C-Programming and DS

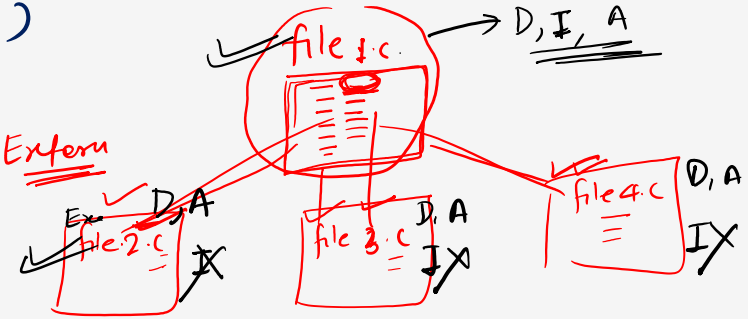
Today Class Topics

-Basics of pointer
-Pointer to pointer
-Call by value
-Call by reference
-Pointer to an Array
-Multidimensional Arrays

C-Programming and DS

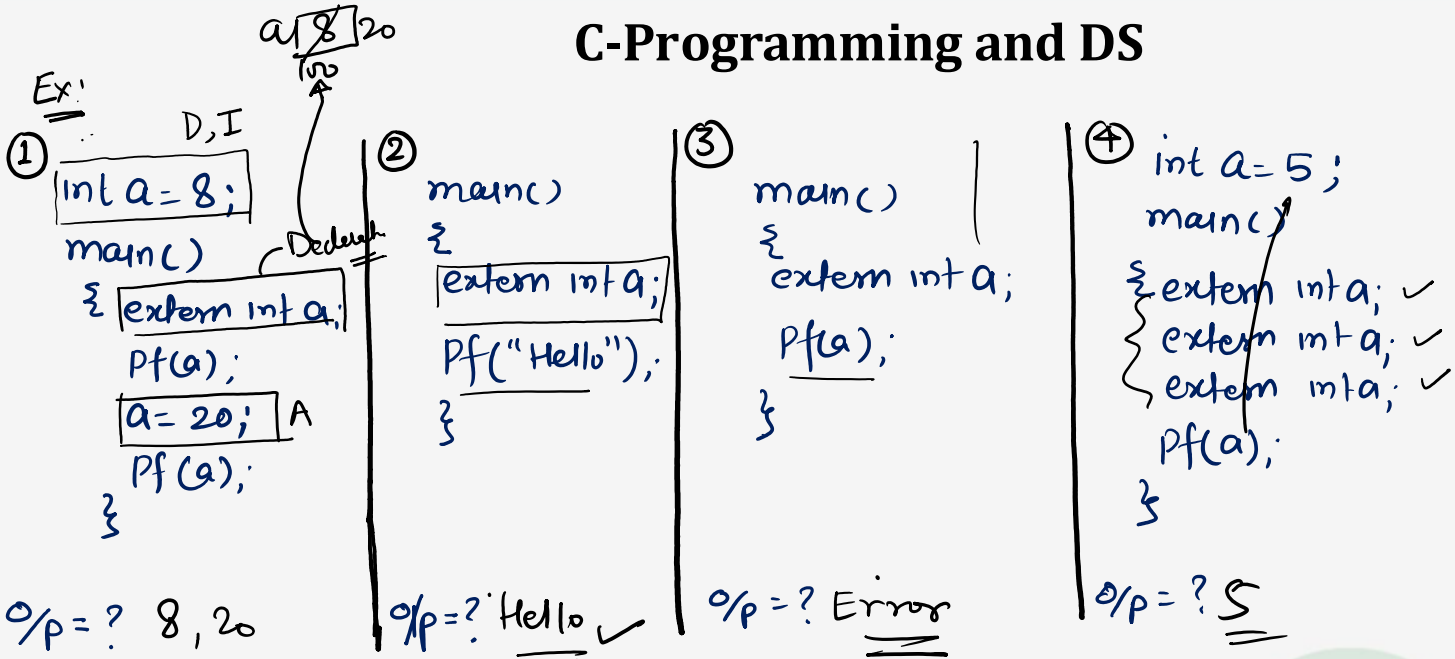
Extern Storage class :- No memory will be allocated to extern variable and they make use of memory outside of variable (Global variable)

- Default value → 0
- Storage → RAM
- Scope → Global
- Lifetime → till the end of the program

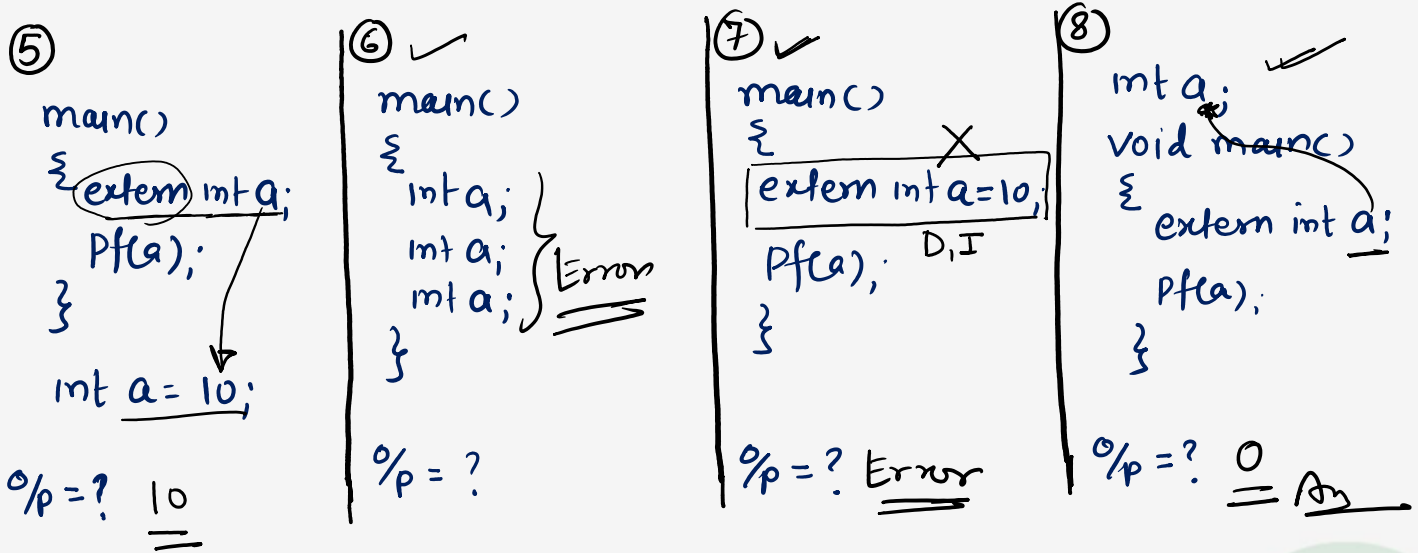


- Extern is short name of external
- used when a particular file need to access a variable from another file
- This will save the memory and provide reusability
- Extern will not work for local variables
- Global variables are available only in the same class or file but the scope of extern variable may be beyond one class or file

C-Programming and DS



C-Programming and DS



a=10;

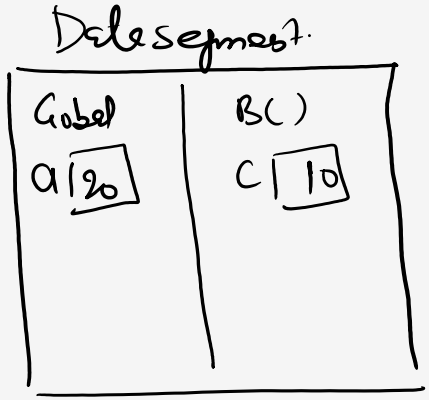
C-Programming and DS

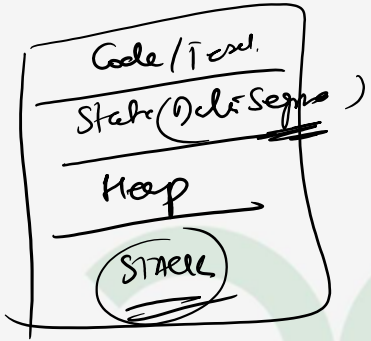
D, I

Ex: `int a=20;`

```
main()
{
    int a=10;      Def
    int b;         D
    b=20;          Assignment
    B();
}
```

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





C-Programming and DS

Ex:-

D.I

```
int a = 20;
main()
{
```

D.I

```
Static int a = 30;
BC();
}
```

```
BC()
{
```

D.I.

```
Static int a = 40;
CC();
}
```

```
CC()
{
```

D

```
extern int a;
}
```

Data segments

Ex:-

D,I

```
a = 10;
```

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

```
BC()
{
    a = 50;     D, I
    b = 20;     D, I
    b = 10;     A
}
```

C-Programming and DS

Scope :- It is categorized into 2 types

i) Static Scoping or Lexical Scoping (CompileTime)

ii) Dynamic Scoping or fluid Binding (Runtime)

Scoping rule :- Scoping rules are applicable only for the "free variable"

free variable : The variable which is not declared in the function but used in the function is called as free variable

Static Scoping :- In Static Scoping free variables are resolved by using global variables

Dynamic Scoping :- In dynamic Scoping free variables are resolved from previous calling function

C-Programming and DS

NOTE:-

1) In the worst case the dynamic scoping behaves like static scoping, if no variable is present in the previous calling function then global variable will be printed

2) C language by default follows static scoping

C-Programming and DS

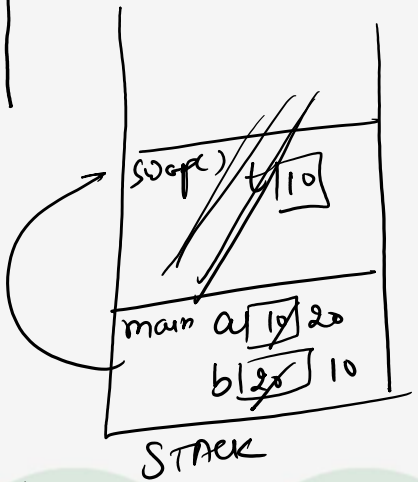
Ex:-
```
int a=50, b=60;
main()
{
  int a,b;
  a=10; b=20;
  swap();
  Pf(a,b);
}
swap()
{
  int t;
  t=a;
  a=b;
  b=t;
  Pf(a,b);
}
```

Data segment

Global

a	50
b	60

free variable



What is the o/p using static and dynamic scoping

Static scoping	Dynamic scoping
60, 50	20, 10
10, 20	20, 10

C-Programming and DS

Ex:-

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

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

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

```
D()
{
  int a=10;
  Pf(a);
}
```

Static scoping

free variable

O/p = ?

Data segment

a 5

Static Scoping	Dynamic scoping
5	30
	20
	10
	5
	Error

C-Programming and DS

GATE Question:

Global

```
Variable x : real
procedure show()
begin
    printf(x);
end
Procedure Small()
begin
    Variable x : real
    x = 0.125;
    show();
end
begin
    x = 0.25;
    show();
    small();
end
```

show, small (local scopes), main

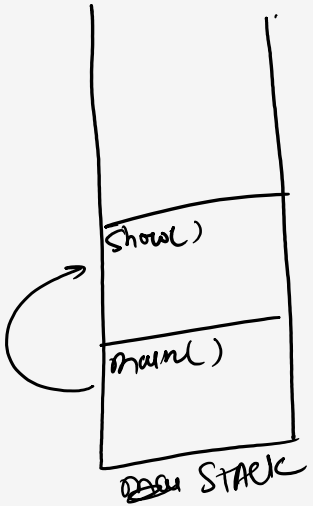
o/p = ?

x

0.25

Static Scoping	Dynamic Scoping
0.25	0.25
0.25	0.125

Ans



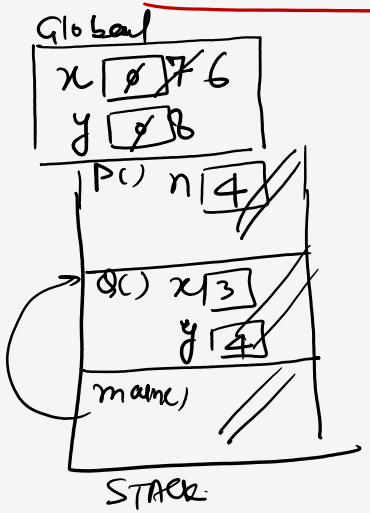
C-Programming and DS

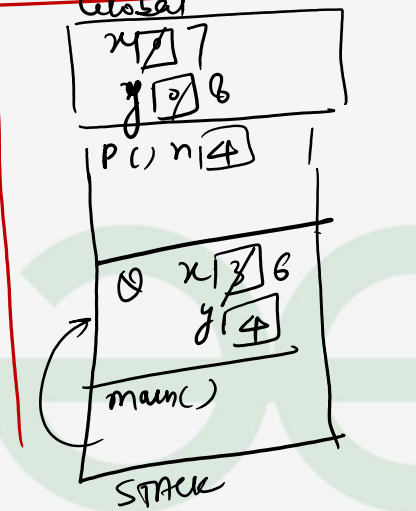
Question.

```
var x,y : integer
Procedure P( var n integer)
begin
    x = (n+2) * (n-3);
end
Procedure Q()
begin
    var x,y : integer
    x=3;
    y=4;
    P(y);
    Printf(x);
end
begin
    x=7, y=8;
    Q();
    Printf(x);
end
main()
```

o/p = ?

Static Scoping	Dynamic Scoping
3, 6	6, 7





C-Programming and DS

2-3 mins

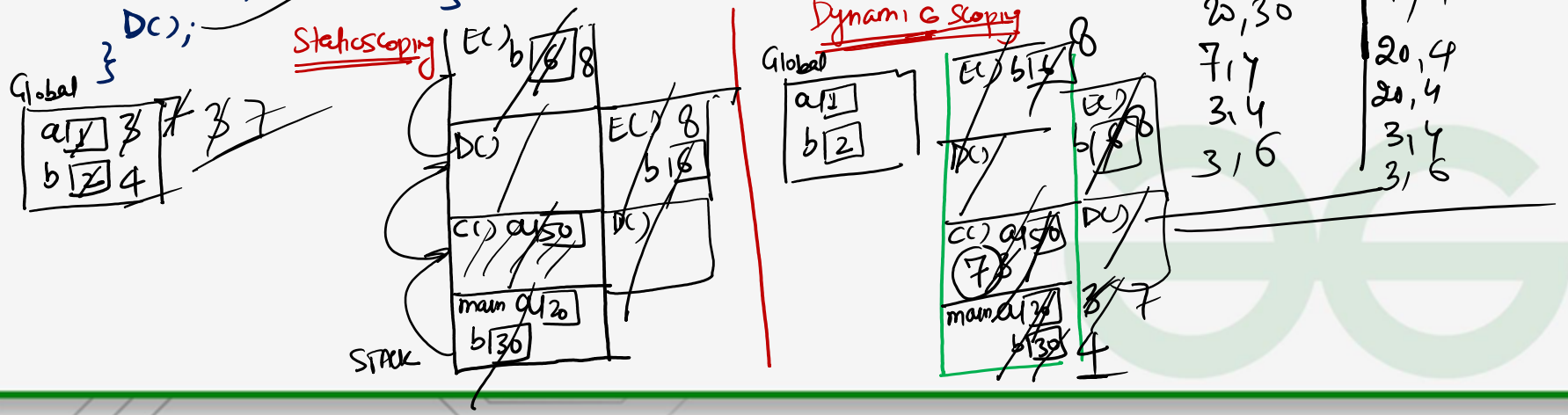
Question:-

```
int a=1, b=2;
main()
{ int a=20, b=30;
  Pf(a,b);
  C();
  Pf(a,b);
  D();
}
C()
{ int a=50;
  Pf(a,b);
  D();
}
D()
{ Pf(a,b);
  a=3, b=4
  Pf(a,b);
  E();
}
E()
{ int b=6;
  Pf(a,b);
  a=7, b=8;
}
```

O/p = ?

Ans

Static scoping	Dynamic scoping
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	3,4
3,6	3,6



C-Programming and DS

Imp

⇒ Pointers :-

int *ptr
float *ptr
char *ptr

char → 1 Byte
int → 2 byte
float → 4 byte

1) Pointer is a variable which is used to store the address
2) The two different operators used in the Pointers
   a) (&) : Address operator
   b) (*) : Indirection operator / dereference operator / value at location / object at location
3) Pointer variable takes same byte of memory irrespective of its datatype (it can be integer Pointer, Character Pointer, float pointer etc)
4) Pointer variable can dereference (access) no. of bytes depending on its data type
5) Integer Pointer can dereference 2 bytes, Character Pointer can dereference 1 byte, float pointer can dereference 4 bytes
6) Pointer variable can be declared using (*)

NOTE:- 1) Address can not be negative
2) Address size will differ from system to system depending on O.S.

C-Programming and DS

Basics:-

① `int a = 10;` `int b;`

a | 10 | 100

`b = &a` ✗

② `int a = 10;`

`int *b = &a;` D&I

(or)

`int *b;` → D

`b = &a` → A

`b = a` ✗

a | 10 | 100

b | 100 | 200

`// Pf("%d", a*b);` Error.

10 10

`Pf("%d", a**b);` ⇒ 100

10 * 10 = 100

* 100

Pf(a) = 10

Pf(&a) = 100

Pf(*&a) = 10

Pf(b) = 100

Pf(*b) = 10

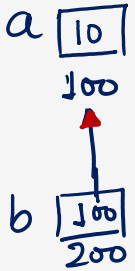
Pf(**b) = *10 → Error

**100

*10

C-Programming and DS

⇒ int a = 10;
int *b = &a;
Pf(b) = ? 100
Pf(*b) = ? 10
Pf(&b) = ? 200
Pf(*&b) = ? 100
Pf(**&b) = ? 10
Pf(**b) = ? Error



Pf(***&b) = ? Error

C-Programming and DS

Parameter Passing Technique:-

1) Call by value
2) Call by reference or Call by Address

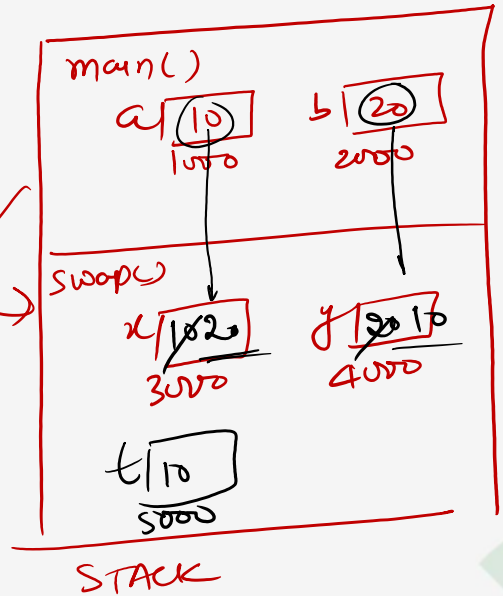
NOTE: C-language only support call by value but we can achieve Call by reference by Passing Address

C-Programming and DS

Call by value :-

```
main()
{ int a=10, b=20;
  Swap(a,b); // Actual params
  printf("%d%d", a,b); //10,20
}

Swap(int x, int y) // formal params
{ int t;
  t=x;
  x=y;
  y=t;
}
```

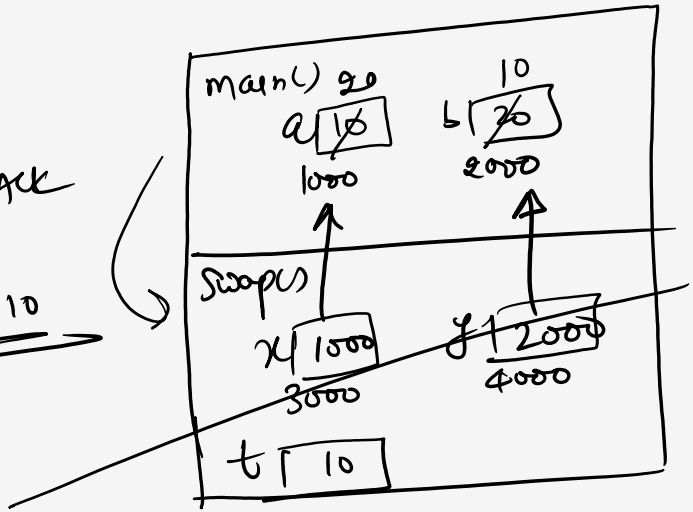


C-Programming and DS

Call by reference:-

```
main()
{ int a=10, b=20;
  swap(&a,&b); //1000, 2000
  printf("%d%d", a,b); //20, 10
}
swap(int *x, int *y)
{ int t;
  t=*x;
  *x=*y;
  *y=t;
}
```

STACK



t = 10

t = *1000

*

& = Address

* = Value at

*1000

Value at 1000

C-Programming and DS

NOTE: if language Support call by reference then Pass Address implicitly

Ex:-

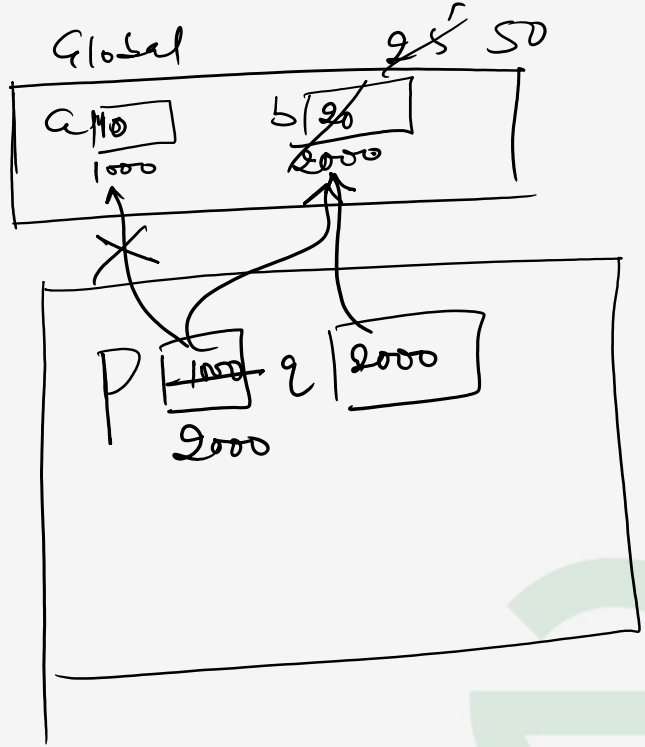
```
main()
{
  int a=10, b=20;
  swap(a,b); // 1000, 2000
  printf("%d%d",a,b);
}
swap(int x, int y) / int *x, int *y
{
  int t;
  t=x; // t=*x
  x=y; // *x=*y
  y=t; // *y=t
}
```

C-Programming and DS

Ex:-

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

What is o/p printed ?



C-Programming and DS

GATE

Ex:-

```
int x;
void Q(int z)
{ z = z+x;
  printf(z);
}
void P(int *y)
{ int x = *y+2;
  Q(x);
  *y = x-1;
  printf(x);
}
main(void)
{ x=5;
  P(&x);
  printf(x);
}
```

x 5 1000

x 7 3000

What is the o/p printed ?

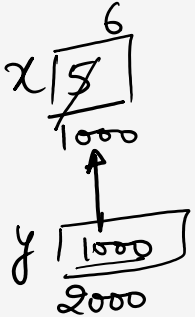
a) 12 7 6

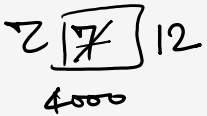
b) 12 6 5

c) 12 6 6

d) 12 7 5

12, 7, 6





C-Programming and DS

GATE:

```
a=3;
void n(x)
{
  x=x*a;
  printf(x);
}
void m(y)
{
  a=1;
  a=y-a;
  n(a);
  printf(a);
}
void main()
{
  m(a);
}
```

D & I

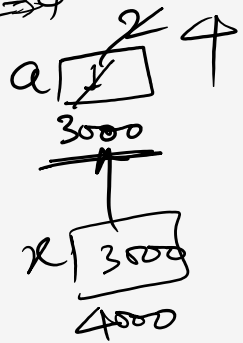
D. I

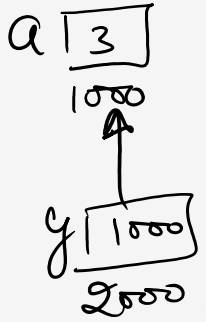
free variable

What will be output of the program, when parameters are passed by reference and dynamic scoping is assumed

a) 6, 2 b) 6, 6 c) 4, 2 d) 4, 4

x = &x * 9
= 2 * 2 ⇒ 4





4, 4

5 mint Break

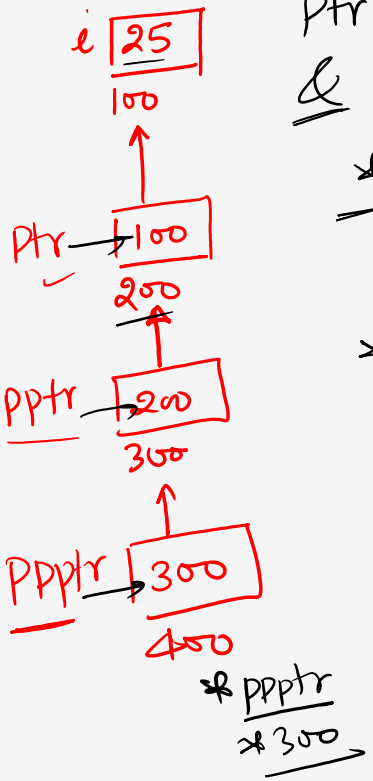
C-Programming and DS

int *ptr

⇒ Pointer to Pointer:-

Ex:
```
void main()
{
    int i;
    int *ptr;
    int **pptr;
    int ***ppptr;
    i=25;
    ptr=&i;
    pptr=&ptr;
    ppptr=&pptr;
}
```

ptr = &i



ptr = &&i

&

*100

**200

*100

25

*ppptr

*300

i → 25
&i → 100
ptr → 100
&ptr → 200
*ptr → 25
pptr → 200
**pptr → 26
*pptr → 100
&pptr → 300
ppptr → 300
&ppptr → 400
*ppptr → 200
**ppptr → 100
***ppptr → 25
****ppptr → Error
****ppptr → 25

C-Programming and DS

Ex:-

```
main()
{ int a=10, *b, **c, ***d;
  b=&a;
  c=&b;
  d=&c;
  Pf(a, b, c, d, *b, *c, *d, **c, **d, ***d);
```

a | 10 | 1000
b | 1000 | 2000
c | 2000 | 3000
d | 3000 | 4000

12 —

10, 1000, 2000, 3000, 10, 1000, 2000, 10, 1000, 10

C-Programming and DS

GATE Question:-

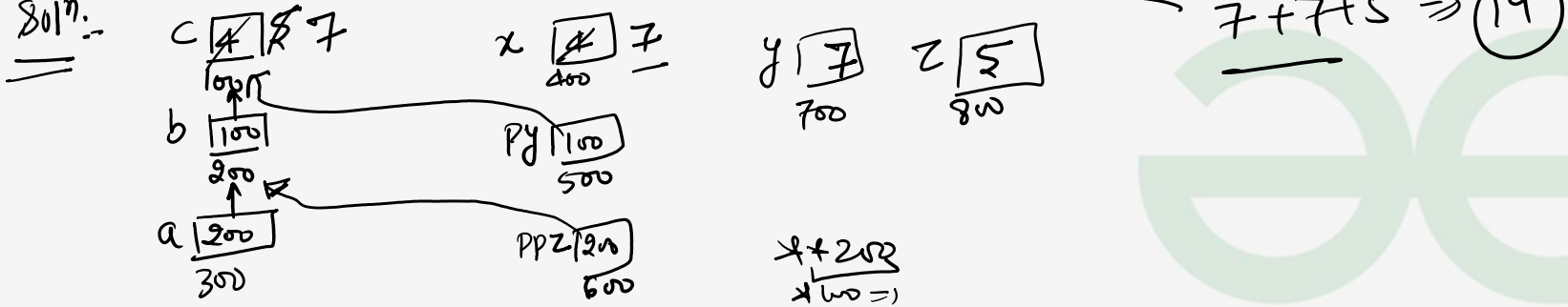
```
void main()
{ int c, *b, **a;
  c = 4;
  b = &c;
  a = &b;
  Pf("%d", f(c,b,a));
}
```

```
int f(int x, int *py, int **ppz)
{ int y, z;
  **ppz = **ppz + 1;
  z = **ppz;
  *py = *py + 2;
  y = *py;
  x = x + 3;
  return x+y+z;
}
```

4, 100, 200

What is the value Printed (19) Ans ?

Soln:-



C-Programming and DS

Question:-

```
void main()
{
  int a;
  int *b=&a;
  scanf("%d",b);
  printf("%d",*b+10);
}
```

O/p = ?

a) Address of a + 10

b) ~~Value entered~~ + 10

c) Address of b + 10;

d) Error

*b + 10 = 30

let a = 20

C-Programming and DS

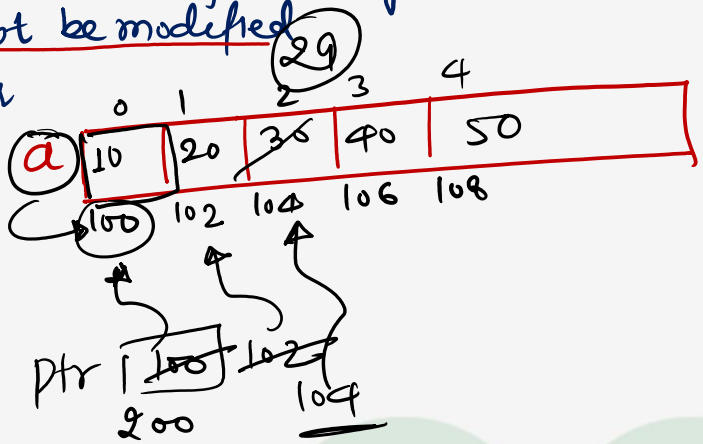
Pointer to Array :

1) Array name without Subscript contains base address of the Array
2) This Base address is constant and it can not be modified
3) Array name is considered as Constant Pointer

Ex:-

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

29



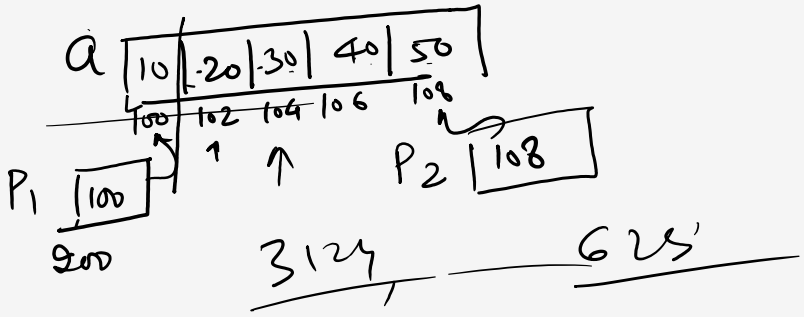
++ptr = ptr = ptr + 1

--*ptr = *ptr = *ptr - 1

Pointer Arithmetic:-

100 + 4 * 2
108

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



1) We can add integer Constant to a Pointer

$\Rightarrow P_1 = P_1 + 1 = 100 + 1 \times 2 = 102$

$a + 4 \Rightarrow 100 + 4 * 2 = 108$

2) We can Subtract integer Constant to the Pointer

$P_2 = P_2 - 2 \Rightarrow 108 - 2 \times 2$

$\Rightarrow 108 - 4 = 104$

3) We can Subtract two pointers $P_2 - P_1 \Rightarrow P_2 \geq P_1$

$P_2 - P_1 = \frac{8}{2} = 4$

(108 − 100)

4) We can not add floating Point no. to a Pointer because it has no meaning

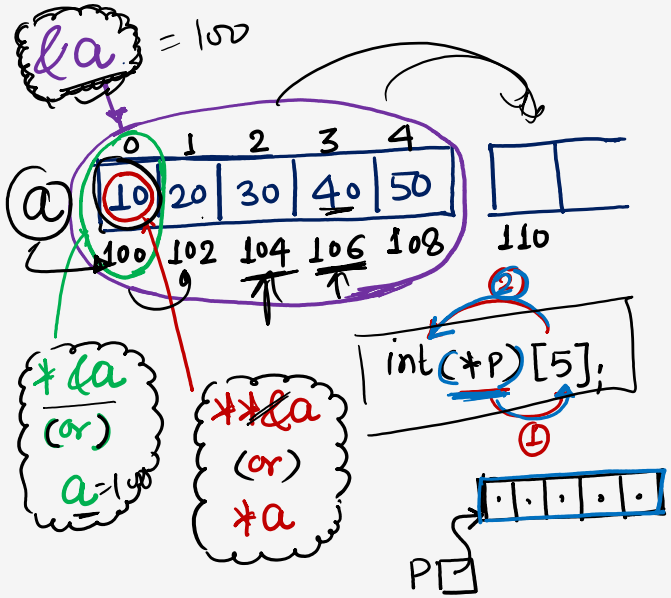
$P_1 + 2.5$ ✗, $P_2 + 1.2$ ✗

5) We can not Perform addition, Multiplication and division between the Pointers because it has no meaning

$P_1 + P_2$ ✗, $P_1 * P_2$ ✗, P_1 / P_2 ✗

1D-Array:-

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



Means P is a Pointer to Array of 5 elements where every element is integer

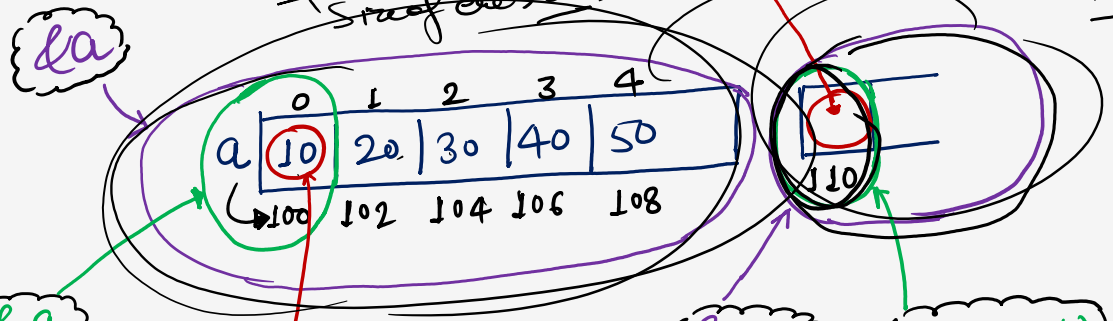
Expression	Print Value	Type	Meaning
1) &a	100	int(*)[]	Base Address of entire 1D Array
2) *&a	100	int *	Base Address of 0th element of 1D Array
3) a	100	int *	" "
4) *a	10	int	value of 0th element
5) **a	error	?	Referring illegal location
6) (&a)+1	110	int(*)[]	Base Add of next Contiguous 1D Array
7) a+1	102	int *	Base Add of 1st element of the Array
8) *a+1	11	int	Adding 1 to the value of 0th element
9) *(a+3) (a[3])	40	int	Value of 3rd element of the Array
10) a[2] *(a+2) *(2+a) 2[a]	30	int	—
11) **&a	10	—	—

Ex:- Pf(*(&a+1)-a);

110 − 100

= 10/2 = 5

→ Size of each element



*&a (or) a ⇒

⇒

**&a or *a

&a+1

*(&a+1)

⇒ 2D 3D

Ex:- Pf((&a+1)-a)

10 Byte − 2 byte

Entire 1D Array

110 − 100

?

Ex:- Pf((&a+1)-&a)); =?

110 − 100 ⇒ 10/10 = 1

10

Passing Array to function :-

Ex:-

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

o/p = ?

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

Thank You !