- **What are the main differences between pointers and arrays?**

Apart from what we have discussed above, the key differences can be found while implementing the pointer and array. For example, when the arrays are implemented, the fixed size of the memory block is allocated. On the other hand, where the pointers are implemented, the memory is dynamically allocated. Thus, allocating the memory in both the pointers and arrays can be considered the key difference. However, it is not the only difference that lies between the arrays and pointer because some other differences also do exist that are as follows:

i. An array usually stores the variables ofsimilar data types, and the data types of the variables must match the type of array. However, the pointer variable stores the address of a variable of a type similar to a type of pointer variable type.
ii. We can generate an array of pointers, i.e. array whose variables are the pointer variables. On the other hand, we can also create a pointer that points to an array.
iii. In general, arrays are static, which means once the size of the array is declared, it cannot be resized according to users requirements. While on the other hand, pointers are dynamic, which means the memory allocated can be resized later at any point in time.
iv. Arrays are allocated at compile-time, while pointers are allocated at runtime
v. If we talk about the size of an array, it usually depends on the number of variables are stored in it. While in the case of the pointer variable, it stores the address of the variable only. To understand it more clearly, you can consider the following given examples:
vi. The "sizeof" operator

If we use the "sizeof(array)", it will return the amount of memory used by all elements stored in the array. However, if we use the it for the pointer, e.g. "sizeof(pointer)", it only returns the amount of memory used by the pointer variable itself.

- **Array of Pointers in C =**

When we want to point at multiple variables or memories of the same data type in a C program, we use an array of pointers.

Declaration of an Array of Pointers in C

An array of pointers can be declared just like we declare the arrays of char, float, int, etc. The syntax for declaring an array of pointers would be:

data_type *name_of_array [array_size];

Now, let us take a look at an example for the same,

int *ary[55]

This one is an array of a total of 55 pointers. In simple words, this array is capable of holding the addresses a total of 55 integer variables.

Example

Let us take a look at a program that demonstrates how one can use an array of pointers in C:

```
#include<stdio.h>

#define SIZE 10

int main()

{

int *arr[3];

int p = 40, q = 60, r = 90, i;

arr[0] = &p;

arr[1] = &q;

arr[2] = &r;

for(i = 0; i < 3; i++)

{

printf("For the Address = %d\t the Value would be = %d\n", arr[i], *arr[i]);

}

return 0;

}
```

The output generated out of the program mentioned above would be like this:

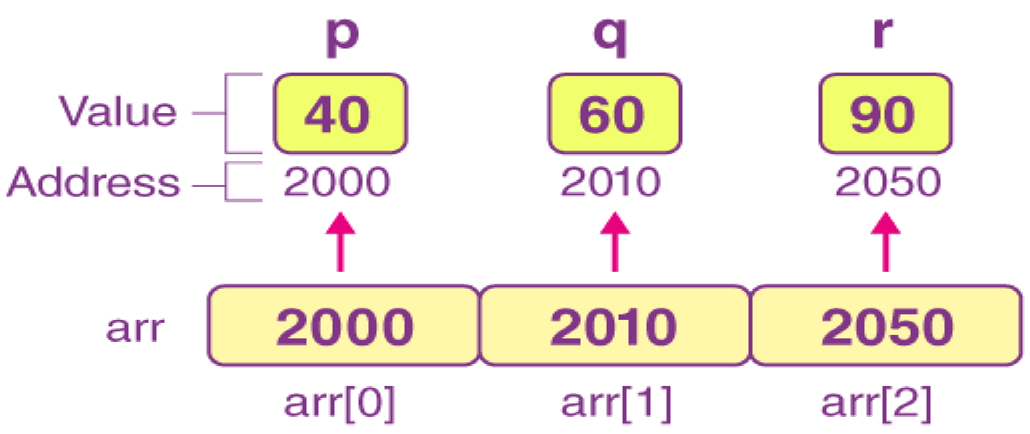
```
For the Address = 387130656 the Value would be = 40

For the Address = 387130660 the Value would be = 60

For the Address = 387130664 the Value would be = 90
```

How Does it Work?

Take a look at how we assigned the addresses of the pointers p, q, and r. We first assign the p variable's address to the 0th element present in the array. In a similar manner, the 1st and 2nd elements of the array will be assigned with the addresses of q and r. At this very point, the array, arr would look like this:



The arr[i] provides the address of the array's ith element. Thus, the arr[0] would return the address of the variable p, the arr[1] would return the address of the pointer q, and so on. We use the * indirection operator to get the value present at the address. Thus, it would be like this:

*arr[i]

Thus, *arr[0] would generate the value at the arr[0] address. In a similar manner, the *arr[1] would generate the value at the arr[1] address, and so on.

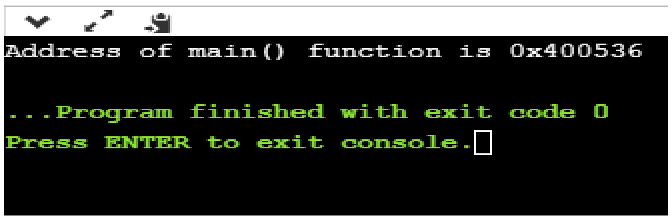
- **C Function Pointer**

we can create a pointer of any data type such as int, char, float, we can also create a pointer pointing to a function. The code of a function always resides in memory, which means that the function has some address. We can get the address of memory by using the function pointer.

Let's see a simple example.

```
#include <stdio.h>
int main()
{
   printf("Address of main() function is %p",main);
   return 0;
}
```

The above code prints the address of **main()** function.



```
Address of main() function is 0x400536

...Program finished with exit code 0
Press ENTER to exit console.
```

Declaration of a function pointer

Till now, we have seen that the functions have addresses, so we can create pointers that can contain these addresses, and hence can point them.

Syntax of function pointer

```
1. return type (*ptr_name)(type1, type2…);
```

For example:

```
1. int (*ip) (int);
```

In the above declaration, *ip is a pointer that points to a function which returns an int value and accepts an integer value as an argument.

Calling a function through a function pointer

We already know how to call a function in the usual way. Now, we will see how to call a function using a function pointer.

Suppose we declare a function as given below:

```
1. float func(int , int);      // Declaration of a function.
```

Calling an above function using a usual way is given below:

```
1. result = func(a , b);     // Calling a function using usual ways.
```

Calling a function using a function pointer is given below:

```
1. result = (*fp)( a , b);    // Calling a function using function pointer.
```

Or

```
1. result = fp(a , b);         // Calling a function using function pointer, and indirection
```

Let's understand the function pointer through an example.

```
#include <stdio.h>

int add(int,int);

int main()

{

  int a,b;
```

```
   int (*ip)(int,int);
   int result;
   printf("Enter the values of a and b : ");
   scanf("%d %d",&a,&b);
   ip=add;
   result=(*ip)(a,b);
   printf("Value after addition is : %d",result);
    return 0;
}
int add(int a,int b)
{
    int c=a+b;
    return c;
}
```