Bài 51: CON TRỎ TRỎ ĐẾN CON TRỎ (POINTERS TO POINTERS)

Xem bài học trên website để ủng hộ Kteam: Con trỏ trỏ đến con trỏ (Pointers to pointers)

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, bạn đã được giới thiệu về CON TRỎ VOID (Void pointers) trong C++. Lưu ý rằng bạn nên hạn chế lạm dụng con trỏ void, trừ khi không còn cách giải quyết nào khác

Hôm nay, chúng ta sẽ tiếp tục tìm hiểu một phần nâng cao của con trỏ trong C++, cụ thể là **Con trỏ trỏ đến con trỏ (Pointers to pointers)**.

Nội dung

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

- CON TRỎ TRONG C++ (Pointer)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Con trỏ trỏ đến con trỏ (Pointers to pointers)
- Mảng con trỏ (Arrays of pointers)
- Cấp phát động mảng 2 chiều (2D dynamically allocated arrays)

- Con trỏ trỏ đến con trỏ trỏ đến con trỏ...

Con trỏ trỏ đến con trỏ (Pointers to pointers)

Con trỏ trỏ đến con trỏ (Pointers to pointers) là một con trỏ chứa địa chỉ của một con trỏ khác.

Bạn đã biết **con trỏ thông thường** sử dụng **một** dấu sao (*) khi khai báo:

```
int *ptr; // con trỏ trỏ đến giá trị kiểu int
```

Con trỏ trỏ đến con trỏ sử dụng **hai** dấu sao (**) khi khai báo:

```
int **ptr_ptr; // con trỏ trỏ đến con trỏ trỏ đến giá trị kiểu int
```

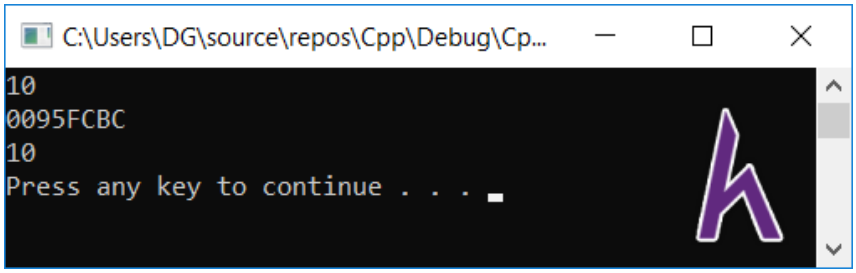
Con trỏ trỏ đến con trỏ hoạt động như một con trỏ thông thường. Chúng ta có thể sử dụng toán tử **dereference (*)** để truy cập giá trị của con trỏ.

```
int value = 10;

int *ptr = &value;
cout << *ptr << "\n"; // in giá trị tại địa chỉ ptr trỏ đến (biến value)

int **ptr_ptr = &ptr; // con trỏ "ptr_ptr" trỏ đến con trỏ "ptr" trỏ đến biến "value"
cout << *ptr_ptr << "\n"; // in giá trị tại địa chỉ ptr_ptr trỏ đến (địa chỉ ptr (&ptr))
cout << **ptr_ptr << "\n"; // dereference 2 lần để in giá trị tại địa chỉ ptr trỏ đến (biến value)
```

Output:



Trong ví dụ trên, khi sử dụng 1 toán tử **dereference (*)** cho **ptr_ptr**, nghĩa là chúng ta đang truy xuất đến giá trị tại **địa chỉ** mà con trỏ **ptr_ptr** nắm giữ (địa chỉ con trỏ ptr).

Khi **dereference (*)** 2 lần con trỏ **ptr_ptr**, chúng ta được giá trị tại **địa chỉ con trỏ ptr trỏ đến** (biến value).

Tương tự như con trỏ thông thường, con trỏ trỏ đến con trỏ có thể gán bằng null:

```
int **ptrptr = nullptr;
```

Lưu ý: rằng bạn **không thể** gán trực tiếp một con trỏ trỏ đến con trỏ bằng giá trị:

```
int value = 10;
int **ptrptr = &&value; // lỗi
```

Mảng con trỏ (Arrays of pointers)

Con trỏ trỏ đến con trỏ có thể được dùng để quản lý mảng một chiều các con trỏ:

```
int *ptr1 = NULL;
int *ptr2 = NULL;

int **ptr_ptr = new int*[2];
ptr_ptr[0] = ptr1;
ptr_ptr[1] = ptr2;
```

Mảng một chiều các con trỏ hoạt động như một mảng thông thường, ngoại trừ việc mỗi phần tử mảng là một con trỏ.

Cấp phát động mảng 2 chiều (2D dynamically allocated arrays)

Con trỏ trỏ đến con trỏ được sử dụng phổ biến cho việc cấp phát động mảng 2 chiều.

Với mảng 2 chiều thông thường, việc khai báo đơn giản như sau:

```
int arr[2][3];
```

Tuy nhiên, cấp phát động mảng 2 chiều phức tạp hơn:

- **Bước 1:** cấp phát động một mảng các con trỏ.
- **Bước 2:** lặp qua mảng con trỏ và cấp phát một mảng động cho từng phần tử mảng.

Chú ý: Mảng 2 chiều động là mảng một chiều động các mảng một chiều động!

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

int main()
{
	int row, col;
	// nhập số dòng, cột
	cout << "Nhap so dong: ";
	cin >> row;
	cout << "Nhap so cot: ";
	cin >> col;

	// cấp phát động
	int **arr = new int*[row]; // Cấp phát vùng nhớ cho ROW con trỏ kiểu (int *): dòng
	for (int i = 0; i < row; i++)
	{
```

```
                arr[i] = new int[col]; // Mỗi con trỏ kiểu (int *) sẽ quản lý COL phần tử kiểu int: cột
        }

        // nhập mảng 2 chiều
        cout << "Nhap mang:" << endl;
        for (int i = 0; i < row; i++)
        {
                for (int j = 0; j < col; j++)
                {
                        cout << "a[" << i << "][" << j << "] = ";
                        cin >> arr[i][j];
                }
        }

        // xuất mảng 2 chiều
        for (int i = 0; i < row; i++)
        {
                for (int j = 0; j < col; j++)
                {
                        cout << arr[i][j] << " ";
                }
                cout << endl;
        }

        // Giải phóng vùng nhớ cho từng phần tử mảng
        for (int i = 0; i < row; i++)
        {
                delete[] arr[i];
        }

        // Giải phóng cho mảng
        delete[] arr;

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

Vì việc cấp phát động và giải phóng mảng hai chiều rất phức tạp, nên lập trình viên thường sử dụng **mảng 1 chiều kích thước x*y** thay thế cho mảng 2 chiều x dòng, y cột:

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

// xây dựng hàm chuyển đổi
int getSingleIndex(int row, int col, int numberOfColumnsInArray)
{
	return (row * numberOfColumnsInArray) + col;
}

int main()
{
	// giả sử cần mảng 2 chiều 5 dòng, 10 cột
	// cấp phát mảng 1 chiều
	int row = 5;
	int col = 10;
	int n = row * col; // 5 * 10 = 50 phần tử
	int *arr = new int[n];

	// gán arr[1,2] = 3 sử dụng mảng 1 chiều arr
	arr[getSingleIndex(1, 2, col)] = 3;

	// xuất giá trị arr[1,2]
	cout << arr[getSingleIndex(1, 2, col)] << "\n";

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

Con trỏ trỏ đến con trỏ trỏ đến con trỏ...

Chúng ta có thể khai báo những con trỏ như sau:

```
int ***ptrx3;

int ****ptrx4;
```

Tuy nhiên, trong thực tế, những con trỏ như thế này không được sử dụng nhiều vì nó khá phức tạp.

Kết luận

Qua bài học này, bạn đã nắm được khái niệm Con trỏ trỏ đến con trỏ (Pointers to pointers) trong C++. Bạn nên tránh sử dụng con trỏ trỏ đến con trỏ trừ khi không có giải pháp nào khác, vì chúng phức tạp để sử dụng và có khả năng gây ra những lỗi tiềm ẩn về vùng nhớ.

Trong bài tiếp theo, mình sẽ giới thiệu cho các bạn khái niệm LỚP DỰNG SẴN VECTOR trong C++.

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**".