Bài: Khuôn mẫu hàm trong C++ (Function templates)

Xem bài học trên website để ủng hộ Kteam: Khuôn mẫu hàm trong C++ (Function templates)

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 ĐỆ QUY TRONG C++ (Recursion) và những bài toán cơ bản về đệ quy.

Hôm nay, chúng ta sẽ cùng tìm hiểu về **Khuôn mẫu hàm trong C++ (Function templates)**.

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
- NẠP CHỒNG HÀM TRONG C++ (Function overloading)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Đặt vấn đề
- Khuôn mẫu hàm trong C++ (Function templates)

Đặt vấn đề

Chúng ta có hàm hoán vị 2 số nguyên:

C++:

```
void Swap(int& x, int& y)
{
	int temp = x;
	x = y;
	y = temp;
}
```

Hàm trên dùng để hoán vị 2 số nguyên. Nếu ta cần hoán vị 2 số kiểu **float** hoặc **double**, ... câu trả lời thường sẽ là sử dụng kỹ thuật **nạp chồng hàm** được giới thiệu trong bài NẠP CHỒNG HÀM TRONG C++ (Function overloading).

C++:

```
void Swap(double& x, double& y)
{
	double temp = x;
	x = y;
	y = temp;
}
```

Tuy nhiên, những hàm nạp chồng xử lý giống nhau, chỉ khác về tham số, điều này vi phạm nguyên tắc trùng lặp code trong lập trình, khó quản lý và bảo trì với những hàm phức tạp.

Khuôn mẫu hàm (Function templates) đã sinh ra để giải quyết những vấn đề trên.

Khuôn mẫu hàm trong C++ (Function templates)

Trong C++, **khuôn mẫu hàm** là một cái khuôn dùng để tạo ta nhiều hàm có định nghĩa giống nhau. Và nó có thể được tái sử dụng nhiều lần nếu chúng ta muốn.

Khi hàm khuôn mẫu được gọi, **compiler sẽ tạo ra một bản sao** của hàm đó, và thay thế kiểu dữ liệu tương ứng của các tham số khuôn mẫu.

Ví dụ:

C++:

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

template <typename Type>
void Swap(Type& x, Type& y)
{
	Type Temp = x;
	x = y;
	y = Temp;
}

int main()
{
	int x = 5, y = 10;
	double a = 5.5, b = 3.5;

	Swap(x, y);
	Swap(a, b);

	int c = 5, d = 10;
	Swap(c, d);

	return 0;
}
```

Trong ví dụ trên:

- **Type** chỉ là một tên riêng thể hiện cho một kiểu dữ liệu tổng quát.
- **class** ta có thể thay thế bằng "**typename**", ở đây nó không có sự khác biệt.

Giải thích:

Khi gặp lời gọi hàm **Swap(x, y)**, trình biên dịch lúc này sẽ tạo ra một hàm **void Swap(int& x, int& y)** từ khuôn mẫu hàm, và biên dịch hàm thành mã máy.

Khi gặp lời gọi hàm **Swap(a, b)**, trình biên dịch cũng tạo một phiên bản khác **void Swap(double& x, double& y)** tương ứng. Khi gặp **Swap(c, d)**, vì trình biên dịch đã tạo ra một hàm **void Swap(int& x, int& y)** trước đó, nên lúc này trình biên dịch không cần tạo ra phiên bản mới.

Với những hàm sử dụng nhiều hơn một kiểu tham số, ví dụ 2 tham số có 2 kiểu dữ liệu khác nhau, chúng ta có thể sửa lại như sau:

C++:

```
template <typename T1, typename T2>
// template function here
```

Hàm khuôn mẫu giúp bạn tiết kiệm rất nhiều thời gian, vì bạn chỉ cần viết một hàm và nó sẽ hoạt động với nhiều kiểu khác nhau. Sử dụng khuôn mẫu hàm giúp dễ quản lý và bảo trì mã hơn, vì mã trùng lặp được giảm đáng kể.

Kết luận

Qua bài học này, bạn đã nắm được khái niệm Khuôn mẫu hàm tron C++ (Function templates).

Trong bài tiếp theo, mình sẽ giới thiệu cho các bạn về TYPEDEF 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**".