/

# Let’s solve it

42

# Pointer & Functions

## Pass by address

Pass by value ( copy actual argument data into memory of formal arguments )

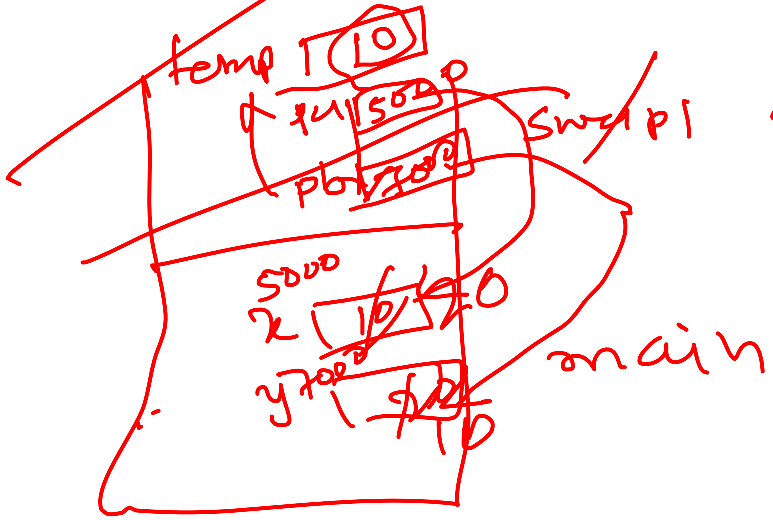
```
int main()
{
    int x = 10, y = 20;

    swap1( x , y );

    swap2( &x , &y );
```

```
void swap1( int a, int b )
{
    int temp;
    temp = a; a = b; b = temp;
}

void swap2(int *pa , int *pb)
{
    int temp;
    temp = *pa;  // x // 10
    *pa = *pb; //20  *pb = temp;
}
```

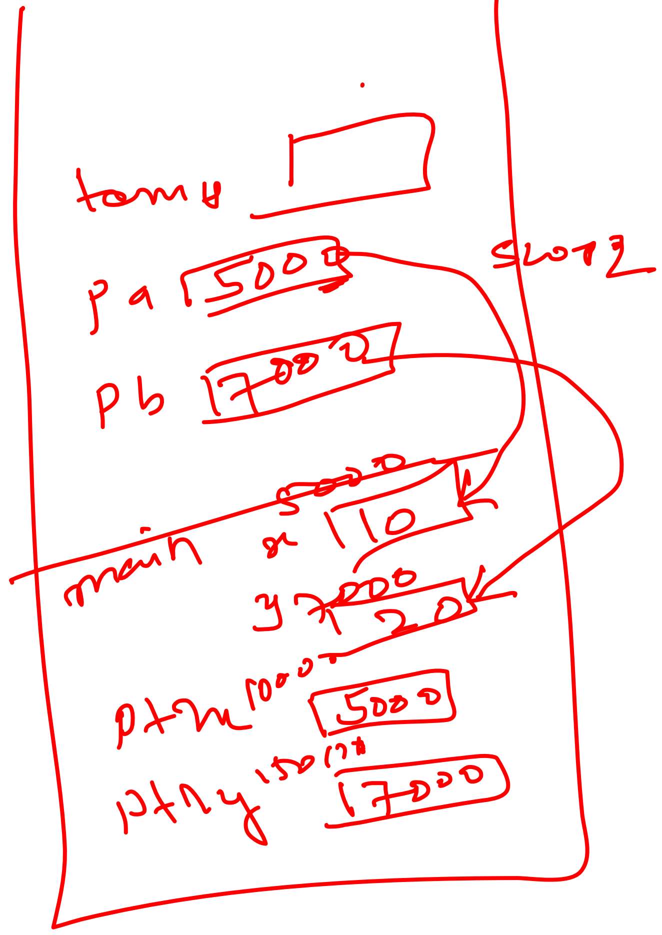


return can return only one thing.

pass by address can support multiple information exchange indirectly.

main

```
int *ptrx = &x;
int *ptry = &y;
swap~~3~~2( ptrx, ptry );
```



~~swap3~~

Is this pass by value or address?

ptrx
ptry

x, ~~x~~
y

```
Swap2( int *pa, int *pb)
{

}
```

What can you change from swap2 of main fn memory?
only x & y
Not ptrx & ptry.

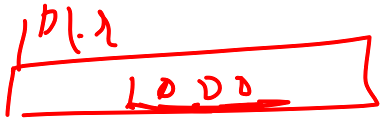
```
xyz( &ptrx,   &ptry );

int*  *pptrx = & ptrx;
int*  *pptry = & ptry ;
xyz(  pptrx , pptry  );        // pptrx }pass by
                               //  pptry } value
                                  ptrx } pass by
                                  ptry } address
                                  their
                                  memory
                                  can be
                                  changed.
void
xyz( int** [1000] ppa , int** [1500] ppb )
{ ~~int temp;~~
temp = ~~**ppa~~ ;   This
**ppa = **ppb ;   has nothing
} **ppb = temp ;    to do with swapping.
```

# Initialization of pointer

```
int *ptr;    int x = 1000;

ptr = x; ✗
```



```
ptr = 1000; ✗

ptr = &x;

int * ptr = 0;
```

NULL initialization

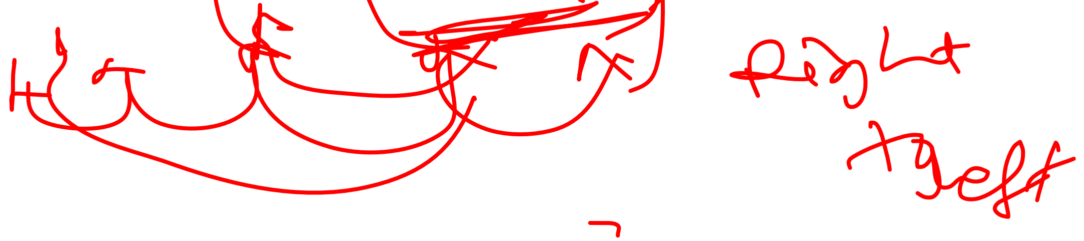
It is good practice to initialize pointer with NULL, if nothing else.

```
int *ptr;
*ptr // Access random memory.
```

Segmentation fault.

```
int x = 10;
int *p1, *p2, *p3;

p1 = (p2 = (p3 = &x));
```



```
int x, y, z;
int *p;

p = &x;

p = &y;

p = &z;
```

```
int x = 10,  y = 20;

int *px = &x,   *py = &y ;      //   ~~*py~~
                                      px(*py)

*px  +  *py      //  x + y

*px  -  *py      //  x - y

*px  /  *py

*px   *   *py
```

Deref   multiplication   Dereferencing