

Bài 22: TRUYỀN THAM CHIẾU CHO HÀM (PASSING ARGUMENTS BY REFERENCE)

Xem bài học trên website để ủng hộ Kteam: [Truyền Tham Chiếu cho Hàm \(Passing Arguments by Reference\)](#)

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 GIÁ TRỊ TRONG C++ \(Passing Arguments by Value in C++\)](#) trong C++. Kỹ thuật này có rất nhiều nhược điểm, bạn có thể xem lại để nắm rõ hơn.

Để khắc phục được những nhược điểm của phương pháp Truyền Giá Trị, hôm nay, mình sẽ giới thiệu cho các bạn về kỹ thuật **Truyền Tham Chiếu trong C++ (Passing Arguments by Reference in C++)**

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\(Passing Arguments by Value\)](#)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Truyền tham chiếu cho hàm (Passing arguments by reference)
- Truyền tham chiếu hằng (Pass by const reference)
- Tổng kết về phương pháp truyền tham chiếu cho hàm (Passing arguments by reference)

Truyền tham chiếu cho hàm (Passing arguments by reference)

Mặc dù **truyền giá trị cho hàm (Passing arguments by value)** là phương pháp thường được sử dụng nhất, vì tính **linh hoạt và an toàn**. Nhưng nó vẫn có 2 hạn chế:

- Gây **giảm hiệu suất** trong trường hợp đối số là **kiểu cấu trúc (structs)** hoặc **các lớp (classes)**, đặc biệt là nếu hàm đó **được gọi nhiều lần**. Vì mỗi lần gọi hàm đều phải sao chép giá trị của đối số vào tham số của hàm.
- Hàm **chỉ có thể trả về một giá trị duy nhất** bằng câu lệnh **return**. Trong nhiều trường hợp, hàm cần trả về nhiều thông tin hơn, cách này không đáp ứng được.

Phương pháp **Truyền tham chiếu cho hàm (Passing arguments by reference)** ra đời để khắc phục 2 nhược điểm đó.

Trước tiên, bạn cần biết cơ bản về biến tham chiếu:

- Trong C++, **tham chiếu (reference)** là một loại biến hoạt động như một bí danh của biến khác.
- Khai báo bằng cách sử dụng **ký hiệu "&"** giữa kiểu dữ liệu và tên biến.
- **Mọi thay đổi trên biến tham chiếu cũng chính là thay đổi trên biến được tham chiếu.**

Chi tiết về biến tham chiếu sẽ được hướng dẫn chi tiết trong bài [BIẾN THAM CHIẾU \(Reference variables\)](#).

Để **truyền tham chiếu cho hàm (Passing arguments by reference)**, bạn chỉ cần khai báo các **tham số (parameters)** của hàm **dưới dạng tham chiếu (references)**:

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

void callByReferences(int &y)    // y is a reference variable
{
    cout << "y = " << y << endl;

    y = 69;

    cout << "y = " << y << endl;
    // y is destroyed here
}


int main()
{
    int x(1);
    cout << "x = " << x << endl;

    callByReferences(x);

    cout << "x = " << x << endl;

    return 0;
}
```

Outputs:



```
C:\Windows\system32\cmd.exe
x = 1
y = 1
y = 69
x = 69
Press any key to continue . . .
```

Trong chương trình trên, khi hàm **callByReferences(int &y)** được gọi, **y** sẽ **trở thành một tham chiếu đến đối số x**. Mọi thay đổi của biến **y** bên trong hàm **callByReferences(int &y)** cũng chính là thay đổi trên biến **x**.

Chú ý: Khi truyền tham chiếu cho hàm, đối số chỉ có thể là biến (variables).

Trả về nhiều giá trị thông qua tham số đầu ra (Returning multiple values via out parameters)

Đôi khi bạn cần một hàm trả về nhiều giá trị. Tuy nhiên, hàm chỉ có một giá trị trả về. Một trong những cách để hàm trả về nhiều giá trị là sử dụng tham số tham chiếu:

Ví dụ:

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

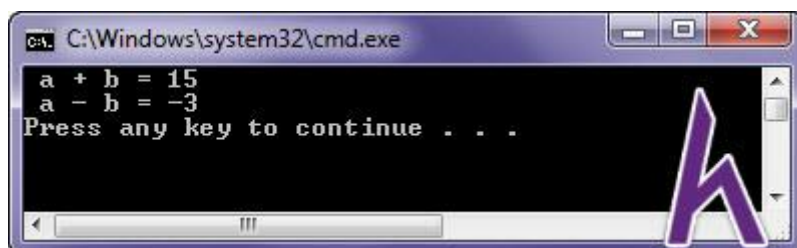
void calculator(int x, int y, int &addOut, int &subOut)
{
    addOut = x + y;
    subOut = x - y;
}

int main()
{
    int a(6), b(9);
    int add, sub;

    // calculator will return the addOut and subOut in variables add and sub
    calculator(a, b, add, sub);

    cout << " a + b = " << add << endl;
    cout << " a - b = " << sub << endl;

    return 0;
}
```

Outputs:

Trong chương trình trên, biến **add** và **sub** truyền vào hàm **calculator** ở dạng tham chiếu, nên giá trị của nó đã thay đổi sau lời gọi hàm.

Chú ý: Trong một hàm, các **tham số** có thể **truyền theo nhiều cách**.

Truyền tham chiếu hằng (Pass by const reference)

Truyền tham chiếu cho hàm đã **giải quyết được vấn đề hiệu suất** của phương pháp **Truyền giá trị (Pass by value)**. Nhưng truyền tham chiếu cho phép hàm **thay đổi giá trị của các đối số (arguments)**, điều này sẽ là **nguy hiểm tiềm ẩn nếu bạn chỉ muốn đọc** các đối số đó (read only).

Nếu bạn biết rằng một **hàm sẽ không thay đổi giá trị của đối số**, nhưng **không muốn truyền giá trị (pass by value)** vì **vấn đề hiệu suất**, giải pháp tốt nhất là **truyền tham chiếu hằng (Pass by const reference)**.

Tham chiếu hằng (const reference) là một tham chiếu mà **không cho phép biến được tham chiếu thay đổi thông qua biến tham chiếu**. Đối số của tham chiếu hằng có thể là **biến số, hằng số hoặc biểu thức**.

Ví dụ:

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

void printValue(const int &value) // value is a const reference
{
    // compile error: a const reference cannot have its value changed!
    value = 69;

    cout << value << endl;
}

int main()
{
    int x(1);

    printValue(x);    // argument is a variable
    //printValue(5);  // argument is a const
    //printValue(x + 5); // argument is a expression

    return 0;
}
```

Trong chương trình trên, vì **value** là tham chiếu hằng, giá trị của nó không thể thay đổi. Nên dòng lệnh **value = 69;** bên trong hàm **printValue(const int &value)** đã tạo ra một lỗi biên dịch.

Ưu điểm khi truyền tham chiếu hằng:

- **Đảm bảo các đối số sẽ không bị thay đổi ngoài ý muốn.** Compiler sẽ thông báo lỗi nếu bạn cố thay đổi một tham chiếu hằng.
- Giúp lập trình viên **biết hàm đó sẽ không làm thay đổi giá trị của đối số.**
- Bạn **không thể truyền đối số là một hằng số (const argument)** cho tham số của hàm là một **biến tham chiếu (non-const reference parameter)**. Nhưng khi **tham số của hàm là một tham chiếu hằng**, bạn **có thể truyền đối số là một biến số hoặc một hằng số** cho nó.

Chú ý: Khi truyền tham chiếu cho hàm, luôn sử dụng một tham chiếu hằng (const reference), trừ khi bạn cần thay đổi giá trị của các đối số

Tổng kết về phương pháp truyền tham chiếu cho hàm (Passing arguments by reference)

Ưu điểm:

- Hàm **có thể thay đổi giá trị của các đối số**. Ngược lại, bạn có thể sử dụng tham chiếu hằng (const reference) nếu không muốn hàm thay đổi giá trị đối số.
- Không mất **thời gian** và **bộ nhớ** để sao chép giá trị của đối số vào tham số của hàm.
- Hàm có thể **trả về nhiều giá trị** thông qua tham chiếu.

Nhược điểm:

- Đối số khi truyền tham chiếu non-const **bắt buộc phải là biến số (variables)**.
- **Khó phân biệt** được tham số khi truyền tham chiếu non-const là tham số input, output hay cả 2.

Khi nào nên sử dụng:

- Khi đối số là **kiểu cấu trúc (structs)** hoặc các **lớp (classes)**. Sử dụng tham chiếu hằng (const reference) nếu không muốn hàm thay đổi giá trị của đối số.
- Khi có **nhu cầu thay đổi giá trị của đối số** sau khi thực hiện hàm.

Khi nào không nên sử dụng:

- Khi đối số có kiểu dữ liệu cơ bản (sử dụng truyền giá trị).

Kết luận

Qua bài học này, bạn đã nắm được phương pháp [Truyền Tham Chiếu trong C++ \(Passing Arguments by Reference in C++\)](#). Và những ưu điểm, nhược điểm, khi nào nên và không nên sử dụng của phương pháp trên.

Qua 3 bài học về hàm, 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 tiếp theo: [TIỀN KHAI BÁO & ĐỊNH NGHĨA HÀM \(Forward declarations and Definitions of Functions\)](#).

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**".