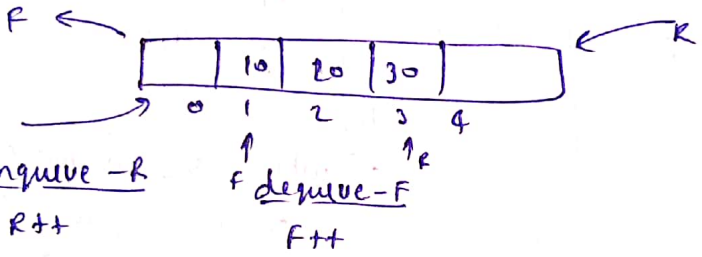
enqueue-R
R++

dequeue-F
F++

dequeue-R
R--

* it is not following the queue property.
* it is following both first in first out / Last in first out

Output Restricted Queue: dequeue operation is restricted but enqueue can be done from both front and rear



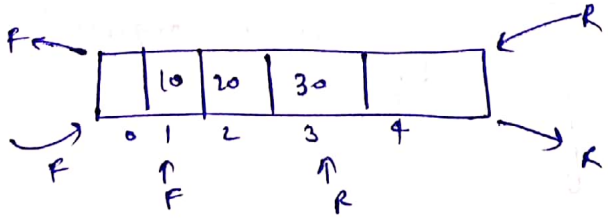
enqueue-R
R++

dequeue-F
F++

enqueue-F
F--

* it is not following the queue property.
* it is following both FIFO / LIFO

Double Ended Queue: enqueue / dequeue can be done for both front and rear



enqueue-R
R++

dequeue-F
F++

enqueue-F
F--

dequeue-R
R--

* It is not following queue property.
* It is following FIFO / LIFO

Priority Queue :

1. Ascending priority Queue (delete min)
2. Descending priority Queue (delete max)

elements : 15, 30, 20, 5, 35, 8, 17

(1) Ascending priority Queue

5, 8, 15, 17, 20, 30, 35

(2) Descending Priority Queue :

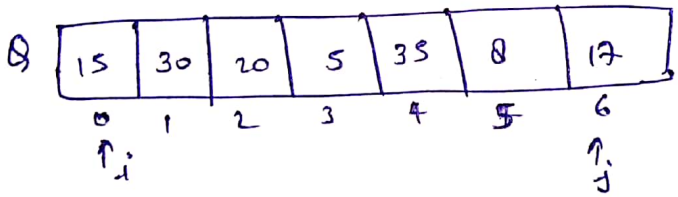
35, 30, 20, 17, 15, 8, 5

Implimenting Ascending Priority Queue : (delete min)

① Using unsorted Array :

elements : 15, 30, 20, 5, 35, 8, 17

int Q[N] N=7



For dequeue :

① Scan the entire array find the position of min element.
② x = Q[min] ; // 5
③ j will always point to last index
④ Q[min] = Q[j];
⑤ j--;
⑥ return x;

T.C. = O(n)

For enqueue :

Q[j] = x;

T.C = O(1)

i = -1, 0

j = -1, 0, 1, 2, 3, 4, 5, 6

(2) Using Sorted array:

enqueue:

$T.C. = O(n)$

dequeue:

$T.C. = O(1)$

TREE's:

Tree is a non-linear data structure used to store a data elements.

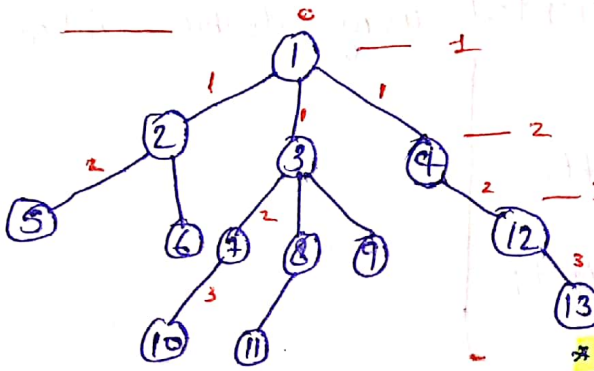
Types of Tree:

1. n-ary tree
2. Binary tree

Degree of a tree:

Degree of a tree decides no. of child nodes, a node can have.

e.g.



* leaf nodes also called leaves.
* degree of tree = 3
* height = 3
* leaf nodes = 5, 6, 10, 11, 9, 13
* height of root :- 0

e.g. 27

* levels in the given tree = 4
* level of a root = 1 e.g. = 61

Height: the longest path from root node to any child / leaf node.

* Ancestors of (10) :- 7, 3, 1
  " " (13) :- 12, 4, 1
* Ancestors means parent → grandparent → great grand parent (purvaj).
* Siblings of (2) :- 3, 4
* Right Sibling of (3) :- 4
* Right Sibling of (7) :- 8, 9 (only immediate right)
* Right Sibling of (6) :- not available
* Left most child (3) :- 7
* Right most child (2) :- 6
* How many child of (12) :- 1

→ Degree of a Node: The total count of subtree attached to that node is called the degree of the node. The degree of a tree node must be 0. The degree of a tree is the maximum degree of a node among all the nodes in the tree. The degree of the node {3} is 3.

Binary TREE :

1. A tree in which every node can contain at most two child node. that is called binary tree.
2. Binary tree can be represented in the two different ways
   ① Using Array
   ② Using Linked List

1. Using Array : not require for tree.
2. Using Linked List :

→ Binary tree will be represented using double linked list

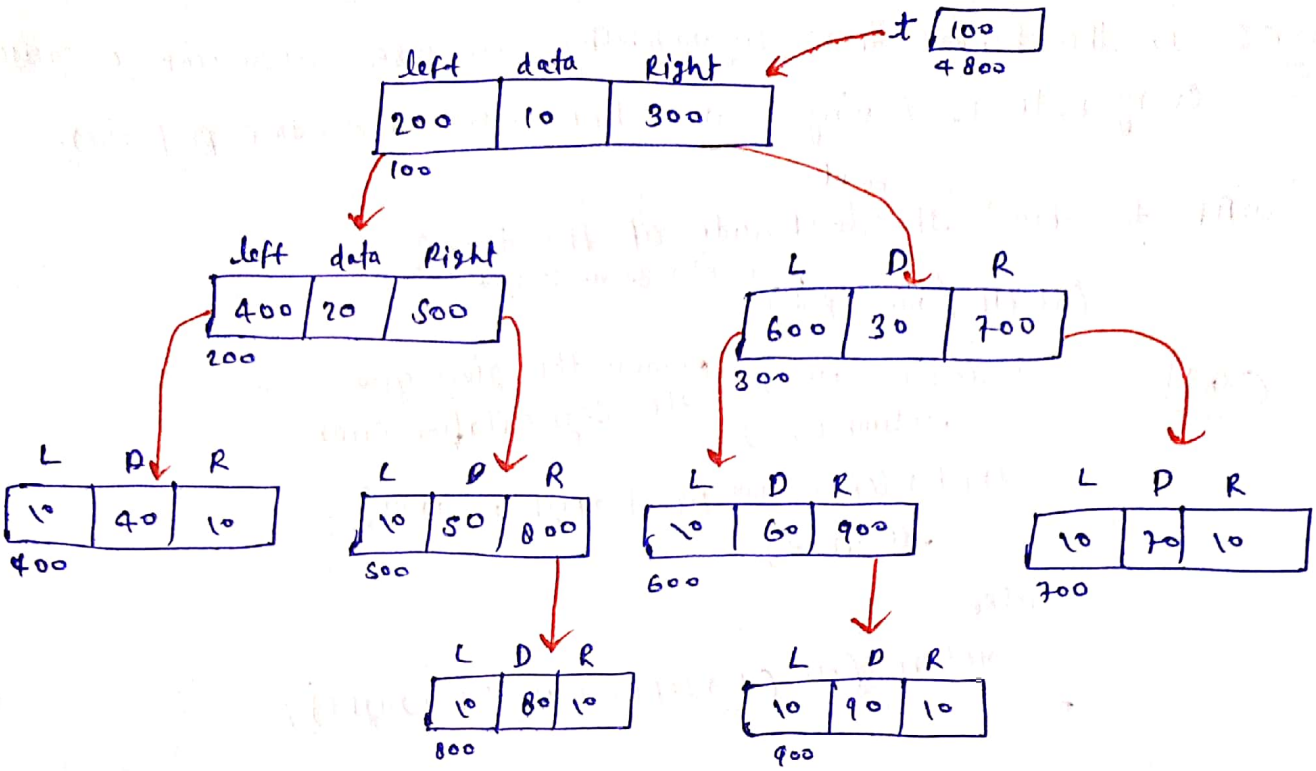
```
struct BTnode
{
    struct BTnode *left;
    int data;
    struct BTnode *right;
};
```

typedef ~~BTnode~~ struct BTnode node;

node a = {NULL, 10, NULL}
node b = {NULL, 20, NULL}
node c = {NULL, 30, NULL}
node d = {NULL, 40, NULL}

NOTE :

1. A tree with n nodes has (n-1) edges.
2. A complete binary tree with n internal nodes has (n+1) leaves.
3. The maximum number of nodes in a binary tree of height h is $(2^{h+1} - 1)$
4. The "vertices" are the "Nodes" in a tree and the "edges" are the "lines connecting them".




```
a.left = &b;
a.right = &c;
```

```
node *t = null;
t = &q;
```

1. Pf(t) ⇒ 100
2. Pf(t → data) ⇒ 10
3. If(t == NULL) ⇒ If this condition is true it means binary tree empty.
4. Pf(t → left) ⇒ 200
5. node *p = t → right ⇒
   P is a pointer variable pointing to right sub tree.
6. t → left is given, it means.. left sub tree is given.
7. If(t → left == Null) ⇒ if this is true there is no left sub tree
8. If(t → left == NULL && t → right == NULL) ⇒ that is called leaf node if this condition is true.
9. leaf nodes ⇒ 40, 80, 90, 70
10. Internal Nodes ⇒ if any node ~~are~~ doesnot contain null then this is Internal node — 10, 20, 30, 50, 60
11. Pf(t → left → right → right → data) ⇒ 80
12. Pf(t → right → lef → data) ⇒ 60

NOTE: In the binary tree programming we use recursion because every node is having same behaviours and same properties.

Quest: WAP to find the no. of leaf node of the tree?

```
int NL (node *t)        // t → pointer to root node
{ if (t==NULL)          // Remov this gives you the segmentation error
    return 0;
  if(t->left==NULL && t->Right==NULL)
    return 1;
  else
    return (NL(t->left) + NL(t->right));
}
```

T.C = O(n)

Quest: WAP to find the no. of internal nodes of the tree?

```
int NI (node *t)        // non leaf or internal node both are same.
{
  if(t==NULL)           // Remove this then program gives you segmentation error.
    return 0;
  if(t->left==NULL && t->Right==NULL)
    return 0;
  else
    return (1 + NI(t->left) + NI(t->right));
}
```

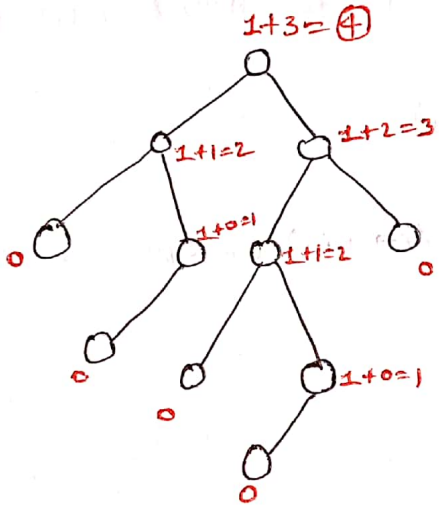
T.C = O(n)

Quest: WAP to find the total no. of nodes in the given binary tree

```
int NN (node *t)
{ if (t==NULL)
    return 0;
  if (t->left==NULL && t->right==NULL)
    return 1;
  else
    return (1 + NN(t->left) + NN(t->right));
}
```

Quest: WAP to find height of a given binary tree?

```
int height (node *t)
  { int HL, HR;
    if ( t == NULL)
      return 0;
    if (t->left == NULL && t->right == NULL)
       return 0;
    else
     { HL = height (t->left);
       HR = height (t->Right);
      return (HL>HR ? HL+1 : HR+1);
     }
}
```



Quest: WAP to find no. of levels in the given binary tree?

```
int levels (node *t)
  { int LL, LR;
    if (t == null)
      return 0;
    if (t->left == NULL && t->right==NULL)
       return 1;
    else
     { LL = levels (t->left);
       LR = levels (t->Right);
      return (LL>LR ? LL+1 : LR+1);
     }
}
```

Ques: WAP to find whether given 2 binary trees are equal or not?

Rough work:

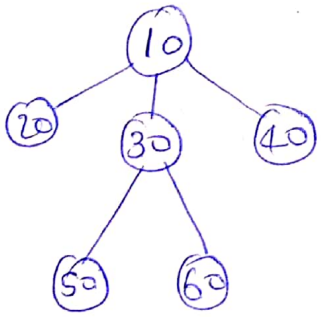
① Empty, Empty
return 1;

② empty && nonempty OR nonempty && empty
return 0;

③ check the data of both root node will be same or not

```
if 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->data == t2->data)
     {
      return (equal(t1->left, t2->left) && equal(t1->Right, t2->Right))
     }
    return 0;
  }
```

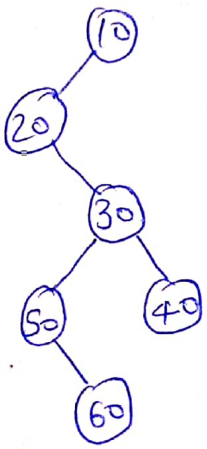
e.g.



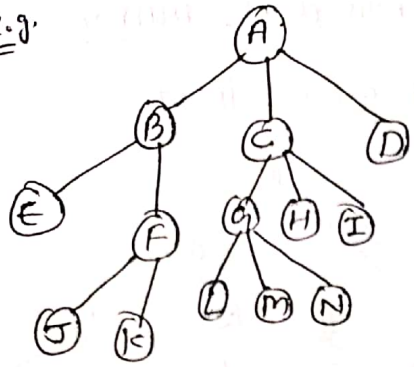
⇒ Binary tree

⇓

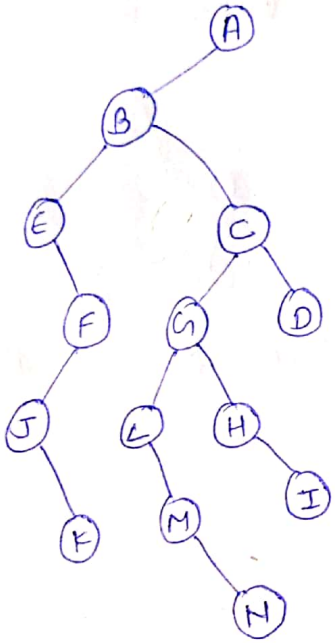
(leftmost child, Right Sibling)



e.g.



→ Binary tree (left most child, Right Sibling.)



Qust: WAP to find the given binary tree is strict binary tree?

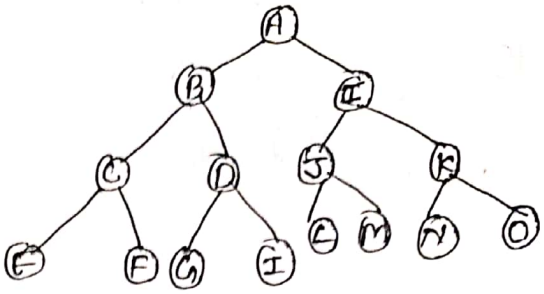
Strict Binary Tree: A Binary tree every node has exactly zero or two child is called as a strict binary tree.

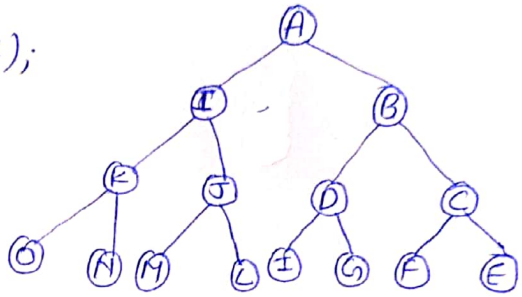
```
int sbt(node *b)
{ if(t==NULL)
    return 1;
  if(t->left==NULL && t->Right==NULL)
    return 1;
  if(t->left!=NULL && t->Right!=NULL)
  return (sbt(t->left) && sbt(t->Right));
  else
  return 0;
}
```

Ques: WAP to convert the given binary tree into as is mirror image?

```
node * Mirror-Image (node *t)
  { if (t == null)
      return t;
    if (t->left == NULL && t->Right == NULL)
      return t;
    else
    { node * temp = t -> right;
      t -> right = Mirror-Image (t -> left);
      t -> left = Mirror-Image (temp);
      return (temp);
    }
```

mirror image

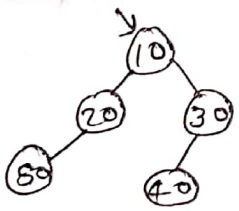




TREE TRAVERSAL'S :

1. Pre Order :- Root, left, Right
2. InOrder :- left, Root, Right
3. Post Order :- left, Right, Root

e.g.



Pre Order :- 10, 20, 50, 30, 40
InOrder :- 50, 20, 10, 40, 30
Postorder :- 50, 20, 40, 30, 10

```
PreOrder (node *t)
  { if (t)
    {
      Pf (t -> data);
      Preorder (t -> left);
      preOrder (t -> Right);
    }
}
```

```
InOrder (node *t)
  { if (t)
    {
      Inorder (t -> lef);
      Pf (t -> data);
      Inorder (t -> right);
    }
}
```

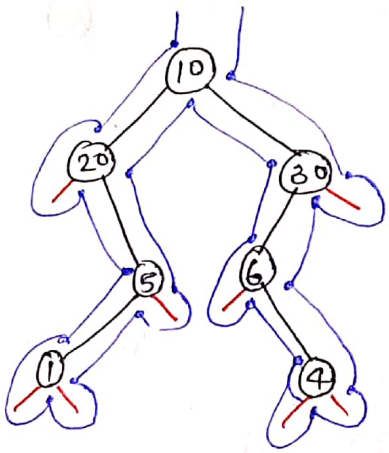
```
Post Order (node *t
  { if (t)
```

Preorder: 1st Time visiting

Inorder: 2nd Time visiting

Postorder: 3rd Time visiting.

e.g.

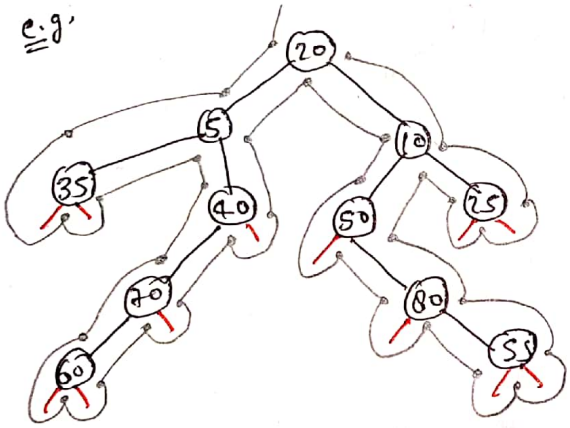


Pre : 10, 20, 5, 1, 30, 6, 4

In : 20, 1, 5, 10, 6, 4, 30

Post : 1, 5, 20, 4, 6, 30, 10

e.g.



Pre : 20, 5, 35, 40, 70, 60, 10, 50, 80, 55, 25

In : 35, 5, 60, 70, 40, 20, 50, 80, 55, 10, 25,

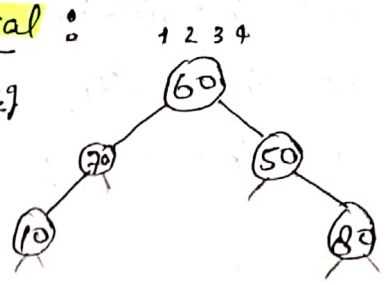
Post : 35, 60, 70, 40, 5, 55, 80, 50, 25, 10, 20

NOTE: To apply the above technique every node should have 2 child nodes. If it is not have 2 child nodes put some dummy nods.

Double Order and Triple Order Traversal :

```
Do (node *t)
 { if (t)
    {
   1 Pf(t→data);
   2 Do( t→left);
   3 Pf(t→ data);
   4 Do(t→right);
    }
 }
```

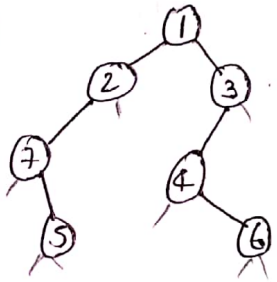
1.eg



O/P → 60, 70, 10, 10, 70, 60, 50, 80,

```
TO (node *t)
 { if(t)
    {
   1 Pf(t→ data);
   2 TO(t→ left);
   3 Pf(t→ data);
   4 To(t→ Right);
   5 Pf(t→ data);
    }
}
```

e.g

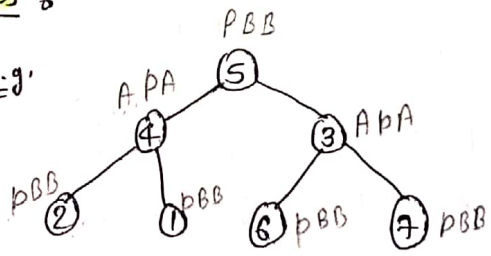


O/P → 1 2, 7 7, 5, 5, 5, 7, 2, 2, 1, 3, 4, 4, 6, 6, 6, 4, 3,

Indirect 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);
    }
}
```

e.g.



what is the A(s) = ?

5, 2, 4, 1, 6, 3, 7

Constructing unique Binary Tree :

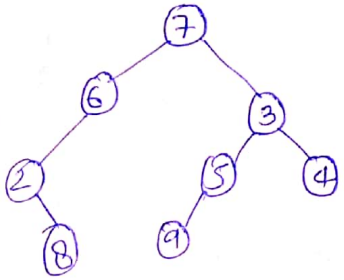
A unique binary tree can be constructed if any two traversals are given and one of them should be in-order.

* Preorder, Inorder ✓
* Postorder, Inorder ✓

i.e Postorder : 8, 2, 6, 9, 5, 4, 3, 7

Inorder : 2, 8, 6, 7, 9, 5, 3, 4

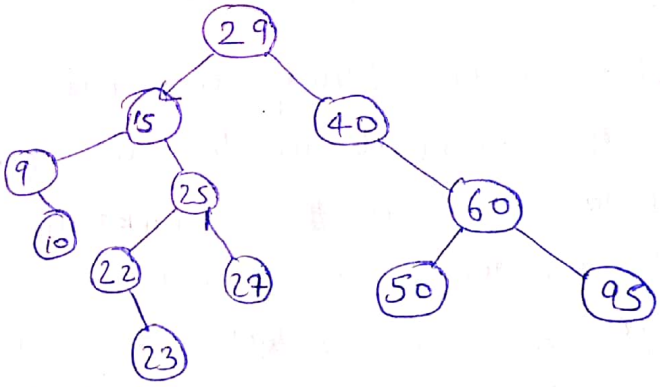
L R



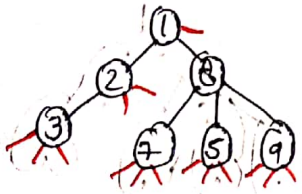
e.g Postorder : 10, 9, 23, 22, 27, 25, 15, 50, 95, 60, 40, 29

Inorder : 9, 10, 15, 22, 23, 25, 27, 29, 40, 50, 60, 95

L R



Quest: Find inorder traversal of given tree:



Inorder : 3, 2, 1, 7, 8, 5, 9

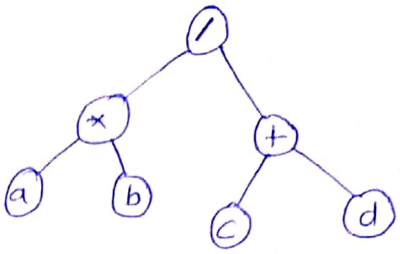
Here we print only inorder means 2nd traversal.

Quest: Construct the unique binary tree for given reverse Polish Notation
↳ Postfix

Postfix : a b * c d + /

Infix : (a * b) / (c + d)

(a * b) / (c + d)
L R



Quest: Construct the binary tree for the given reverse polish Notation.

Postfix : 3 4 * 5 - 2 ↑ 6 7 * 1 + -

Infix :

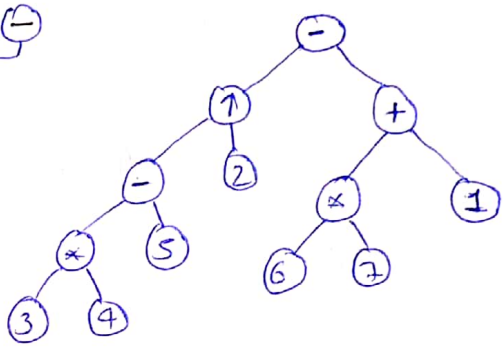
(3 * 4) - 5 - 2 ↑ 6 7 * 1 + -

(3 * 4) 5 - 2 ↑ (6 * 7) 1 + -

(3 * 4) - 5) 2 ↑ (6 * 7) 1 + -

(3 * 4 - 5) ↑ 2) ((6 * 7) + 1) -

((3 * 4 - 5) ↑ 2) - ((6 * 7) + 1)
L R

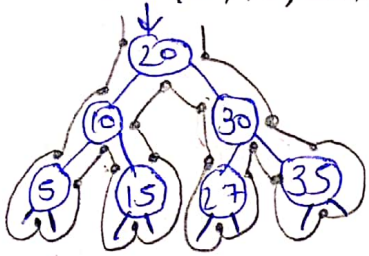


Binary Search Tree : (B.S.T.)

1. BST is also a like binary tree but with the restriction that all the elements in the left sub tree should be smaller to the parant and all the element in the right sub tree should be greater to the parent.
2. This rule shoud be followed for every node.
3. All the data elements must be unique.
4. BST is used for searching an element.

Construct the BST with given data elements:

20, 10, 15, 30, 35, 27, 5



Pre order : 20, 10, 5, 15, 30, 27, 35

In order : 5, 10, 15, 20, 27, 30, 35

Post order : 5, 15, 10, 27, 35, 30, 20

NOTE:

1. Inorder traversal of BST is always sorted order.
2. Last element in the preorder and last element in the inorder both are same. (This is true only for complete BT)

Ques: Consider the numbers (1 to 100) in B.S.T. when Searching for an element 43, which of the following sequences cannot be the sequence of nodes examined?

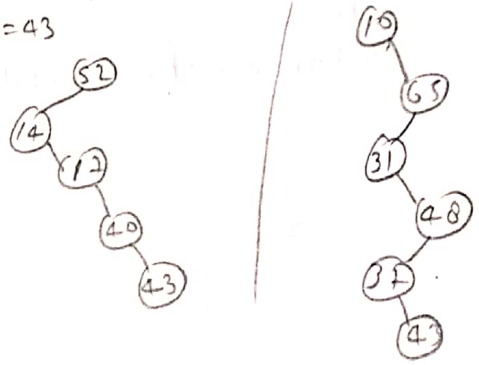
(a) 52, 14, 17, 40, 43

(b) 10, 65, 31, 48, 37, 43

(c) 3, 50, 40, 60, 43

(d) 81, 61, 52, 14, 41, 43

Key = 43



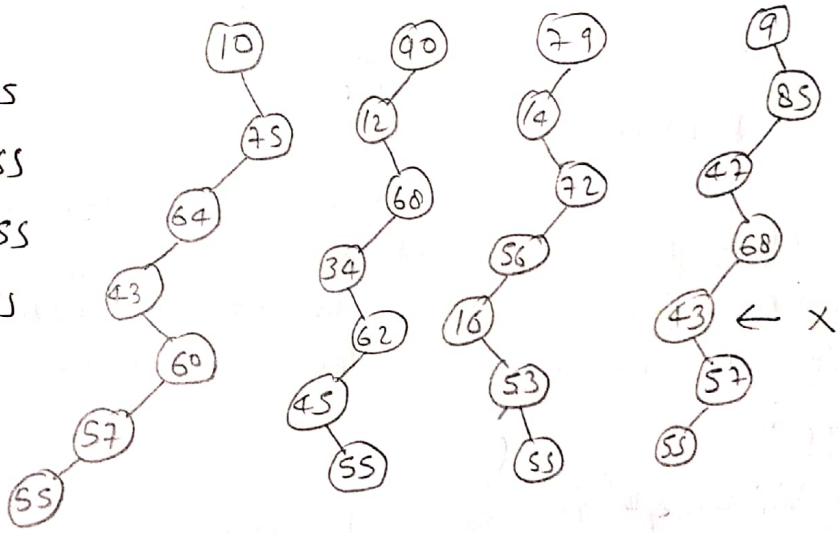
Ques: Key = 55

(a) 10, 75, 64, 43, 60, 57, 55

(b) 90, 12, 68, 34, 62, 45, 55

(c) 79, 14, 72, 56, 16, 53, 55

(d) 9, 85, 47, 68, 43, 57, 55



Quest: WAP to Search a element in the BST ?

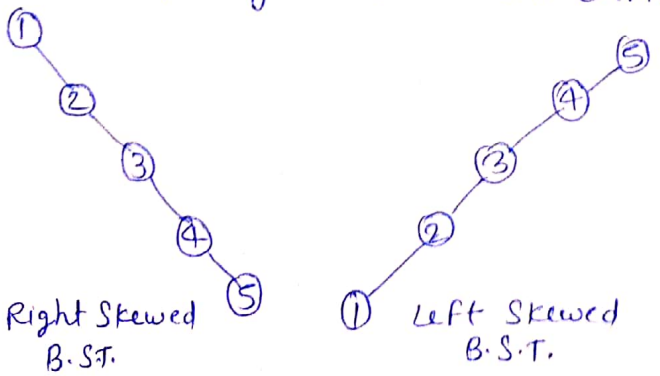
```
node * Search(node *t, int n)        → element to search
{
  if (t == NULL)
    return t;
  if( n == t->data)
    return t;
  if(n < t->data)
  return Search(t->left, n);
  else
  return Search(t->right, n);
}
```

Rough work :

1. empty ⇒ NULL
2. Compare the root node data with element n.
3. if it is Smaller element then Search in LST otherwise Search in RST

NOTE :

1. Time Complexity to search an element in B.S.T = $\boxed{O(n)}$



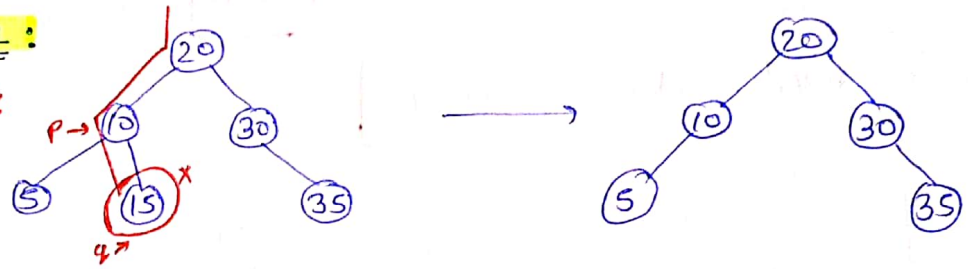
2. Time Complexity to Insert (n) elements in B.S.T = $\boxed{O(n^2)}$

3. To improve the worst case Search time complexity from order of O(n) to ordrer log(n) [O(logn)] AVL tree will be used.

Deletions in BST :

CASE 1 :

Leaf Node :



delete 15

* If it is the leaf node then simply delete that and update in its Parent pointer as a Null.

CASE 2 :

Internal Node

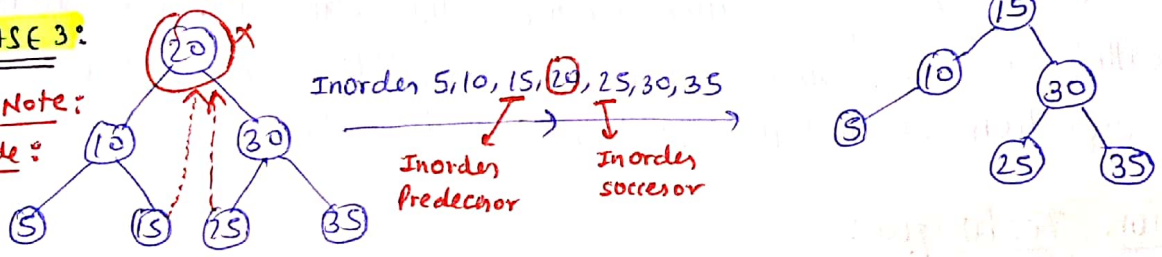
delete 30

* After deleting the node 30 the child of 30 should be connected to Parent of 30

*** CASE 3 :

Root Note :

2 Node :

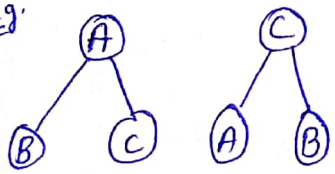


delete 20

* New root will be updated with the inorder predecesor (largest element in left subtree) or inorder successor (smallest element in right subtree)

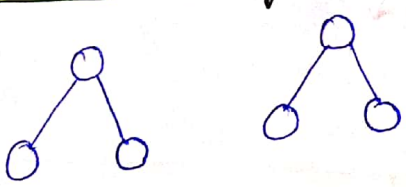
Labelled Binary Tree :

e.g.



Unlabelled Binary Tree :

e.g

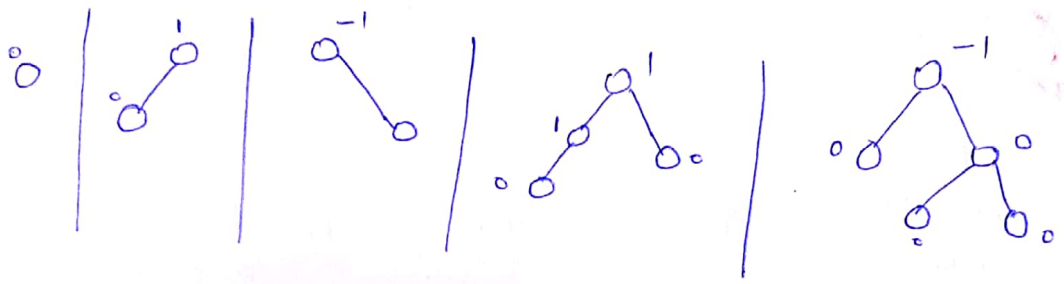


AVL Tree (Height Balanced Tree B.S.T):

(Adelson, Valsky, Landis)

Balanc factor = height of L.S.T. – height of R.S.T.

Accepted Balanced factor = 0, +1, -1

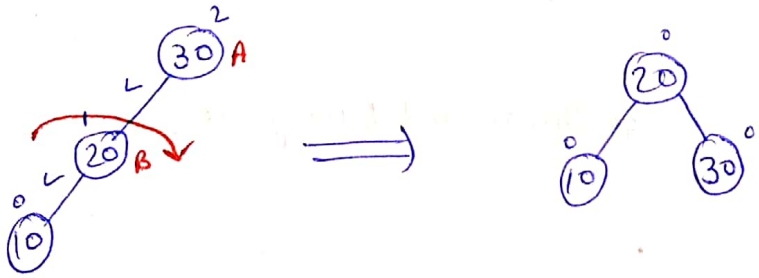


Construction of AVL Tree:

1. The processor is same as creating the BST by inserting one element at a time.
2. when the new node is inserted the balance factor of all its ancestor will be updated.
3. If node is having balance factor other than accepted than it is called as imbalance. Then we need to apply the proper rotation technique and make it balanced.

Rotation Technique:

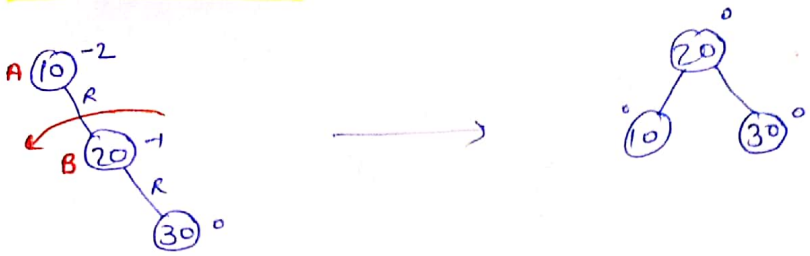
1. L.L. Imbalance :-



make node 'A' as child of node 'B'

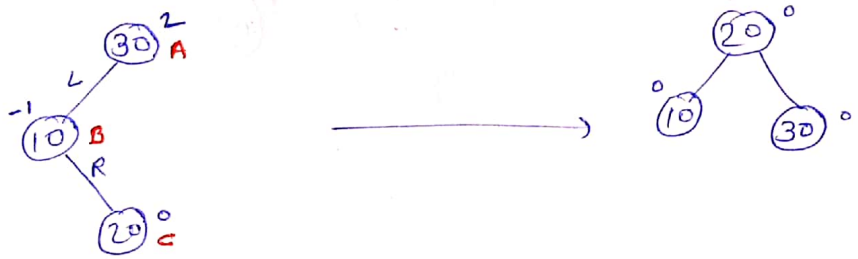
Internally it involves single Rotation.

2. R.R. Imbalance :

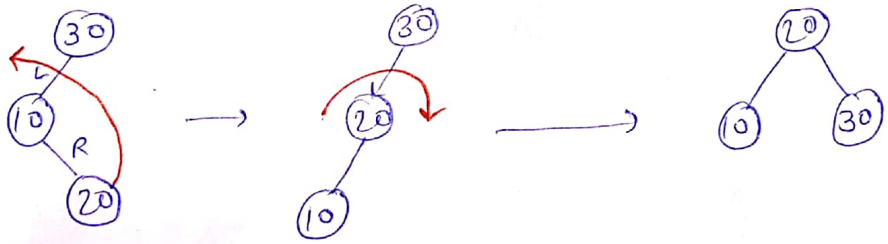


* make node 'A' as child of node 'B'
* Internally involves single rotation

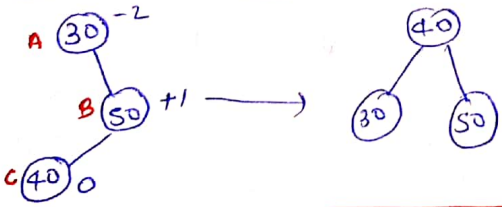
3. LR Imbalance :



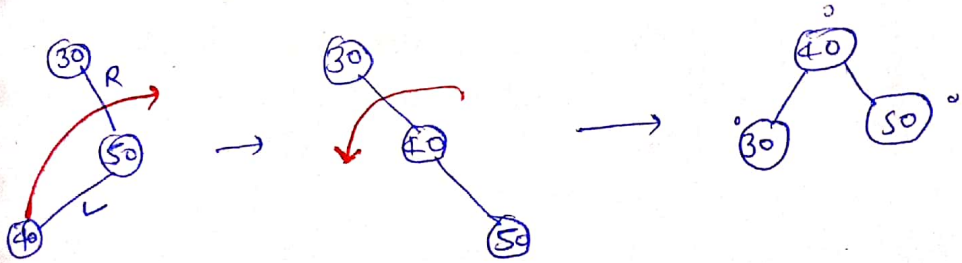
* make node 'A' and node 'B' as child of node 'C'
* Internally it involves double rotations



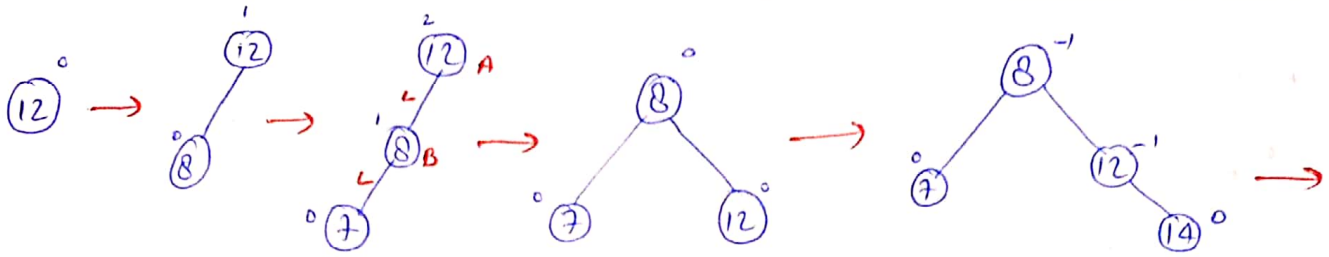
4. RL Imbalance :

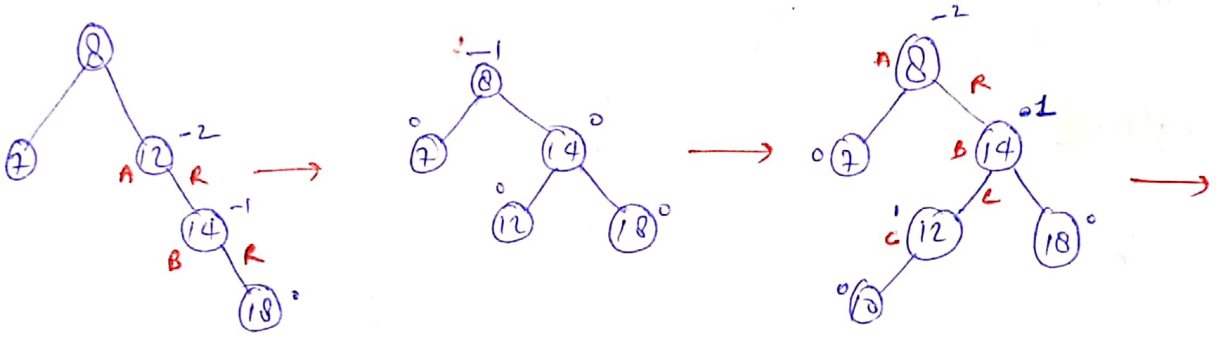


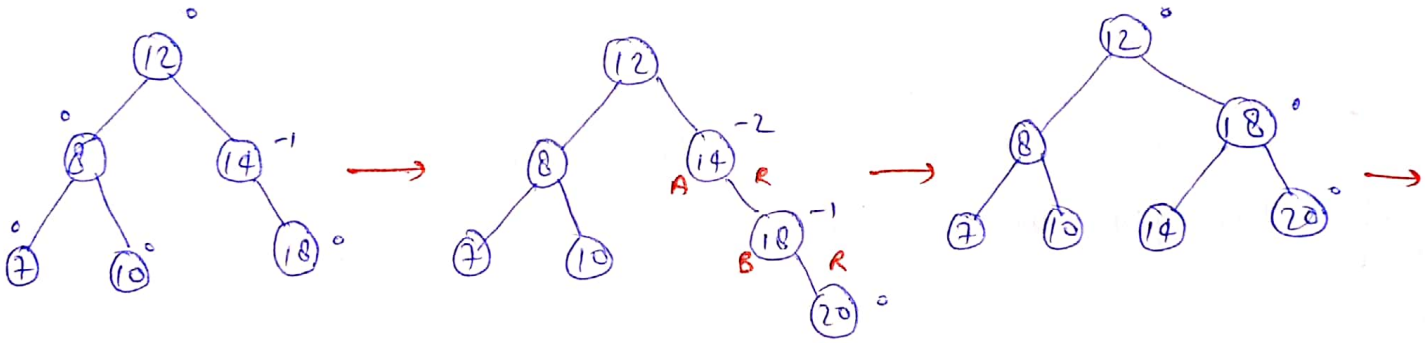
* make node 'A' and node 'B' as child of node 'C'.
* It internally involves double rotations.

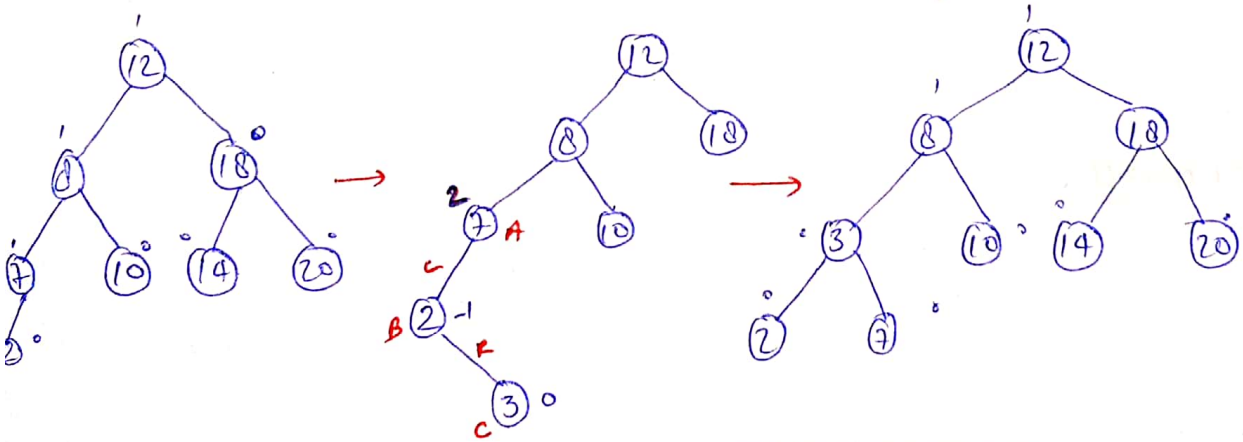


Ques: Construct the AVL Tree with given nodes 12, 8, 7, 14, 18, 10, 20, 3





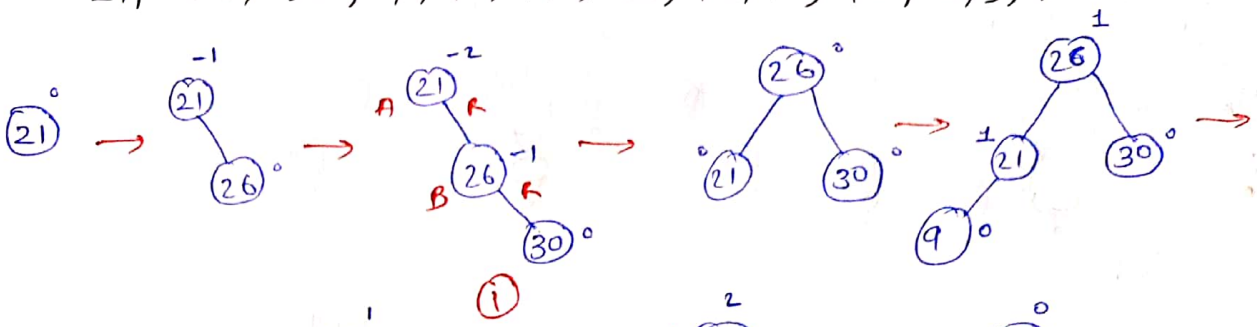




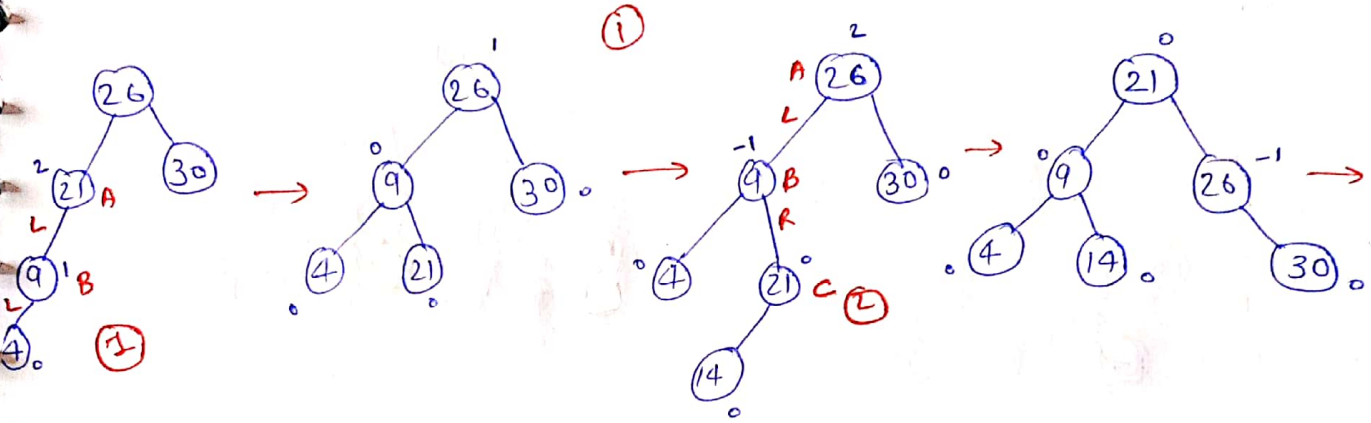
No. of rotation = 7

Ques: Construct AVL TREE

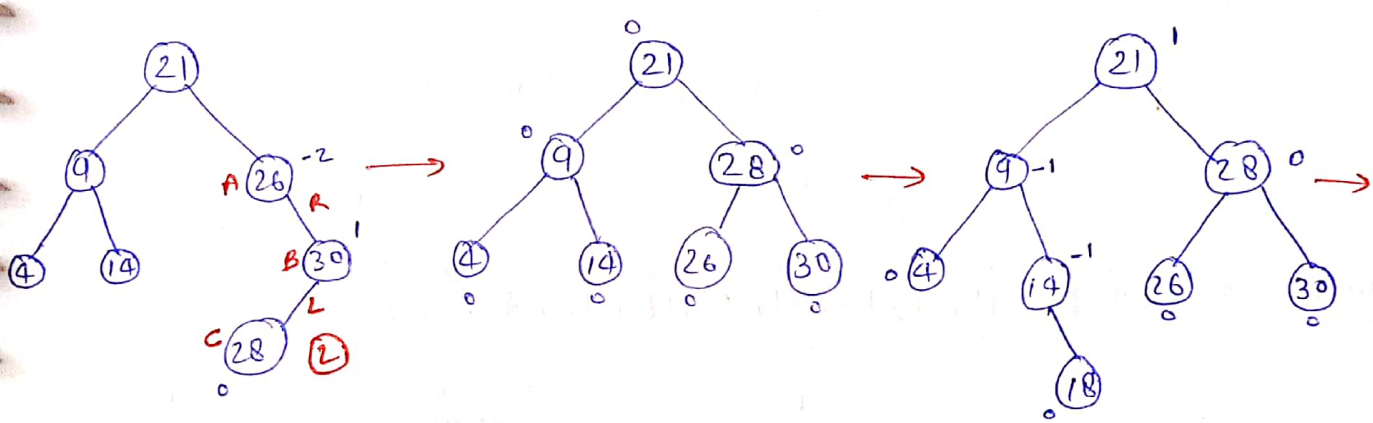
21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7



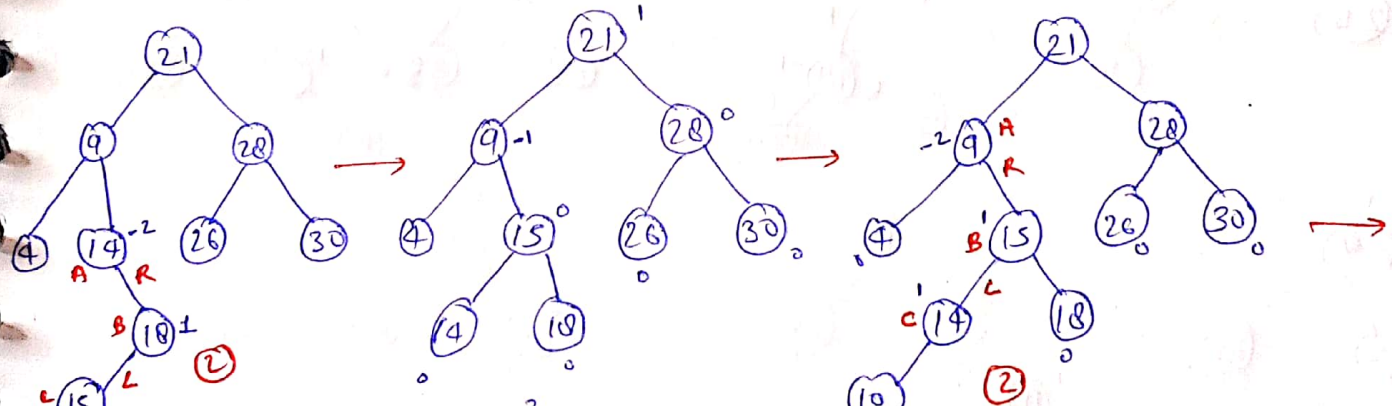
(1)



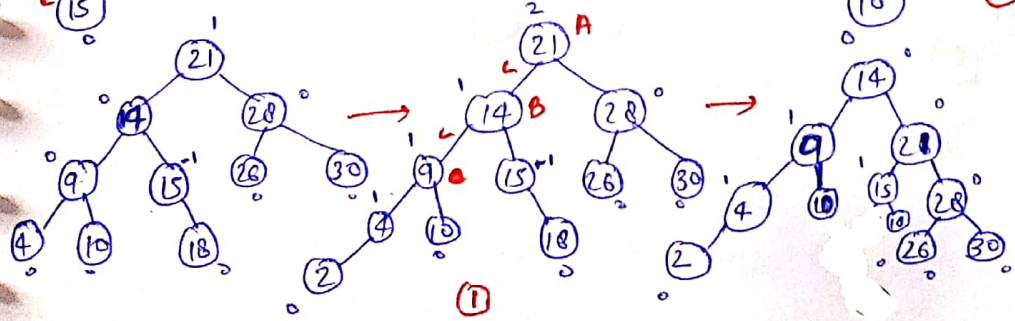
(1) (2)



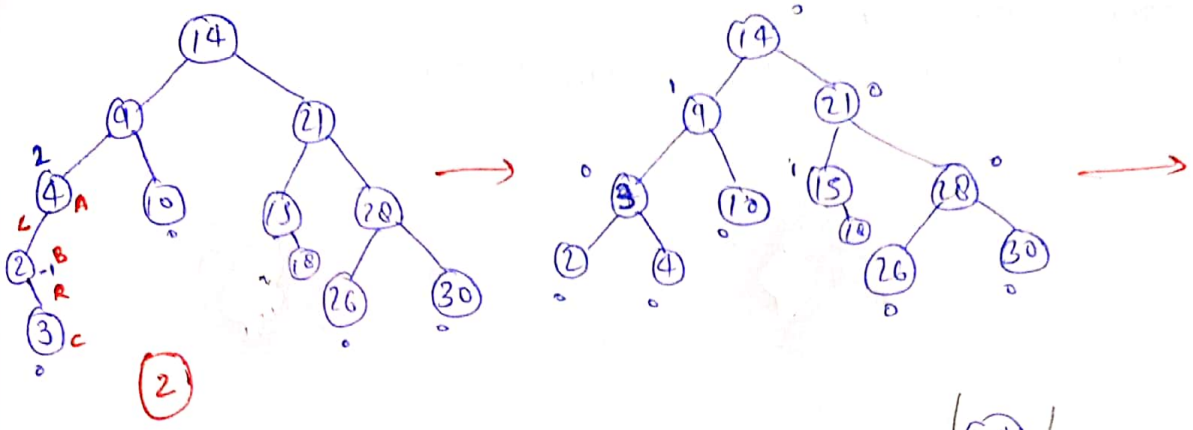
(2)



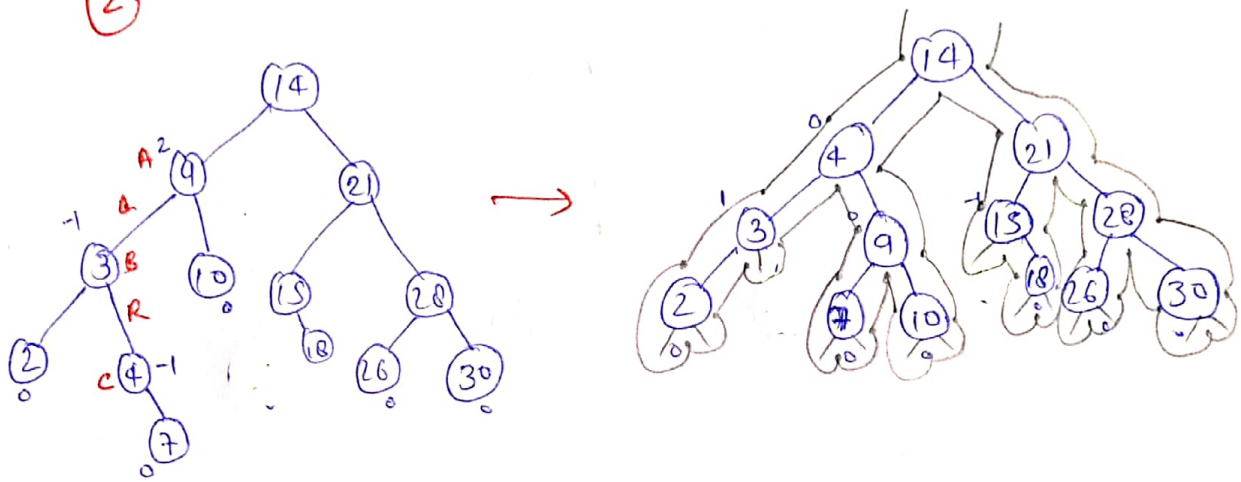
(2)



(1)



(2)

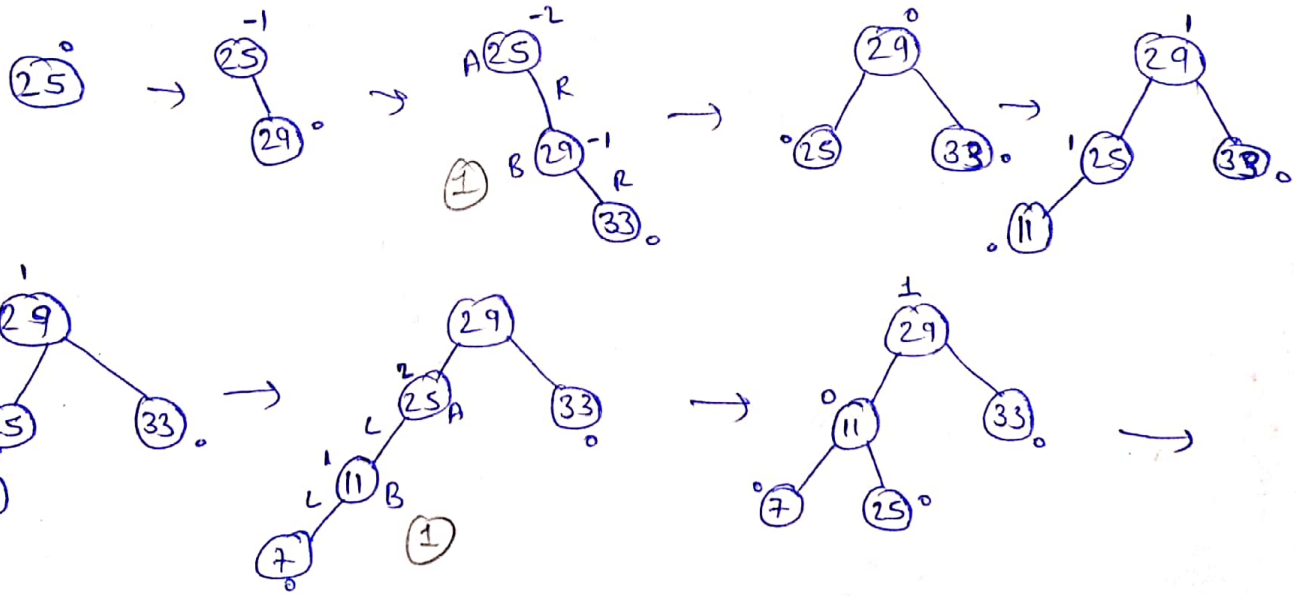


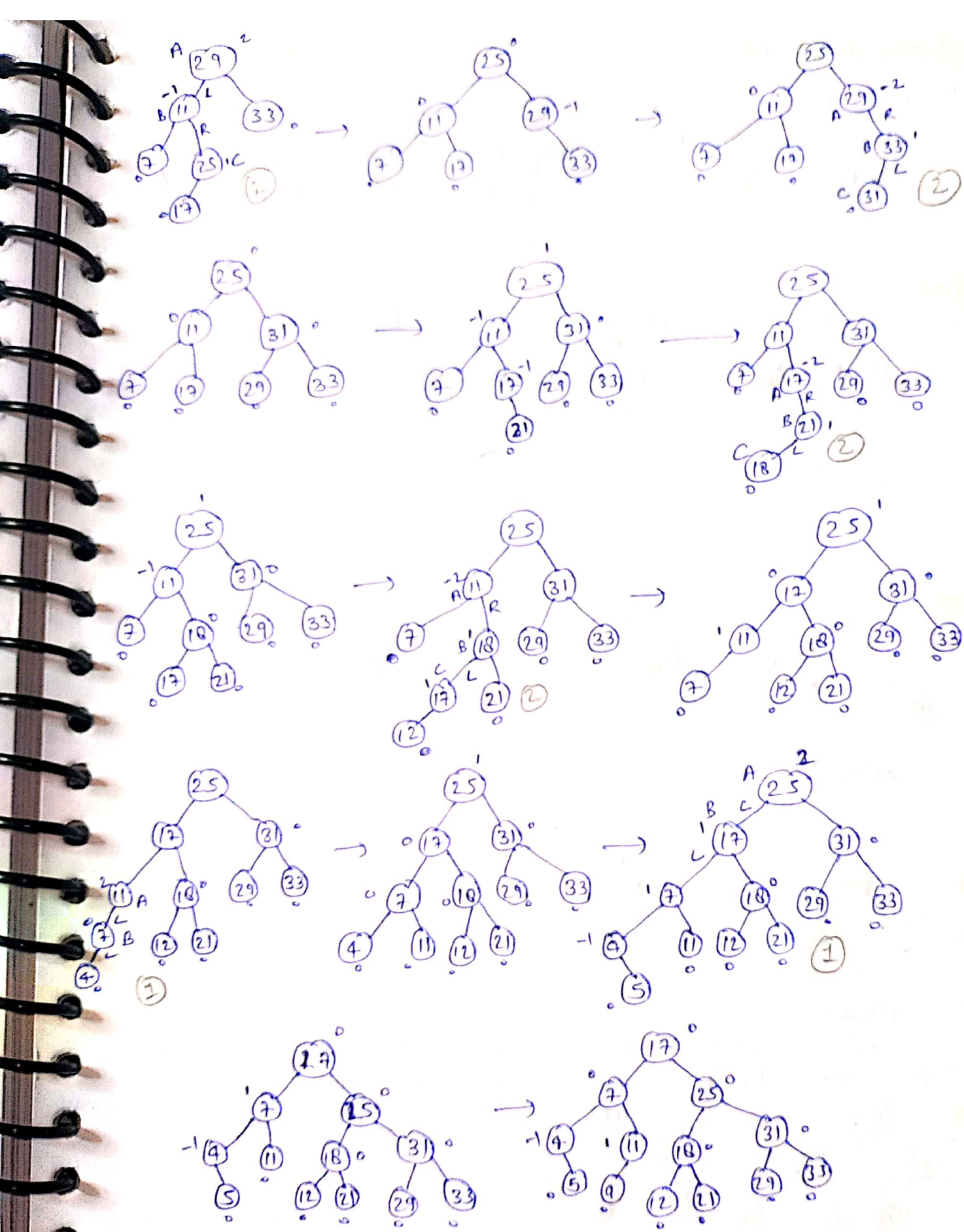
(2)

No. of rotation = 15

Preorder : 14, 4, 3, 2, 9, 7, 10, 21, 15, 18, 28, 28, 30

Ques: 25, 29, 33, 11, 7, 17, 31, 21, 18, 12, 4, 5, 9

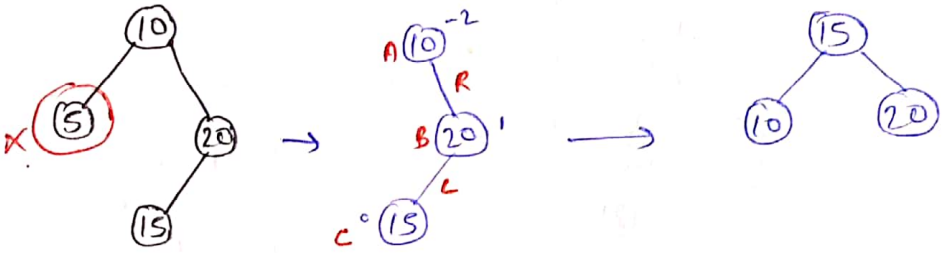




Deletion in AVL TREE:

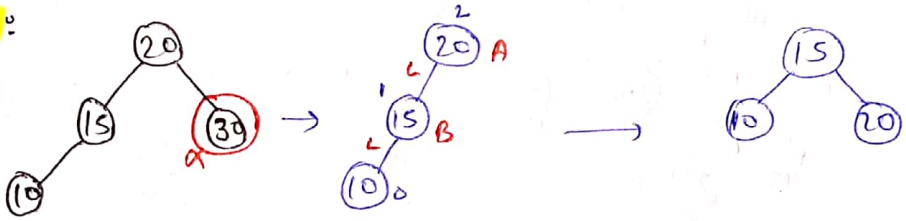
* Deletion in AVL tree is same as BST but after deletion check the balance factor of every node and if there is any imbalance then apply the proper rotation technique and make it balanced.

CASE 1:
leaf Node



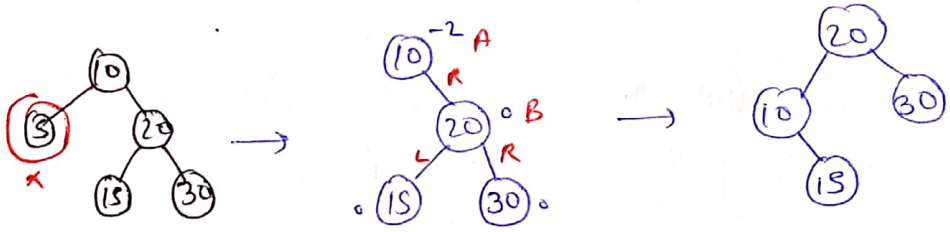
delete 5

CASE 2:
leaf Node



delete 30

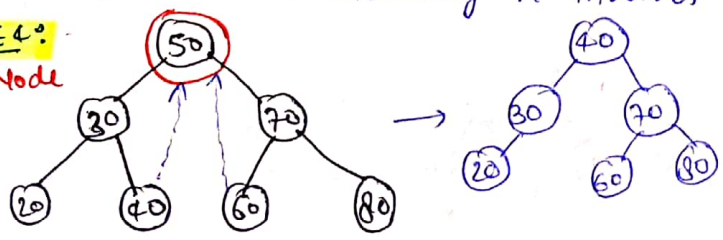
CASE 3:
leaf node



delete 5

* In the above scenario it is advisable to follow RR imbalance rotation technique because internally it involves only one rotation.

CASE 4:
root Node



delete 50

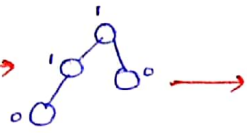
* In the above scenario New root will be updated with the inorder predesser or inorder successor.

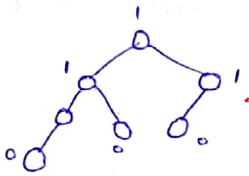
Finding Minimum no. of nodes in AVL Tree with height 'h' :

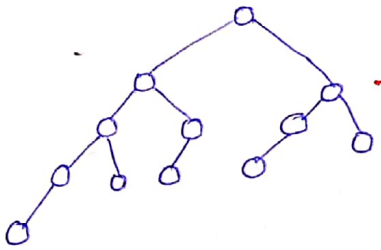
$h = 0 \longrightarrow$  $\longrightarrow$ nodes = 1

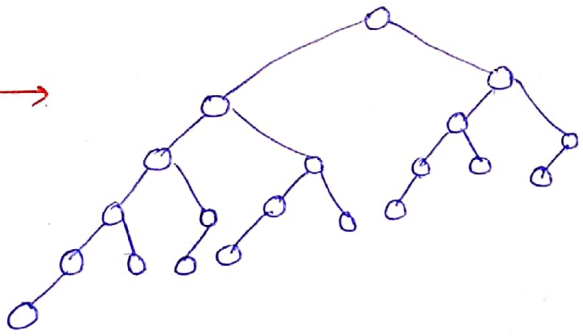
Max. No. of Nodes = $2^{h+1} - 1$ with height 'h'

$h = 1 \longrightarrow$  $\longrightarrow$ nodes = 2

$h = 2 \longrightarrow$  $\longrightarrow$ nodes = 4

$h = 3 \longrightarrow$  $\longrightarrow$ nodes = 7

$h = 4 \longrightarrow$  $\longrightarrow$ nodes = 12

$h = 5 \longrightarrow$  $\longrightarrow$ nodes = 20

height (h)	Min. of nodes in AVL Tree
0	1
1	2
2	4
3	7
4	12
5	20
6	33
7	54
8	88
9	143
10	232
11	376

Prest. + Prev. +1

min. no. of nodes in AVL Tree of height (h) = min. no. of nodes with height (h-1) + min. no. of nodes with height (h-2) + 1

Note:

* Worst Cas Search time complexity is O(logn) because of AVL is height balance tree.

Hashing:

1. Hashing is a technique used for searching an element.
2. It is a mathmatical computation over a key.
3. The worst case search time complexity in the hashing is Constant $O(1)$.

Goal of the Hashing:

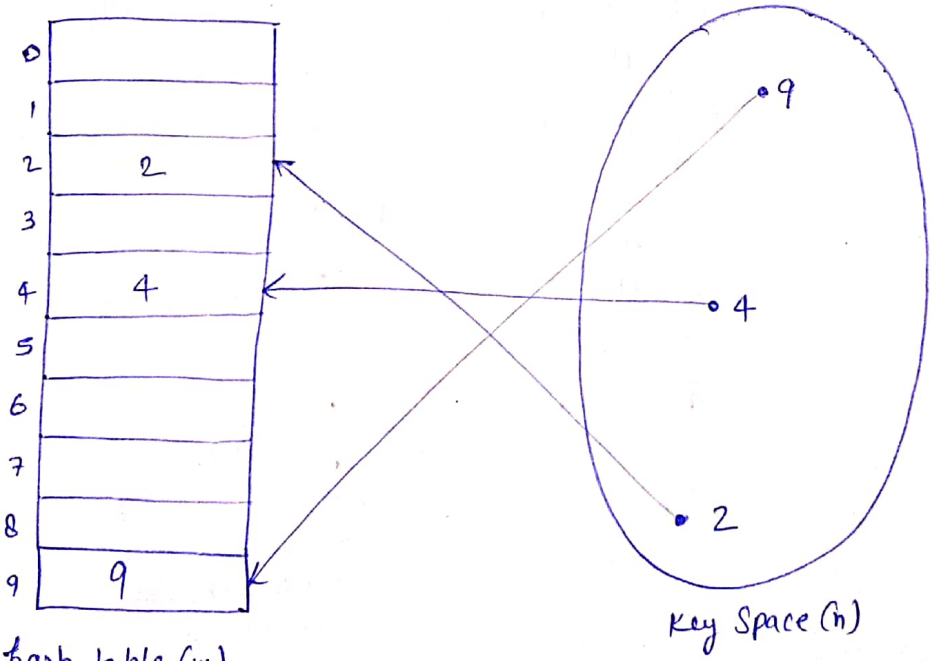
1. To design a hash function which gives less no. of collisions.
2. The following componants are used in the hashing:
   (1) Key Space (n)
   (2) Hash table (or) hash bucket (m)
   (3) Hash Function

Direct Address Table (DAT):

hash Table size (m) = 10

```
int ht[10];
```

* In the DAT keys will be directly store in the hash table in the respective indexes.



NOTE:

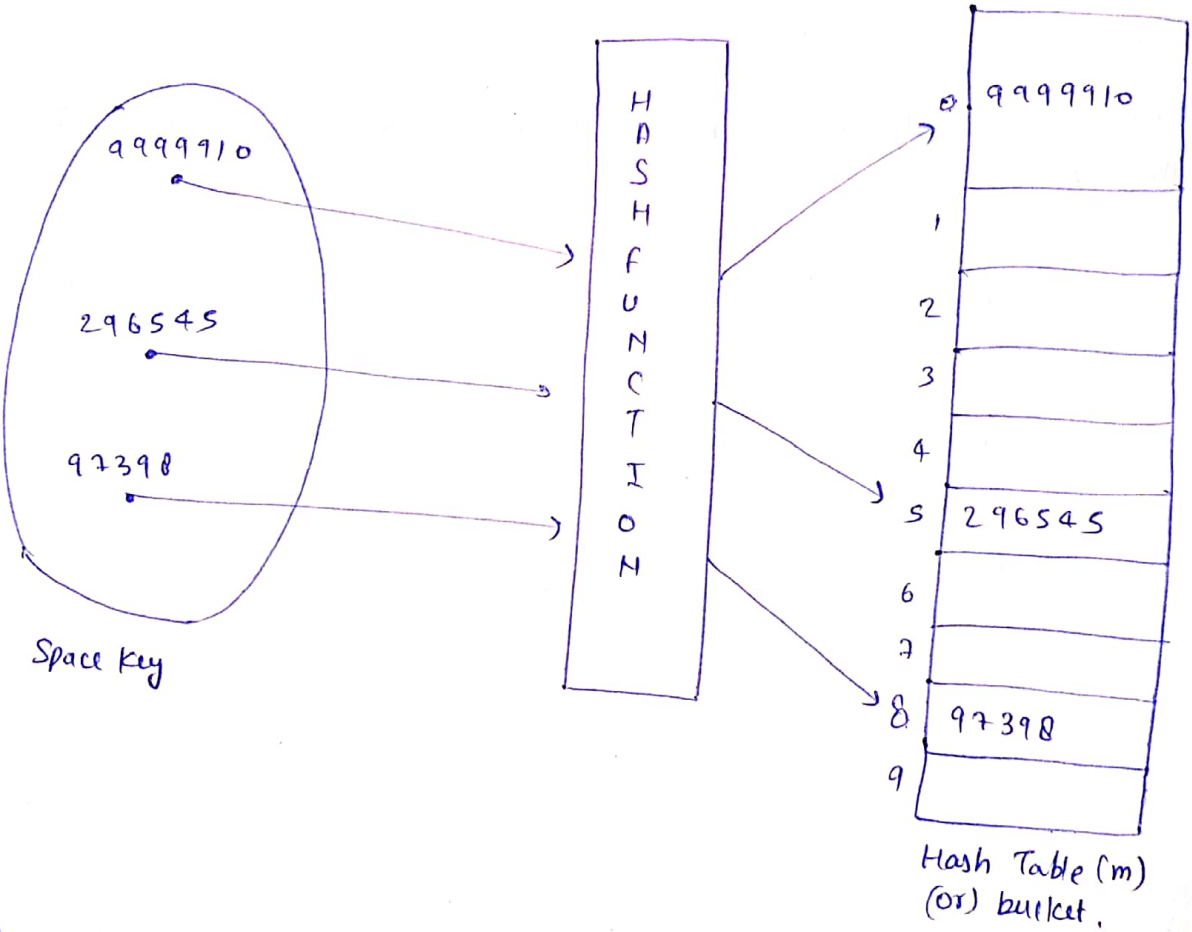
* DAT major advantage is Search T.C. will be O(1) every case.
* If one key having large size like key = 2^{1000}, 999910 then we need to maintain very large hash table.
* To avoid this problem the concept of hash function is introduce.

Hash function:

Hash Table size (m) = 10 (0---9)

1. Division Modulo Method:

H.F (key) = key mod m;



1. By using the hash function we can comfortably store large keys in the hash table.
2. If the two keys are mapped on to same location at hash table then it is called as a collision.
   i.e key = 888

~~1. Div~~

NOTE:

1. Do not choose 'm' value as exactly power of 2, because if $m = 2^k$, then

H.f(key) = LSB 'k' bits always.

if $m = 8$, $m = 2^3$, $k = 3$

	m ↓	m ↓
(1) key = 86 = 1010110 =	86 mod 8 = 6	86 mod 7 = 2
(2) key = 102 = 1100110 =	102 mod 8 = 6	102 mod 7 = 4
(3) key = 118 = 1110110 =	118 mod 8 = 6	118 mod 7 = 6
(4) key = 126 = 1111110 =	126 mod 8 = 6	126 mod 7 = 0

2. Choose the m value as prime no. which is not nearer to powers of 2, so that it gives less no. of collisions.

2. Mid Square Method :

Hash table size (m) = 1000 (0 -- 999)

key = 9867

$(key)^2 = (9867)^2 = 97\underset{2\ 3\ 4}{(357)}689$

Index	key
357	9867

Hash Table

0 ... 99 / 100 ... 999

* we have take 3 digit just because hash table size is (0 -- 999).
* The counter example is difficult.

3. Folding Method :

(a) Fold Boundary Method :

Hash Table size (m) = 1000 (0 --- 999)

Key = 134 671 235

```
  134
+ 235
-----
  369
```

index	Hash Table
369	134 671 235

* if o/p is (1)369 then do 369 + 1 = 370

* Counter example is very easy.

(b) Fold shifting Method :

Hash Table size (m) = 1000 (0 --- 999)

Key = 134 671 235

```
  134
  671
+ 235
------
 1040
 +  1
------
  040
   1
------
  041
```

index	Hash Table
041	134 671 235

* Counter example is easy.

(c) Digit Extraction Method :

H.T. (m) = 1000 (0 --- 999)

Key = 13(4) 6(7)1 (2)35

0 1 2 3 4 5 6 7 8

Index	Hash Table
472	134 671 235

* Counter example is easy.

Collision Resolution Techniques (C.R.T.):

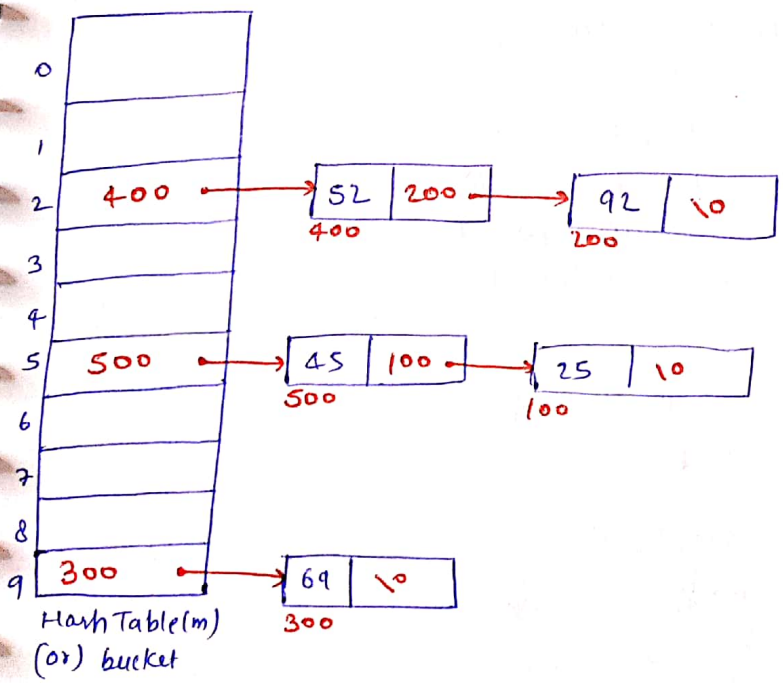
- Chaining (outside)
- Open addressing (inside)
  - Linear Probing
  - Quadratic Probing
  - Double Hashing

1. Chaining:

1. Chaining will be implementing with the help of linked list
2. Keys will be stored outside the hash table in the form of linked list node. (@) `Struct node *HT[10];`
3. Hash Table size(m) = 10 (0---9)

H.f(key) = key mod m

Keys = 25, 92, 69, 52, 45, 59, 23, 49, 55



Time Complexity

Searching: B.C = O(1)
W.C = O(n)

insertion: B.C = O(1)
W.C = O(1)

deletion: B.C = O(1)
W.C = O(n)

* The main advantage of chaining is we can handle any no. of collisions until the memory is full.
* The main disadvantage is even though the space is available inside the hash table we are using outside space.
* The longest chain possible with the 'n' case is n
* The major advantage of chaining compare to other C.R.T. is deletion.

because it is easy.

2. Open Addressing:

Open addressing is also known as closed hashing

(a) Linear Probing:

$$L.P(key, i) = (H.F(key) + i) \bmod n$$

$i = 0, 1, 2, \ldots\ldots (m-1)$

$$H.F(key) = key \bmod n$$

e.g. Keys(n) = 25, 38, 43, 68, 79, 46, 58, 65, 20

0	79
1	58
2	20
3	43
4	
5	25
6	46
7	65
8	38
9	68

Hash table (m)

① $L.P(25, 0) = (5+0)\%10 = 5$

② $L.P(38, 0) = (8+0)\%10 = 8$

③ $L.P(43, 0) = (3+0)\%10 = 3$

④ $LP(68, 0) = (8+0)\%10 = 8$

$LP(68, 1) = (8+1)\%10 = 9$

⑤ $LP(79, 0) = (9+0)\%10 = 9$

$LP(79, 1) = (9+1)\%10 = 10\%10 = 0$

⑥ $LP(46, 0) = (6+0)\%10 = 6$

⑦ $LP(58, 0) = (8+0)\%10 = 8$

$LP(58, 1) = (8+1)\%10 = 9$

$LP(58, 2) = (8+2)\%10 = 10\%10 = 0$

$LP(58, 3) = (8+3)\%10 = 11\%10 = 1$ } ③

⑧ $LP(65, 0) = (5+0)\%10 = 5$

$LP(65, 1) = (5+1)\%10 = 6$

$LP(65, 2) = (5+2)\%10 = 7$ } ②

⑨ $LP(20, 0) = (0+0)\%10 = 0$

$LP(20, 1) = (0+1)\%10 = 1$

$LP(20, 2) = (0+2)\%10 = 2$ } ②

Total collision = 9

#Load factor (α):

* The no. of keys stored in one slot is called as the load factor (α).

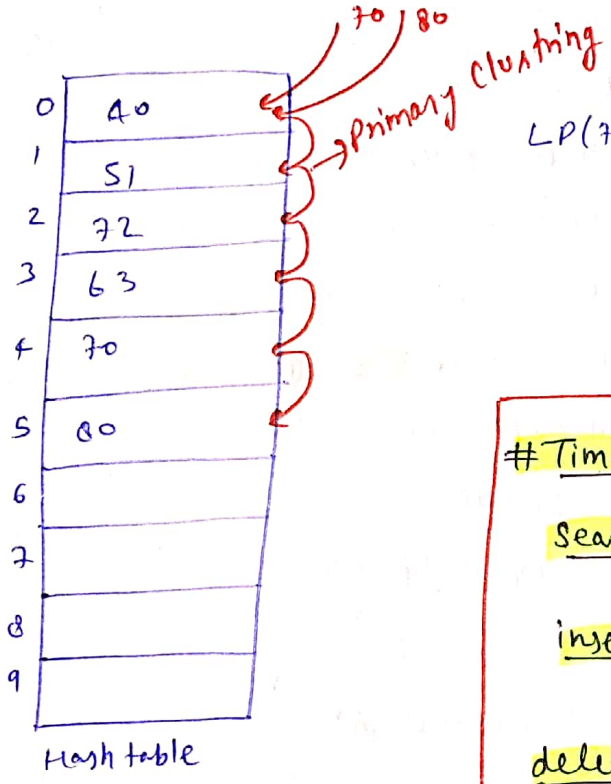
in 'm' slots ⟶ we are storing 'n' keys

in '1' slots ⟶ ?

$\frac{1}{m} * n = \frac{n}{m}$

$\alpha = \frac{n}{m}$ $\quad 0 \le \alpha \le 1$

Ques: Keys: 63, 51, 40, 72, 70, 80, [illegible].



LP(70,0) = 0+0 = 0
= 0+1 = 1
= 0+2 = 2
= 0+3 = 3
= 0+4 = 4

LP(80,0) = 0+0 = 0
= 0+1 = 1
= 0+2 = 2
= 0+3 = 3
= 0+4 = 4
= 0+5 = 5

#Time Complexity:

Searching: B.C. = O(1), W.C = O(m)

insertion: B.C = O(1), W.C = O(m)

deletion: B.C. = O(1), W.C = O(m)

#Primary Clustring:

If the two keys are mapped on to same starting location in the hash table then they both follow the same path in the linear manner. Because of this search time complexity will increase.

2. To avoid this problem we use quadratic Probing.

NoTe:

1. Deletion will create a problem to other keys but we can manage by storing the special symbol like - $, #
2. If more no. of deletion occur then perform rehashing.

Quadratic Probing:

$$Q.P(key, i) = (H.F(key) + C_1 * i + C_2 * i^2) \bmod m$$

$C_1 = 1$, $C_2 = 1$, $i = 0, 1, 2, \ldots\ldots q(m-1)$

$H.F(key) = key \bmod m$

$m = 10$ (0 — 9)

e.g. keys: 25, 98, 57, 75, 97, 18, 78, 46

0	18
1	75
2	
3	
4	78
5	25
6	46
7	57
8	98
9	97

Hash Table

① $QP(25,0) = 5 + 0 + 0 = 5 \% 10 = 5$

② $QP(98,0) = 8 + 0 + 0 = 8 \% 10 = 8$

③ $QP(57,0) = 7 + 0 + 0 = 7 \% 10 = 7$

④ $QP(75,0) = 5 + 0 + 0 = 5 \% 10 = 5$

$QP(75,1) = 5 + 1 + 1 = 7 \% 10 = 5$ (1)

$QP(75,2) = 5 + 2 + 4 = 11 \% 10 = 1$ (2)

⑤ $QP(97,0) = 7 + 0 + 0 = 7 \% 10 = 7$

$Q(97,1) = 7 + 1 + 1 = 9 \% 10 = 9$ (3)

⑥ $QP(18,0) = 8 + 0 + 0 = 8 \% 10 = 8$

$QP(18,1) = 8 + 1 + 1 = 10 \% 10 = 0$ (4)

⑦ $QP(78,0) = 8 + 0 + 0 = 8 \% 10 = 8$

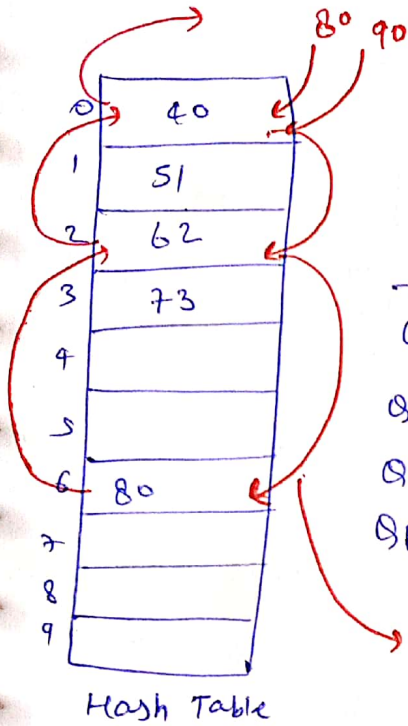
$QP(78,1) = 8 + 1 + 1 = 10 \% 10 = 0$ (5)

$QP(78,2) = 8 + 2 + 4) = 14 \% 10 = 4$ (6)

⑧ $QP(46,0) = 6 + 0 + 0 = 6 \% 10 = 6$

No. of Collision = 6

Quest: keys = 40, 51, 62, 73, 80, 90



$QP(80,0) = 0+0+0 = 0\%10 = 0$

$QP(80,1) = 0+1+1 = 2\%10 = 2$

$QP(80,2) = 0+2+4 = 6\%10 = 6$

$QP(90,0) = 0+0+0 = 0\%10 = 0$

$QP(90,1) = 0+1+1 = 2\%10 = 2$

$QP(90,2) = 0+2+4 = 6\%10 = 6$

$QP(90,3) = 0+3+9 = 12\%10 = 2$

→ This is called Secondary Clustering.

Secondary Clustering:

If the two keys are mapped on to same starting locating in the hash table then they both follow the same path in a quadritic manner because of this Search time complexity will increase.

2. To avoid this problem will use double hashing.

Time Complexity:

Searching: B.C = O(1)
W.C = O(m)

insertion: B.C = O(1)
W.C = O(m)

deletion: B.C = O(1)
W.C = O(m)

Double Hashing :

$H.f_1(key) = k \bmod m$

$H.f_2(key) = 1 + (k \bmod (m-2))$ $\quad m = 10$

$D.H.(key, j) = (H.f_1(key) + j * H.f_2(key)) \bmod m$

$j = 0, 1, 2 \ldots 9\ (m-1)$

Ques: Keys : 25, 98, 57, 75, 97, 18, 78, 65, 46

	Hash Table (m)
0	
1	97
2	78
3	65
4	18
5	25
6	46
7	57
8	98
9	75

① $D.H.(25,0) = 5 + 0 = 5 \% 10 = 5$

② $D.H(98,0) = 8 + 0 = 8 \% 10 = 8$

③ $D.H(57,0) = 7 + 0 = 7 \% 10 = 7$

④ $DH(75,0) = 5 + 0 = 5 \% 10 = 5$

$DH(75,1) = 5 + 4 = 9 \% 10 = 9$

⑤ $DH(97,0) = 7 + 0 = 7 \% 10 = 7$

$DH(97,1) = 7 + 2 = 9 \% 10 = 9$

$DH(97,2) = 7 + 4 = 11 \% 10 = 1$

⑥ $DH(18,0) = 8 + 0 = 8 \% 10 = 8$

$DH(18,1) = 8 + 3 = 11 \% 10 = 1$

$DH(18,2) = 8 + 4 = 12 \% 10 = 2$

⑦ $DH(78,0) = 8 + 0 = 8 \% 10 = 8$

$DH(78,1) = 8 + 6 = 14 \% 10 = 4$ → check

⑧ $DH(65,0) = 5 + 0 = 5 \% 10 = 5$

$DH(65,1) = 5 +$

Time Complexity:

Search: B.C. = O(1)
W.C = O(m)

insertion: B.C. = O(1)
W.C = O(m)

deletion: B.C = O(1)
W.C = O(m)

Conclusion:

if $n \leq m$, by using perfect hashing (→ is advance data structure) we can achive worst case Search time complexity as O(1) only 99%.

NOTE:

① The expected no. of Probes in a unsuccesful search of open addressing technique is

$$\frac{1}{1-\alpha}$$

, where 'α' is load factor

② The expected no. of Probes in a succesful search of open addressing technique is

$$\frac{1}{\alpha} \log \frac{1}{1-\alpha}$$

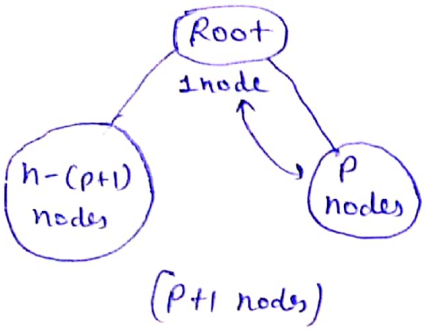
where 'α' is load factor

$$\alpha = \frac{n}{m}$$

IMPORTANT NOTES :

1. The first number to be inserted in the tree must be

No. 1, 2, 3 --- n are inserted in BST



we know total nodes = n

left (root) + 1 + p = n

left (root) = n - 1 - p

left (root) = n - (1+p)

= n - (p+1)

So, root element will be = left element + 1

= n - (p+1) + 1

= n - p - 1 + 1

First no. to be inserted → $= n - p$

2. The minimum nomber of nodes in the tree is = $2^{h-1} + 1$

for example goto PYQ → 5.24

3. The maximum number of nodes in a binary tree of height (h) = ?

$2^{h+1} - 1$

4. If every internal node has exactly K children. Then the number of leaves/leaf in such a tree with 'n' internal nodes = $n(K-1)+1$

∵ also if every internal node has exactly 'K' children then (K-1) key contain no leaves/leaf.

5. If you have internal node (I) and leaves/laaf (L) then Calculate Complete n-ary tree the value of 'n' is

$I[n-1] + 1 = L$

6. The maximum number of Binary contain 'n' no.

$\Rightarrow \frac{1}{n+1} \left({}^{2n}C_n \right)$

$= \frac{1}{n+1} \times \frac{(2n)!}{(2n-n)!\, n!}$

7. Vertices and node are same.

8. # HANDSHAKING THEOREMS

In Graph Theory, Handshaking Theorem states in any given graph, Sum of degree of all the vertices is twice the number of edges contained in it.

$$\sum_{i=1}^{n} d(v_i) = 2 \times |E|$$

→ The sum of degree of all the vertices is always even.

→ The sum of degree of all the vertices with odd degree is always even

→ The number of vertices with odd degree are always even.

E.g. A simple graph G has 24 edges and degree of each vertax is 4 find the no. of vertices.

No. of edges = 24

Degree of each vertax = 4

Using Hand shaking. Theorem.

Sum of degree of all vertices = 2x No. of edges

$n \times 4 = 2 \times 24$

$\boxed{n = 12}$