

Bài 45: CÁC PHÉP TOÁN TRÊN CON TRỎ VÀ CHỈ MỤC MẢNG TRONG C++ (POINTERS AND ARRAYS)

Xem bài học trên website để ủng hộ Kteam: [Các phép toán trên Con trỏ và Chỉ mục 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 sự liên quan giữa [CON TRỎ & MẢNG TRONG C++ \(Pointers and arrays\)](#). Đó là những kiến thức khá quan trọng mà bạn cần nắm trong C++.

Hôm nay, chúng ta sẽ cùng tìm hiểu về **Các phép toán trên Con trỏ và Chỉ mục mảng trong C++**, cụ thể hơn, nó là các toán tử và thao tác dùng cho con trỏ và mảng.

Nội dung

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

- [BIẾN TRONG C++](#) (Variables)
- [MẢNG MỘT CHIỀU TRONG C++](#) (Arrays)
- [CON TRỎ TRONG C++](#) (Pointer)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Phép toán số học trên Con trỏ (Pointer arithmetic)
- Mảng tĩnh được lưu trong bộ nhớ như thế nào?
- Ứng dụng giữa Con trỏ số học, Mảng, và Chỉ mục mảng
- Sử dụng con trỏ lặp qua một mảng

Phép toán số học trên Con trỏ (Pointer arithmetic)

C++ cho phép bạn thực hiện các phép toán **cộng** hoặc **trừ** số nguyên trên con trỏ (**+**, **-**, **++**, **--**).

Nếu **ptr** trỏ đến một số nguyên, **ptr + 1** là **địa chỉ của số nguyên tiếp theo** trong bộ nhớ sau **ptr**. Ngược lại, **ptr - 1** là địa chỉ của số nguyên trước đó trong bộ nhớ trước **ptr**.

Chú ý: **ptr + 1** không trả về địa chỉ vùng nhớ sau **ptr**, mà **ptr + 1** trả về địa chỉ vùng nhớ của đối tượng tiếp theo thuộc kiểu mà **ptr** trỏ tới.

Ví dụ: Nếu **ptr** trỏ đến một số nguyên (4 byte hoặc 8 byte), **ptr + 5** có nghĩa là cách 5 số nguyên (20 byte hoặc 40 byte) sau **ptr**. Nếu **ptr** trỏ đến một ký tự char (1 byte), **ptr + 5** có nghĩa là cách 5 ký tự (5 byte) sau **ptr**.

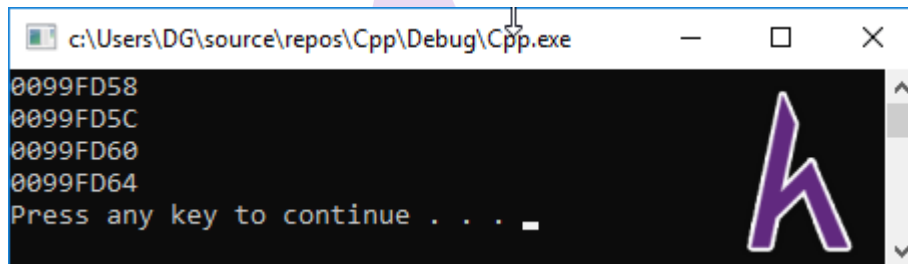
```
#include <iostream>
using namespace std;

int main()
{
    int value = 5;
    int *ptr = &value;

    cout << ptr << '\n';
}
```

```
cout << ptr + 1 << '\n';  
cout << ptr + 2 << '\n';  
cout << ptr + 3 << '\n';  
  
system("pause");  
return 0;  
}
```

Output:

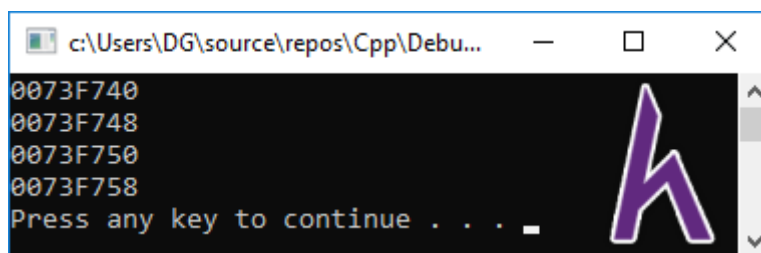


Trong chương trình trên, bạn có thể thấy, mỗi địa chỉ trong số này khác nhau **4 đơn vị**, bằng kích thước của kiểu **int** là **4 byte**.

Ví dụ 2: Sử dụng kiểu **double** thay vì kiểu **int** và xem sự khác biệt:

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    double value = 5;  
    double *ptr = &value;  
  
    cout << ptr++ << '\n';  
    cout << ptr++ << '\n';  
    cout << ptr++ << '\n';  
    cout << ptr << '\n';  
  
    system("pause");  
    return 0;  
}
```

Output:



Lúc này, mỗi địa chỉ trong số này khác nhau **8 đơn vị**, tương đương với kích thước của kiểu **double** là **8 byte**.

Mảng tĩnh được lưu trong bộ nhớ như thế nào?

Trong bài [MẢNG MỘT CHIỀU TRONG C++ \(Arrays\)](#), bạn đã biết được 2 tính chất của mảng tĩnh trong C++:

- Kích thước được **xác định ngay khi khai báo** và **không bao giờ thay đổi** (mảng tĩnh).
- C++ luôn chỉ định **một khối nhớ liên tục** cho một biến kiểu mảng.

Trong C++, các phần tử của mảng tĩnh được sắp xếp một cách tuần tự trong bộ nhớ, nghĩa là `array[0]`, `array[1]`, `array[2]`, ... được nằm cạnh nhau, và theo thứ tự.

Để chứng minh điều đó, ta cùng xem chương trình bên dưới:

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

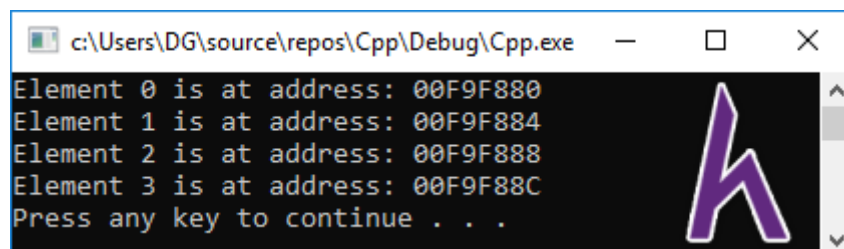
int main()
{
    int array[] = { 5, 8, 2, 7 };

    cout << "Element 0 is at address: " << &array[0] << "\n";
    cout << "Element 1 is at address: " << &array[1] << "\n";
    cout << "Element 2 is at address: " << &array[2] << "\n";
    cout << "Element 3 is at address: " << &array[3] << "\n";

    system("pause");
}
```

```
    return 0;  
}
```

Output:



Bạn có thể thấy, mỗi địa chỉ trong số này **liên tục nhau**, khác nhau **4 đơn vị**, bằng kích thước của kiểu **int** là **4 byte**.

Ứng dụng giữa Con trỏ số học, Mảng, và Chỉ mục mảng

Trong phần trên, bạn đã biết được các phần tử của một mảng được **sắp xếp tuần tự** trong bộ nhớ.

Trong bài [CON TRỎ & MẢNG TRONG C++](#) (Pointers and arrays), bạn đã học được rằng một mảng tĩnh có thể được **chuyển đổi ngầm định thành một con trỏ trỏ đến phần tử đầu tiên** (phần tử 0) của mảng.

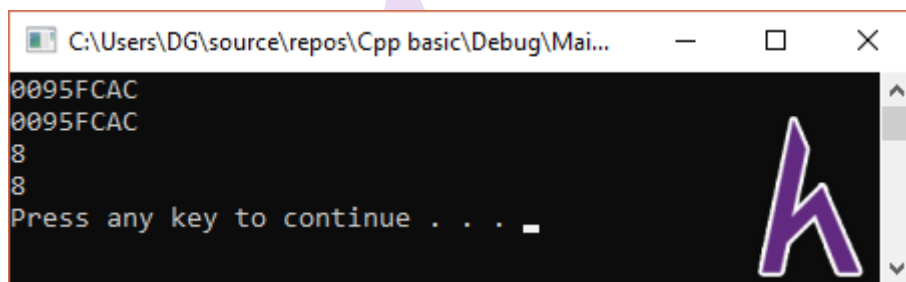
Cũng trong phần trên, bạn đã học được rằng việc cộng 1 vào con trỏ **ptr** sẽ trả về địa chỉ vùng nhớ của đối tượng tiếp theo **thuộc kiểu** mà **ptr** trỏ tới.

Vì vậy, ta có thể kết luận rằng nếu cộng **n** vào một mảng, kết quả sẽ là 1 con trỏ trỏ đến địa chỉ phần tử thứ **n** của mảng.

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    int array[] = { 5, 8, 2, 7 };  
    cout << &array[1] << '\n'; // địa chỉ vùng nhớ phần tử 1  
    cout << array + 1 << '\n'; // địa chỉ vùng nhớ phần tử 1  
}
```

```
cout << array[1] << '\n'; // 8
cout << *(array + 1) << '\n'; // 8

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

Output:

Trong chương trình trên, **array[n]** tương đương với ***(array + n)** với **n** là 1 số nguyên.

Ghi chú: Dấu ngoặc vuông thường được sử dụng hơn vì tính đơn giản và dễ hiểu.

Sử dụng con trỏ lặp qua một mảng

Trong phần trên, bạn đã biết được rằng cộng **n** vào một mảng, kết quả sẽ là 1 con trỏ trỏ đến địa chỉ phần tử thứ **n** của mảng.

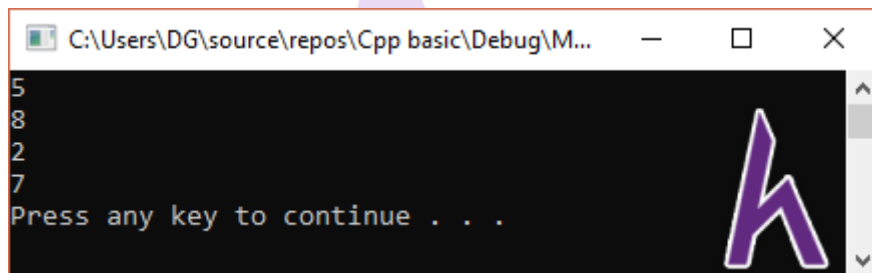
Tính chất đó có thể được ứng dụng trong trường hợp duyệt một mảng:

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

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

```
for (int* ptr = array; ptr < array + arrayLength; ptr++)  
{  
    cout << *ptr << '\n';  
}  
  
system("pause");  
return 0;  
}
```

Output:



Thông thường, cách này không được sử dụng vì nó phức tạp và dễ nhầm lẫn hơn cách sử dụng dấu ngoặc vuông [] đối với mảng.

Kết luận

Qua bài học này, bạn đã nắm được Các phép toán trên Con trỏ và Chỉ mục mảng trong C++. Đây là những kiến thức này khá quan trọng về con trỏ và mảng mà bạn cần nắm vững để có thể áp dụng trong những bài học sau này.

Trong bài tiếp theo, mình sẽ giới thiệu cho các bạn về [CẤP PHÁT ĐỘNG TRONG C++ \(Dynamic memory allocation\)](#).

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**".