Array Parameters and Pointers

In C++, you can pass arrays as function parameters to modify the elements of an array in a function.

Here is an example of how you can pass an array as a function parameter in C++:

```
#include <iostream>

void increment(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        arr[i]++;  // increment the value stored at each elem
    }
}

int main()
{
    int arr[] = { 1, 2, 3, 4, 5 };
    int size = sizeof(arr) / sizeof(arr[0]);

    increment(arr, size);  // pass the array and the size of

    for (int i = 0; i < size; i++)
    {
        std::cout << arr[i] << " ";
    }

    return 0;
}
```

Output

```
2 3 4 5 6
```

the function increment takes an int array and an int as parameters, and modifies the values stored at each element of the array. This has the effect of modifying the elements of arr in the main function, even though arr is not returned by the increment function.

Passing arrays as function parameters can be a useful tool in C++, as it allows you to modify the elements of an array in a function without returning the array itself. However, it's important to be careful when working with arrays, as they can be tricky to work with and can lead to problems if used incorrectly.

Here are a few tips for working with arrays as function parameters safely in C++:

- Always initialize arrays before using them to avoid undefined behavior.
- Check for out-of-bounds accesses to avoid reading or writing to memory locations that are not part of the array.
- Use array bounds checking tools, such as AddressSanitizer, to detect and debug array-related errors, such as out-of-bounds accesses and buffer overflows.

By following these guidelines, you can help ensure that your C++ programs are safe and stable when using arrays as function parameters.