It can go on till any no: of time.

But usually: it is pointer to a pointer.

Sizes:-

```
int *alpha;
char *ch;
float *s;
```

> Var, holding addrs of int, char, float vars

```
Printf("%d %d %d", sizeof(alpha),
                   sizeof(ch),
                   sizeof(s));
```

O/P: (4 4 4).

size of ptr = 2 bytes (16 bit)
4 bytes (32 bit)
8 bytes (64 bit)

↙

not dependent on the datatype the pointer points to.

All pointers regardless of the type of data type, have the same size based on system Architecture.

call by value & call by reference.

↓

value of each actual arg is copied into corresponding formal arguments.

Hence changes made to the formal Parameters has no effect on actual arguments

```
void swap (int x, int y);
main()
{
    int a=10, b=20;
    swap (a,b);
    Print (a,b);
}
void swap (int x, int y);
{ int temp;
  temp= x;
  x = y;
  y = temp;
  Print x, y;
}
```

o/p:-

x=20, y=10

a=10, b=20.

Ref. the addr. of actual arguments in the call are copied into the formal args.

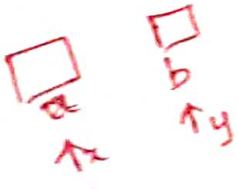
Hence values will be affected.

```
void swap (int *, int *);
main()
{
    int a=10, b=20;
    swap (&a, &b);
    Printf ("a,b);
}
```

→ explain with boxes

x = &a;

y = &b



o/p

a=20,

b=10

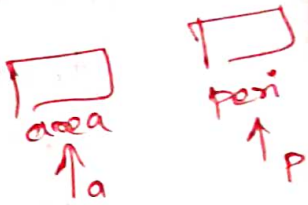
```
void swap (int *x, int *y);
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
}
```

Utility of call by reference :-

return → returns only 1 value from a function at a time.

overcome the limitation by using call by reference.

```
Void areaperi (int , float * , float * )
main()
{
    int radius;
    float area, perimeter;
    scanf ("%d", &radius);
    areaperi(radius, &area, &perimeter);
    Printf (Area);
            Perimeter);
}
void areaperi (int r, float * a , float *p)
{
    *a = 3.14 * r * r ;
    *p = 2 * 3.14 * r ;
}
```



O/P:

rad = 5.

Area = 78.5

Peri = 31.4

R= value

Area, Peri = Addr.

↓

output both the values of area & perimeter.

Conclusions:-

1-> value shouldnot be changed → pass by value.

if value needs to be changed → pass by reference.

If returns more than 1 value } → return indirectly by call by ref.

Uses:-

1) Return multiple values from a func.

2) to acess | manipulate array | strings.

3) if array or structure is passed by val, to a function a copy is created leading to wastage of memory space.

avoided by passing address as pointers.

4) dynamic memory allocation. (allocate memory at runtime).

⇒ H/w: write a program that defines a func that calculates power of one number raised to another and factorial value of a number in one call.

class:- write a function that receives 5 integers and returns the sum, average and of these numbers. call this func from main() & print the results in main().

```
void stats (int *, int *, double *);
main()
{
        int sum, avg;
        double stdev;
        stats (&sum, &avg, &stdev);
    Printf  sum, avg, stdev.;
}

void stats (int * sum, int * avg, int * stdev)
{
    int  n1, n2, n3, n4, n5;
    Scanf ("%d.... %d", &n1, &n2, &n3, &n4,
                                    &n5);
    * sum = n1+n2+ n3+n4+n5;
    * avg =  * sum / 5;
  * stdev  =  sqrt (( pow (( n1 - * avg), 2.0)+
                pow ((n2- *avg ), 2.0) + \
                pow ((n3- *avg) , 2.0) +
                pow ((n4 - *avg) , 2.0) + \
                pow ((n5 - *avg) , 2.0)) /4);
```

$$\sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}}$$

O/p:

10 20 30 40 50

Sum = 150

Avg = 30

SD = 15.811388