Bài: Biến trong C++ (Variables in C++)

Xem bài học trên website để ủng hộ Kteam: Biến trong C++ (Variables in C++)

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, GHI CHÚ TRONG C++ (Comments in C++), bạn đã đã nắm được các loại comment trong C++, và đã biết sử dụng nó như thế nào cho hợp lý.

Hôm nay, mình sẽ giới thiệu các bạn về **Biến trong C++ (Variables)**.

Nội dung:

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Biến trong C++
- Khởi tạo biến trong C++ (Defining a variable)
- Định nghĩa biến ở đâu (Where to define variables)

Biến trong C++

Giống như toán học, trong lập trình, các bạn cũng sẽ giải những bài toán.

Ví dụ: Bạn có bài toán giải phương trình bậc nhất: 2x - 6 = 0. Và bạn viết một chương trình để giải nó, như thế này:

C++:

```
#include <iostream>
int main()
{
	// Xuất thông báo "2x - 6 = 0" ra màn hình console
	std::cout << "2x - 6 = 0" << std::endl;

	// Xuất kết quả nghiệm x
	std::cout << "x = " << (6 / 2) << std::endl;

	return 0;
}
```

Nhìn có vẻ đã ổn, bây giờ bạn tiếp tục muốn giải một phương trình bậc nhất khác: 6x + 9 = 0.

Vấn đề phát sinh từ đây, khi bạn muốn giải những bài toán với những số khác nhau, bạn phải viết lại nhiều lần. Nhưng bạn muốn chỉ viết chương trình một lần nhưng sử dụng trong mọi trường hợp, vậy nên khái niệm **biến** trong lập trình đã ra đời.

> Trong lập trình, **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, và có thể lấy thông tin đó ra để sử dụng. Khi một biến được khai báo, một vùng trong bộ nhớ sẽ dành cho các biến.

Bài học hôm nay, mình chỉ đề cập đến các biến số nguyên (integer variables). Số nguyên là các số nguyên dương (1, 2, 3, ...), các số đối (-1, -2, -3, ...) và số 0. Biến số nguyên (integer variables) là những biến dùng để lưu trữ các số nguyên.

Ví dụ:

C++:

```
// Khai báo biến số nguyên nVarName
// Giả sử nVarName được cấp vùng nhớ tại địa chỉ 0x0069
int nVarName;
```

Đây gọi là một câu lệnh khai báo, khi chương trình được chạy, đến dòng lệnh này, một vùng trong bộ nhớ RAM sẽ được cấp cho biến nVarName này. Ví dụ trong trường hợp này, biến nInterger được cấp một vùng nhớ tại địa chỉ 0x0069 trong RAM, vậy mỗi khi chương trình chạy đến dòng lệnh nào chứa biến nInterger, chương trình sẽ vào vùng nhớ 0x0069 để lấy giá trị của nó.

Sau khi một biến được khai báo, bạn muốn biến nVarName có một giá trị để sử dụng, câu lệnh gán với toán tử gán = (**assignment operator)** sẽ làm việc đó.

Ví dụ:

C++:

```
// Gán giá trị 96 cho vùng nhớ tại địa chỉ 0x0069
nVarName = 96;
```

Khi chương trình chạy đến câu lệnh này, vùng nhớ tại địa chỉ 0x0069 sẽ được gán giá trị 96.

C++:

```
// In giá trị biến nVarName tại địa chỉ 0x0069 ra màn hình
std::cout << nVarName;
```

Khởi tạo biến trong C++ (Defining a variable)

Ngay khi biến được định nghĩa, bạn lập tức có thể cung cấp một giá trị cho biến. Đó gọi là biến khởi tạo. Trong C++, có 2 cách cơ bản để khởi tạo 1 biến:

C++:

```
// Khởi tạo sao chép giá trị cho biến với toán tử gán =
int nVarName = 69;      // copy initialization
// Khởi tạo trực tiếp giá trị cho biến với dấu ngoặc đơn ()
int nVarName(69);       // direct initialization
```

Mặc dù khởi tạo trực tiếp giống như một lời gọi hàm, nhưng compiler có thể phân biệt được giữa các biến và các function để có thể xử lý chính xác.

Uniform initialization in C++11

Khởi tạo sao chép và khởi tạo trực tiếp chỉ sử dụng cho một số loại biến, bạn không thể sử dụng để khởi tạo một biến chứa danh sách các giá trị. **Vì vậy, C++ 11 đã cung cấp một cơ chế khởi tạo sử dụng cho tất cả các loại dữ liệu**, gọi là **khởi tạo đồng nhất (Uniform initialization).**

C++:

```
// Uniform initialization
int nVarName{69};       // Khởi tạo giá trị 69
int nVarName{}; // Khởi tạo giá trị 0 (hoặc empty, tùy kiểu dữ liệu)

// Không như khởi tạo trực tiếp và khởi tạo sao chép, Compiler sẽ ném một
// thông báo lỗi nếu giá trị khởi tạo đồng nhất không cùng kiểu
int nVarName{6.9};
```

Khởi tạo đồng nhất (Uniform initialization) cũng được gọi là **danh sách khởi tạo** (**list initialization**) sẽ được hướng dẫn chi tiết từ bài **Mảng 1 chiều (Arrays).**

Chú ý:

- Không như một số ngôn ngữ lập trình khác, C/C++ không tự động khởi tạo một giá trị cho biến. Khi bạn khai báo một biến mà không khởi tạo hoặc gán giá trị cho biến đó, thì giá trị của biến đó có thể là một giá trị rác nào đó trong vùng nhớ.
- Vì vậy, nếu bạn sử dụng một biến mà chưa được khởi tạo, chương trình sẽ cho ra những kết quả không mong muốn (hoặc có thể gặp lỗi ngay lúc compile chương trình).

→ Bạn nên khởi tạo biến ngay khi khai báo nếu có thể.

Gán giá trị cho biến trong C++ (Variable assignment):

C++:

```
int nVarName;
nVarName = 96;  // Gán giá trị 96 cho biến nVarName
```

Định nghĩa nhiều biến (Defining multiple variables):

C++ cho phép bạn định nghĩa nhiều biến cùng kiểu dữ liệu trong một câu lệnh, cách nhau bởi dấu phẩy "**,**".

Ví dụ:

C++:

```
// Các câu lệnh bên dưới là như nhau
int nKteam1;
int nKteam2;
int nKteam3, nKteam4;
```

Chú ý:

- Một số sai lầm các lập trình viên mới hay mắc phải:

C++:

```
int nKteam1 = 1, nKteam2 = 2;   // copy initialization
int nKteam3(3), nKteam4(4);     // direct initialization
int nKteam5{5}, nKteam6{6};     // uniform initialization
```

- Chỉ nên định nghĩa nhiều biến trong một câu lệnh. Không nên khởi tạo nhiều biến trong một câu lệnh, vì có thể gây nhầm lẫn. **Ví dụ:**

C++:

```
// Định nghĩa nhiều biến trong một câu lệnh
int nKteam1, int nKteam2;       // Sai: Compile error
int nKteam3, nKteam4;   // Đúng

// Định nghĩa nhiều kiểu dữ liệu trong một câu lệnh
int nKteam1, double dKteam2;    // Sai: Compile error
int nKteam3; double dKteam4;    // Đúng: Không nên

// Đúng và nên viết
int nKteam3;
double dKteam4;
```

Định nghĩa biến ở đâu (Where to define variables)

Trong compiler C cũ bắt buộc lập trình viên khai báo tất cả các biến ở đầu một hàm. Phong cách này ngày nay đã lỗi thời.

C++ cho phép định nghĩa các biến ở **bất kỳ đâu** trong hàm, khuyến khích gần nơi sử dụng biến đó. Một số lý do nên định nghĩa biến gần nơi sử dụng:

- Định nghĩa tất cả biến ở đầu hàm sẽ khó xác định được ý nghĩa của biến đó cho đến khi tìm ra nơi sử dụng biến đó.
- Định nghĩa biến ở nơi sử dụng sẽ chắc chắn rằng biến này không ảnh hưởng bởi những dòng lệnh phía trên, giúp thu hẹp phạm vi ảnh hưởng tốt hơn.
- Giảm thiểu khả năng biến định nghĩa mà không sử dụng, hoặc sử dụng biến chưa được khởi tạo (uninitialized), định nghĩa ngay khi sử dụng sẽ đảm bảo biến đó sẽ được dùng ngay.

> **Chú ý:** Nên định nghĩa biến càng gần nơi sử dụng càng tốt.

Kết luận

Qua bài học này, bạn đã nắm được Biến trong C++ (Variables), và đã biết nguyên lý hoạt động và một số kinh nghiệm về biến trong C++. Và bài học này đang đề cập về biến của một số nguyên. Trong C++ vẫn còn rất nhiều kiểu dữ liệu khác, bạn sẽ được học nó trong bài học tiếp theo: KIỂU DỮ LIỆU CƠ BẢN TRONG C++ (Data types)

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**".