

Bài 44: CON TRỎ VÀ MẢNG TRONG C++ (POINTERS AND ARRAYS)

Xem bài học trên website để ủng hộ Kteam: [Con trỏ và mảng trong C++ \(Pointers and arrays\)](#)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage [How Kteam](#) nhé!

Dẫn nhập

Ở bài học trước, mình đã chia sẻ cho các bạn về [CON TRỎ NULL TRONG C++ \(NULL pointers\)](#). Lưu ý rằng ta nên khởi tạo con trỏ là null nếu nó chưa trỏ đến một địa chỉ cụ thể nào khác nhé.

Hôm nay, chúng ta sẽ cùng tìm hiểu về sự liên quan giữa **Con trỏ và mảng trong C++ (Pointers and arrays)**.

Nội dung

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về:

- [BIẾN TRONG C++](#)
- [CON TRỎ TRONG C++](#)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Sự tương đồng giữa con trỏ và mảng trong C++
- Sự khác nhau giữa con trỏ và mảng trong C++
- Xem lại việc truyền mảng vào hàm (passing arrays to functions)
- Cơ bản về Truyền địa chỉ cho hàm (pass by address)

Sự tương đồng giữa con trỏ và mảng trong C++

Trong bài [MẢNG MỘT CHIỀU TRONG C++ \(Arrays\)](#), bạn đã biết cách khai báo và khởi tạo giá trị cho mảng 1 chiều:

```
int array[4] = { 5, 8, 2, 7 }; // mảng tĩnh 4 phần tử
```

Hiện tại, bạn đã biết biến **array** là 1 mảng 4 số nguyên, giá trị của mảng trên lần lượt là:

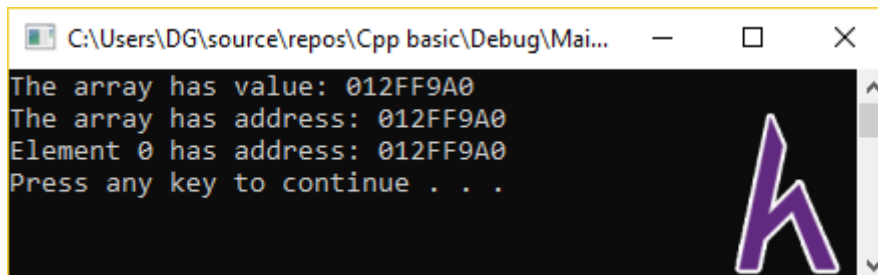
```
array[0] = 5  
array[1] = 8  
array[2] = 2  
array[3] = 7
```

Nhưng bản thân biến **array** đang chứa giá trị gì? Để trả lời câu hỏi này, chúng ta cùng nhìn vào ví dụ bên dưới:

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    int array[4] = { 5, 8, 2, 7 };  
  
    // in giá trị của biến array  
    cout << "The array has value: " << array << '\n';  
  
    // in địa chỉ của biến array  
    cout << "The array has address: " << &array << '\n';  
  
    // in địa chỉ của phần tử array[0]  
    cout << "Element 0 has address: " << &array[0] << '\n';  
  
    system("pause");  
}
```

```
    return 0;
}
```

Output:



```
C:\Users\DG\source\repos\Cpp basic\Debug\Mai...
The array has value: 012FF9A0
The array has address: 012FF9A0
Element 0 has address: 012FF9A0
Press any key to continue . . .
```

Trong chương trình trên, ta thấy **giá trị** và **địa chỉ** của biến array tương đương với **địa chỉ của phần tử array[0]**. Nó giống như cách hoạt động của 1 con trỏ.

Chú ý: Bản thân mảng chứa địa chỉ của phần tử đầu tiên của mảng, mảng có thể được thao tác **GIỐNG** như 1 con trỏ (nhưng mảng và con trỏ **KHÔNG** hoàn toàn như nhau).

Trong hầu hết các trường hợp, mảng và con trỏ có thể xử lý giống nhau:

```
#include <iostream>
using namespace std;

int main()
{
    int array[4] = { 5, 8, 2, 7 };

    // toán tử trỏ (*) trả về giá trị phần tử đầu tiên: array[0]
    cout << *array << '\n'; // 5

    // khai báo con trỏ ptr trỏ vào mảng array
    int *ptr = array;
    cout << *(ptr) << '\n'; // 5

    char name[] = "Kteam"; // C-style string (mảng char)
    cout << *name << '\n'; // K

    system("pause");
    return 0;
}
```

Trong ví dụ trên, ta sử dụng **toán tử trả (*)** để lấy giá trị phần tử đầu tiên của mảng, hoặc sử dụng con trỏ **ptr** trỏ vào mảng **array** và thao tác với mảng **array** bằng con trỏ **ptr**.

Sự khác nhau giữa con trỏ và mảng trong C++

Chú ý: Một sai lầm phổ biến trong C++ là tin rằng mảng và con trỏ đến mảng là giống hệt nhau. Mặc dù chúng đều trỏ đến phần tử đầu tiên của mảng, nhưng chúng là 2 kiểu khác nhau.

```
#include <iostream>
using namespace std;

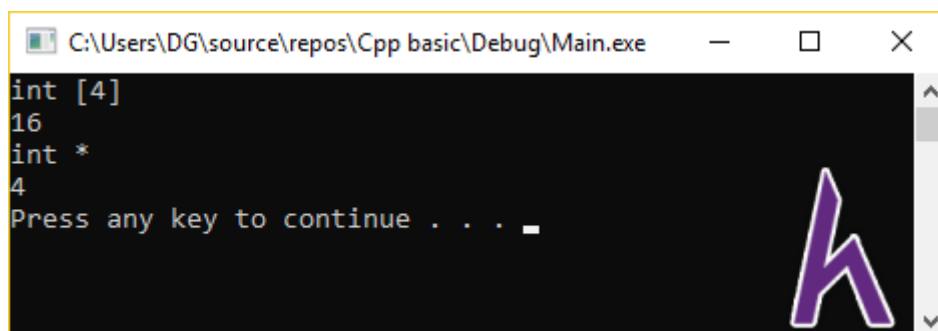
int main()
{
    int array[4] = { 5, 8, 2, 7 };

    cout << typeid(array).name() << '\n';
    cout << sizeof(array) << '\n';

    int *ptr = array;
    cout << typeid(ptr).name() << '\n';
    cout << sizeof(ptr) << '\n';

    system("pause");
    return 0;
}
```

Output:



```
C:\Users\DG\source\repos\Cpp basic\Debug\Main.exe
int [4]
16
int *
4
Press any key to continue . . .
```

Trong trường hợp trên, mảng là kiểu **"int [4]"** kích thước 16 byte, trong khi một con trỏ tới mảng sẽ thuộc kiểu **"int *"** kích thước 4 byte.

Vì vậy, sử dụng mảng một chiều có thể biết được chính xác số lượng phần tử của mảng, trong khi con trỏ không làm được điều này.

Xem lại việc truyền mảng vào hàm (passing arrays to functions)

Trong bài [MẢNG MỘT CHIỀU TRONG C++ \(Arrays\)](#), bạn đã biết việc sao chép 1 số lượng lớn các phần tử sẽ gây **tốn rất nhiều vùng nhớ** và **giảm hiệu suất**.

Vì vậy, khi truyền một mảng cho một hàm, mảng sẽ được chuyển đổi ngầm định thành một con trỏ tới mảng, và con trỏ được truyền vào hàm:

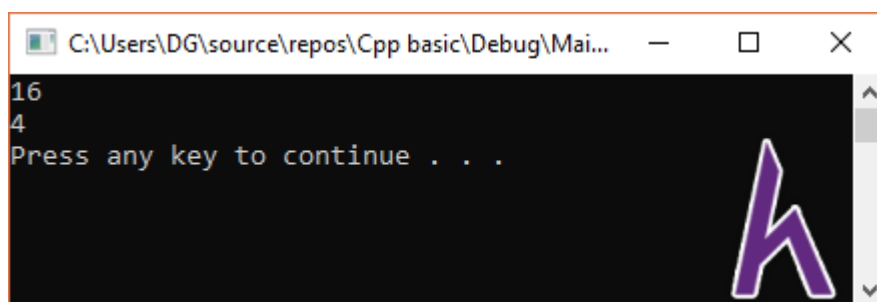
```
#include <iostream>
using namespace std;

void printSize(int *array)
{
    // tham số array là con trỏ int*
    cout << sizeof(array) << '\n'; // kích thước con trỏ int*, không phải kích
thước mảng
}

int main()
{
    int array[] = { 5, 8, 2, 7 };
    cout << sizeof(array) << '\n'; // kích thước mảng: sizeof(int) * array length
}
```

```
printSize(array); // đổi số array được chuyển thành con trỏ int* tại đây

system("pause");
return 0;
}
```

Output:

Lưu ý rằng điều này xảy ra ngay cả khi tham số được khai báo là một mảng cố định:

```
void printSize(int a[100]);

void printSize(int a[]);
```

Chú ý: Để tránh nhầm lẫn về việc đang thao tác với mảng hay con trỏ, ưu tiên sử dụng cú pháp con trỏ (*) cho tham số hàm là mảng.

Cơ bản về Truyền địa chỉ cho hàm (pass by address)

Mảng được chuyển đổi ngầm định thành con trỏ khi được chuyển đến một hàm. Đó là lý do tại sao thay đổi mảng trong hàm sẽ thay đổi mảng thực tế được truyền vào.

```
#include <iostream>
using namespace std;
```

```
// tham số ptr đang chứa địa chỉ mảng array
void changeArray(int *ptr)
{
    // thay đổi mảng bên trong hàm đồng nghĩa mảng bên ngoài cũng thay
    đổi
    *ptr = 1;
    ptr[1] = 3;
}

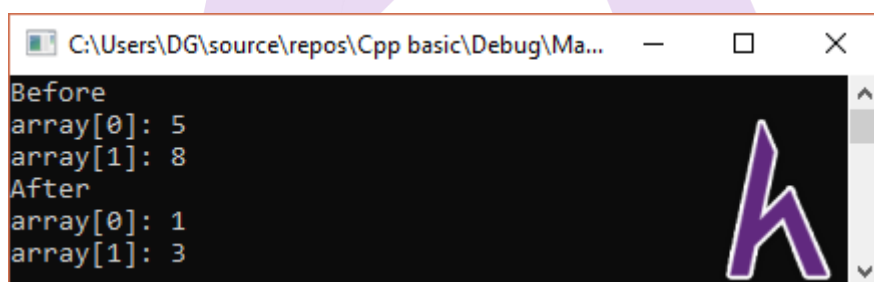
int main()
{
    int array[] = { 5, 8, 2, 7 };
    cout << "Before" << '\n';
    cout << "array[0]: " << array[0] << '\n';
    cout << "array[1]: " << array[1] << '\n';

    changeArray(array);

    cout << "After" << '\n';
    cout << "array[0]: " << array[0] << '\n';
    cout << "array[1]: " << array[1] << '\n';

    system("pause");
    return 0;
}
```

Output:



Khi hàm **changeArray()** được gọi, mảng chuyển đổi ngầm thành một con trỏ, và **giá trị của con trỏ đó (địa chỉ phần tử đầu tiên của mảng)** được sao chép vào tham số **ptr** của hàm **changeArray()**.

Mặc dù giá trị trong **ptr** là một bản sao địa chỉ của mảng, nhưng **ptr** vẫn trỏ vào mảng thực tế (không phải là bản sao). Do đó, mọi thao tác thay đổi trên vùng nhớ mà con trỏ trỏ đến, sẽ làm thay đổi mảng bên ngoài.

Kết luận

Qua bài học này, bạn đã nắm được sự liên quan giữa Con trỏ và mảng trong C++ (Pointers and arrays).

Trong bài tiếp theo, mình sẽ giới thiệu cho các bạn về CON TRỎ SỐ HỌC & LẬP CHI MỤC MẢNG TRONG C++ (Pointers and arrays).

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên **"Luyện tập – Thử thách – Không ngại khó"**.