SNS COLLEGE OF TECHNOLOGY

An Autonomous Institution

Coimbatore – 35

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF AI & ML

PROGRAMMING FOR PROBLEM SOLVING

I YEAR - I SEM

UNIT IV – FUNCTIONS AND POINTERS

TOPIC 7 – POINTER ARRAYS

INTRODUCTION

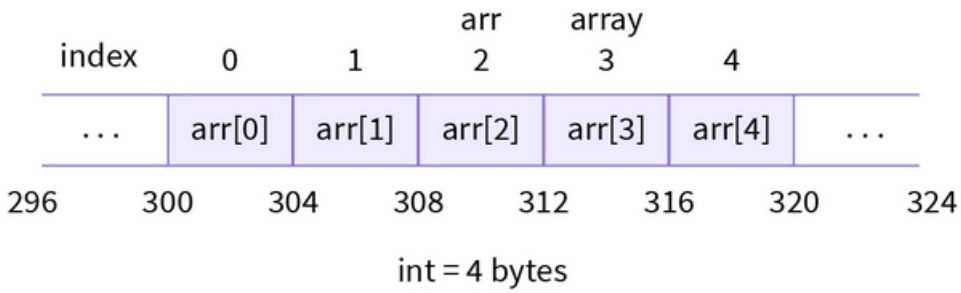
- Pointers and Array representations are very much related to each other and can be interchangeably used in the right context.
- Arrays can be single or multidimensional and are stored in contiguous memory blocks in our system, so it is easy for pointers to get associated with the arrays.
- An array name is generally treated as a pointer to the first element of the array and if we store the base address of the array in another pointer variable, then we can easily manipulate the array using pointer arithmetic in a C Program.

In C Language, we can declare an integer array using the below statement:

```
int arr[5];
```

The above statement will allocate 5 integer blocks and will occupy a memory of 20 Bytes in the system (5 * 4 = 20, 5 is size of the array and 4 bytes is the space occupied by an integer block, total = 20).

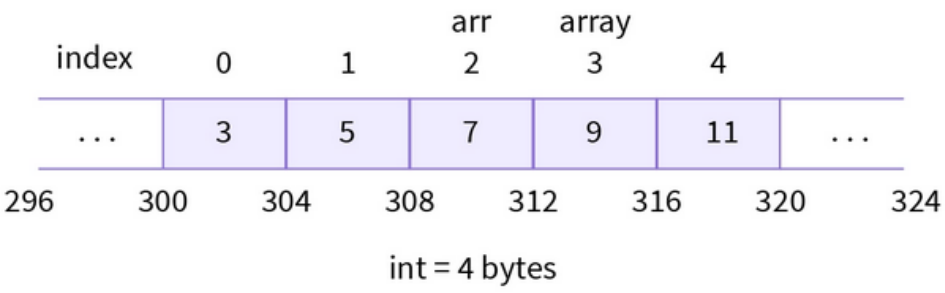
Below is the representation of how the array is stored in the system's memory. Let the base address allocated by the system to the array is **300**.



Note: All the consecutive array elements are at a distance of 4 bytes from each other as an int block occupies 4 bytes of memory in the system (64-bit Architecture). Also, each array element contains a garbage value because we have not initialized the array yet.

RELATIONSHIP BETWEEN POINTERS AND ARRAYS IN C

Let an array representation as shown below



With respect to the concept of the pointer, let us see some important points related to arrays in general:

- 'arr' serves two purposes here, first it is the name of the array and second, arr itself represents the base address of the array i.e. 300 in the above case, if we print the value in arr then it will print the address of the first element in the array.
- As the array name arr itself represents the base address of the array, then by default arr acts as a pointer to the first element of the array.
- arr is the same as &arr and &arr[0] in C Language.
- If we use dereferencing operator (*) on any of the above representations of array address, we will get the value of the very first element of the array.

```c
#include <stdio.h>

int main() {
  // array declaration and initialization
  int arr[5] = {3, 5, 7, 9, 11};

  // printing the addresses and values represented by arr, &arr and &arr[0]
  printf("arr : %u, Value : %d\n", arr, *arr);

  printf("&arr : %u, Value : %d\n", &arr, *(arr));

  printf("&arr[0] : %u, Value : %d\n", &arr[0], *( &arr[0]));

  return 0;
}
```

```
[Success] Your code was executed successfully
arr : 63744176, Value : 3
&arr : 63744176, Value : 3
&arr[0] : 63744176, Value : 3
```

Output address will be different at every run.

We can see that arr, &arr and &arr[0] are printing the same addresses and values in the output window. So, it is clear from the above program and output that arr, &arr and &arr[0] represent the same address in the system's memory.

Syntax Representing Array in Terms of Pointers in C

In a C Program, we denote array elements as arr[i], where i is the index value. Below is a similar syntax in terms of **pointers** of how we can represent the array elements using the dereferencing operator (*) on the array name i.e. using the pointers property of the array.

```
*(arr + i)
```

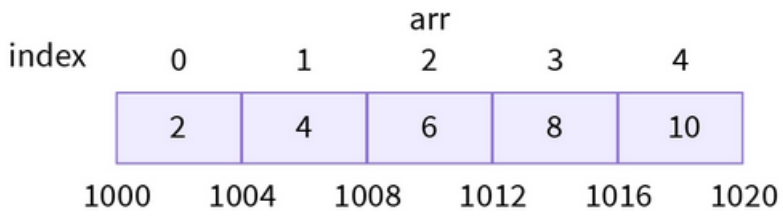
- * is a dereferencing operator used to extract the value from the address (arr + i).
- *(arr + i) is the same as arr[i] in a C Program.
- arr represents the array name and i represents the index value.

```c
int main()
{
  // array declaration and initialization
  int arr[5] = {2, 4, 6, 8, 10}, i;

  for(i = 0; i < 5; i++)
  {
        // printing the elements address and value at
        // arr[i] using *(arr + i) syntax
        printf("[index %d] Address : %u, Value : %d\n", i, (arr + i), *(arr + i));
  }

  return 0;
}
```

```
[Success] Your code was executed successfully

[index 0] Address : 2364420656, Value : 2
[index 1] Address : 2364420660, Value : 4
[index 2] Address : 2364420664, Value : 6
[index 3] Address : 2364420668, Value : 8
[index 4] Address : 2364420672, Value : 10
```

Explanation:

- We have declared and initialized an integer array arr, array representation:



- (arr + i) represents the address of the value at index i, so *(arr + i) will give the value at i^{th} index (address(arr + i) = address(arr[i])), it is used to print the addresses of the array elements as the value of i changes from 0-4.
- * is a dereferencing operator used for printing the value at the provided address. *(arr + i) will print the values of the array at consecutive addresses as the value of i changes from 0-4.

Note: From the above example we can conclude that, &arr[0] is equal to arr and arr[0] is equal to *arr. Similarly,

- &arr[1] is equal to (arr + 1) and arr[1] is equal to *(arr + 1).
- &arr[2] is equal to (arr + 2) and arr[2] is equal to *(arr + 2) and so on.
- ...
- Finally, we can write the above expressions in a fundamental form:
- &arr[i] is equal to (arr + i) and arr[i] is equal to *(arr + i).

Pointer to Array in C

In a pointer to an array, we just have to store the base address of the array in the pointer variable. We know in the arrays that the base address of an array can be represented in three forms,

```
*ptr = &arr;
```

```
*ptr = arr;
```

```
*ptr = &arr[0];
```

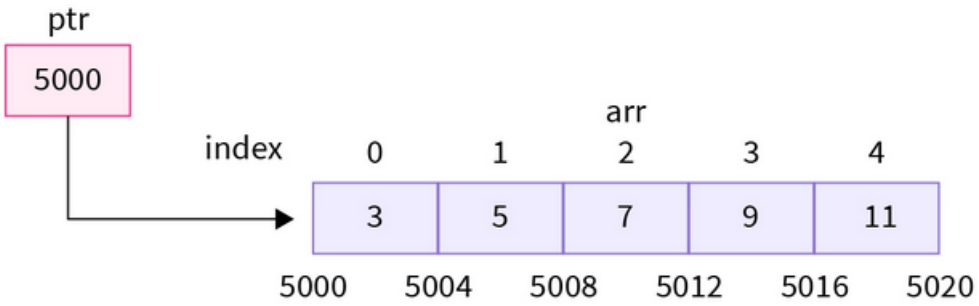
In all the above cases, ptr will store the base address of the array. Now, Let's see an example where we are printing array elements using a pointer to array. We will add consecutive integer values to the pointer ptr using a for loop and with the help of addition arithmetic we are going to print the array elements.

```c
#include <stdio.h>

int main()
{
        // array declaration and initialization
        int arr[5] = {3, 5, 7, 9, 11}, i;

        // both `arr` and `&arr` return the address of the first element of the
        int *ptr = arr;

        // printing the elements of array using addition arithmetic on pointer
        for(i = 0; i < 5; i++)
        {
                printf("%d ", *(ptr + i));
        }

        return 0;
}
```

```
[Success] Your code was executed successfully
3 5 7 9 11
```

Explanation:

- We have declared and initialized an integer array arr, array representation:



- (ptr + i) will give the address of the array elements as the value of i changes from 0-4 as address(ptr + i) = address(arr[i]).
- * is the dereferencing operator used for printing the value at the provided address. *(ptr + i) will print the values of the array as the value of i changes.

An arithmetic operation on a pointer means that we are modifying the *address value* of the pointer and not the value pointed by the pointer.

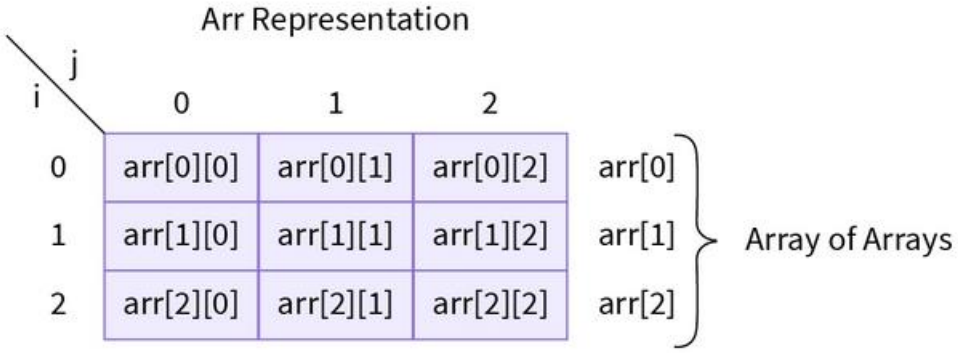
POINTER TO MULTIDIMENSIONAL ARRAYS IN C

Multi-dimensional arrays are defined as an array of arrays. 2-D arrays consist of 1-D arrays, while 3-D arrays consist of 2-D arrays as their elements.

POINTER TO 2D ARRAYS

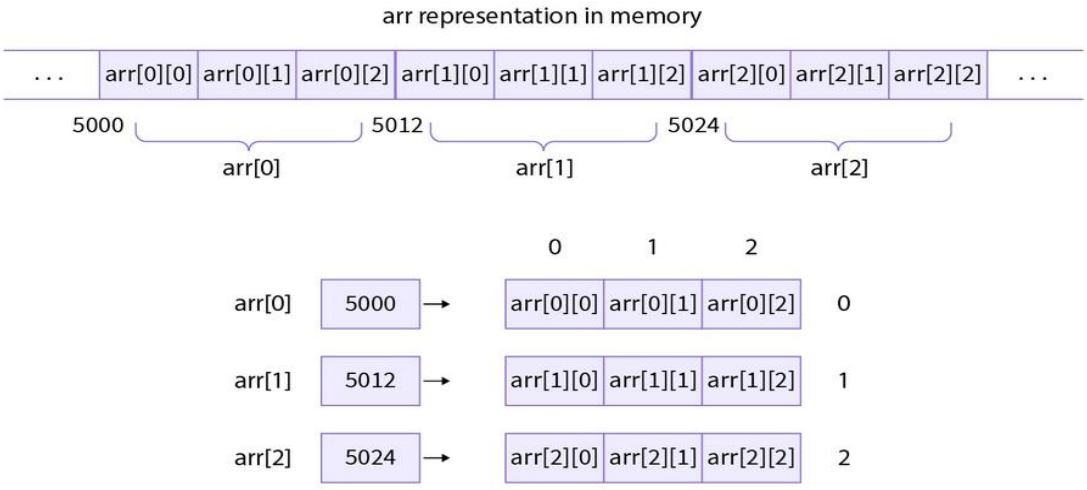
A 2-D array is an array of arrays, we can understand 2-D array as they are made up of n 1-D arrays stored in a linear manner in the memory. 2-D arrays can also be represented in a matrix form.

In the matrix form, there are rows and columns, so let's look at the representation of a 2-D array matrix below where i represents the row number and j represents the column number, arr is the array name.



Here, array contains 3 1-D arrays as its element, so array name arr acts as a pointer to the 1st 1-D array i.e. arr[0] and not to the first element of the array i.e. arr[0][0]. As we know our system's memory is organized in a sequential manner so it is not possible to store a 2-D array in rows and columns fashion, they are just used for the logical representation of 2-D arrays.

In the above representation, we have combined 3 1-D arrays that are stored in the memory to make a 2-D array, herearr[0],arr[1], arr[2] represents the base address of the respective arrays. So, arr[0], arr[1] and arr[2] act as a pointer to these arrays and we can access the 2-D arrays



Syntax for representing 2-D array elements:

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

((arr + i) + j) represents the element of an array arr at the index value of i^{th} row and j^{th} column; it is equivalent to the regular representation of 2-D array elements as arr[i][j].

POINTER TO 3D ARRAYS IN C

When the elements of an array are 2-D arrays, then the array formed is known as 3-Dimensional Array. 3-Dimensional arrays can also be known as array of matrices.

Syntax for Representing 3-D array elements:

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

ARRAY OF POINTERS IN C

As we know, arrays are collections of elements stored in contiguous memory locations. An array of pointers is similar to any other array in C Language. It is an array which contains numerous pointer variables and these pointer variables can store address values of some other variables having the same data type.

Syntax to declare a normal array

```
data_type (array_name)[sizeof_array];
```

Example:

```
int arr[10];
```

Syntax to declare a pointer array:

```
data_type (*array_name)[sizeof_array];
```

Example:

```
int *ptr[10];
```

We are using * operator to define that the ptr array is an array of pointers.

An application of an array of pointers is that it becomes easy to store strings in a char pointer array and it also reduces the memory consumption.

DIFFERENCE BETWEEN POINTER AND ARRAYS

	Pointer	array
1.	It is declared as -: ***var_name;**	It is declared as -: **data_type var_name[size];**
2.	It is used to store the address of different variables of the same data type.	It is used to store the multiple variable of same data type
3.	We can generate a pointer to the array.	We can generate a array of pointer
4.	It is designed to store the address of variable	It is designed to store the value of variable.
5.	A pointer variable can store the address of only one variable at a time.	A array can store the number of elements the same size as the size of the array variable.

CONCLUSION

- Array name generally acts as a **pointer to the array** and contains the starting address of the array.
- Array elements can be accessed and manipulated using a pointer containing the starting address of the array.
- Syntax for representation of 2-D arrays elements in terms of pointers is *(*(arr + i) + j) (arr[i][j]) and for 3-D arrays elements is *(*(*(arr + i) + j) + k) (arr[i][j][k]).
- Array of pointers are used to store multiple address values and are very useful in case of storing various string values.