Bài 23: TIỀN KHAI BÁO VÀ ĐỊNH NGHĨA HÀM (FORWARD DECLARATIONS AND DEFINITIONS OF FUNCTIONS)

Xem bài học trên website để ủng hộ Kteam: Tiền khai báo và Định nghĩa Hàm (Forward declarations and Definitions of Functions)

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 kỹ thuật TRUYỀN THAM CHIẾU TRONG C++ (Passing Arguments by Reference in C++).

Ở những bài học trước, mình đều đặt hàm main() ở cuối file code của chương trình, nhưng mình chưa giải thích tại sao. Vấn đề đó sẽ được giải thích cụ thể trong bài hôm nay: **Tiền khai báo và Định nghĩa Hàm (Forward declarations and Definitions of Functions)**.

Nội dung

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

- CƠ BẢN VỀ HÀM & GIÁ TRỊ TRẢ VỀ (Basics of Function and Return values)
- TRUYỀN GIÁ TRỊ CHO HÀM (Passing Arguments by Value)
- TRUYỀN THAM CHIẾU CHO HÀM (Passing Arguments by Reference)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Lỗi "identifier not found"
- Tiền khai báo và nguyên mẫu hàm (Forward declaration and function prototypes)
- Khai báo và định nghĩa trong C++ (Declarations and definitions in C++)

Lỗi "identifier not found"

Quan sát chương trình bên dưới:

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

int main()
{
	printValue(69);

	return 0;
}

void printValue(int x)
{
	cout << x << endl;
}
```

Bạn mong đợi kết quả hiện lên màn hình sẽ là: "**69**". Nhưng thực tế, khi compile chương trình sẽ báo lỗi:

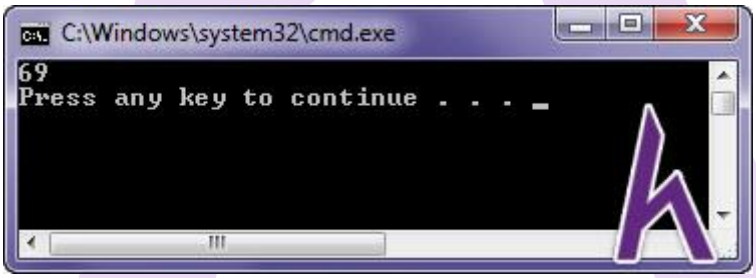
> **'printValue': identifier not found**

Trong C++, các dòng lệnh được **compile từ trên xuống dưới**. Khi compiler đọc đến câu lệnh **printValue(69)** ở dòng 6, compiler không hiểu hàm **printValue(int x)** là gì vì đến dòng 11 hàm mới được định nghĩa. Dẫn đến lỗi **"identifier not found"**.

Để giải quyết vấn đề trên, bạn có thể sắp xếp lại vị trí của hàm trong chương trình:

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

void printValue(int x)
{
	cout << x << endl;
}

int main()
{
	printValue(69);

	return 0;
}
```

Outputs:



Với cách này, khi hàm ***main*()** gọi hàm **printValue(int x)**, compiler đã biết hàm **printValue(int x)** vì nó được định nghĩa bên trên.

Vì đây là một chương trình đơn giản, nên bạn dễ dàng thay đổi thứ tự của các hàm. Tuy nhiên trong một chương trình lớn, có rất nhiều hàm chồng chéo lẫn nhau, bạn không thể biết một cách chính xác thứ tự từng hàm được gọi.

Ví dụ:

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

void sayHello()
{
	cout << "Hello Howkteam.com" << endl;
	sayHi();
}

void sayHi()
{
	cout << "Hi Howkteam.com" << endl;
	sayHello();
}

int main()
{
	sayHello();

	return 0;
}
```

Trong chương trình trên, bạn không thể thay đổi thứ tự các hàm, vì hàm **sayHello()** và hàm **sayHi()** đang gọi chồng chéo lẫn nhau. Nếu định nghĩa hàm **sayHello()** trước, thì sẽ có lỗi **'sayHi': identifier not found**, ngược lại sẽ có lỗi **'sayHello': identifier not found**.

Để giải quyết vấn đề này, bạn cần biết về khái niệm **tiền khai báo (forward declaration)** và **nguyên mẫu hàm (function prototypes)**.

Tiền khai báo và nguyên mẫu hàm (Forward declaration and function prototypes)

Tiền khai báo (forward declaration) giúp compiler **biết được về sự tồn tại** của một định danh (một type, function, hoặc class) **trước khi thực sự định nghĩa nó**.

Khi compiler gặp một lời gọi hàm đã tồn tại câu lệnh tiền khai báo (forward declaration), nó sẽ hiểu rằng đó là một lời gọi hàm và kiểm tra tính hợp lệ của lời gọi hàm (danh sách đối số, kiểu trả về), mặc dù compiler chưa biết hàm đó thực hiện những công việc gì và nó được định nghĩa ở đâu.

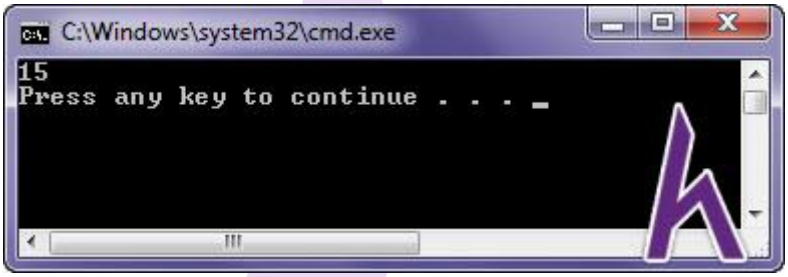
Để tạo tiền khai báo cho một hàm, chúng ta sử dụng một câu lệnh khai báo gọi là **nguyên mẫu hàm (function prototypes)**. Nguyên mẫu hàm bao gồm: **kiểu trả về của hàm**, **tên hàm**, **tham số**, nhưng **không có thân hàm** và được **kết thúc bằng dấu chấm phẩy ";"**.

Ví dụ:

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

// Function prototype includes return type, name, parameters, and semicolon.
// No function body!
void printValue(int x);
int add(int x, int y);

int main()
{
	printValue(add(6, 9));

	return 0;
}

// even though the body of printValue() isn't defined until here
void printValue(int x)
{
	cout << x << endl;
```

```
}

// even though the body of add() isn't defined until here
int add(int x, int y)
{
	return x + y;
}
```

Outputs:



Trong chương trình trên, khi gọi hàm **printValue(add(6, 9))** compiler đã biết được sự tồn tại của 2 hàm trên thông qua câu lệnh tiền khai báo hàm.

Nguyên mẫu hàm **có thể không cần tên tham số cụ thể**, bạn có thể viết lại như sau:

```
// Function prototype includes return type, name, parameters, and semicolon.
// No function body!
void printValue(int);
int add(int, int);
```

> **Kinh nghiệm:** Nguyên mẫu hàm nên có tên của tham số, vì nó sẽ giúp bạn hiểu rõ mục đích của từng tham số mà không cần đi đến phần định nghĩa của hàm.

Khai báo và định nghĩa trong C++ (Declarations and definitions in C++)

Khai báo (declarations) là **giới thiệu sự tồn tại** của một định danh (một type, function, hoặc class) và **mô tả nó**. **Khai báo (declarations)** giúp compiler **biết được về sự tồn tại** của một định danh trước khi chúng ta định nghĩa nó. Compiler **không cần phải cấp vùng nhớ khi khai báo**.

Ví dụ về khai báo:

```
extern int n;
extern int add(int, int);
double div(int, double); // extern can be omitted for function declarations
class foo; // no extern allowed for type declarations
```

Từ khóa **extern** cho biết rằng định danh này được định nghĩa ở chỗ khác. Có nghĩa **compiler không cần phải cấp vùng nhớ** cho định danh này.

Định nghĩa (definitions) là trình bày rõ kiểu dữ liệu hoặc giá trị khởi tạo của biến (variables), cấu trúc và nội dung của hàm (functions), lớp (classes). Compiler **cần phải cấp vùng nhớ khi định nghĩa**.

Ví dụ về định nghĩa:

```
int n;
int add(int a, int b) { return a + b; }
double div(int i, double d){ return i / d; }
class foo {};
```

Một định danh có thể được khai báo nhiều lần, những dòng lệnh bên dưới hợp lệ trong C++:

```
double f(int, double);
double f(int, double);
extern double f(int, double); // the same as the two above
extern double f(int, double);
```

Tuy nhiên, định danh chỉ được định nghĩa một lần duy nhất. Nếu một định danh được định nghĩa 2 lần, compiler sẽ thông báo lỗi.

> **Chú ý:** Khi một định danh được khai báo nhưng không định nghĩa, nếu định danh đó không được sử dụng trong chương trình, thì chương trình vẫn được biên dịch và chạy thành công. Nhưng nếu **sử dụng một định danh khi chưa định nghĩa** chúng, compiler vẫn biên dịch thành công, nhưng **trình liên kết (linker) sẽ thông báo lỗi**.

Kết luận

Qua bài học này: Tiền khai báo và Định nghĩa Hàm (Forward declarations and Definitions of Functions), bạn đã giải quyết được những thắc mắc về hàm trong bài học trước. Hàm (functions) là khái niệm rất quan trọng, bạn cần nắm rõ khái niệm này để tiếp tục học những bài nâng cao hơn về sau.

Trong bài tiếp theo, mình sẽ chia sẻ về cấu trúc điều khiển trong C++: GIỚI THIỆU VỀ CẤU TRÚC ĐIỀU KHIỂN (Control flow introduction).

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**".