# C-Programming

## #) Scope of a variable :-

1.) Static Scoping (default)

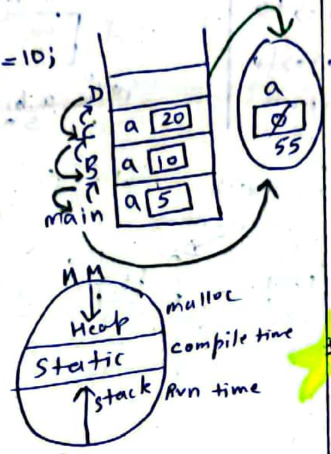
2.) Dynamic "

```
Ex:) int a=55; ——▷ mem. at compile time
main()
{ int a=5
  B();
}

B()
{ int a=10;
  C();
}

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

D()
{
  pf(a)
}
```

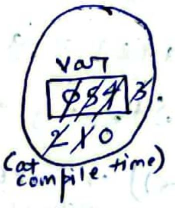


Dynamic scoping : o/p = 20

Static scoping : o/p=55

## #) Static variable :

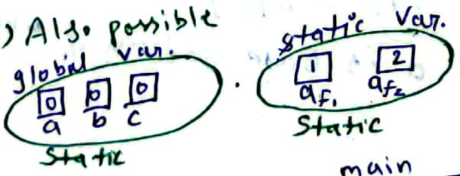
```
Ex:) main()
{ Static int var=5;
  if(var) { pf(var--);
            main();
          }
}
```



o/p: 54321

```
main()
1. ✗
2. if(5)
  3. ✓
  4. main()
    1. ✗
    2. if(4)
      3. ✓
      4. main()
                 1. ✗
                 2. if(3)
                   3. ✓
                   4. main()
                     1. ✗
                     2. if(2)
                       3. ✓
                       4. main()
                         1. ✗
                         2. if(0)
```

Notes:) Also possible

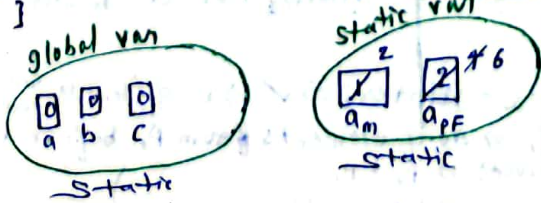


```
Ex:) int a,b,c =0;
main()
{1 Static int a=1;
 2 prtFun();
 3 a+=1;
 4 prtFun();
 5 pf("\n %d %d",a,b);
}

prtFun()
{1 static int a=2;
 2 int b=1;
 3 a+=++b;
 4 pf("\n %d %d",a,b);
}
```

o/p:
4,2
6,2
2,0

```
main
1. ✓
2. pFun()
   1. ✓
   2. [~~1~~] 2
       b
   3. ✓
   4. Pf(4,2)
3. ✓
4. pFun()
   1. ✗
   2. [~~1~~] 2
       b
   3. ✓
   4. Pf(6,2)
5. Pf(2, 0)
```



Note:) auto int a=1; ⟶ => inside stack
register int a=2; ⟶ i.e. same as int a = 2;

## #) Extern Variable: variable sharing memory with global variable

```
Ex:) int a=5;
main() { extern int a;
         a=20;
         pf(a);
       }
```

global
$a_m$
~~5~~ 20
a
Static

1

Note:) Global extern variable will allocate memory if it is initialized, but not local extern variable.

Note:) Extern keyword power is that u can create 1 lakh local variables with same name.

## #) Associativity & Precedence :

| | |
|---|---|
| ( ) [ ] —> . | L to R |
| ! ~ ++ -- + - * (pointer) (type) sizeof | R to L |
| * / % | L to R |
| + − | L to R |
| << >> | L to R |
| < <= > >= | L to R |
| == != | L to R |
| & Bitwise AND | L to R |
| ^ Bitwise XOR | L to R |
| \| Bitwise OR | L to R |
| && | L to R |
| \|\| | L to R |
| ?: | R to L |
| = += -= *= /= %= &= ^= \|= <<= >>= | R to L |
| , | L to R |

## #) a[3][5] = {10,20,30 . . .150}

2D

$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline & 1000 & 1002 & 1004 & 1006 & 1008 \\ 0 & 10 & 20 & 30 & 40 & 50 \\ & 1010 & 1012 & 1014 & 1016 & 1018 \\ 1 & 60 & 70 & 80 & 90 & 100 \\ & 1020 & 1022 & 1024 & 1026 & 1028 \\ 2 & 110 & 120 & 130 & 140 & 150 \end{array}$$

| | |
|---|---|
| a ⇒ 1000<br>&a ⇒ 1000<br>&a+1 ⇒ 1030 (array skip) | a+1 ⇒ 1010 (1 row skip)<br>a+2 ⇒ 1020 (2 row skip) |

*(a+1) ⇒ 1010 (1th row selected)

$*(a+1)+\underset{\text{1 element}}{\underset{\downarrow}{1}} \Rightarrow 1012$

*(*(a+1)+1) ⇒ 70

a[15]={10,20 . . . 150}

1 D

| | | |
|---|---|---|
| 0 | 10 | 1000 |
| 1 | 20 | 1002 |
| 2 | 30 | 1004 |
| 3 | 40 | 1006 |
| 4 | 50 | 1008 |
| 5 | 60 | 1010 |
| 6 | 70 | 1012 |
| 7 | 80 | 1014 |
| 8 | 90 | 1016 |
| 9 | 100 | 1018 |
| 10 | 110 | 1020 |
| 11 | 120 | 1022 |
| 12 | 130 | 1024 |
| 13 | 140 | 1026 |
| 14 | 150 | 1028 |

| | |
|---|---|
| a ⇒ 1000<br>a+1 ⇒ 1002<br>a+3 ⇒ 1006 | &a ⇒ 1000<br>&a+1 ⇒ 1030 (array skip) |

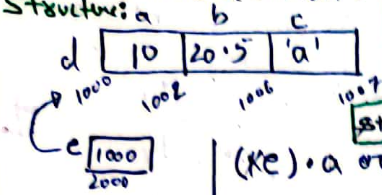
Note:)
Static Scoping = early binding = ↑ Exec.ⁿ efficiency
Dynamic " = late binding = ↓ " "

Note:) wrong Practice in pointers:-
int a = address ✗
int *a = data ✗

Note: In ASCII
A=65 a=97

## #) Structure & union:

Structure:



```
struct node{
    int a;
    float b;
    char c; }d;
d.a = 10;
d.b = 20.5;
d.c = 'a';
struct node *e = &d;
```

(*e)·a or e→a => 10

*e => error

e+1 => 1007

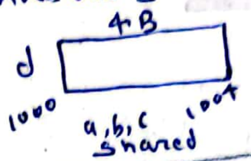
d => 1000

pf(%d, d) => 10

pf(%d, *e) => 10

pf(e) => 1000

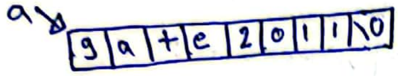
Union: Stores only one variable at a time, when you store another variable, the previous one gets deleted



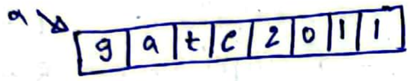
2

## #) Arrays:

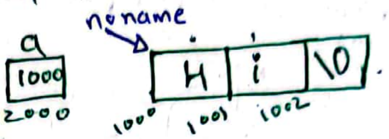
•) char a[] = "gate2011";



Char a[] = {'g', 'a', 't', 'e', '2', '0', '1', '1'};



Char *a = "Hi";



•) *(a+2) = 'g' => possible

a = a+2 => Not-possible

<Address Constant>

a = a+2 => possible

*(a+2) = 'g' => May not possible

<String constant>

•) strlen(a) = 8

sizeof(a) = 9

•) pf("%c", 1000 or (a+5)) => Error may come

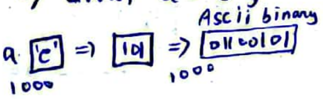
•) a[2] is *(a+2)    •) a[i] is *(a+i) is *(i+a) is i[a]

•) a[i][j] is *(*(a+i)+j)    •) a[i][j][k] is *(*(*(a+i)+j)+k)

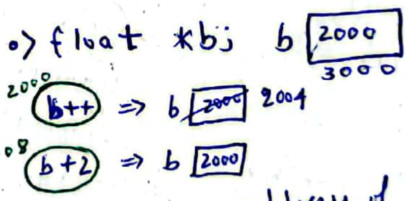
•) if(0)
if(0.0)
if('\0')
if(NULL)

cond.$^n$ fail

•) char a = 'e';

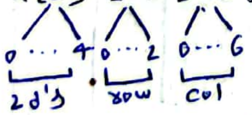


•) float *b;



•) In 2d Array a[1] is base address of 1th row, a is base address of whole 2d Array.

•) Ex:) int a[5] [3] [7]; BA = 1000



a => 1000 (i.e. 000)
*a => 1000 (i.e. 0th 2d selected)
**a => 1000 (0th 2d, 0th row selected)
***a => value (a[0][0][0])

a => 1000
*a => 1000 (0th 2d selected)
*a + 5 => 1070
rows

**a => 1000 (0th 2d, 0th row selected)
**a + 5 => 1010.
elements

a => 1000
a+1 => 1042 (∵ 1 2d's skip)
a+3 => 1126 (3 2d's skip)
*(a+3) => 1126 (3rd 2d selected)

a[2] + 9 => *(a+2) + 9
= *(1000 + 2×21×2) + 9
= *(1084) + 9
= 2nd 2d selected + 9 rows
= 1084 + 9 × 7 × 2
= 1210

a => 1000
&a => 1000
&a+1 => 1000 + 5×3×7×2
= 1210
↓
1 3d's skip

3· Pointer$_1$ + or / or * Pointer$_2$ => ✗ Error

Note:) 1·) Pointer Var ± Integer constant => ✓ => Skipping elements

int *p;    P + 5 => Skip 5 elements in forward
P − 5 => Skip 5 elements in backward

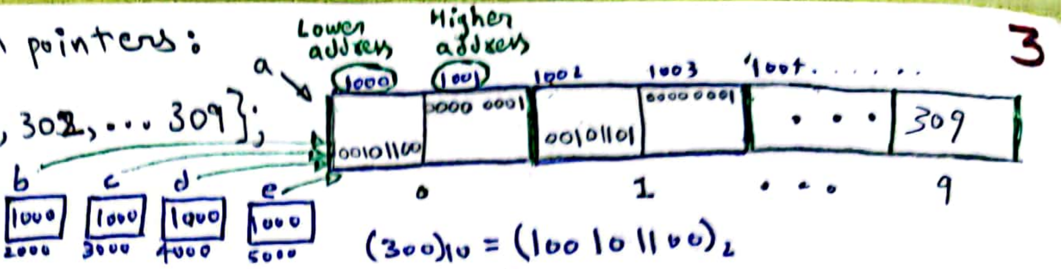
2·) Pointer$_1$ − Pointer$_2$ => ✓ => no. of elements b/w the pointers

$P_2 - P_1$ => No. of elements from $P_1$ before $P_2$

( provided: 1) $P_2 \geq P_1$
2) $P_1 P_2$ points same array )

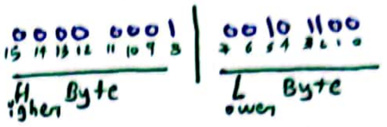
CamScanner

## #> Typecasting in pointers:



```
① int a[10] = {300, 301, 302, ... 309};
② int *b = a;
③ char *c = (char *)a;
④ float *d = (float *)a;
⑤ void *e = a;
⑥ int *f = (int *)e;
⑦ b = d; //Error
⑧ b = c; // No Error
```

$(300)_{10} = (100101100)_2$



Little Endian ⇒ Lower address contain Lower Byte

| b ⇒ 1000 | c ⇒ 1000 | d ⇒ 1000 | e ⇒ 1000 |
|---|---|---|---|
| *b ⇒ 300 | *c ⇒ 44 | *d ⇒ 4B data from 1000 onwards in IEEE 754 | *e ⇒ Error |
| b+1 ⇒ 1002 | c+1 ⇒ 1001 | d+1 ⇒ 1004 | e+1 ⇒ Error |

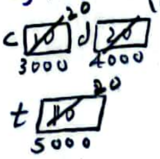
## #> Parameter passing techniques :-

### 1.) Call by value :- (call by copy)

main() a [10] 1000 b [20] 2000

```
main()
{ int a=10, b=20;
  pf(a,b);
  Swap(a, b);
  pf(a,b);
}
```

```
swap( int c, int d)
{ int t;
  t = c;
  c = d;
  d = t;
}
```



other names :-
call by name
call by need

o/p: 10,20
10,20

- Used when we want to pass very less no. of parameters.

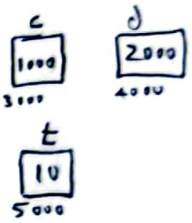
Note: C lang. is CSL bec.
1. Variable declared before use
2. Matching formal & actual parameters of fn
} PDA can't handle

### 2.) Call by reference :-

main() a [10] 1000 b [20] 2000

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

```
swap(int c, int d)
{ int t;
  t = *c;
  *c = *d;
  *d = t;
}
```



o/p: 10, 20
20, 10

- used when we want to pass very large no. of parameters.

- Here, How a compiler will behave, which supports call by reference :- That lang. will himself keep '&' in swap(a, b) & '*' in swap(c, d).

Note:)

```
int a=3, b=5;
unsigned int c=0;
x = (a-b) > c ? 1 : 0;
```

Here, (3-5) > 0 [∵ c is unsigned int]
2 > 0

so, x = 1

Note:) *p1 [2000] *p2 [2008]

pf(p2-p1);

$\Rightarrow \frac{2008-2000}{8} = 1$ = o/p

Note:) In # define F(a,b) (a*b)
a, b are formal parameters

Note:

| x | 11 | -11 |
|---|---|---|
| ~x | -12 | 10 |

बड़ा हो & sign change

छोटा हो & sign change

~ is Bitwise complement (1's)

bec. 11 = 0000 1011
-12 = 1111 0100 } 1's

CamScanner