**Buổi Học về C++: Hàm và Tham số**

**I. Giới Thiệu**

Trình bày mục tiêu của buổi học: Học về hàm, tham số và cách sử dụng chúng.

**II. Phần 1: Khái Niệm và Sử Dụng Hàm**

a. Hàm là gì?

***- Định nghĩa hàm và giải thích vì sao chúng quan trọng trong lập trình.***

Định Nghĩa Hàm:

Hàm trong lập trình là một khối mã có thể được tái sử dụng, được đặt tên và thực hiện một tác vụ cụ thể. Mỗi hàm thường được thiết kế để thực hiện một công việc cụ thể và có thể nhận đầu vào từ bên ngoài, thực hiện các thao tác, và trả về một kết quả (nếu cần). Hàm giúp tạo ra một cấu trúc chương trình rõ ràng và dễ quản lý bằng cách chia nhỏ một nhiệm vụ lớn thành các phần nhỏ hơn.

Cấu trúc của hàm:

- Tên hàm: được viết theo dạng: operator <Ký hiệu toán tử>

- Cú pháp của hàm:

<Kiểu trả về> operator <Ký hiệu của toán tử> (các đối số)

{

// Thân hàm toán tử;

}

- Ví dụ: Hàm toán tử cộng hai số thực bất kỳ được viết như sau:

float operator +(float x, float y)

{

return x + y;

}

Quan Trọng của Hàm trong Lập Trình:

Tái Sử Dụng Mã (Code Reusability):

Hàm cho phép lập trình viên viết một lần và sử dụng nhiều lần. Khi có một khối mã cần được thực hiện nhiều lần trong chương trình, việc đặt nó trong một hàm giúp giảm sự lặp lại và dễ dàng bảo trì.

Tính Tổ Chức (Modularity):

Hàm giúp chia chương trình thành các phần nhỏ hơn và dễ quản lý hơn. Mỗi hàm có thể tập trung vào một nhiệm vụ cụ thể, làm cho mã nguồn trở nên rõ ràng và dễ đọc hơn.

Giảm Độ Phức Tạp (Complexity Reduction):

Bằng cách chia nhỏ một nhiệm vụ lớn thành các hàm nhỏ, chúng ta giảm độ phức tạp của chương trình. Việc này làm cho mã nguồn trở nên dễ hiểu và dễ bảo trì hơn.

Debugging Dễ Dàng (Easy Debugging):

Khi có lỗi, việc xác định lỗi và sửa chúng trở nên dễ dàng hơn với các hàm. Mỗi hàm được kiểm thử độc lập, giúp tìm ra lỗi mà không cần kiểm tra toàn bộ chương trình.

Phân Loại Các Nhiệm Vụ (Task Classification):

Hàm giúp phân loại nhiệm vụ và chức năng trong chương trình. Các hàm có thể được sắp xếp thành các thư viện, modules hoặc namespaces, giúp tổ chức mã nguồn theo cách hợp lý.

Thuận Tiện Cho Đội Ngũ Phát Triển (Team Development):

Trong các dự án phức tạp, nhiều lập trình viên có thể làm việc cùng nhau. Sử dụng hàm giúp phân chia công việc một cách hiệu quả và làm cho việc phát triển chung trở nên thuận tiện.

Tăng Tính Tương Tác (Interactivity):

Hàm cũng có thể giúp tương tác giữa các phần của chương trình thông qua việc truyền tham số và nhận kết quả trả về.

Tóm lại, hàm là một yếu tố quan trọng trong lập trình, mang lại tính tái sử dụng, tổ chức, và giảm độ phức tạp, giúp làm cho mã nguồn dễ bảo trì và phát triển.

***- So sánh hàm với khối lệnh thông thường.***

```cpp

// Ví dụ đơn giản về hàm

#include <iostream>

void welcomeMessage() {

std::cout << "Chào mừng đến với buổi học C++ của CLB HIT!\n";

}

int main() {

welcomeMessage(); // Gọi hàm

return 0;

}

```

***b. Khai báo và Định nghĩa Hàm***

***- Cú pháp cơ bản của khai báo và định nghĩa hàm.***

Ví dụ minh họa với các hàm đơn giản như in thông điệp.

```cpp

// Ví dụ về khai báo và định nghĩa hàm

#include <iostream>

// Khai báo hàm

void printMessage();

int main() {

printMessage(); // Gọi hàm

return 0;

}

// Định nghĩa hàm

void printMessage() {

std::cout << "Đây là một ví dụ về hàm!\n";

}

```

***c. Gọi Hàm***

```cpp

// Ví dụ về truyền tham số

#include <iostream>

// Hàm có tham số

void greetUser(std::string name) {

std::cout << "Chào bạn, " << name << "!\n";

}

int main() {

std::string userName = "Alice";

greetUser(userName); // Gọi hàm với tham số

return 0;

}

```

- Cách gọi hàm và truyền tham số.

bool isPrime(int n){

if (n < 2) {

return false;

}

// Số nguyên tố chẵn duy nhất là 2

if (n == 2) {

return true;

}

// Nếu n là số chẵn, không phải là số nguyên tố

if (n % 2 == 0) {

return false;

}

// Kiểm tra các ước số lẻ từ 3 đến căn bậc hai của n

for (int i = 3; i <= sqrt(n); i += 2) {

if (n % i == 0) {

return false;

}

}

***- So sánh giữa việc truyền giá trị và tham chiếu.***

Truyền Giá Trị (Pass by Value)

Đặc Điểm Chính:

Tham số nhận được một bản sao của giá trị được truyền vào hàm.

Bất kỳ thay đổi nào đối với tham số trong hàm không ảnh hưởng đến biến gốc.

Ví Dụ:

cpp

Copy code

#include <iostream>

void incrementValue(int num) {

num++;

}

int main() {

int value = 5;

incrementValue(value);

std::cout << "Giá trị sau khi tăng: " << value << "\n";

return 0;

}

Kết quả: Giá trị sau khi tăng: 5

Tham Chiếu (Pass by Reference)

Đặc Điểm Chính:

Tham số nhận được địa chỉ (tham chiếu) của biến gốc.

Mọi thay đổi đối với tham số trong hàm ảnh hưởng trực tiếp đến giá trị của biến gốc.

Ví Dụ:

cpp

Copy code

#include <iostream>

void incrementValue(int &num) {

num++;

}

int main() {

int value = 5;

incrementValue(value);

std::cout << "Giá trị sau khi tăng: " << value << "\n";

return 0;

}

Kết quả: Giá trị sau khi tăng: 6

So Sánh

Hiệu Suất:

Truyền giá trị có thể hiệu quả hơn trong một số trường hợp vì không có chi phí thêm của việc quản lý địa chỉ.

Bảo Toàn Giá Trị Gốc:

Truyền giá trị giữ nguyên giá trị của biến gốc, không làm thay đổi giá trị nếu có thay đổi trong hàm.

Tham chiếu có thể làm thay đổi giá trị của biến gốc.

Sử Dụng Tài Nguyên:

Tham chiếu không tạo bản sao, giúp tiết kiệm tài nguyên khi xử lý dữ liệu lớn.

Truyền giá trị tốn tài nguyên hơn do cần phải tạo bản sao.

Rủi Ro:

Truyền giá trị an toàn hơn, vì không có nguy cơ thay đổi giá trị ngoài ý muốn.

Tham chiếu có thể tạo ra các lỗi logic nếu không được sử dụng cẩn thận.

Sử Dụng Thích Hợp:

Truyền giá trị thích hợp cho các hàm không cần thay đổi giá trị đối số gốc.

Tham chiếu thích hợp cho việc muốn thay đổi giá trị đối số gốc.

Lựa chọn giữa truyền giá trị và tham chiếu phụ thuộc vào yêu cầu cụ thể của bài toán và hiệu suất mong muốn.

#include <iostream>

void modifyArray(int arr[], int size) {

for (int i = 0; i < size; ++i) {

arr[i] \*= 2;

}

}

int main() {

int myArray[] = {1, 2, 3, 4, 5};

modifyArray(myArray, 5);

for (int i = 0; i < 5; ++i) {

std::cout << myArray[i] << " "; // Kết quả: 2 4 6 8 10

}

return 0;

}

**III. Phần 2: Tham Số Trong Hàm**

***a. Tham số của Hàm***

- Định nghĩa và giải thích ý nghĩa của tham số trong hàm.

- Các loại tham số: bắt buộc, tùy chọn, mặc định.

Định Nghĩa và Ý Nghĩa Của Tham Số Trong Hàm:

Trong lập trình, tham số là các giá trị mà một hàm có thể nhận từ bên ngoài khi được gọi. Các tham số giúp hàm thực hiện công việc của mình dựa trên dữ liệu cụ thể được chuyển vào nó. Điều này tạo ra sự linh hoạt và tái sử dụng mã, vì cùng một hàm có thể được sử dụng với các giá trị khác nhau tùy thuộc vào ngữ cảnh sử dụng.

Ý Nghĩa Cụ Thể của Tham Số:

Chuyển Dữ Liệu:

Tham số chủ yếu được sử dụng để chuyển dữ liệu từ ngoại vi (gọi hàm) vào hàm. Những giá trị này thường là thông tin cần thiết cho hàm thực hiện công việc cụ thể.

Thực Hiện Công Việc Động:

Tham số giúp hàm thực hiện công việc động dựa trên dữ liệu cụ thể mà nó nhận được. Điều này tạo ra khả năng tái sử dụng mã và linh hoạt trong việc sử dụng hàm.

Truyền Thông Tin Giữa Các Phần của Chương Trình:

Tham số cũng được sử dụng để truyền thông tin giữa các phần khác nhau của chương trình. Điều này giúp tạo ra sự tương tác và tính toàn cục trong chương trình.

Các Loại Tham Số:

Tham Số Bắt Buộc (Required Parameters):

Là những tham số mà hàm phải nhận để thực hiện công việc của mình.

Nếu không truyền giá trị cho tham số bắt buộc, chương trình có thể không biên dịch hoặc hàm không thể thực hiện đúng.

void printName(std::string name) {

std::cout << "Tên của bạn là: " << name << "\n";

}

Tham Số Tùy Chọn (Optional Parameters):

Là những tham số mà có thể hoặc không được truyền vào hàm.

Các tham số này thường được gán một giá trị mặc định, và nếu không truyền giá trị, hàm sẽ sử dụng giá trị mặc định.

void showMessage(std::string text, int times = 1) {

for (int i = 0; i < times; ++i) {

std::cout << text << "\n";

}

}

Tham Số Mặc Định (Default Parameters):

Là tham số có giá trị mặc định được đặt trước trong danh sách tham số.

Nếu không có giá trị được truyền cho tham số này, hàm sẽ sử dụng giá trị mặc định.

void greetUser(std::string name, std::string greeting = "Chào bạn!") {

std::cout << greeting << " " << name << "\n";

}

Bằng cách kết hợp các loại tham số này, lập trình viên có thể tận dụng sự linh hoạt và đa dạng trong việc sử dụng hàm, làm tăng tính tái sử dụng mã và dễ bảo trì.

***b. Sử Dụng Tham Số Trong Hàm***

- Minigame: Viết một hàm tính tổng của hai số, sử dụng tham số.

- Bàn luận về cách tham số tác động lên logic của hàm.

**IV. Phần 3: Tham Chiếu và Tham Trị**

***a. Tham Chiếu và Tham Trị là gì?***

- Giải thích khái niệm tham chiếu và tham trị.

“Tham số” là dữ liệu đầu vào của phương thức.

“Tham chiếu” là cách để phương thức lấy dữ liệu.

“Tham trị” là cách tham chiếu để sử dụng khi không có nhu cầu thay đổi giá trị của biến đầu vào

- So sánh cách chúng hoạt động trong hàm.

void

***b. Ví Dụ Thực Hành***

- Minh họa sự khác biệt giữa truyền tham chiếu và tham trị qua một số ví dụ.

- Nhấn mạnh tác động của thay đổi tham chiếu đối với biến gốc.

Để minh họa sự khác biệt giữa truyền tham chiếu và tham trị, cùng xem xét một số ví dụ:

1. Truyền Tham Trị:

#include <iostream>

// Hàm tăng giá trị của một số

void incrementValue(int num) {

num++;

std::cout << "Giá trị bên trong hàm (tham trị): " << num << "\n";

}

int main() {

int value = 5;

std::cout << "Giá trị trước khi gọi hàm: " << value << "\n";

incrementValue(value);

std::cout << "Giá trị sau khi gọi hàm (tham trị): " << value << "\n";

return 0;

}

Kết quả:

Giá trị trước khi gọi hàm: 5

Giá trị bên trong hàm (tham trị): 6

Giá trị sau khi gọi hàm (tham trị): 5

Trong ví dụ này, giá trị của biến value không thay đổi bên ngoài hàm sau khi gọi hàm incrementValue vì chúng ta đã truyền tham trị.

2. Truyền Tham Chiếu:

cpp

Copy code

#include <iostream>

// Hàm tăng giá trị của một số sử dụng tham chiếu

void incrementValueWithReference(int &num) {

num++;

std::cout << "Giá trị bên trong hàm (tham chiếu): " << num << "\n";

}

int main() {

int value = 5;

std::cout << "Giá trị trước khi gọi hàm: " << value << "\n";

incrementValueWithReference(value);

std::cout << "Giá trị sau khi gọi hàm (tham chiếu): " << value << "\n";

return 0;

}

Kết quả:

Giá trị trước khi gọi hàm: 5

Giá trị bên trong hàm (tham chiếu): 6

Giá trị sau khi gọi hàm (tham chiếu): 6

Lần này, giá trị của biến value đã thay đổi bên ngoài hàm sau khi gọi hàm incrementValueWithReference vì chúng ta đã truyền tham chiếu.

Nhấn Mạnh Tác Động của Thay Đổi Tham Chiếu Đối với Biến Gốc:

Khi chúng ta truyền tham chiếu, mọi thay đổi được thực hiện trên tham chiếu đều ảnh hưởng trực tiếp đến biến gốc. Điều này làm cho tham chiếu trở thành một cách mạnh mẽ để thực hiện sự tương tác giữa hàm và biến bên ngoài hàm.

Hãy nhớ rằng sự lựa chọn giữa tham chiếu và tham trị phụ thuộc vào yêu cầu cụ thể của bài toán và mong muốn về hiệu suất.

**V. Tổng Kết và Bài Tập**

a. Tổng hợp những điểm chính đã học.

b. Giao bài tập: Viết một chương trình sử dụng hàm với tham số tham chiếu và tham trị để thực hiện một nhiệm vụ cụ thể.