Bài 4: Extern - Static - Volatile

# Extern

Khái niệm Extern trong ngôn ngữ lập trình C được sử dụng để thông báo rằng một biến hoặc hàm đã được khai báo ở một nơi khác trong chương trình hoặc trong một file nguồn khác. Điều này giúp chương trình hiểu rằng biến hoặc hàm đã được định nghĩa và sẽ được sử dụng từ một vị trí khác, giúp quản lý sự liên kết giữa các phần khác nhau của chương trình hoặc giữa các file nguồn.

Ví dụ:

File main.c

| #**include** <stdio.h>  **int** value = 90;  **extern** **void** **display**();  **int** **main**()  {  printf("hello\n");  display(); } |
| --- |

File other.c

| #**include** <stdio.h>  **extern** **int** value; **void** **display**() {  printf("value: %d\n", value); } |
| --- |

Ứng dụng:

* Chia sẻ biến và hàm giữa các file nguồn:
* Extern cho phép bạn chia sẻ biến và hàm giữa nhiều file nguồn trong một chương trình.
* Điều này hữu ích khi bạn muốn tách chương trình thành các phần nhỏ để quản lý dễ dàng hơn.
* Chia sẻ biến và hàm giữa các module hoặc thư viện:
* Extern có thể được sử dụng để kết nối các module hoặc thư viện trong một dự án lớn.
* Khai báo hàm trong trường hợp định nghĩa sau:
* Nếu bạn muốn sử dụng một hàm trước khi nó được định nghĩa trong mã nguồn, bạn có thể sử dụng extern để khai báo hàm.
* Biến toàn cục giữa các tệp nguồn:
* Khi có một biến toàn cục được sử dụng trong nhiều file nguồn, extern giúp các file nguồn biết về sự tồn tại của biến đó.
* Chia sẻ hằng số giữa các file nguồn:
* Nếu bạn có một hằng số được sử dụng ở nhiều nơi, bạn có thể sử dụng extern để chia sẻ giá trị của hằng số đó giữa các file nguồn.

# Static

## Static local variables

Khi static được sử dụng với local variables (biến cục bộ - khai báo biến trong một hàm), nó giữ giá trị của biến qua các lần gọi hàm và giữ phạm vi của biến chỉ trong hàm đó.

Ví dụ:

| #**include** <stdio.h>  **void** **exampleFunction**() {  **static** **int** count = 0; // Biến static giữ giá trị qua các lần gọi hàm  count++;  printf("Count: %d\n", count); }  **int** **main**() {  exampleFunction(); // In ra "Count: 1"  exampleFunction(); // In ra "Count: 2"  exampleFunction(); // In ra "Count: 3"  **return** 0; } |
| --- |

Ứng dụng:

Lưu trữ trạng thái giữa các lần gọi hàm: Sử dụng biến static để theo dõi trạng thái trạng thái giữa các lần gọi hàm mà không cần sử dụng biến toàn cục.

## Static global variables

Khi static được sử dụng với global variables ( biến toàn cục - khai báo biến bên ngoài hàm), nó hạn chế phạm vi của biến đó chỉ trong file nguồn hiện tại.

Ví dụ:

File main.c

| #**include** <stdio.h>  **extern** **void** **display**(); //extern int s\_g\_value; **extern** **int** g\_value;  **int** **main**() {  printf("hello\n");  g\_value = 40;    display();   **return** 0; } |
| --- |

File other.c:

| #**include** <stdio.h>  **int** g\_value = 30; **static** **int** s\_g\_value = 20;   **void** **display**() {  printf("static global value: %d\n", s\_g\_value);  printf("global value: %d\n", g\_value); } |
| --- |

## Static trong class

Khi một thành viên của lớp được khai báo là static, nó thuộc về lớp chứ không thuộc về các đối tượng cụ thể của lớp đó. Các đối tượng của lớp sẽ chia sẻ cùng một bản sao của thành viên static, và nó có thể được truy cập mà không cần tạo đối tượng. Nó thường được sử dụng để lưu trữ dữ liệu chung của tất cả đối tượng.

Ví dụ:

| #**include** <iostream>  **typedef** **enum** {  red = 0,  blue,  green,  purple,  black,  yellow } Pen\_Color;  **void** **print\_color\_pen**(Pen\_Color color) {  **switch** (color)  {  **case** red:  std::cout << "Red\n";  **break**;  **case** blue:  std::cout << "Blue\n";  **break**;  **case** green:  std::cout << "Green\n";  **break**;    **default**:  **break**;  } }   **class** **PEN** { **public**:  Pen\_Color pen\_color;  **static** **int** pen\_length;   PEN(Pen\_Color color);  Pen\_Color **get\_color**()  {  **return** pen\_color;  }  **void** **set\_length**(**int** length)  {  pen\_length = length;  } };  **int** PEN::pen\_length;  PEN::PEN(Pen\_Color color) {  pen\_color = color;  pen\_length = 10; }   **int** **main**(**int** argc, **char** **const** \*argv[]) {  PEN **blue\_pen**(blue);  PEN **red\_pen**(red);  PEN **green\_pen**(green);   blue\_pen.set\_length(9);   std::cout << "Color: ";  print\_color\_pen(blue\_pen.get\_color());  std::cout << "Length: " << blue\_pen.pen\_length << '\n';   std::cout << "Color: ";  print\_color\_pen(red\_pen.get\_color());  std::cout << "Length: " << red\_pen.pen\_length << '\n';   std::cout << "Color: ";  print\_color\_pen(green\_pen.get\_color());  std::cout << "Length: " << green\_pen.pen\_length << '\n';   **return** 0; } |
| --- |

# Volatile

Từ khóa volatile trong ngôn ngữ lập trình C được sử dụng để báo hiệu cho trình biên dịch rằng một biến có thể thay đổi ngẫu nhiên, ngoài sự kiểm soát của chương trình. Việc này ngăn chặn trình biên dịch tối ưu hóa hoặc xóa bỏ các thao tác trên biến đó, giữ cho các thao tác trên biến được thực hiện như đã được định nghĩa.

Ví dụ:

# Register

Trong ngôn ngữ lập trình C, từ khóa register được sử dụng để chỉ ra ý muốn của lập trình viên rằng một biến được sử dụng thường xuyên và có thể được lưu trữ trong một thanh ghi máy tính, chứ không phải trong bộ nhớ RAM. Việc này nhằm tăng tốc độ truy cập. Tuy nhiên, lưu ý rằng việc sử dụng register chỉ là một đề xuất cho trình biên dịch và không đảm bảo rằng biến sẽ được lưu trữ trong thanh ghi. Trong thực tế, trình biên dịch có thể quyết định không tuân thủ lời đề xuất này.

Ví dụ:

| #**include** <stdio.h> #**include** <time.h>  **int** **main**() {  // Lưu thời điểm bắt đầu  **clock\_t** start\_time = clock();   // Đoạn mã của chương trình  **for** (**int** i = 0; i < 1000000; ++i) {  // Thực hiện một số công việc bất kỳ  }   // Lưu thời điểm kết thúc  **clock\_t** end\_time = clock();   // Tính thời gian chạy bằng miligiây  **double** time\_taken = ((**double**)(end\_time - start\_time)) / CLOCKS\_PER\_SEC;   printf("Thoi gian chay cua chuong trinh: %f giay\n", time\_taken);   **return** 0; } |
| --- |