C++ Pointers and Arrays

programiz.com/cpp-programming/pointers-arrays

Join our newsletter for the latest updates.

In this tutorial, we will learn about the relation between arrays and pointers with the help of examples.

In C++, Pointers are variables that hold addresses of other variables. Not only can a pointer store the address of a single variable, it can also store the address of cells of an array.

Consider this example:

```
int *ptr;
int arr[5];

// store the address of the first
// element of arr in ptr
ptr = arr;
```

Here, *ptr* is a pointer variable while *arr* is an `int` array. The code `ptr = arr;` stores the address of the first element of the array in variable *ptr*.

Notice that we have used `arr` instead of `&arr[0]`. This is because both are the same. So, the code below is the same as the code above.

```
int *ptr;
int arr[5];
ptr = &arr[0];
```

The addresses for the rest of the array elements are given by `&arr[1]`, `&arr[2]`, `&arr[3]`, and `&arr[4]`.

Point to Every Array Elements

Suppose we need to point to the fourth element of the array using the same pointer *ptr*.

Here, if *ptr* points to the first element in the above example then `ptr + 3` will point to the fourth element. For example,

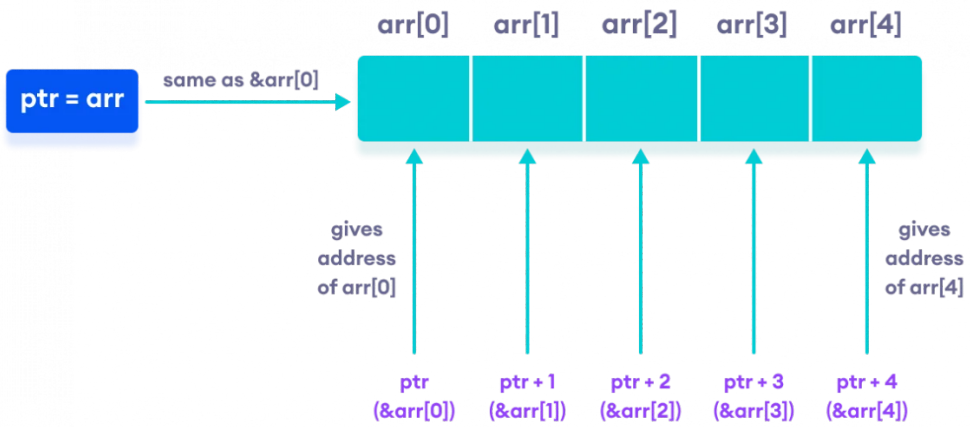
```
int *ptr;
int arr[5];
ptr = arr;

ptr + 1 is equivalent to &arr[1];
ptr + 2 is equivalent to &arr[2];
ptr + 3 is equivalent to &arr[3];
ptr + 4 is equivalent to &arr[4];
```

Similarly, we can access the elements using the single pointer. For example,

```
// use dereference operator
*ptr == arr[0];
*(ptr + 1) is equivalent to arr[1];
*(ptr + 2) is equivalent to arr[2];
*(ptr + 3) is equivalent to arr[3];
*(ptr + 4) is equivalent to arr[4];
```

Suppose if we have initialized `ptr = &arr[2];` then

```
ptr - 2 is equivalent to &arr[0];
ptr - 1 is equivalent to &arr[1];
ptr + 1 is equivalent to &arr[3];
ptr + 2 is equivalent to &arr[4];
```



Working of C++ Pointers with Arrays

Note: The address between *ptr* and *ptr + 1* differs by 4 bytes. It is because *ptr* is a pointer to an `int` data. And, the size of int is 4 bytes in a 64-bit operating system.

Similarly, if pointer *ptr* is pointing to `char` type data, then the address between *ptr* and *ptr + 1* is 1 byte. It is because the size of a character is 1 byte.

Example 1: C++ Pointers and Arrays

```
// C++ Program to display address of each element of an array

#include <iostream>
using namespace std;

int main()
{
    float arr[3];

    // declare pointer variable
    float *ptr;

    cout << "Displaying address using arrays: " << endl;

    // use for loop to print addresses of all array elements
    for (int i = 0; i < 3; ++i)
    {
        cout << "&arr[" << i << "] = " << &arr[i] << endl;
    }

    // ptr = &arr[0]
    ptr = arr;

    cout<<"\nDisplaying address using pointers: "<< endl;

    // use for loop to print addresses of all array elements
    // using pointer notation
    for (int i = 0; i < 3; ++i)
    {
        cout << "ptr + " << i << " = "<< ptr + i << endl;
    }

    return 0;
}
```

Output

```
Displaying address using arrays:
&arr[0] = 0x61fef0
&arr[1] = 0x61fef4
&arr[2] = 0x61fef8

Displaying address using pointers:
ptr + 0 = 0x61fef0
ptr + 1 = 0x61fef4
ptr + 2 = 0x61fef8
```

In the above program, we first simply printed the addresses of the array elements without using the pointer variable *ptr*.

Then, we used the pointer *ptr* to point to the address of *a[0]*, `ptr + 1` to point to the address of *a[1]*, and so on.

In most contexts, array names decay to pointers. In simple words, array names are converted to pointers. That's the reason why we can use pointers to access elements of arrays.

However, we should remember that pointers and arrays are not the same.

There are a few cases where array names don't decay to pointers. To learn more, visit: When does array name doesn't decay into a pointer?

Example 2: Array name used as pointer

```
// C++ Program to insert and display data entered by using pointer notation.

#include <iostream>
using namespace std;

int main() {
    float arr[5];

   // Insert data using pointer notation
    cout << "Enter 5 numbers: ";
    for (int i = 0; i < 5; ++i) {

        // store input number in arr[i]
        cin >> *(arr + i) ;

    }

    // Display data using pointer notation
    cout << "Displaying data: " << endl;
    for (int i = 0; i < 5; ++i) {

        // display value of arr[i]
        cout << *(arr + i) << endl ;

    }

    return 0;
}
```

Output

```
Enter 5 numbers: 2.5
3.5
4.5
5
2
Displaying data:
2.5
3.5
4.5
5
2
```

Here,

1. We first used the pointer notation to store the numbers entered by the user into the array *arr*.

   ```
   cin >> *(arr + i) ;
   ```

   This code is equivalent to the code below:

   ```
   cin >> arr[i];
   ```

   Notice that we haven't declared a separate pointer variable, but rather we are using the array name *arr* for the pointer notation.

   As we already know, the array name *arr* points to the first element of the array. So, we can think of *arr* as acting like a pointer.

2. Similarly, we then used `for` loop to display the values of *arr* using pointer notation.

   ```
   cout << *(arr + i) << endl ;
   ```

   This code is equivalent to

   ```
   cout << arr[i] << endl ;
   ```