Bài 39: TỪ KHÓA AUTO TRONG C++11.(THE AUTO KEYWORD)

Xem bài học trên website để ủng hộ Kteam: Từ khóa auto trong C++11.(The auto keyword)

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 CÁC THAO TÁC TRÊN MẢNG KÝ TỰ (C-style strings) trong C++.

Hôm nay, mình sẽ giới thiệu cho các bạn về **Từ khóa auto trong C++11 (The auto keyword)**, một từ khóa khá hữu ích trong lập trình C++.

Nội dung

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

- BIẾN TRONG C++ (Variables)
- CƠ BẢN VỀ HÀM và GIÁ TRI TRẢ VỀ (Basics of Functions and Return values)
- TIỀN KHAI BÁO và ĐỊNH NGHĨA HÀM (Forward declarations and Definitions of Functions)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Từ khóa auto trước C++11 (C++98/C++03)
- Từ khóa auto từ C++11

- Kiểu trả về của hàm với từ khóa auto từ C++14

Từ khóa auto trước C++11 (C++98/C++03)

Trong bài BIẾN CỤC BỘ trong C++ (Local variables), bạn đã biết được **biến cục bộ** có **thời gian tự động**, có nghĩa là chúng **được tạo tại thời điểm định nghĩa**, và bị **hủy khi ra khỏi khối lệnh mà biến đó được định nghĩa**.

Trước khi phiên bản C++11 ra đời, từ khóa **auto** là từ khóa **ít được sử dụng nhất** trong C++. Nó được sử dụng để xác định 1 biến có **thời gian tự động**.

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

int main()
{
	// xác định rõ ràng nKteam1 là 1 biến cục bộ
	auto int nKteam1(38); // trước phiên bản C++11

	// mặc định đã là biến cục bộ
	int nKteam2(38); // ý nghĩa tương tự câu lệnh trên

	return 0;
}
```

Tuy nhiên, trình biên dịch có thể **tự động biết được** một biến có phạm vi **cục bộ** hay **toàn cục**. Vì vậy, từ phiên bản **C++11 trở về sau**, sử dụng từ khóa auto để xác định phạm vi của biến là **không cần thiết**, nên chức năng này đã bị **loại bỏ**.

Từ khóa auto từ phiên bản C++11

Khai báo biến với từ khóa auto

Từ phiên bản **C++11 trở về sau**, ý nghĩa của các từ khóa **auto** đã thay đổi, nó trở thành 1 từ khóa khá **hữu ích** và **thường được sử dụng** trong C++.

Xét các câu lệnh bên dưới:

```
char cKteam = 'K';
int nKteam = 1;
float fKteam = 1.0F;
double dKteam = 1.0;
```

Với các câu lệnh trên, bạn phải xác định kiểu dữ liệu của biến được khai báo.

Tuy nhiên, C++ có thể xác định kiểu dữ liệu của giá trị khởi tạo ('K': **char**, **1: int**, **1.0F: float**, **1.0: double**). **Vậy tại sao bạn phải xác định kiểu dữ liệu cho biến, trong khi compiler có thể xác định kiểu dữ liệu cho biến đó thông qua giá trị khởi tạo?**

Từ phiên bản C++11 trở về sau, từ khóa auto được dùng để tự động nhận dạng kiểu dữ liệu thông qua kiểu dữ liệu của giá trị khởi tạo ra nó.

```
auto cKteam = 'K'; // 'K' là kiểu char => cKteam kiểu char
auto nKteam = 1; // 1 là kiểu int => nKteam kiểu int
auto fKteam = 1.0F; // 1.0F là kiểu float => fKteam kiểu float
auto dKteam = 1.0; // 1.0 là kiểu double => dKteam kiểu double
```

Bạn có thể xem kiểu dữ liệu của biến khi sử dụng từ khóa auto thông qua hàm **typeid().name()**.

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

int main()
{
// 'K' là kiểu char => cKteam kiểu char
		auto cKteam = 'K';
		cout << "type of cKteam: " << typeid(cKteam).name() << endl;

// 1 là kiểu int => nKteam kiểu int
```

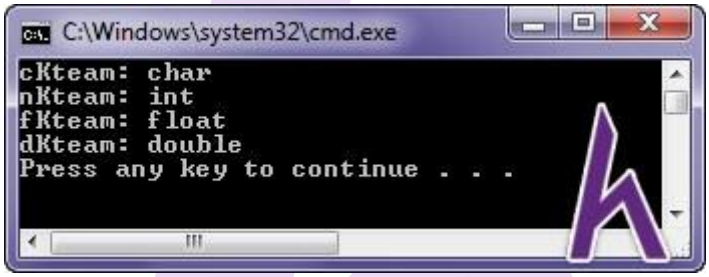
```
	auto nKteam = 1;
	cout << "type of nKteam: " << typeid(nKteam).name() << endl;

	// 1.0F là kiểu float => fKteam kiểu float
	auto fKteam = 1.0F;
	cout << "type of fKteam: " << typeid(fKteam).name() << endl;

	// 1.0 là kiểu double => dKteam kiểu double
	auto dKteam = 1.0;
	cout << "type of dKteam: " << typeid(dKteam).name() << endl;

	return 0;
}
```

Output:



Từ khóa auto có thể được dùng cho giá trị trả về của 1 hàm:

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

int add(int a, int b)
{
	return a + b;
}

int main()
{
	// hàm add() return kiểu int => biến sum kiểu int
	auto sum = (3, 8);
cout << "type of sum: " << typeid(sum).name() << endl;

	return 0;
```

```
}
```

> **Chú ý:** Biến bắt buộc phải có **giá trị khởi tạo** khi sử dụng từ khóa **auto**.

Đối với những kiểu dữ liệu cơ bản, có thể bạn thấy từ khóa **auto** này chưa cần thiết. Trong tương lai, bạn sẽ gặp những kiểu dữ liệu khá **phức tạp** và **dài dòng**, lúc này sử dụng từ khóa auto sẽ **tiết kiệm thời gian**, và giúp code trở nên **đẹp** hơn.

Từ khóa auto không thể sử dụng làm tham số hàm

Từ khóa auto không thể sử dụng làm tham số hàm:

```
int add(auto a, auto b)
{
	return a + b;
}
```

> **Chú ý:** từ khóa **auto** xác định kiểu dữ liệu tại **thời gian biên dịch**, nên nó **không** được sử dụng cho **tham số hàm**.

Nếu bạn muốn viết 1 hàm thực hiện trên nhiều kiểu dữ liệu khác nhau, bạn có thể sử dụng khuôn mẫu hàm (function template). Chi tiết sẽ được hướng dẫn trong bài KHUÔN MẪU HÀM TRONG C++ (Function templates).

Trailing return type syntax in C++11

C++11 hỗ trợ khai báo hàm sử dụng cú pháp **trailing return type.** Trong trường hợp này, từ khóa **auto không dùng để xác định kiểu dữ liệu tự động**, nó chỉ là 1 phần của cú pháp.

Hai cách khai báo hàm bên dưới là như nhau:

```
int add(int a, int b);
auto add(int a, int b) -> int;
```

Cú pháp **trailing return type** về cơ bản sẽ giúp code của bạn trở nên dễ học hơn, ngoài ra nó còn được sử dụng với từ khóa **decltype, lambda expression**, và nhiều tính năng khác trong những phiên bản **C++14/C++17**. Những tính năng này sẽ được hướng dẫn sau.

Kiểu trả về của hàm với từ khóa auto từ C++14

Từ phiên bản **C++14**, từ khóa **auto** có thể dùng để **tự động xác định kiểu trả về của hàm**.

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

// a + b có kiểu int => hàm add() kiểu int
auto add(int a, int b)
{
	return a + b;
}

int main()
{
	// hàm add() return kiểu int => biến sum kiểu int
	auto sum = (3, 8);
	cout << "type of sum: " << typeid(sum).name() << endl;
```

```
        return 0;
}
```

Vì **a** và **b** có kiểu **int**, nên hàm **add()** sẽ có kiểu trả về là **int**.

Kết luận

Qua bài học này, bạn đã biết được cách thức hoạt động Từ khóa auto trong C++11 (The auto keyword).

Trong bài tiếp theo, mình sẽ giới thiệu cho các bạn VÒNG LẶP FOR EACH TRONG C++11 (For each loops).

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**".