

Bài 42: CON TRỎ CƠ BẢN TRONG C++

Xem bài học trên website để ủng hộ Kteam: [Con trỏ cơ bản trong C++](#)

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 [LỚP STD::ARRAY TRONG C++11](#). Từ C++11, lớp **std::array** thường được ưu tiên sử dụng thay thế cho mảng tĩnh, vì nó được hỗ trợ sẵn nhiều phương thức và dễ dàng quản lý hơn

Hôm nay, chúng ta sẽ cùng tìm hiểu về khái niệm **Con trỏ (Pointer)**, một trong những phần khó hiểu nhất của ngôn ngữ lập trình C++.

Nội dung

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

- [BIẾN TRONG C++](#)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Nhìn lại khái niệm biến (variable)
- Toán tử địa chỉ (&): address-of operator
- Toán tử trỏ đến (*): dereference operator
- Con trỏ (Pointer)

Nhìn lại khái niệm biến trong C++ (Variables)

Trong bài học [BIẾN TRONG C++](#), bạn đã biết **Biến (variable)** là tên của một **vùng trong bộ nhớ RAM**, được sử dụng để lưu trữ thông tin.

Ví dụ:

```
// Khai báo biến số nguyên n
// Giả sử n được cấp vùng nhớ tại địa chỉ 0x0069
int n;
```

Khi dòng lệnh này được thực thi, **một vùng trong bộ nhớ RAM** sẽ được cấp cho biến **n**. Trong trường hợp này, biến **n** được cấp một vùng nhớ tại **địa chỉ 0x0069 trong RAM**, vậy mỗi khi chương trình chạy đến dòng lệnh nào chứa biến **n**, chương trình sẽ **vào vùng nhớ 0x0069 để lấy giá trị của nó**.

Toán tử địa chỉ (&): address-of operator

Toán tử địa chỉ (&) cho phép chúng ta xem địa chỉ bộ nhớ nào được gán cho một biến.

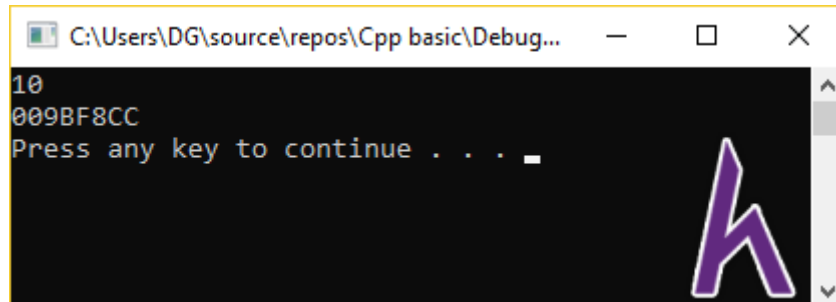
Ví dụ:

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

int main()
{
    int x = 10;
    cout << x << '\n'; // print the value of variable x
    cout << &x << '\n'; // print the memory address of variable x

    system("pause");
}
```

```
return 0;  
}
```

Output:

Xét kết quả trên, ta thấy **009BF8CC** chính là **địa chỉ của biến x trong bộ nhớ RAM**, địa chỉ này sẽ thay đổi sau mỗi lần chạy chương trình.

Chú ý: Toán tử địa chỉ (&) tương tự như **toán tử bitwise (&)**, nhưng **Toán tử địa chỉ (&)** là toán tử 1 ngôi, trong khi **toán tử bitwise (&)** là 2 ngôi.

Toán tử trở đến (*): dereference operator

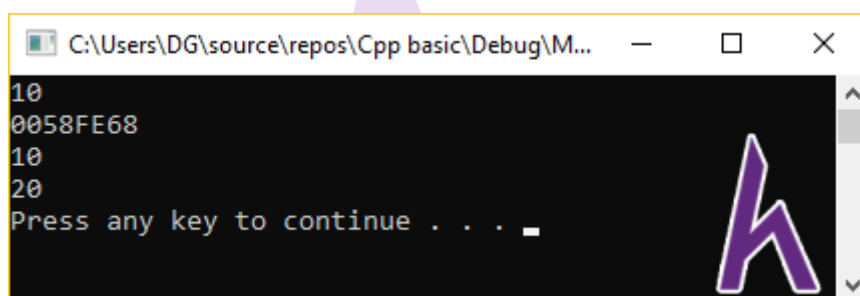
Toán tử trở đến (*) cho phép chúng ta **truy cập (get/set) giá trị** tại một địa chỉ cụ thể.

Ví dụ:

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
  
    int x = 10;  
    cout << x << '\n'; // print the value of variable x  
    cout << &x << '\n'; // print the memory address of variable x  
}
```

```
cout << *&x << '\n'; /// print the value at the memory address of variable  
x  
  
*&x = 20;  
cout << x << '\n'; // print the value of variable x  
  
return 0;  
}
```

Output:



Trong ví dụ trên, ta dùng **toán tử trở (*)** thay đổi giá trị tại địa chỉ của biến x thành 20 (không trực tiếp thay đổi biến x).

Chú ý: Toán tử trở (*) tương tự như **toán tử nhân (*)**, tuy nhiên **toán tử trở (*)** là toán tử 1 ngôi, trong khi **toán tử nhân (*)** là 2 ngôi.

Con trỏ (Pointer)

Con trỏ (pointer) là một biến chứa một địa chỉ bộ nhớ làm giá trị của nó.

Khai báo con trỏ (Declaring a pointer)

Biến con trỏ (pointer variable) được khai báo giống như các biến thông thường, nhưng có thêm **một dấu sao (*)** giữa kiểu dữ liệu và tên biến.

```
int *iPtr; // con trỏ đến 1 địa chỉ chứa giá trị số nguyên  
double *dPtr; // con trỏ đến 1 địa chỉ chứa giá trị số thực
```

```
int* iPtr2; // đúng cú pháp (nhưng không nên sử dụng)
int * iPtr3; // đúng cú pháp (nhưng không nên sử dụng)

int *iPtr4, *iPtr5; // khai báo 2 con trỏ đến các biến số nguyên
```

C++ cho phép đặt dấu sao **cạnh kiểu dữ liệu, cạnh tên biến, hoặc ở giữa**. Tuy nhiên, khi khai báo nhiều biến con trỏ, dấu sao phải được đặt cạnh mỗi biến.

```
int* iPtr6, iPtr7; // iPtr6 là 1 con trỏ, nhưng iPtr7 chỉ là 1 biến số nguyên
```

Chú ý: Khi khai báo một **biến con trỏ**, hãy đặt **dấu sao (*) bên cạnh tên biến**.

Tuy nhiên, khi trả về một con trỏ từ một hàm, sẽ rõ ràng hơn nếu đặt dấu sao (*) bên cạnh kiểu trả về:

```
int* doSomething();
```

Chú ý: Khi khai báo một **hàm** trả về 1 con trỏ, hãy **đặt dấu sao (*) bên cạnh kiểu dữ liệu trả về**.

Tương tự như các biến thông thường, **con trỏ không được khởi tạo khi khai báo**. Nếu con trỏ không được khởi tạo một giá trị, chúng sẽ chứa giá trị rác.

Gán giá trị cho con trỏ (Assigning a value to a pointer)

Vì **con trỏ (pointer)** là một biến chứa một **địa chỉ bộ nhớ**, nên khi gán giá trị cho con trỏ, giá trị đó phải là 1 địa chỉ.

Ví dụ:

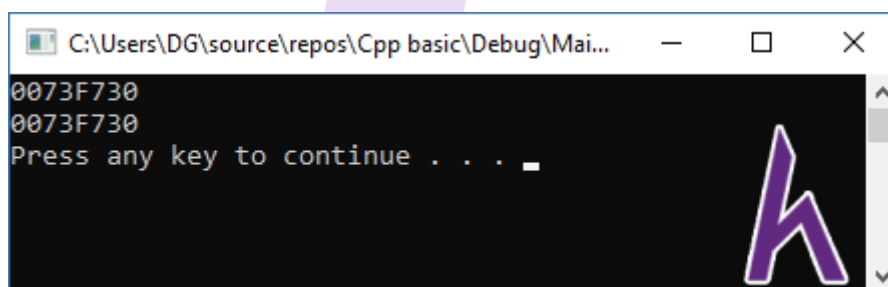
```
#include <iostream>
using namespace std;

int main()
{
    int value = 10;
    int *ptr = &value; // khởi tạo con trỏ ptr là địa chỉ biến value

    cout << &value << '\n'; // in địa chỉ biến value
    cout << ptr << '\n'; // in địa chỉ của con trỏ ptr đang giữ

    return 0;
}
```

Output:



Trong ví dụ trên, con trỏ ptr đang nắm giữ địa chỉ của biến value, nên ta có thể nói **con trỏ ptr trỏ đến biến value**.

Chú ý: Kiểu dữ liệu của con trỏ dùng để xác định kiểu dữ liệu của biến mà nó trỏ đến.

```
int iValue = 5;
double dValue = 7.0;

int *iPtr = &iValue; // ok
double *dPtr = &dValue; // ok

iPtr = &dValue; // sai – con trỏ int không thể trỏ đến địa chỉ biến double
dPtr = &iValue; // sai – con trỏ double không thể trỏ đến địa chỉ biến int

int *ptr = 5; // sai – con trỏ chỉ có thể giữ 1 địa chỉ
```

```
double *dPtr = 0x0012FF7C; // sai
```

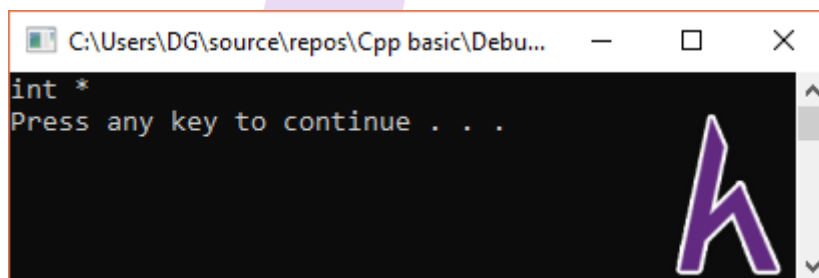
Trong ví dụ trên, ta có thể thấy **&iValue** là 1 địa chỉ của biến kiểu **int**, nên nó có thể gán cho con trỏ kiểu **int**. Để hiểu rõ hơn, ta có thể kiểm tra kiểu dữ liệu của **&iValue**:

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

int main()
{
    int iValue = 5;
    cout << typeid(&iValue).name() << "\n";

    return 0;
}
```

Output:



Truy cập vào vùng nhớ mà con trỏ trỏ đến (Dereferencing pointers)

Khi biến con trỏ đã được trỏ đến một địa chỉ nào đó, ta có thể truy xuất giá trị tại địa chỉ đó bằng **toán tử trỏ (*)**.

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

int main()
{
    int value = 10;
    cout << &value << "\n"; // in địa chỉ biến value
}
```

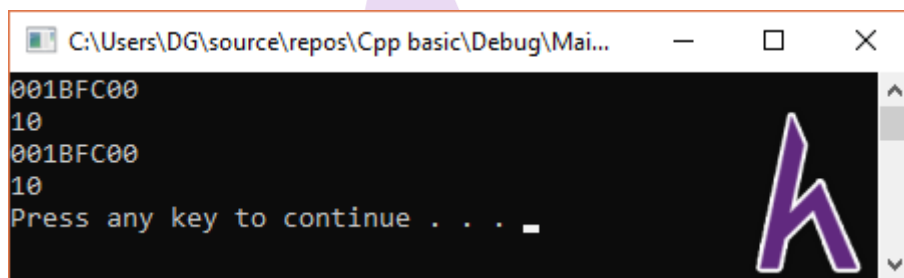
```

cout << value << "\n"; // in giá trị biến value

int *ptr = &value; // ptr trỏ đến biến value
cout << ptr << "\n"; // in địa chỉ con trỏ ptr trỏ đến, tương đương &value
cout << *ptr << "\n"; // in giá trị tại địa chỉ mà ptr trỏ đến, tương đương
value

return 0;
}

```

Output:

Sau khi được gán, giá trị con trỏ có thể được gán lại cho một giá trị khác:

```

int value1 = 5;
int value2 = 7;

int *ptr;

ptr = &value1; // ptr points to value1
cout << *ptr << "\n"; // prints 5

ptr = &value2; // ptr now points to value2
cout << *ptr << "\n"; // prints 7

```

Chú ý: Khi địa chỉ của **biến value** được gán cho **con trỏ ptr**:

- **ptr** tương đương với **&value**
- ***ptr** được xử lý giống như **value**

Vì ***ptr** được xử lý giống như **value**, nên ta có thể gán giá trị cho ***ptr** như 1 biến thông thường:


```
value = 5;
int *ptr = &value; // ptr points to value

*ptr = 7; // *ptr tương đương với value
cout << value; // prints 7
```

Kích thước của con trỏ (size of pointers)

Kích thước của con trỏ phụ thuộc vào **kiến trúc mà tập tin thực thi được biên dịch**.

- Kiến trúc **x86**, con trỏ sẽ có kích thước 32-bit (4 bytes)
- Kiến trúc **x64**, con trỏ sẽ có kích thước 64-bit (8 bytes)

```
char *chPtr; // biến kiểu char có kích thước 1 byte
int *iPtr; // biến kiểu int có kích thước 4 bytes
struct Something
{
    int nX, nY, nZ;
};
Something *somethingPtr; // biến kiểu Something là 12 bytes

cout << sizeof(chPtr) << '\n'; // 4 bytes
cout << sizeof(iPtr) << '\n'; // 4 bytes
cout << sizeof(somethingPtr) << '\n'; // 4 bytes
```

Chú ý: Kiểu dữ liệu của con trỏ thay đổi không tác động đến kích thước bộ nhớ của con trỏ.

Truy cập con trỏ không hợp lệ

Khi truy cập vào một con trỏ, ứng dụng sẽ đến vị trí bộ nhớ được lưu trữ trong con trỏ và truy xuất nội dung của bộ nhớ. Vì lý do bảo mật, nếu một ứng dụng cố gắng truy cập vào một vị trí bộ nhớ không được hệ điều hành phân bổ, hệ điều hành có thể tắt ứng dụng.

Chú ý: Nếu truy xuất giá trị của con trỏ khi nó chưa gán địa chỉ cụ thể, chương trình có thể bị đóng bởi hệ điều hành.

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

void foo(int *&p)
{
    // p là tham chiếu đến con trỏ (sẽ được đề cập ở những bài sau)
    // Hàm này dùng để đánh lừa compiler rằng con trỏ p đã bị thay đổi
    // Mục đích là để chương trình được biên dịch thành công
}

int main()
{
    int *p; // Khai báo 1 con trỏ mang giá trị rác
    foo(p); // Đánh lừa compiler rằng con trỏ p đã được gán

    cout << *p; // In ra giá trị tại địa chỉ rác

    return 0;
}
```

Trong ví dụ trên, mặc dù chương trình đã biên dịch thành công, nhưng khi thực thi, ta nhận được thông báo lỗi vi phạm quyền truy cập **“Exception thrown: read access violation”**.

Kết luận

Qua bài học này, bạn đã nắm được cơ bản về Con trỏ trong C++ (Pointer). Con trỏ được xem là một trong những phần khó hiểu nhất của ngôn ngữ C++, nhưng nó sẽ đơn giản hơn nhiều nếu bạn nắm được những khái niệm căn bản trong bài học này.

Trong bài tiếp theo, mình sẽ giới thiệu cho các bạn khái niệm CON TRỎ NULL TRONG C++ (NULL pointers).

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"**.

