08/10/2023

* **Pointer to an array →**

Pointer to an array is a pointer that points to first element of an array.

Code→
```
#include <bits/stdc++.h>
using namespace std;
int main(){
int nums[] = {1,2,3};
int (*ptr)[3] = &nums; //pointer to array of 3 integers
cout << (*ptr)[1] << endl;

int *ptr2 = nums; //pointer to 1st index of nums
cout << *(ptr2 + 2) << endl;
}
```

O/P = 2
3

* **Array of pointers →**

An array of pointers is an array where each element is a pointer to a memory location.

Code→
```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int nums[5]={1,2,3,4,5};
    int *arr[5];
    for(int i=0; i<5; i++){
        arr[i] = &nums[i];
        cout << *arr[i] << endl;
}}
```

O/P = 1
2
3
4
5

Note → While making pointer to an array,

```
int nums[3] = {1,2,3};
int (*ptr)[3] = &nums;
```

the parentheses around *ptr is necessary, because the de-reference operator * has lower precedence than the array subscript operator [].

* **What happens when an Array is passed to a function ?**

```
#include<bits/stdc++.h>
using namespace std;

int solve (int arr[], int size){        // This is pointer to arr.
   cout <<"size of arr inside solve: "<< sizeof(arr)<<endl;
  }

int main(){
int arr[] = {1,2,3,4};
cout <<" size of arr inside main: "<< sizeof(arr)<<endl;
 solve(arr,4);
}
```

O/P = size of arr inside main: 16
size of arr inside solve : 8 (Pointer address is architecture dependent)

Note - When an array is passed to a function, not the whole array is passed. Actually, the base address of array is passed to the function. We can also write int arr[] in solve function as int *arr.

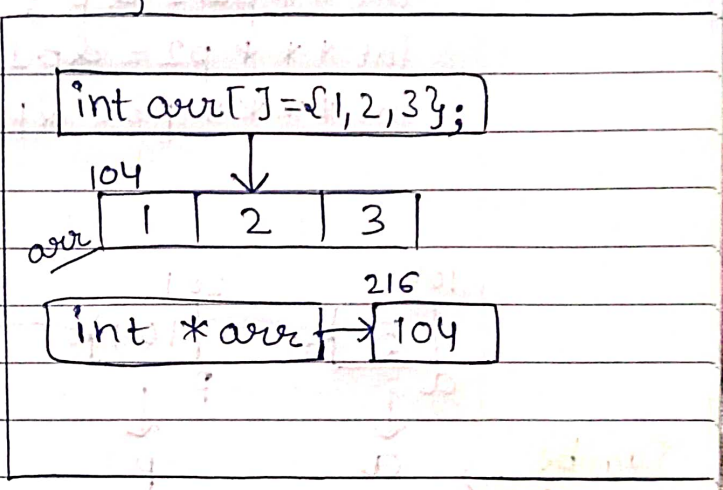
Q1-

```
main()
{
int arr[]={1,2,3};
solve(arr, 3);

//Print
  arr → 104
  &arr → 104
}
```

```
solve(int *arr, int size){

//Print
  arr → 104
  &arr → 216
}
```



Q2→

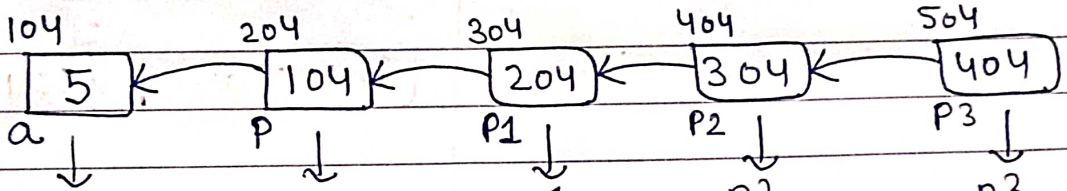
```
main()
{
int arr[]={10,20,30};
solve(arr, 3);

//Print array ⟶ 10 25 30
}
```

```
solve(int arr[], int size)
{
*(arr+1)+=5;
}
```

* Pointer to Pointer →

```
int a = 5;
int *p = &a;
int **p1 = &p ; //double Pointer
int *** p2 = &p1 ; // triple Pointer
int **** p3 = &p2; // multi Pointer
```

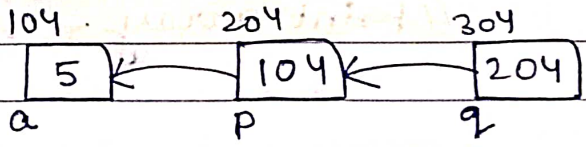


Can be accessed by →

a	p	p1	p2	p3
*p	*p1	*p2	*p3	
**p1	**p2	**p3		
***p2	***p3			
****p3				

Q→

```
int a = 5;
int *p = &a;
int **q = &p;
```

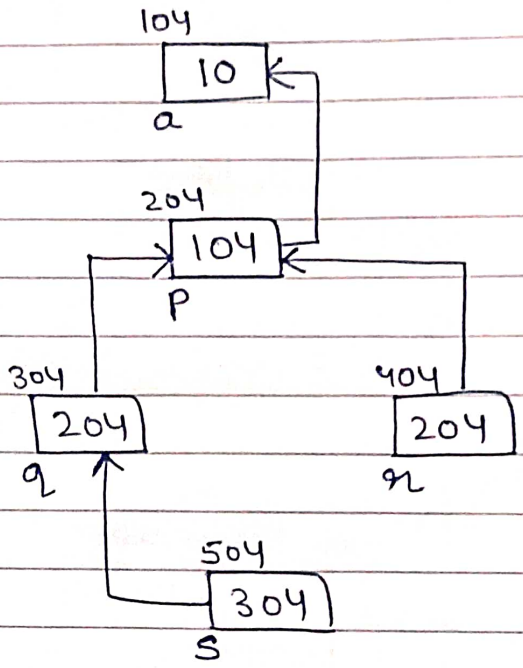


// Print →

a : 5	p : 104	q : 204
&a : 104	&p : 204	&q : 304
	*p : 5	*q : 104
		**q : 5

Q→

```
int a=10;
int *p=&a;
int **q=&p;
int ***r=&p;
int ****s=&q;
```



//Print

```
*s    : 204
**r   : 10
***s  : 10
*q    : 104
*p    : 10
p     : 104
&s    : 504
&r    : 404
&q    : 304
**s+1 : 104+(1x4)=108 (This is because 104 is the address.)
```