Bài: Đệ quy trong C++ (Recursion)

Xem bài học trên website để ủng hộ Kteam: Đệ quy trong C++ (Recursion)

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, Kteam đã chia sẻ cho các bạn về CON TRỎ HÀM TRONG C++. Đó là những kiến thức khá quan trọng mà bạn cần nắm trong C++.

Hôm nay, chúng ta sẽ cùng tìm hiểu về **Đệ quy trong C++ (Recursion)**.

Nội dung

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

- CƠ BẢN VỀ HÀM VÀ GIÁ TRỊ TRẢ VỀ

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Đệ quy là gì?
- Điều kiện dừng (điều kiện cơ sở)
- Một số bài toán đệ quy kinh điển
- Đệ quy so với vòng lặp

Đệ quy là gì?

Trong lập trình, **một hàm được gọi là đệ quy nếu bên trong thân hàm có một lời gọi đến chính nó**.

Hàm đệ quy luôn có **điều kiện dừng** được gọi là "điểm neo". Khi đạt tới điểm neo, hàm sẽ không gọi chính nó nữa.

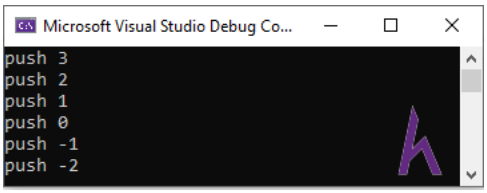
Khi được gọi, hàm đệ quy thường được truyền cho một tham số, thường là kích thước của bài toán lớn ban đầu. Sau mỗi lời gọi đệ quy, tham số sẽ nhỏ dần, nhằm phản ánh bài toán đã nhỏ hơn và đơn giản hơn. Khi tham số đạt tới một **giá trị cực tiểu** (tại điểm neo), hàm sẽ chấm dứt.

Ví dụ:

C++:

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

void countDown(int count)
{
	cout << "push " << count << '\n';
	countDown(count - 1); // countDown() tự gọi lại chính nó
}

int main()
{
	countDown(3);

	return 0;
}
```

Output:



Khi **countDown(3)** được gọi, thì "**push 3**" được in ra và **countDown(2)** được gọi. Sau đó **countDown(2)** in "**push 2**" và gọi **countDown(1)**. Sau đó **countDown(1)** in "**push 1**" và gọi **countDown(0)**. Theo trình tự, trong hàm **countDown(n)** sẽ gọi **countDown(n-1)**, việc này được lặp lại vô hạn, đây là một hàm đệ quy lặp vô hạn.

Trong bài HÀM NỘI TUYẾN TRONG C++ (Inline functions), bạn đã biết khi một hàm được gọi, nó sẽ được đưa vào **ngăn xếp** (stack), hàm đệ quy cũng vậy, mỗi lần gọi chính nó thì nó lại được đưa vào ngăn xếp (stack), nếu như không có điểm dừng, hoặc gọi mãi mà chưa tới điểm dừng, sẽ dễ xảy ra tình trạng tràn bộ nhớ ngăn xếp (stack).

Điều kiện dừng (điều kiện cơ sở)

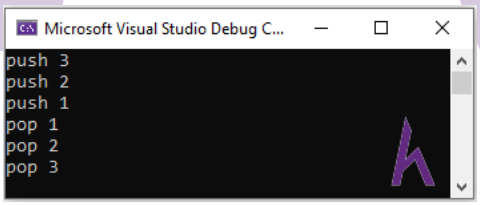
Hàm đệ quy phải có một điều kiện kết thúc đệ quy, nếu không chương trình sẽ lặp vô hạn (đến khi tràn bộ nhớ ngăn xếp). **Điều kiện dừng** của hàm đệ quy gọi là điều kiện cơ sở.

Ví dụ:

C++:

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

void countDown(int count)
{
	cout << "push " << count << '\n';

	if (count > 1) // điều kiện dừng
		countDown(count - 1);

	cout << "pop " << count << '\n';
}

int main()
{
	countDown(3);
	return 0;
}
```

Output:



Trong chương trình trên, do có **điều kiện kết thúc** (count > 1), nên trong hàm **countDown(1)** không gọi **countDown(0)**, vì vậy "**pop 1**" được in ra và kết thúc hàm **countDown(1)**. Lúc này, hàm **countDown(1)** bật ra khỏi ngăn xếp, hàm **countDown(2)** tiếp tục thực thi tại vị trí sau lời gọi hàm **countDown(1)**, do đó "**pop 2**" được in ra. Tuần tự đến khi thoát ra toàn bộ các lời gọi hàm đệ quy.

Một số bài toán đệ quy kinh điển

Bài toán tính giai thừa

Cho **n** là một số tự nhiên (n>=0). Hãy tính giai thừa của n (**n!**) biết rằng 0! = 1 và n! = (n-1)! * n.

Phân tích:

Theo giả thiết, ta có : **n! = (n-1)! * n**. Như vậy :

- Để tính **n!** ta cần phải tính **(n-1)!**
- Để tính **(n-1)!** ta phải tính **(n-2)!**
- ...

Cứ như vậy, cho tới khi gặp trường hợp **0!**. Khi đó ta lập tức có được kết quả là **1**, không cần phải tính thông qua một kết quả trung gian khác.

C++:

```
long GiaiThua(int n)
{
	if (n == 0)
	{
		return 1; // điều kiện dừng
	}

	return n * GiaiThua(n - 1); // gọi đệ quy
}
```

Dãy Fibonaci

Dãy Fibonaci là dãy vô hạn các số tự nhiên. Số Fibonaci thứ **n**, ký hiệu **F(n)**, được định nghĩa như sau :

- **F(n) = 0**, nếu **n = 0**
- **F(n) = 1,** nếu **n = 1**
- **F(n) = F(n-1) + F(n-2)**, nếu **n > 1**

Yêu cầu: tính số fibonaci thứ n với n cho trước.

C++:

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

int fibonacci(int number)
{
	if (number == 0)
		return 0; // điều kiện dừng
	if (number == 1)
		return 1; // điều kiện dừng
	return fibonacci(number - 1) + fibonacci(number - 2);
}

int main()
{
	// in dãy 15 số fibonacci
	for (int count = 0; count < 15; ++count)
		cout << fibonacci(count) << " ";

	return 0;
}
```

Output: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

Đệ quy so với vòng lặp

Một câu hỏi thường gặp là: "**Tại sao lại sử dụng hàm đệ quy trong khi bạn có thể thực hiện với vòng lặp (vòng lặp for hoặc while)?**".

Thông thường, các bài toán đệ quy có thể giải quyết bằng vòng lặp. Tuy nhiên, hàm đệ quy giúp code dễ đọc và đơn giản hơn.

Ví dụ: in dãy fibonaci

C++:

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

int main()
{
	int n1 = 0;
	int n2 = 1;

	for (int i = 1; i <= 15; ++i)
	{
		cout << n1 << " ";

		int nextTerm = n1 + n2;
		n1 = n2;
		n2 = nextTerm;
	}

	return 0;
}
```

Output: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

Sử dụng vòng lặp thường có hiệu suất cao hơn so với phiên bản hàm đệ quy. Vì khi một hàm được gọi, chương trình sẽ tốn một lượng chi phí cho việc đưa hàm vào ngăn xếp (stack).

> **Chú ý:** Ưu tiên sử dụng vòng lặp thay vì đệ quy.

Kết luận

Qua bài học này, bạn đã nắm được khái niệm Đệ quy trong C++ (Recursion) và những bài toán cơ bản về đệ quy.

Trong bài tiếp theo, mình sẽ giới thiệu cho các bạn về KHUÔN MẪU HÀM TRONG C++ (Function templates).

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**".