# Relationship Between Arrays and Pointers in C programming

While Arrays and Pointers are distinct entities, they often interact in ways that can make them seem closely related. This post will delve into the connections between arrays and pointers, illustrating their relationship with examples.

## Arrays in C

An array in C is a collection of elements of the same type, stored in contiguous memory locations. You can declare an array as follows:

```
int arr[5] = {1, 2, 3, 4, 5};
```

Here, `arr` is an array of five integers. The elements can be accessed using an index, with `arr[0]` referring to the first element, `arr[1]` to the second, and so on.

## Pointers in C

A pointer is a variable that stores the memory address of another variable. You can declare a pointer to an integer as follows:

```
int *ptr;
int value = 10;
ptr = &value; // ptr now holds the address of value
```

Here, `ptr` is a pointer to an integer, and it holds the address of the variable `value`.

## Relationship Between Arrays and Pointers

The relationship between arrays and pointers in C can be summarized as follows:

1. The name of an array acts as a pointer to its first element.
2. You can use pointer arithmetic to traverse an array.
3. Pointers can be used to dynamically allocate arrays.

Same is illustrated below:

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
    int arr[5] = {1, 2, 3, 4, 5};
    int *ptr = arr; // ptr now points to the first element of arr

    printf("%d\n", *ptr); // Outputs: 1
    printf("%d\n", *(ptr + 1)); // Outputs: 2
    printf("%d\n", *(ptr + 2)); // Outputs: 3

    // Pointer Arithmetic
    for (int i = 0; i < 5; i++) {
        printf("%d ", *(ptr + i)); // Outputs: 1 2 3 4 5
    }

    printf("\n"); // new line

    // Dynamic Arrays with Pointers
    int *arr1 = (int *)malloc(5 * sizeof(int));
    for (int i = 0; i < 5; i++) {
        arr1[i] = i + 1; // Initialize the array
    }

    for (int i = 0; i < 5; i++) {
        printf("%d ", arr1[i]); // Outputs: 1 2 3 4 5
    }

    free(arr1); // Don't forget to free the allocated memory

}
```

Output:

```
1
2
3
1 2 3 4 5
1 2 3 4 5
```

### Array Name as a Pointer

When you use the name of an array in an expression, it is converted to a pointer to the first element of the array. For example:

```
int arr[5] = {1, 2, 3, 4, 5};
int *ptr = arr; // ptr now points to the first element of arr
```

Here, `ptr` points to `arr[0]`. You can access array elements using this pointer:

```
printf("%d\n", *ptr); // Outputs: 1
printf("%d\n", *(ptr + 1)); // Outputs: 2
printf("%d\n", *(ptr + 2)); // Outputs: 3
```

This shows that `arr[i]` is equivalent to `*(arr + i)`.

## Pointer Arithmetic

Pointer arithmetic allows you to traverse an array using a pointer. Consider the following example:

```
int arr[5] = {1, 2, 3, 4, 5};
int *ptr = arr;

for (int i = 0; i < 5; i++) {
    printf("%d ", *(ptr + i)); // Outputs: 1 2 3 4 5
}
```

Here, `*(ptr + i)` accesses the `i-th` element of the array. This demonstrates how pointers can be used to iterate over an array.

## Dynamic Arrays with Pointers

Pointers are essential for dynamically allocating arrays. The `malloc` function from the standard library allocates memory dynamically:

```
int *arr1 = (int *)malloc(5 * sizeof(int));
for (int i = 0; i < 5; i++) {
    arr1[i] = i + 1; // Initialize the array
}

for (int i = 0; i < 5; i++) {
    printf("%d ", arr1[i]); // Outputs: 1 2 3 4 5
}

free(arr1); // Don't forget to free the allocated memory
```

In this example, `malloc` allocates memory for an array of five integers, and `arr` acts as a pointer to the first element of this dynamically allocated array.

We will dive in to dynamic memory allocation in future post.

**In summary, the key points are:**

- The array name acts as a pointer to its first element.
- Pointer arithmetic can be used to traverse arrays.
- Pointers enable dynamic memory allocation for arrays.