## **Bài 13: Hàm ẩn danh Lambda Function trong C++**

## **Khái niệm lambda function**

Lambda function (hay hàm ẩn danh) trong C++ là một hàm không có tên, cho phép bạn định nghĩa hàm ngay tại chỗ mà không cần phải khai báo trước. Lambda function được giới thiệu trong C++11 và thường được sử dụng để viết mã ngắn gọn, dễ đọc hơn, đặc biệt trong các tình huống mà hàm chỉ cần sử dụng một lần hoặc cần được truyền như một đối số.

### Cú pháp cơ bản:

[capture](parameters) -> return\_type {

// Thân hàm

}

## **Ví dụ lambda function**

### 1. Capture

Phần này xác định cách mà lambda function "bắt" các biến từ phạm vi bên ngoài. Có một số cách để thực hiện:

**[]**: Không bắt biến nào. Bạn không thể sử dụng bất kỳ biến nào từ phạm vi bên ngoài trong lambda.

auto lambda = []() {

    std::cout << "Hello!" << std::endl;

};

**[&]**: Bắt tất cả các biến trong phạm vi bên ngoài theo tham chiếu. Điều này có nghĩa là nếu bạn thay đổi giá trị của một biến trong lambda, biến đó cũng sẽ thay đổi bên ngoài.

int a = 5;

auto lambda = [&a]() {

    a += 10; *// Thay đổi giá trị của a*

};

**[=]**: Bắt tất cả các biến theo giá trị. Lambda sẽ tạo bản sao của tất cả các biến được sử dụng trong thân hàm, do đó, việc thay đổi chúng trong lambda không ảnh hưởng đến các biến bên ngoài.

int a = 5;

auto lambda = [=]() {

*// Không thay đổi giá trị của a ở đây*

};

**[x, &y]**: Bắt biến x theo giá trị và biến y theo tham chiếu. Bạn có thể chỉ định các biến cụ thể để bắt.

int x = 10, y = 20;

auto lambda = [x, &y]() {

    y += 5; *// Thay đổi y, nhưng x không thay đổi*

};

### 2. Parameters

Phần này định nghĩa các tham số mà lambda function nhận vào. Cú pháp tương tự như các hàm thông thường.

auto lambda = [](int a, int b) {

    return a + b;

};

Ở ví dụ trên, lambda function nhận hai tham số a và b.

### 3. Return Type

Bạn có thể chỉ định kiểu trả về của lambda function bằng cách sử dụng dấu ->. Tuy nhiên, trong nhiều trường hợp, trình biên dịch có thể tự suy luận được kiểu trả về, vì vậy bạn có thể bỏ qua phần này.

auto lambda = [](int a, int b) -> int {

    return a + b; *// Trả về kiểu int*

};

### 4. Body

Thân hàm là nơi bạn viết mã thực hiện. Tất cả các phép toán mà bạn muốn thực hiện sẽ được viết ở đây.

auto lambda = [](int a, int b) {

    return a \* b; *// Thực hiện phép nhân*

};

### Cách gọi lambda function

#### Gọi Lambda Ngay Lập Tức

#### Gán Lambda vào Biến và Gọi Sau

#### Truyền Lambda như Một Tham Số

## **Hoạt động lambda function**

Khi một lambda function được gọi, nó hoạt động như một hàm thông thường. Bạn có thể sử dụng nó để thực hiện các phép toán, xử lý dữ liệu, hoặc bất kỳ tác vụ nào khác. Lambda có thể "bắt" các biến từ phạm vi bên ngoài, cho phép chúng tương tác với các biến đó.

Ví dụ:

#include <iostream>

int main() {

int x = 5;

auto lambda = [&x]() {

x += 10; // Thay đổi giá trị của x

};

lambda();

std::cout << "x after lambda: " << x << std::endl; // Kết quả: 15

return 0;

}

## **Quản lí bộ nhớ lambda function**

### Bộ nhớ cho lambda function được quản lý tự động. Nếu lambda không bắt các biến theo tham chiếu, nó sẽ tạo một bản sao của chúng. Nếu lambda bắt biến theo tham chiếu, nó sẽ giữ tham chiếu tới biến đó. Khi lambda không còn được sử dụng, bộ nhớ sẽ tự động giải phóng.

Ví dụ:

#include <iostream>

int main() {

    int x = 5;

    auto lambda = [&x]() {

        x += 10; *// Thay đổi giá trị của x*

    };

    lambda();

    std::cout << "x after lambda: " << x << std::endl; *// Kết quả: 15*

    return 0;

}

## **Ứng dụng lambda function**

### Lambda functions thường được sử dụng trong các tình huống như:

* Sắp xếp và lọc dữ liệu trong các container (STL).
* Để sử dụng trong các thuật toán như std::sort, std::for\_each.
* Xử lý sự kiện trong lập trình GUI hoặc lập trình bất đồng bộ.

## **Nhược điểm lambda function**

### Mặc dù lambda function mang lại nhiều lợi ích, nhưng cũng có một số nhược điểm cần lưu ý:

* **Độ phức tạp**: Với các lambda phức tạp, mã có thể trở nên khó đọc và bảo trì.
* **Khó khăn trong việc gỡ lỗi**: Do không có tên, việc gỡ lỗi lambda có thể khó khăn hơn so với các hàm truyền thống.
* **Quản lý phạm vi**: Việc quản lý biến bắt theo tham chiếu cần cẩn trọng, vì nếu biến đó ra khỏi phạm vi, sẽ dẫn đến lỗi.

## **Lưu ý**

### 1. **Capture Biến**

* **Capture Tất Cả**: Dùng [=] để capture tất cả biến bên ngoài theo giá trị. Nếu bạn muốn thay đổi biến, bạn cần capture theo tham chiếu bằng [&].
* **Capture Cụ Thể**: Có thể chỉ định biến cụ thể trong dấu ngoặc vuông, ví dụ: [a, &b] để capture a theo giá trị và b theo tham chiếu.

### 2. **Không Gọi Biến Không Capture**

* Nếu bạn sử dụng biến bên ngoài mà không capture, trình biên dịch sẽ báo lỗi. Đảm bảo rằng bạn đã capture các biến cần thiết.

### 3. **Kiểu Trả Về**

* Nếu hàm lambda không có kiểu trả về rõ ràng, trình biên dịch sẽ tự động suy luận kiểu trả về. Để rõ ràng hơn, bạn có thể chỉ định kiểu trả về bằng cách sử dụng cú pháp ->.

### 4. **Tránh Sử Dụng Lambda Quá Phức Tạp**

* Mặc dù lambda có thể rất mạnh mẽ, nhưng nếu chúng trở nên quá phức tạp hoặc dài, có thể tốt hơn là định nghĩa một hàm thông thường để giữ mã nguồn dễ đọc và dễ hiểu hơn.

### 5. **Lambda và Tham Chiếu**

* Nếu bạn capture theo tham chiếu, hãy chắc chắn rằng biến đó còn tồn tại khi lambda được gọi. Nếu biến đó ra khỏi phạm vi, bạn sẽ gặp lỗi truy cập bộ nhớ.

## Luyện tập

### Bài tập 1: Kiểm tra số chẵn hay lẻ

Viết một lambda function để kiểm tra xem một số nguyên có phải là số chẵn hay không. In ra kết quả kiểm tra cho một số cho trước.

### Bài tập 2: Tính bình phương của một số

Viết một lambda function để tính bình phương của một số nguyên và in ra kết quả.

### Bài tập 3: Tính tổng từ 1 đến n

Viết một lambda function để tính tổng các số nguyên từ 1 đến n và in ra kết quả.