# Bài 9: Hàm cơ bản trong C++

1. **Tầm Quan Trọng Của Hàm Trong Lập Trình C++**
   1. Giới thiệu

Hàm là một trong những khái niệm cơ bản và quan trọng nhất trong lập trình, đặc biệt là trong ngôn ngữ C++. Chúng đóng vai trò trung tâm trong việc tổ chức và quản lý mã nguồn, giúp lập trình viên viết mã hiệu quả hơn và giảm thiểu lỗi.

* 1. Tổ chức mã nguồn

**Tạo cấu trúc rõ ràng**: Hàm giúp chia mã nguồn thành các khối nhỏ, dễ quản lý hơn. Thay vì viết một đoạn mã dài và phức tạp, bạn có thể chia nó thành các hàm nhỏ với các nhiệm vụ cụ thể. Điều này giúp mã nguồn trở nên rõ ràng hơn và dễ đọc hơn.

**Cải thiện khả năng bảo trì**: Khi mã nguồn được tổ chức tốt, việc sửa chữa hoặc cập nhật trở nên dễ dàng hơn. Nếu bạn cần thay đổi một phần của chương trình, bạn chỉ cần cập nhật hàm liên quan, không cần phải lùng sục khắp mã nguồn.

* 1. Tái sử dụng mã nguồn

**Giảm thiểu sự trùng lặp**: Một trong những lợi ích lớn nhất của hàm là khả năng tái sử dụng mã nguồn. Thay vì viết cùng một đoạn mã nhiều lần, bạn chỉ cần viết hàm một lần và gọi nó bất cứ khi nào cần. Điều này giúp tiết kiệm thời gian và giảm nguy cơ lỗi do việc sao chép mã nguồn.

**Khả năng mở rộng**: Khi cần thêm tính năng mới hoặc thay đổi cách tính toán, bạn chỉ cần cập nhật hàm mà không cần phải viết lại mã ở nhiều nơi khác nhau. Điều này làm cho việc mở rộng chương trình trở nên dễ dàng và nhanh chóng hơn.

* 1. Hỗ trợ kiểm thử

**Kiểm thử đơn lẻ**: Hàm cho phép bạn kiểm thử các phần riêng biệt của chương trình một cách độc lập. Bạn có thể kiểm tra từng hàm một để đảm bảo rằng nó hoạt động đúng trước khi tích hợp vào phần lớn hơn của chương trình.

**Phát hiện và sửa lỗi**: Khi bạn kiểm thử các hàm riêng lẻ, việc phát hiện lỗi trở nên dễ dàng hơn. Điều này giúp bạn phát hiện và sửa lỗi sớm hơn trong quá trình phát triển.

* 1. Giúp lập trình viên viết mã tốt hơn

**Cải thiện kỹ năng thiết kế**: Khi sử dụng hàm, bạn phải nghĩ về cách tổ chức và thiết kế mã của mình. Điều này khuyến khích lập trình viên viết mã có cấu trúc tốt hơn và giúp phát triển kỹ năng lập trình tốt hơn.

**Giúp tránh lỗi logic**: Việc chia mã thành các hàm nhỏ giúp lập trình viên dễ dàng kiểm soát và theo dõi các phần của chương trình, từ đó giúp tránh các lỗi logic và cải thiện chất lượng mã.

1. **Cách Định Nghĩa Hàm Trong C++**
   1. Cú pháp (syntax)
      * **Kiểu trả về**: Xác định loại dữ liệu mà hàm sẽ trả về. Nếu hàm không trả về giá trị, kiểu trả về sẽ là void.
      * **Tên hàm**: Tên của hàm, dùng để gọi hàm khi cần.
      * **Danh sách tham số**: Danh sách các tham số mà hàm nhận vào. Nếu hàm không nhận tham số, danh sách này sẽ trống.
      * **Thân hàm**: Phần chứa các câu lệnh thực thi của hàm. Trong thân hàm, bạn có thể viết mã để thực hiện một nhiệm vụ cụ thể và, nếu cần, trả về một giá trị.

kiểu\_trả\_về tên\_hàm(danh\_sách\_tham\_số) {

// Thân hàm

// Câu lệnh

// return giá\_trị; // (nếu kiểu trả về khác void)

}

* 1. Ví Dụ Cơ Bản Về Định Nghĩa Và Gọi Hàm
     + Ví dụ 1: Hàm Không Có Tham Số Và Không Trả Về Giá Trị

#include <iostream>

void printMessage() {

    std::cout << "Hello, welcome to learning C++!" << std::endl;

}

int main() {

    printMessage(); *// Gọi hàm*

    return 0;

}

* + - **Giải thích**: Hàm printMessage không nhận tham số và không trả về giá trị. Trong hàm main, chúng ta gọi printMessage() để thực hiện hành động.

**Đề Bài 1: In một thông điệp chào mừng**

* **Yêu cầu**: Định nghĩa một hàm tên là printGreeting không nhận tham số và không trả về giá trị. Hàm này sẽ in ra một thông điệp chào mừng như “Welcome to C++ Programming!” Khi gọi hàm này trong hàm main, bạn sẽ thấy thông điệp trên màn hình.

**Đề Bài 2: In thông báo số lượng sách trong thư viện**

* **Yêu cầu**: Tạo một hàm tên là printBookCount không nhận tham số và không trả về giá trị. Hàm này sẽ in thông báo về số lượng sách trong thư viện, ví dụ: “The library has 100 books.” Gọi hàm này từ hàm main để in thông báo ra màn hình.
  + - Ví dụ 2: Hàm Có Tham Số Và Trả Về Giá Trị

#include <iostream>

int add(int a, int b) {

    return a + b;

}

int main() {

    int sum = add(5, 7); *// Gọi hàm với các tham số*

    std::cout << "Sum: " << sum << std::endl;

    return 0;

}

* + - **Giải thích**: Hàm add nhận hai tham số a và b, tính tổng và trả về kết quả. Trong hàm main, chúng ta gọi add(5, 7) và lưu kết quả vào biến sum, sau đó in kết quả ra màn hình.

**Đề Bài 1: Tính tổng hai số nguyên**

* **Yêu cầu**: Viết một hàm addNumbers nhận hai tham số kiểu int và trả về tổng của chúng. Trong hàm main, gọi hàm addNumbers với hai số nguyên và in kết quả tổng ra màn hình.

**Đề Bài 2: Tính diện tích hình vuông**

* **Yêu cầu**: Tạo một hàm calculateSquareArea nhận một tham số kiểu int đại diện cho chiều dài cạnh của hình vuông và trả về diện tích. Trong hàm main, gọi hàm calculateSquareArea với một giá trị và in diện tích ra màn hình.
  + - Ví dụ 3: Hàm Có Tham Số Và Không Trả Về Giá Trị

#include <iostream>

void printNumber(int number) {

    std::cout << "Number: " << number << std::endl;

}

int main() {

    printNumber(10); *// Gọi hàm với tham số*

    return 0;

}

* + - **Giải thích**: Hàm printNumber nhận một tham số number và in giá trị của nó. Trong hàm main, chúng ta gọi printNumber(10) để in số 10 ra màn hình.

**Đề Bài 1: In thông tin của một sản phẩm**

* **Yêu cầu**: Định nghĩa hàm printProductInfo nhận hai tham số: tên sản phẩm (kiểu string) và giá (kiểu float). Hàm này sẽ in thông tin sản phẩm ra màn hình, ví dụ: “Product: Laptop, Price: $999.99.” Gọi hàm này từ hàm main để kiểm tra kết quả.

**Đề Bài 2: In tất cả các phần tử trong mảng**

* **Yêu cầu**: Tạo một hàm printArrayElements nhận một mảng các số nguyên và kích thước của nó. Hàm này sẽ in tất cả các phần tử của mảng ra màn hình. Trong hàm main, gọi hàm này với một mảng mẫu và kích thước của nó.

1. **Hàm có tham số**
   1. Truyền Tham Số Theo Giá Trị (Pass by Value)
      * **Mô Tả**: Khi truyền tham số theo giá trị, một bản sao của giá trị được truyền vào hàm. Thay đổi giá trị của tham số trong hàm không ảnh hưởng đến giá trị của biến gốc.

void modifyValue(int x)

{

    x = 10;

}

int main()

{

    int num = 5;

    modifyValue(num);

    std::cout << "num = " << num << std::endl; *// num vẫn là 5*

    return 0;

}

**Đề Bài 1: Thay đổi giá trị của biến trong hàm**

* **Yêu cầu**: Định nghĩa một hàm updateValue nhận một tham số kiểu int và thay đổi giá trị của tham số đó thành 50. Trong hàm main, khai báo một biến kiểu int, gọi hàm updateValue với biến này, và in giá trị của biến sau khi gọi hàm. Bạn sẽ thấy rằng giá trị của biến không bị thay đổi ngoài hàm.

**Đề Bài 2: So sánh hai số**

* **Yêu cầu**: Tạo một hàm compareNumbers nhận hai tham số kiểu int và so sánh chúng. Hàm này sẽ in ra “The numbers are equal” nếu hai số bằng nhau, và “The numbers are different” nếu chúng khác nhau. Trong hàm main, gọi hàm compareNumbers với hai số khác nhau và hai số giống nhau để kiểm tra kết quả.
  1. Truyền Tham Số Theo Tham Chiếu (Pass by Reference)
     + **Mô Tả**: Khi truyền tham số theo tham chiếu, hàm nhận một tham chiếu đến biến gốc. Thay đổi giá trị của tham số trong hàm sẽ ảnh hưởng đến giá trị của biến gốc.

void modifyValue(int *&*x) {

    x = 10;

}

int main() {

    int num = 5;

    modifyValue(num);

    std::cout << "num = " << num << std::endl; *// num là 10*

    return 0;

}

**Đề Bài 1: Thay đổi giá trị của biến trong hàm**

* **Yêu cầu**: Viết hàm increaseValue nhận một tham số kiểu int theo tham chiếu và tăng giá trị của tham số đó thêm 10. Trong hàm main, khai báo một biến kiểu int, gọi hàm increaseValue với biến này, và in giá trị của biến sau khi gọi hàm. Bạn sẽ thấy rằng giá trị của biến đã thay đổi.

**Đề Bài 2: Hoán đổi hai giá trị**

* **Yêu cầu**: Tạo một hàm swapValues nhận hai tham số kiểu int theo tham chiếu và hoán đổi giá trị của chúng. Trong hàm main, khai báo hai biến kiểu int, gọi hàm swapValues với hai biến này, và in giá trị của hai biến sau khi hoán đổi.

* 1. Truyền Tham Số Hằng (Constant Parameters)
     + **Mô Tả**: Khi tham số được khai báo là hằng (const), giá trị của tham số không thể bị thay đổi trong hàm. Điều này giúp bảo vệ dữ liệu và tránh lỗi vô tình thay đổi giá trị.

void printValue(*const* int x) {

    std::cout << "Value: " << x << std::endl;

*// x = 10; // Lỗi: không thể thay đổi giá trị của x*

}

int main() {

    int num = 5;

    printValue(num); *// num không bị thay đổi*

    return 0;

}

**Đề Bài 1: In giá trị không thay đổi**

* **Yêu cầu**: Định nghĩa một hàm nhận một tham số kiểu int với từ khóa const và in giá trị của tham số đó ra màn hình. Trong hàm main, khai báo một biến kiểu int và gọi hàm displayValue với biến này. Đảm bảo rằng giá trị của tham số không bị thay đổi trong hàm.

**Đề Bài 2: Tính tổng của các giá trị trong mảng**

* **Yêu cầu**: Viết một hàm sumArray nhận một mảng kiểu int và kích thước của mảng với tham số mảng là hằng (const). Hàm này sẽ tính tổng của tất cả các phần tử trong mảng và trả về kết quả. Trong hàm main, khai báo một mảng, gọi hàm sumArray với mảng và kích thước của nó, và in tổng ra màn hình.
  1. Tham số mặc định
     + **Mô Tả**: Tham số mặc định được khai báo trong phần khai báo của hàm, và chúng được chỉ định giá trị mặc định trong danh sách tham số của hàm.

#include <iostream>

void printMessage(*const* std::string *&*message = "Hello, World!") {

    std::cout << message << std::endl;

}

int main() {

    printMessage(); *// In "Hello, World!"*

    printMessage("Custom Message"); *// In "Custom Message"*

    return 0;

}

* **Quy tắc:**
* Tham số mặc định chỉ được phép sau các tham số bắt buộc trong danh sách tham số của hàm.
* Khi gọi hàm, nếu không truyền giá trị cho tham số mặc định, thì tham số đó sẽ sử dụng giá trị mặc định được chỉ định.
* Nếu có tham số không có giá trị mặc định sau một tham số có giá trị mặc định, bạn sẽ gặp lỗi biên dịch.
* Ví dụ về cách hoạt động của tham số mặc định và tham số bắt buộc trong hàm:

#include <iostream>

using namespace std;

*// Hàm với tham số mặc định*

void exampleFunction(int a, int b = 10) {

    cout << "a: " << a << ", b: " << b << endl;

}

int main() {

*// Gọi hàm với cả hai tham số*

    exampleFunction(5, 20); *// Output: a: 5, b: 20*

*// Gọi hàm chỉ với tham số đầu tiên*

    exampleFunction(5); *// Output: a: 5, b: 10*

    return 0;

}

* Ví Dụ Về Quy Tắc Không Đúng

*// Lỗi biên dịch: Tham số mặc định không thể xuất hiện trước tham số không có giá trị mặc định*

void wrongFunction(int a = 5, int b) {

*// ...*

}

**Đề Bài 1: Tính diện tích hình chữ nhật với tham số mặc định**

* **Yêu cầu**: Định nghĩa hàm calculateArea nhận hai tham số kiểu int: chiều dài và chiều rộng của hình chữ nhật, với chiều rộng có giá trị mặc định là 5. Hàm này sẽ tính diện tích của hình chữ nhật và trả về kết quả. Trong hàm main, gọi hàm calculateArea với một giá trị duy nhất và với cả hai giá trị để kiểm tra kết quả.

**Đề Bài 2: Chào mừng người dùng với tham số mặc định**

* **Yêu cầu**: Viết hàm greetUser nhận một tham số kiểu string đại diện cho tên người dùng, với giá trị mặc định là “User”. Hàm này sẽ in thông điệp chào mừng, ví dụ: “Hello, User!” hoặc “Hello, [Tên]!” Nếu tên được cung cấp. Trong hàm main, gọi hàm greetUser không cung cấp tên và cung cấp tên cụ thể để kiểm tra kết quả

1. **Lưu Ý Khi Làm Việc Với Hàm**
   1. Thiếu kiểu trả về
      * **Mô tả**: Khi định nghĩa một hàm có kiểu trả về khác void, bạn phải đảm bảo rằng hàm trả về một giá trị của đúng kiểu. Nếu không, trình biên dịch sẽ báo lỗi.

int getNumber() {

*// Sai: không trả về giá trị*

}

* 1. Tham số không khớp
     + **Mô tả**: Khi khai báo hàm với tham số cụ thể, bạn cần đảm bảo rằng các lời gọi hàm cung cấp đúng số lượng và loại tham số. Nếu không, bạn có thể gặp lỗi biên dịch hoặc lỗi thực thi.

void printSum(int a, int b);

printSum(10); *// Sai: thiếu tham số thứ hai*

* 1. Lỗi cú pháp trong thân hàm
     + **Mô tả**: Lỗi cú pháp như thiếu dấu ngoặc nhọn hoặc dấu chấm phẩy trong phần thân hàm có thể gây ra lỗi biên dịch.

int add(int a, int b) {

    return a + b *// Thiếu dấu chấm phẩy ở cuối*

}

* 1. Hàm không trả về giá trị khi cần
     + **Mô tả**: Khi bạn khai báo một hàm có kiểu trả về khác void, bạn phải đảm bảo rằng hàm trả về một giá trị. Nếu không, trình biên dịch có thể báo lỗi hoặc hành vi không mong muốn có thể xảy ra.

int calculate() {

*// Sai: không trả về giá trị*

}

* 1. Lỗi khi sử dụng tham số hằng
     + **Mô tả**: Khi bạn khai báo tham số là hằng (const), bạn không thể thay đổi giá trị của tham số đó trong hàm. Nếu bạn cố gắng thay đổi, trình biên dịch sẽ báo lỗi.

1. **Tips**

* **Tên hàm rõ ràng** giúp dễ hiểu mã.
* **Chia nhỏ mã** thành các hàm nhỏ hơn để dễ quản lý.
* **Tránh hàm dài** bằng cách chia chúng thành các hàm con.
* **Sử dụng tham số mặc định** một cách hợp lý.
* **Chú ý đến kiểu trả về** và đảm bảo nó chính xác.
* **Sử dụng tham số hằng (const)** nếu không cần thay đổi.
* **Tái sử dụng hàm** để giảm lặp lại mã.
* **Kiểm tra đầu vào** trong hàm để bảo vệ chương trình.
* **Tài liệu hóa hàm** rõ ràng giúp người khác hiểu và sử dụng hàm hiệu quả.