Bài: Hằng số trong C++ (Constants)

Xem bài học trên website để ủng hộ Kteam: Hằng số trong C++ (Constants)

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 nắm được các thao tác NHẬP XUẤT & ĐỊNH DẠNG DỮ LIỆU TRONG C++ (Input and Output), và đã biết được những kinh nghiệm cũng như kỹ thuật liên quan đến nhập xuất trong C++.

Hôm nay, bạn sẽ được học một khái niệm mới có liên quan đến biến (variables) và rất hay gặp trong lập trình, đó là: **Hằng số trong C++ (Constants)**

Nội dung

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

- BIẾN TRONG C++ (Variables)
- KIỂU DỮ LIỆU CƠ BẢN TRONG C++

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Tổng quan về hằng số (Constants)
- Hằng số với từ khóa const
- Hằng số với chỉ thị tiền xử lý #define
- Nên định nghĩa hằng số ở đâu

Tổng quan hằng số (Constants)

Ở bài học trước đây, Biến trong C++ (Variables), bạn đã biết biến (variable) là tên của một vùng trong bộ nhớ RAM, được sử dụng để lưu trữ thông tin. Bạn **có thể gán thông tin** cho một biến, **thay đổi thông tin** của biến trong quá trình chạy chương trình, và có thể **lấy thông tin** đó ra để sử dụng.

Ví dụ:

C++:

```
int nVarName1;          // Khai báo biến nVarName1 kiểu int
int nVarName2{ 69 };    // Khởi tạo giá trị 69 cho biến nVarName2 kiểu int
nVarName2 = 70;         // Gán giá trị 70 cho biến nVarName2
```

Tuy nhiên, trong thực tế có những đối tượng mà giá trị của nó không bao giờ thay đổi (Ví dụ: PI = 3.14159, tốc độ âm thanh v = 343.2 m/s, ...). Lúc này, nếu bạn lưu các giá trị này vào biến, rất có khả năng nó sẽ bị thay đổi trong quá trình chạy chương trình. Vậy nên khái niệm **hằng (constant)** trong lập trình đã ra đời.

Hằng số với từ khóa const

Khai báo một hằng số trong C++

Để khai báo một hằng số trong C++, bạn sử dụng từ khóa **const** trước hoặc sau kiểu dữ liệu của biến:

C++:

```
const double PI{ 3.14159 };               // Cách này thông dụng hơn
double const SPEED_OF_SOUND{ 343.2 };     // Cách này ít được sử dụng
```

Chú ý: Hằng số phải được khởi tạo trong lúc khai báo, và giá trị của hằng số sẽ không thể thay đổi trong suốt chương trình.

C++:

```
const double PI;        // Sai vì hằng số phải được khởi tạo khi khai báo
const double PI{ 3.14159 };     // Khởi tạo hằng số PI
PI = 3;                         // Sai vì hằng số không thể thay đổi giá trị
```

Hằng số có thể được khởi tạo giá trị từ một biến thông thường:

C++:

```
int nHeight = 169;
const int HEIGHT{ nHeight };
```

Hằng số có thể sử dụng làm một tham số hàm (function paramater). Cách này được sử dụng phổ biến và khá hữu ích. Cách sử dụng này nhằm 2 mục đích:

- Giúp lập trình viên biết được các tham số hằng sẽ không bị thay đổi giá trị sau lời gọi hàm.
- Đảm bảo các tham số hằng sẽ không bị thay đổi giá trị bên trong hàm.

C++:

```
// Hàm in năm sinh, với tham số hằng nYear
void printYearOfBirth(const int nYear)
{
        cout << "Year of birth: " << nYear << endl;
}
```

Hằng số với chỉ thị tiền xử lý #define

Ngoài cách sử dụng từ khóa **const** để khai báo một hằng số, C++ vẫn cho phép bạn sử dụng chỉ thị tiền xử lý #define để định nghĩa một **macro** sử dụng như một hằng số. Phương pháp này thường thấy trong những hệ thống cũ, hiện nay ít được sử dụng hơn vì những hạn chế của nó (sẽ được đề cập bên dưới).

- **Cú pháp khai báo**:

```
#define identifier substitution_text
```

Khi các tiền xử lý gặp chỉ thị này, bất kỳ lần xuất hiện tiếp theo của '**identifier**' sẽ được thay bằng '**substitution_text**'. Thông thường, '**identifier**' sẽ được viết hoa toàn bộ, và sử dụng gạch dưới "_" để thay cho khoảng trắng.

Ví dụ:

C++:

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

// Định nghĩa một macro bằng chỉ thị tiền xử lý #define
#define YEAR_OF_BIRTH 2016

int main()
{
	cout << "Year of birth: " << YEAR_OF_BIRTH << endl;

	// Khởi tạo một biến integer với giá trị 2016
	int nYear{ YEAR_OF_BIRTH };

	return 0;
}
```

Ở chương trình trên, khi bạn biên dịch, tiền xử lý sẽ thay thế tất cả những macro YEAR_OF_BIRTH thành giá trị 2016. Sau đó, chương trình sẽ được biên dịch bình thường.

Cách sử dụng tiền xử lý #define giống như việc bạn sử dụng trực tiếp một số vào chương trình. Nhưng nó có nhiều ưu điểm hơn như:

- **Trực quan hơn**: khi nhìn vào một tiền xử lý #define, bạn có thể biết mục đích của nó thông qua tên.
- **Tránh hardcode**: khi tiền xử lý #define được sử dụng ở nhiều nơi, bạn chỉ cần thay đổi giá trị ở nơi khai báo nó, những nơi khác sẽ được tự động cập nhật giá trị.

Mặc dù tiền xử lý #define có những ưu điểm như trên, nhưng bạn nên hạn chế hoặc không nên sử dụng tiền xử lý #define làm hằng số cho chương trình. Có 2 lý do bạn không nên sử dụng nó:

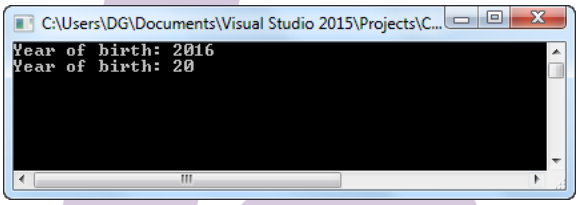
- **Gây khó khăn khi gỡ lỗi (debug) chương trình**: Các **macro** là những chỉ thị tiền xử lý, khi chương trình biên dịch, tiền xử lý sẽ thay thế tất cả những macro trong chương trình thành giá trị của nó. Vì vậy, khi bạn debug chương trình, những macro này sẽ không hiển thị giá trị mà nó nắm giữ, điều này sẽ gây khó khăn cho khi debug chương trình.
- **Các macro luôn có phạm vi toàn cục**: Nghĩa là một macro được #define trong một khối lệnh sẽ ảnh hưởng đến một macro #define trong khối lệnh khác nếu chúng cùng tên. (Chi tiết về biến toàn cục sẽ được hướng dẫn chi tiết trong bài TẦM VỰC CỦA BIẾN TRONG C++ (Variable scope))

Ví dụ:

C++:

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

// Định nghĩa một macro bằng chỉ thị tiền xử lý #define
#define YEAR_OF_BIRTH 2016

// Khai báo prototype
void printYearOfBirth();

int main()
{
	cout << "Year of birth: " << YEAR_OF_BIRTH << endl;

// Định nghĩa một macro trùng tên
#define YEAR_OF_BIRTH 20

	printYearOfBirth();
	return 0;
}

// Định nghĩa hàm
void printYearOfBirth()
{
	cout << "Year of birth: " << YEAR_OF_BIRTH << endl;
}
```

Outputs:



Trong chương trình trên, macro **YEAR_OF_BIRTH** đã bị định nghĩa lại trong thân hàm ***main*()**. Dẫn đến khi gọi hàm printYearOfBirth(), kết quả in ra đã thay đổi thành 20 thay thì 2016.

> **Chú ý:** Nên sử dụng hằng số với từ khóa **const** thay vì chỉ thị tiền xử lý **#define**.

Nên định nghĩa hằng số ở đâu

Một hằng số thường sẽ được sử dụng ở mọi nơi trong chương trình, vì giá trị của nó là không thay đổi (Ví dụ: các hằng số toán học, vật lý, hóa học, ...). Việc định nghĩa một hằng số ở nhiều nơi trong chương trình là điều không nên. Vì vậy, bạn nên **định nghĩa hằng số ở một nơi**, và bạn có thể **sử dụng nó ở toàn chương trình**.

Có nhiều cách để thực hiện việc này, mình sẽ đề cập 2 cách thường được sử dụng:

- **Cách 1:** Ở những chương trình đơn giản, bạn có thể định nghĩa một hằng số ở phạm vi toàn cục để có thể sử dụng ở mọi nơi trong file của bạn. (Chi tiết về biến toàn cục sẽ được giới thiệu chi tiết trong bài TẦM VỰC CỦA BIẾN TRONG C++ (Variable scope)).

Ví dụ:

C++:

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

// Định nghĩa hằng số phạm vi toàn cục
const double PI{ 3.14159 };

// Định nghĩa hàm print PI
void printPI()
{
	cout << "PI = " << PI << endl;
}

int main()
{
	cout << "PI = " << PI << endl;
	return 0;
}
```

- **Cách 2:** Ở những chương trình phức tạp hơn, bạn có thể thực hiện theo các bước:
  1. Tạo một header file định nghĩa các hằng số: để có thể sử dụng ở mọi nơi trong chương trình. (Chi tiết về header file sẽ được giới thiệu chi tiết trong bài LIÊN KẾT NHIỀU FILE TRONG CHƯƠNG TRÌNH C++ (Header files)).
  2. Tạo một namespace bên trong file header: để có thể phân biệt nếu có một biến khác trùng tên với hằng số của bạn. (Chi tiết về namespace sẽ được giới thiệu chi tiết trong bài KHÔNG GIAN TÊN TRONG C++ (Namespaces)).
  3. Định nghĩa tất cả các hằng số bên trong namespace.
  4. #include file header ở nơi bạn cần sử dụng hằng số.

Ví dụ:

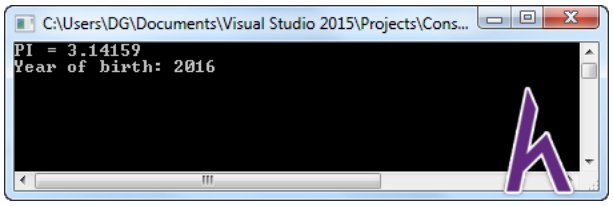
C++:

```
// File Constants.h
#ifndef _CONSTANTS_
#define _CONSTANTS_

// Định nghĩa namespace chứa các hằng số
namespace constants
{
	const double PI{ 3.14159 };
	const double SPEED_OF_SOUND{ 343.2 };   // Tốc độ âm thanh 343.2 m/s
	const int YEAR_OF_BIRTH{ 2016 };
	// ...
}

#endif // !_CONSTANTS_
```

C++:

```
// File main.cpp
#include <iostream>
#include "Constants.h"  // include header file
using namespace std;
//using namespace constants;

int main()
{
	// Sử dụng toán tử phân giải phạm vi :: để truy cập hằng số từ namespace
	cout << "PI = " << constants::PI << endl;
cout << "Year of birth: " << constants::YEAR_OF_BIRTH << endl;
	return 0;
}
```

Outputs:



Kết luận

Qua bài học này, bạn đã nắm được các loại Hằng số trong C++ (Constants), và đã biết được những kinh nghiệm cũng như thắc mắc liên quan đến việc khởi tạo và sử dụng hằng số một cách hiệu quả.

Ở bài tiếp theo, bạn sẽ được học về TOÁN TỬ CƠ BẢN TRONG C++ (Operators), là tiền đề để bạn có thể giải được các bài toán trong lập trình.

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**".