Bài 40: VÒNG LẶP FOR EACH TRONG C++11 (FOR EACH LOOPS)

Xem bài học trên website để ủng hộ Kteam: Vòng lặp for each trong C++11 (For each loops)

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 TỪ KHÓA AUTO TRONG C++11 (The auto keyword), một từ khóa khá hữu ích trong lập trình C++.

Hôm nay, mình sẽ giới thiệu cho các bạn về **Vòng lặp for each trong C++11 (For each loops)**.

Nội dung

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

- TỪ KHÓA AUTO TRONG C++11 (The auto keyword)
- VÒNG LẶP FOR TRONG C++ (For statements)
- MẢNG 1 CHIỀU (Arrays)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

Vòng lặp for each trong C++11

Vòng lặp for each trong C++11

Trong bài MẢNG 1 CHIỀU TRONG C++ (Arrays), bạn đã biết được cách sử dụng **vòng lặp for** để truy cập vào từng phần tử của mảng.

Ví dụ:

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

#define MAX 5

int main()
{
	int arr[MAX] = { 14, 3, 6, 27, 12 };
	for (int i = 0; i < MAX; i++)
	{
		cout << arr[i] << " ";
	}
cout << endl;

	return 0;
}
```

Output: 14 3 6 27 12

Phiên bản **C++11** cung cấp một loại vòng lặp mới, có cú pháp đơn giản, dễ sử dụng hơn for loop, được gọi là **for-each loop** **(Range-based for loop)**.

For-each loop được sử dụng để lặp qua các phần tử trên 1 mảng (hoặc các cấu trúc danh sách khác như **vectors**, **linked lists**, **trees**, và **maps**).

Cú pháp vòng lặp for-each

```
for (element_declaration : array)
		statement;
```

Trong đó:

- **element_declaration**: là một khai báo biến hoặc tham chiếu có kiểu trùng với kiểu của các phần tử trong array. Thường sử dụng từ khóa auto.
- **array**: mảng hoặc cấu trúc danh sách cần lặp.
- **statement**: câu lệnh đơn hoặc khối lệnh.

Ví dụ: sử dụng **for-each loop** để truy cập vào từng phần tử của mảng:

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

int main()
{
	int arr[] = { 14, 3, 6, 27, 12 };
	for (int item: arr)
	{
		// biến item đại diện cho phần tử mảng ở mỗi vòng lặp
		cout << item << " ";
	}
	cout << endl;

	return 0;
}
```

Output: 14 3 6 27 12

Trong chương trình trên, vòng lặp **for-each** sẽ lặp qua từng phần tử trong mảng **arr**, gán giá trị phần tử hiện tại vào biến **item**. Biến **item** nên có cùng kiểu dữ liệu với phần tử trong mảng **arr**.

> **Chú ý:** Biến **element_declaration** đại diện cho phần tử của **array** ở vòng lặp hiện tại, **không phải** là chỉ số của **array**.

Vòng lặp for-each và từ khóa auto

Vì biến **element_declaration** nên có cùng kiểu dữ liệu với phần tử trong mảng, nên cách tốt nhất là sử dụng từ khóa **auto** để khai báo. **Compiler** sẽ tự động xác định kiểu cho biến.

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

int main()
{
	int arr[] = { 14, 3, 6, 27, 12 };
	for (auto item: arr) // compiler tự động xác định kiểu cho item
	{
		cout << item << " ";
	}
	cout << endl;

	return 0;
}
```

Output: 14 3 6 27 12

Vòng lặp for-each và tham chiếu

Trong các ví dụ trên, vòng lặp **for-each** sử dụng các **biến giá trị** cho mỗi lần lặp:

```
int arr[] = { 14, 3, 6, 27, 12 };
for (auto item: arr) // item là 1 bản copy của phần tử hiện tại
{
	cout << item << " ";
}
```

Với phương pháp này, các biến sẽ được sao chép giá trị của phần tử hiện tại trong mảng sau mỗi lần lặp. Điều này gây giảm hiệu suất và tốn vùng nhớ khi sao chép. Để khắc phục trường hợp này, ta sử dụng biến tham chiếu.

Sử dụng biến tham chiếu cho vòng lặp for-each giúp CPU **truy cập trực tiếp** vào phần tử của mảng, không mất thời gian và vùng nhớ để **khởi tạo bản sao** cho biến. Tuy nhiên, tham chiếu có thể làm **thay đổi giá trị** của phần tử mảng.

```
int arr[] = { 14, 3, 6, 27, 12 };
for (auto &item: arr) // item là 1 biến tham chiếu đến phần tử hiện tại
{
	cout << item << " ";
}
```

Trường hợp không muốn thay đổi giá trị của phần tử trong mảng, bạn có thể sử dụng **tham chiếu hằng (const reference)**.

```
int arr[] = { 14, 3, 6, 27, 12 };
for (const auto &item: arr) // item là 1 tham chiếu hằng đến phần tử hiện tại
{
	cout << item << " ";
}
```

> **Chú ý:** Sử dụng **tham chiếu (reference)** hoặc **tham chiếu hằng (const reference)** cho biến khai báo trong vòng lặp for-each vì lý do hiệu suất.

Vòng lặp for-each không làm việc với con trỏ mảng

Để lặp qua 1 mảng, vòng lặp **for-each** phải biết được kích thước của mảng đó. Con trỏ đến 1 mảng không cho biết kích thước của mảng đó, nên vòng lặp for-each sẽ không làm việc.

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

void printArray(int arr[])
{
	for (const auto &item : arr) // lỗi biên dịch vì arr chỉ là 1 con trỏ
```

```
	{
		cout << item << " ";
	}
}

int main()
{
	int arr[] = { 14, 3, 6, 27, 12 };
	printArray(arr);

	return 0;
}
```

> **Chú ý:** Vòng lặp for-each **không** làm việc với con trỏ đến một mảng.

Xem chỉ số phần tử hiện tại trong vòng lặp for-each?

Vòng lặp **for-each không** cung cấp phương thức trực tiếp nào để xem được chỉ số phần tử hiện tại. Với những yêu cầu liên quan đến chỉ số phần tử, bạn có thể sử dụng **VÒNG LẶP FOR TRONG C++** (For statements)

Kết luận

Qua bài học này, bạn đã biết được cách sử dụng **Vòng lặp for each trong C++11** (For each loops). Vòng lặp for-each không chỉ làm việc với fixed arrays, nó làm việc với nhiều loại cấu trúc danh sách khác như vectors, linked lists, trees, và maps. Những cấu trúc này sẽ được giới thiệu trong các bài học về sau.

> **Chú ý:** Vòng lặp for-each chỉ hoạt động từ **C++11**, các compiler cũ sẽ không được hỗ trợ.

Trong bài tiếp theo, mình sẽ giới thiệu cho các bạn **LỚP DỰNG SẴN ARRAY 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**".