Arrays And Pointer Arithmetic:

In this tutorial we will learn about, arrays and pointer arithmetic

There are four arithmetic operators that can be used on Pointers :

- ++
- --
- +
- -

Pointers arithmetic is not same as normal arithmetic i.e. if you want to add 1 to any no. then you will get that no. after adding 1 in it. But in Pointers it is a bit different as:

Base Address : The first byte address of any variable is known as base address.

It means suppose you have int type variable then it's size in my architecture is 4 byte (It may vary in yours) then 4 consecutive blocks will be created in RAM. So when we will point it with any pointer variable at that time the address of 1 block of 4 will come in that pointer variable. It means base address is the first block address of any data type variable.

Pointers Arithmetic :

- We can't add, multiply or divide two addresses. (Subtraction is possible)

E.g:

```
int a, b, *p, *q;

&a*&b;          // Not Possible

&a+&b;          // Not Possible

p*q;            // Not Possible
```

- We can't multiply & divide an address with integer value.

E.g :

```
&a*5;           // Not Possible

p/5;            // Not Possible
```

- We can add or subtract integer to/from an address.

E.g :

```
int a,b;          // Let's Suppose &a is 1000

int *p, *q;

p=&a;

q=&b;

p+1;              // It is possible [1000+1 = 1004]
```

Now you may think why we got 1004 as an output after adding 1 in 1000. So as we know a is an integer. So it'll take 4 bytes of memory & we know pointer always takes base address, but it access other bytes also by itself only.

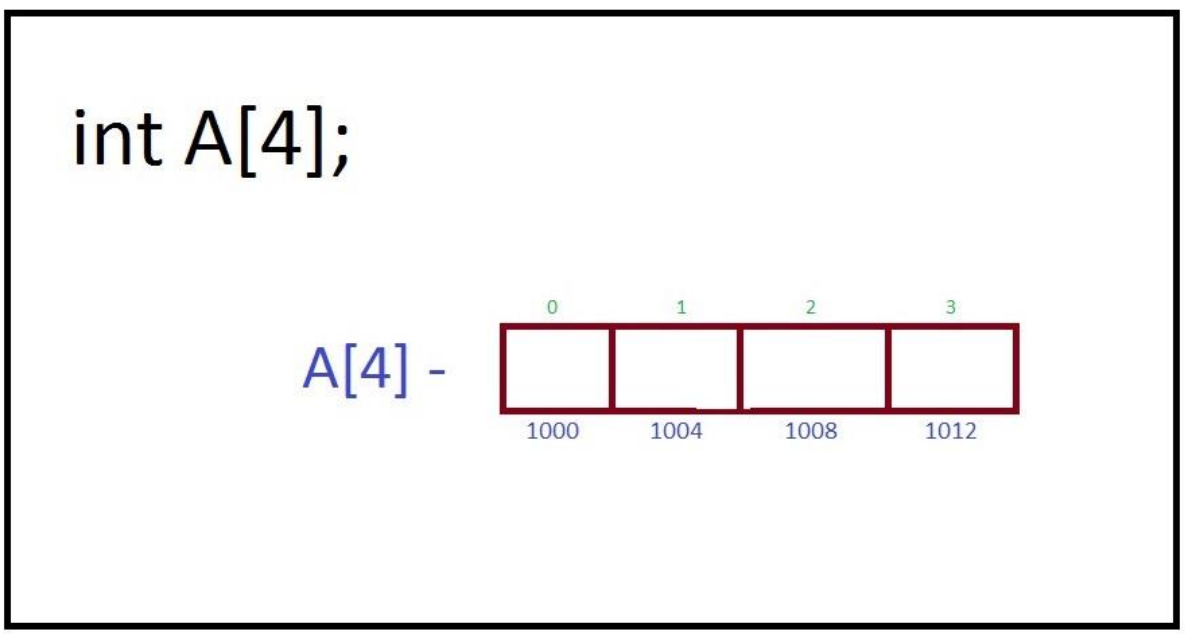
So, when we will add 1000+1 then from 1000 to 1003 all address are allocated to int a. So the next block of memory is after 1003 i.e. 1004. So that's how Pointers arithmetic is done.

To Find arithmetic (+,-) of any pointer :

Pointer + n = pointer + size of (type of pointer)*n

Arrays and Pointers :

- Array always consumes memory location in contiguous fashion/manner.



Suppose address of A[0] is 1000 then the address of next variable in Array will be 1004 (if int type) because we know pointer always takes the base address and when it is incremented or decremented then it contain the address of next block but not of very next byte.

- Pointer when incremented always point to next block of its own type.

When we add or increment in pointer than literal addition doesn't take place instead of that it start pointing next block.

E.g. :

int age[50];

int *p;

p= age;

It means, Either we write &age or &age[0] or age. They all means that the first element of array or index 0 is being used or pointed by pointer.

So to access the other Array variables we can simply do increment or decrements in pointer variable an can access other Array variables too.

```c
#include <stdio.h>
int main()
{
    // char a = '3';
    // char* ptra = &a;
    // printf("%d\n", ptra);
    // ptra--;
    // printf("%d\n", ptra);
    // printf("%d", ptra-2);
    int arr[] = {311,52,3,4,5,6,67};
    int* arrayptr = arr;
    printf("Value at position 3 of the array is %d\n", arr[3]);
    printf("The address of first element of the array is %d \n", &arr[0]);
    printf("The address of first element of the array is %d \n", arr);
    printf("The address of second element of the array is %d \n", &arr[1]);
    printf("The address of second element of the array is %d \n", arr + 1);
    printf("The address of third element of the array is %d \n", &arr[2]);
    printf("The address of third element of the array is %d \n", arr + 2);
    // arr--; // this line will throw an error

    printf("The value at address of first element of the array is %d \n", *(&arr[0]));
    printf("The value at address of first element of the array is %d \n", arr[0]);
```

```c
    printf("The value at address of first element of the array is %d \n", *(arr));
    printf("The value at address of second element of the array is %d \n", *(&arr[1]));
    printf("The value at address of second element of the array is %d \n", arr[1]);
    printf("The value at address of second element of the array is %d \n", *(arr + 1));

    return 0;
}
```