Parallel processors

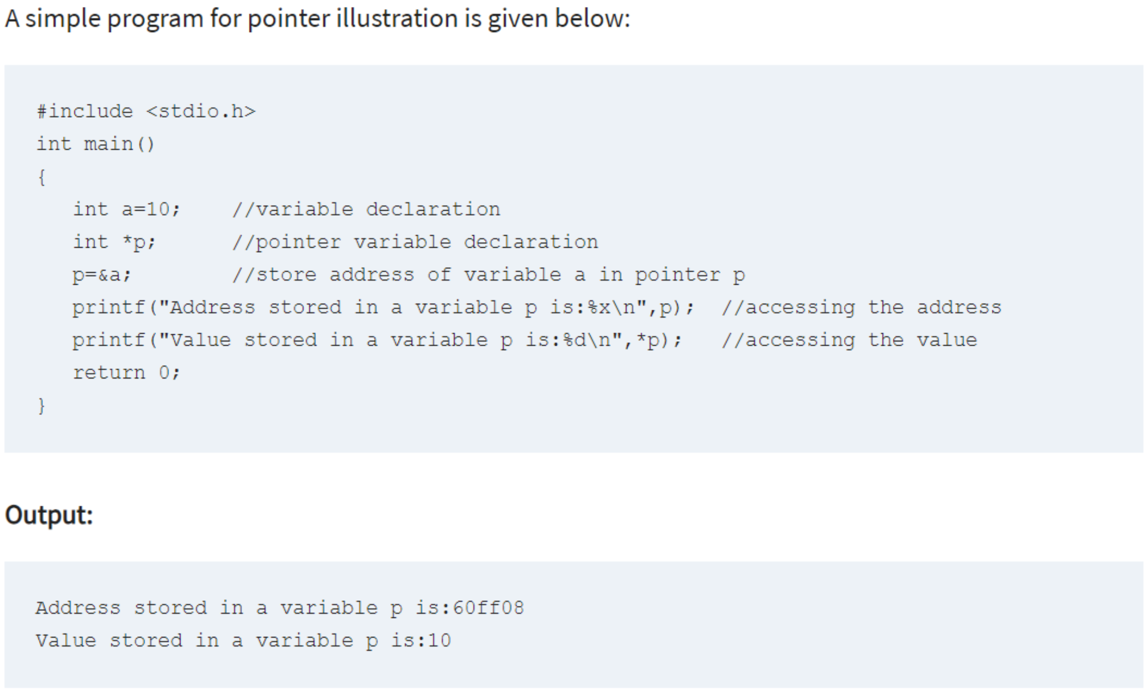
Pointers

DR Mohamed Mahmoud

Abdallah Saad Ahmed

What is Pointer in C?

The **Pointer** in C, is a variable that stores address of another variable. A pointer can also be used to refer to another pointer function. A pointer can be incremented/decremented, i.e., to point to the next/ previous memory location. The purpose of pointer is to save memory space and achieve faster execution time



A simple program for pointer illustration is given below:

```
#include <stdio.h>
int main()
{
   int a=10;    //variable declaration
   int *p;      //pointer variable declaration
   p=&a;        //store address of variable a in pointer p
   printf("Address stored in a variable p is:%x\n",p);  //accessing the address
   printf("Value stored in a variable p is:%d\n",*p);   //accessing the value
   return 0;
}
```

Output:

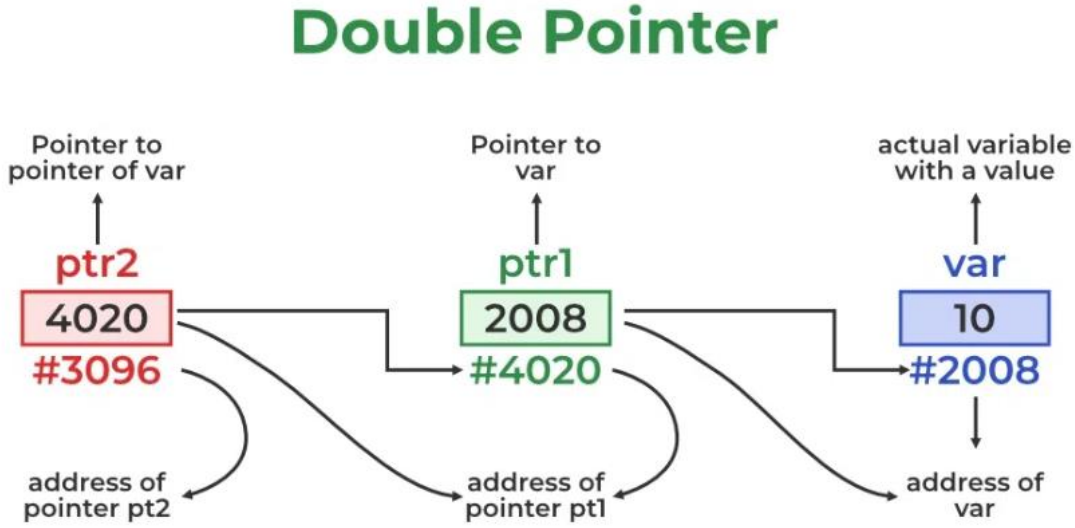
```
Address stored in a variable p is:60ff08
Value stored in a variable p is:10
```

Other types of pointers in ‘c’ are as follows:

- Dangling pointer
- Wild pointer
- Void pointer
- NULL pointer
- Complex pointer
- Near pointer
- Far pointer
- Huge pointer

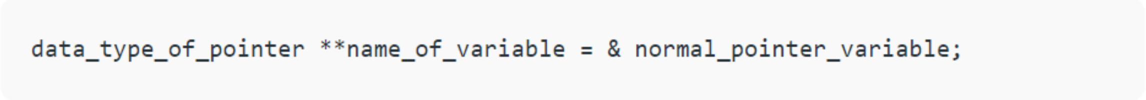
C – Pointer to Pointer (Double Pointer)

The pointer to a pointer in C is used when we want to store the address of another pointer. The first pointer is used to store the address of the variable. And the second pointer is used to store the address of the first pointer. That is why they are also known as ***double-pointers***. We can use a pointer to a pointer to change the values of normal pointers or create a variable-sized 2-D array. A double pointer occupies the same amount of space in the memory stack as a normal pointer.

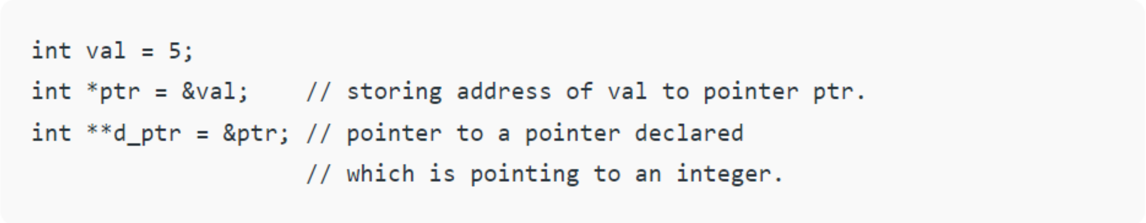


Declaration of Pointer to a Pointer in C

Declaring Pointer to Pointer is similar to declaring a pointer in C. The difference is we have to place an additional '*' before the name of the pointer.



```
data_type_of_pointer **name_of_variable = & normal_pointer_variable;
```

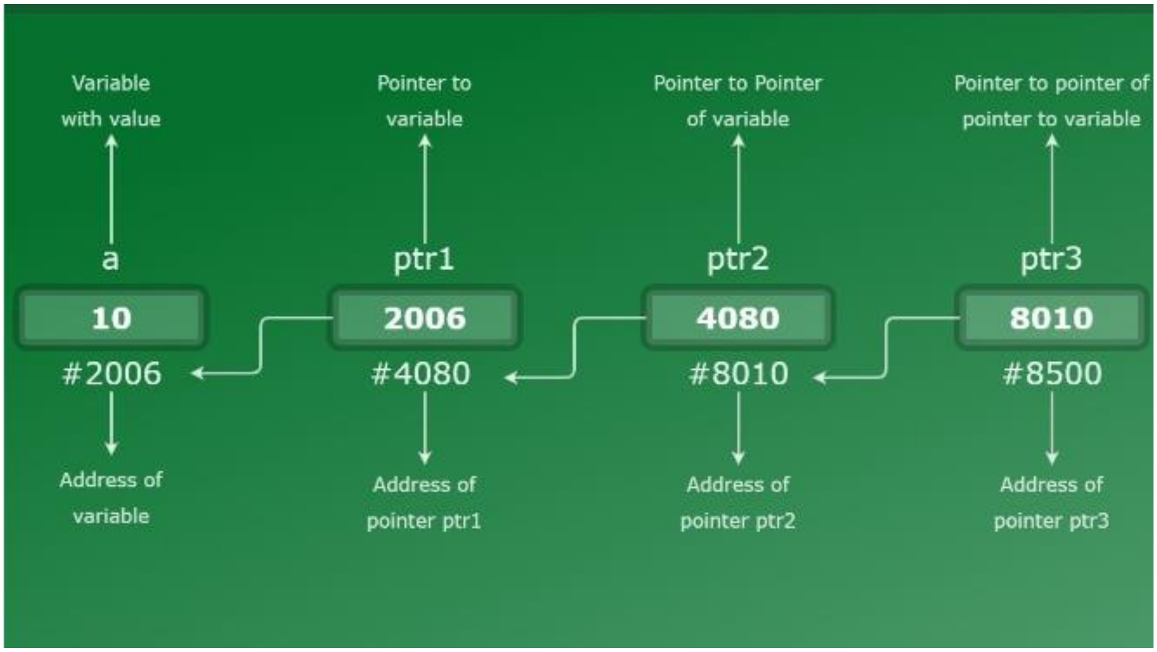


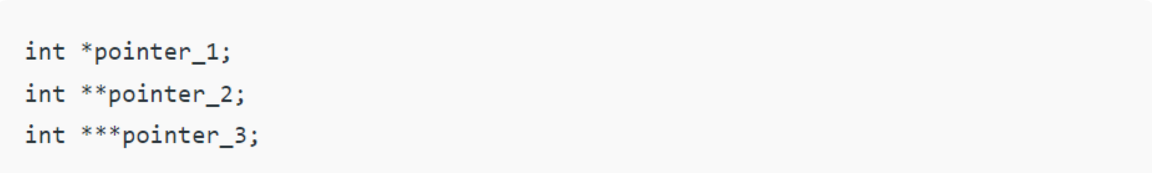
```
int val = 5;
int *ptr = &val;    // storing address of val to pointer ptr.
int **d_ptr = &ptr; // pointer to a pointer declared
                    // which is pointing to an integer.
```

Chain of Pointers

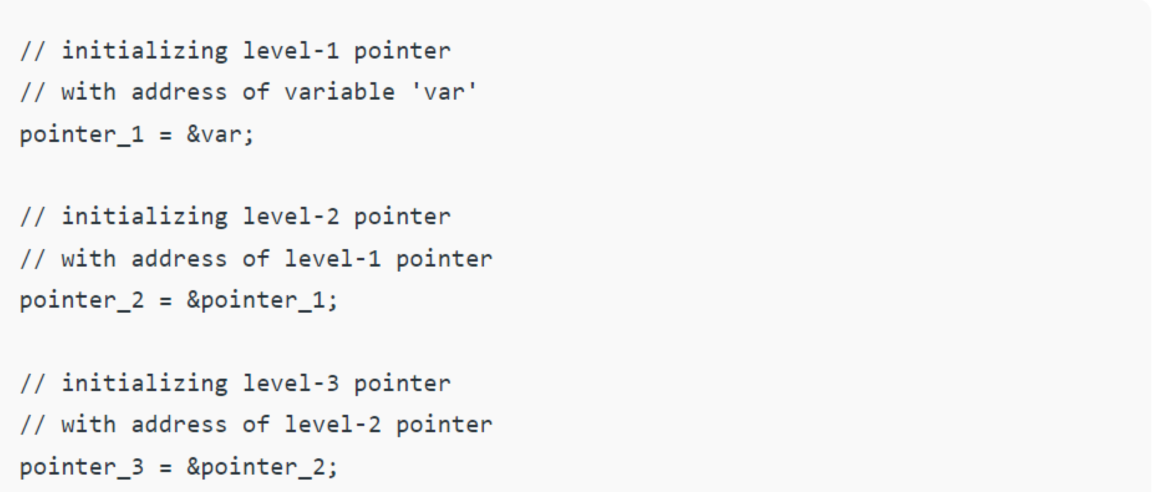
A pointer is used to point to a memory location of a variable. A pointer stores the address of a variable.

Similarly, a **chain of pointers** is when there are multiple levels of pointers. Simplifying, a pointer points to address of a variable, double-pointer points to a variable and so on. This is called **multiple indirections**.





```
int *pointer_1;
int **pointer_2;
int ***pointer_3;
```



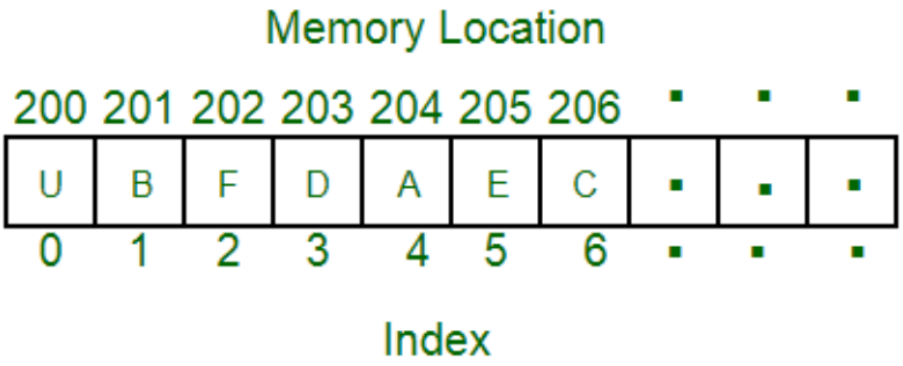
```
// initializing level-1 pointer
// with address of variable 'var'
pointer_1 = &var;

// initializing level-2 pointer
// with address of level-1 pointer
pointer_2 = &pointer_1;

// initializing level-3 pointer
// with address of level-2 pointer
pointer_3 = &pointer_2;
```

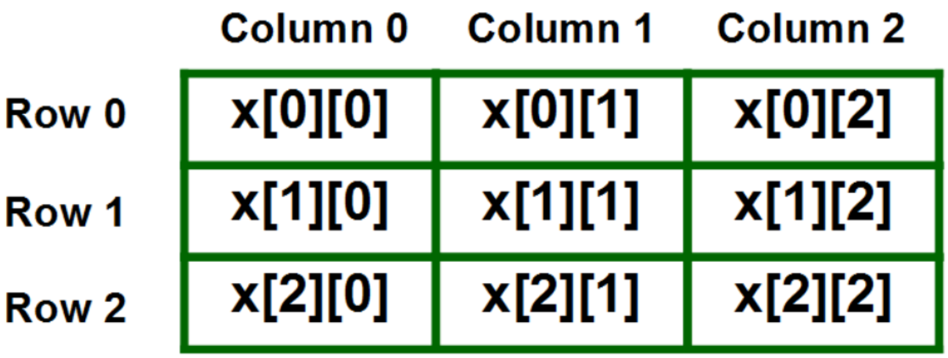
One Dimensional Array:

- It is a list of the variable of similar data types.
- It allows random access and all the elements can be accessed with the help of their index.
- The size of the array is fixed.
- For a dynamically sized array, vector can be used in C++.
- Representation of 1D array:



Two Dimensional Array:

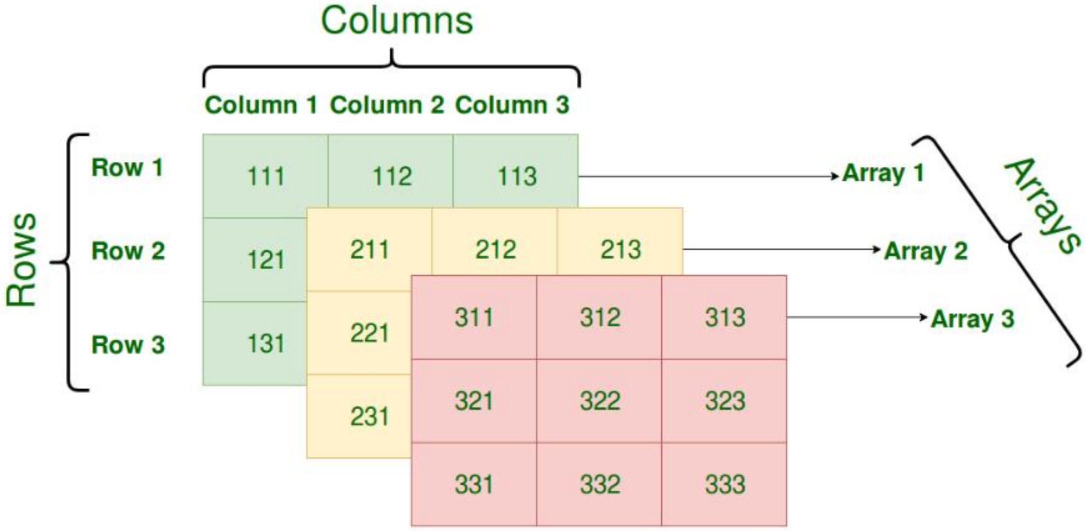
- It is a list of lists of the variable of the same data type.
- It also allows random access and all the elements can be accessed with the help of their index.
- It can also be seen as a collection of 1D arrays. It is also known as the Matrix.
- Its dimension can be increased from 2 to 3 and 4 so on.
- They all are referred to as a multi-dimension array.
- The most common multidimensional array is a 2D array.
- Representation of 2 D array:



	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Three-Dimensional Array in C

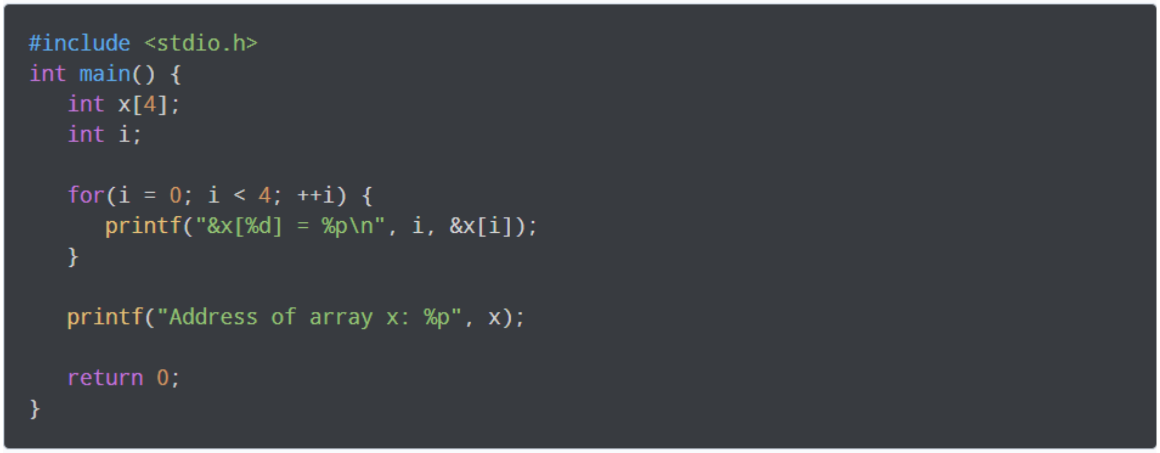
A **Three Dimensional Array** or **3D** array in C is a collection of two-dimensional arrays. It can be visualized as multiple 2D arrays stacked on top of each other.



Graphical Representation of Three-Dimensional Array of Size 3 x 3 x 3

Relationship Between Arrays and Pointers

An array is a block of sequential data. Let's write a program to print addresses of array elements.



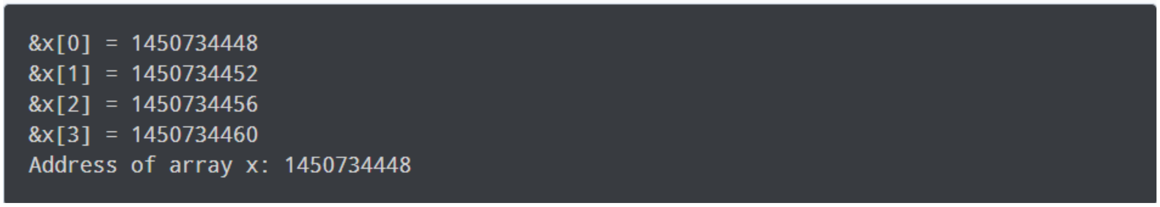
```
#include <stdio.h>
int main() {
   int x[4];
   int i;

   for(i = 0; i < 4; ++i) {
      printf("&x[%d] = %p\n", i, &x[i]);
   }

   printf("Address of array x: %p", x);

   return 0;
}
```

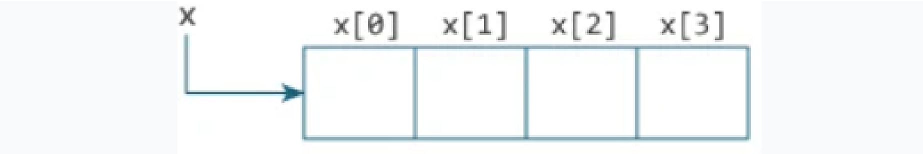
Output



```
&x[0] = 1450734448
&x[1] = 1450734452
&x[2] = 1450734456
&x[3] = 1450734460
Address of array x: 1450734448
```

There is a difference of 4 bytes between two consecutive elements of array x. It is because the size of int is 4 bytes (on our compiler).

Notice that, the address of &x[0] and x is the same. It's because the variable name x points to the first element of the array.



Relation between Arrays and Pointers

From the above example, it is clear that &x[0] is equivalent to x. And, x[0] is equivalent to *x.

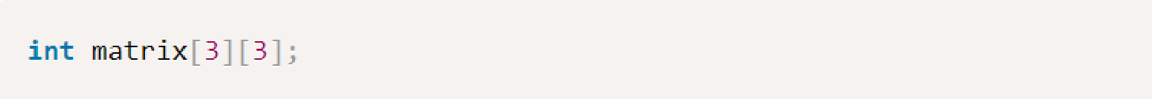
How to access two dimensional array using pointers?

To access a two dimensional array using pointer, let us recall basics from one dimensional array. Since it is just an array of one dimensional array.

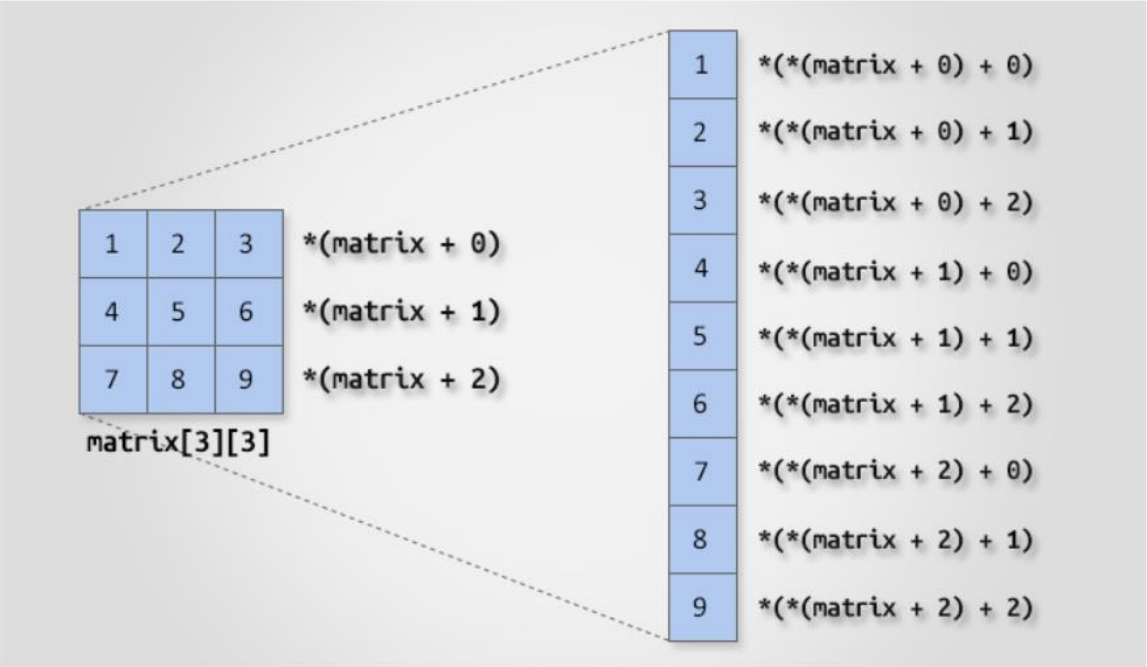
Suppose I have a pointer array_ptr pointing at base address of one dimensional array. To access nth element of array using pointer we use *(array_ptr + n) (where array_ptr points to 0th element of array, n is the nth element to access and nth element starts from 0).

Now we know two dimensional array is array of one dimensional array. Hence let us see how to access a two dimensional array through pointer.

Let us suppose a two-dimensional array

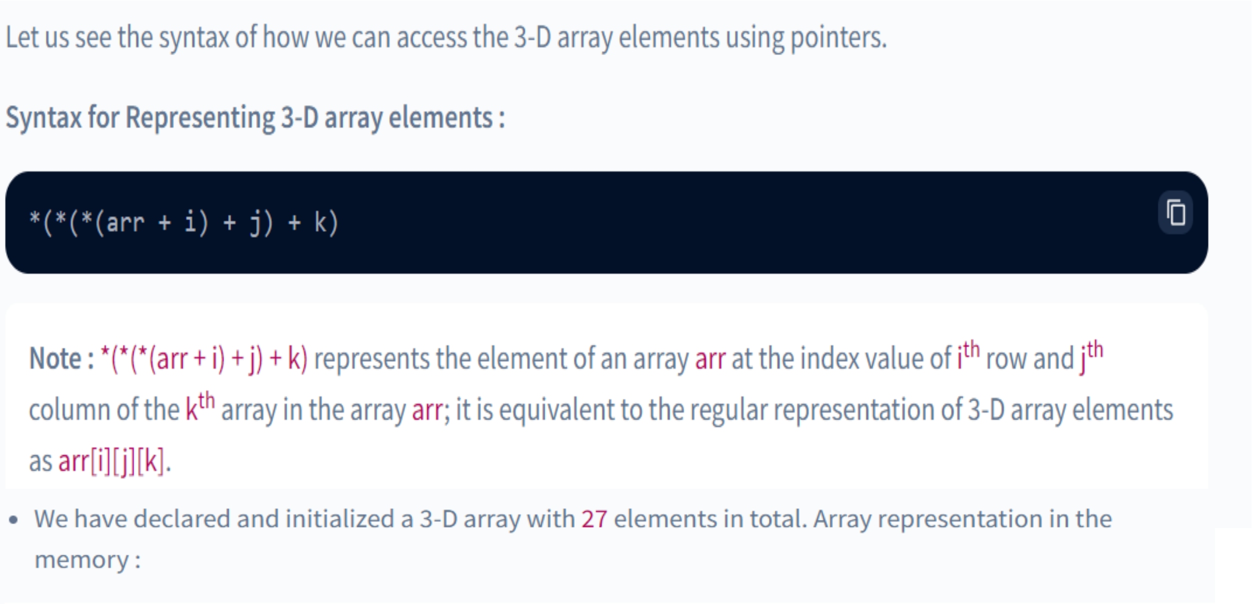


```
int matrix[3][3];
```



Two dimensional array access using pointer

How to access Three dimensional array using pointers?



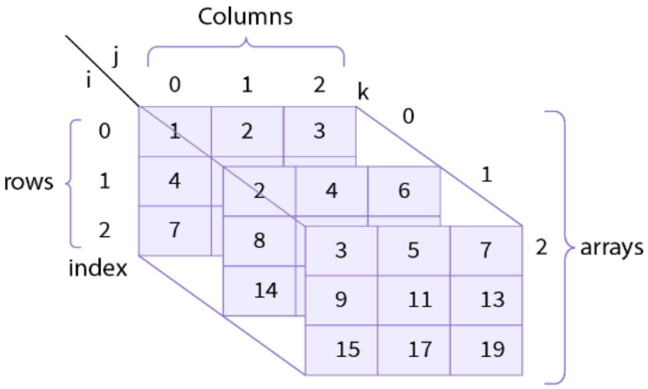
Let us see the syntax of how we can access the 3-D array elements using pointers.

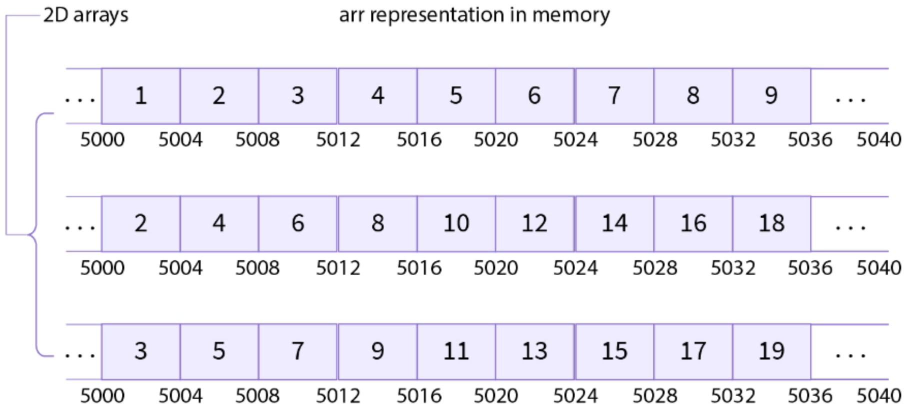
Syntax for Representing 3-D array elements :

```
*(*(*(arr + i) + j) + k)
```

Note : *(*(*(arr + i) + j) + k) represents the element of an array arr at the index value of ith row and jth column of the kth array in the array arr; it is equivalent to the regular representation of 3-D array elements as arr[i][j][k].

- We have declared and initialized a 3-D array with 27 elements in total. Array representation in the memory :





Thanks…