Dưới đây là bản dịch của tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

## 8 Quy tắc

### 8.1 Một môi trường C tiêu chuẩn

#### Quy tắc 1.1: Chương trình không được vi phạm cú pháp tiêu chuẩn C và các ràng buộc, và không được vượt quá giới hạn dịch của môi trường triển khai.

- [Bảng 3 Hướng dẫn MISRA], [IEC 61508-7: Bảng C.1], [ISO 26262-6: Bảng 1]

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Mở rộng

Chương trình chỉ nên sử dụng những tính năng của ngôn ngữ C và thư viện của nó được chỉ định trong phiên bản tiêu chuẩn đã chọn (xem Mục 3.1).

Tiêu chuẩn cho phép các môi trường triển khai cung cấp các mở rộng ngôn ngữ và việc sử dụng những mở rộng này được quy tắc này cho phép.

Trừ khi sử dụng một mở rộng ngôn ngữ, chương trình không được:

- Chứa bất kỳ vi phạm nào của cú pháp ngôn ngữ được mô tả trong Tiêu chuẩn;

- Chứa bất kỳ vi phạm nào của các ràng buộc do Tiêu chuẩn áp đặt.

Chương trình không được vượt quá các giới hạn dịch do môi trường triển khai áp đặt. Các giới hạn dịch tối thiểu được quy định bởi Tiêu chuẩn nhưng môi trường triển khai có thể cung cấp các giới hạn cao hơn.

##### Lưu ý:

- Một triển khai tuân thủ sẽ tạo ra một chẩn đoán cho các vi phạm cú pháp và ràng buộc, nhưng hãy lưu ý rằng:

- Chẩn đoán không nhất thiết phải là một lỗi mà có thể là một cảnh báo;

- Chương trình có thể được dịch và một tệp thực thi có thể được tạo ra mặc dù có sự vi phạm cú pháp hoặc ràng buộc;

- Một triển khai tuân thủ không cần phải tạo ra chẩn đoán khi một giới hạn dịch bị vượt quá; một tệp thực thi có thể được tạo ra nhưng không được đảm bảo sẽ thực thi đúng.

Dưới đây là bản dịch của phần Rationale từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

### Lý do

Các vấn đề liên quan đến các tính năng ngôn ngữ nằm ngoài các phiên bản được hỗ trợ của ISO/IEC 9899 không được xem xét trong quá trình phát triển các hướng dẫn này.

Có bằng chứng kể lại về việc một số triển khai không tuân thủ không phát hiện được các vi phạm ràng buộc, ví dụ như trong [38] trang 135, ví dụ 2 có tiêu đề “Lỗi ghi vào khu vực const”.

#### Ví dụ

Một số trình biên dịch C90 hỗ trợ các hàm nội tuyến (inline functions) bằng cách sử dụng từ khóa \_\_inline. Một chương trình C90 sử dụng \_\_inline sẽ tuân thủ quy tắc này miễn là nó được dịch bằng một trình biên dịch như vậy.

Dưới đây là bản dịch của đoạn văn từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

Nhiều trình biên dịch cho các mục tiêu nhúng cung cấp các từ khóa bổ sung xác định loại đối tượng với các thuộc tính của khu vực bộ nhớ mà đối tượng đó nằm, ví dụ:

- \_\_zpage — đối tượng có thể được truy cập bằng một lệnh ngắn

- \_\_near — con trỏ tới đối tượng có thể được lưu trong 16 bit

- \_\_far — con trỏ tới đối tượng có thể được lưu trong 24 bit

Một chương trình sử dụng các từ khóa bổ sung này sẽ tuân thủ quy tắc này miễn là trình biên dịch hỗ trợ những từ khóa này như một phần mở rộng ngôn ngữ.

#### Xem thêm

- Dir 2.1, Quy tắc 1.2

#### Quy tắc 1.2: Không nên sử dụng các phần mở rộng ngôn ngữ

- Phân loại: Tư vấn

- Phân tích: Không quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Lý do

Một chương trình phụ thuộc vào các phần mở rộng ngôn ngữ có thể kém di động hơn so với một chương trình không sử dụng. Mặc dù Tiêu chuẩn yêu cầu một triển khai tuân thủ phải tài liệu hóa bất kỳ phần mở rộng nào mà nó cung cấp cho ngôn ngữ, có nguy cơ rằng tài liệu này có thể không cung cấp mô tả đầy đủ về hành vi trong mọi hoàn cảnh.

Nếu quy tắc này không được áp dụng, quyết định sử dụng mỗi phần mở rộng ngôn ngữ nên được giải thích trong tài liệu thiết kế của dự án. Các phương pháp đảm bảo việc sử dụng hợp lệ của mỗi phần mở rộng, ví dụ như kiểm tra trình biên dịch và các chẩn đoán của nó, cũng nên được tài liệu hóa.

Cần nhận ra rằng việc sử dụng các phần mở rộng ngôn ngữ trong hệ thống nhúng là cần thiết. Tiêu chuẩn yêu cầu rằng một phần mở rộng không được thay đổi hành vi của bất kỳ chương trình nào tuân thủ nghiêm ngặt. Ví dụ, một trình biên dịch có thể triển khai, như một phần mở rộng, việc đánh giá đầy đủ các toán tử logic nhị phân mặc dù Tiêu chuẩn chỉ định rằng việc đánh giá dừng lại ngay khi kết quả có thể được xác định. Phần mở rộng như vậy không tuân thủ Tiêu chuẩn vì các tác dụng phụ trong toán hạng phải của một toán tử AND logic sẽ luôn xảy ra, dẫn đến một hành vi khác.

#### Xem thêm

- Quy tắc 1.1

#### Quy tắc 1.3: Không được có bất kỳ hành vi nào không xác định hoặc không xác định quan trọng

- Phân loại: Bắt buộc

- Phân tích: Không quyết định được, Hệ thống

- Áp dụng cho: C90, C99

##### Mở rộng

Một số hành vi không xác định và không xác định được xử lý bởi các quy tắc cụ thể. Quy tắc này ngăn chặn tất cả các hành vi không xác định và không xác định quan trọng khác. Phụ lục H liệt kê các hành vi không xác định và những hành vi không xác định được coi là quan trọng.

Dưới đây là bản dịch của phần Rationale từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

### Lý do

Bất kỳ chương trình nào có hành vi không xác định hoặc không xác định có thể không hoạt động theo cách mong đợi. Trong nhiều trường hợp, hiệu ứng là làm cho chương trình không di động nhưng cũng có thể xảy ra các vấn đề nghiêm trọng hơn. Ví dụ, hành vi không xác định có thể ảnh hưởng đến kết quả của một phép tính. Nếu hoạt động chính xác của phần mềm phụ thuộc vào phép tính này thì an toàn hệ thống có thể bị ảnh hưởng. Vấn đề này đặc biệt khó phát hiện nếu hành vi không xác định chỉ xuất hiện trong những tình huống hiếm hoi.

Nhiều hướng dẫn của MISRA C được thiết kế để tránh một số hành vi không xác định và không xác định. Ví dụ, tuân thủ tất cả Quy tắc 11.4, Quy tắc 11.8 và Quy tắc 19.2 đảm bảo rằng không thể trong C tạo ra một con trỏ không phải là const tới một đối tượng được khai báo với kiểu const. Điều này tránh được C90 [Không xác định 39] và C99 [Không xác định 61]. Tuy nhiên, các hành vi khác không được bao phủ bởi các hướng dẫn cụ thể vì:

- Không có khả năng hành vi này sẽ được gặp phải;

- Không có hướng dẫn thực tiễn nào có thể được đưa ra ngoài việc tuyên bố rõ ràng rằng hành vi này nên được tránh.

Thay vì giới thiệu một hướng dẫn cho mỗi hành vi không xác định và không xác định quan trọng, Hướng dẫn MISRA C trực tiếp giải quyết những hành vi được coi là quan trọng nhất và có khả năng xảy ra nhất trong thực tế. Những hành vi không có hướng dẫn cụ thể đều được bao phủ cùng nhau bởi quy tắc này. Phụ lục H liệt kê tất cả các hành vi không xác định và không xác định quan trọng, cùng với các hướng dẫn MISRA C ngăn chặn sự xuất hiện của chúng. Do đó, nó chỉ ra những hành vi nào được kỳ vọng sẽ được ngăn chặn bởi quy tắc này và những hành vi nào được bao phủ bởi các quy tắc khác.

Lưu ý: một số triển khai có thể cung cấp hành vi xác định rõ cho một số hành vi không xác định và không xác định được liệt kê trong Tiêu chuẩn. Nếu dựa vào những hành vi xác định rõ này, bao gồm cả việc sử dụng phần mở rộng ngôn ngữ, sẽ cần phải lệch khỏi quy tắc này đối với những hành vi đó.

#### Xem thêm

- Dir 4.1

### 8.2 Mã không sử dụng

#### Quy tắc 2.1: Một dự án không được chứa mã không thể truy cập

- [IEC 61508-7 Phần C.5.9], [DO-178C Phần 6.4.4.3.c]

- Phân loại: Bắt buộc

- Phân tích: Không quyết định được, Hệ thống

- Áp dụng cho: C90, C99

##### Lý do

Với điều kiện là một chương trình không biểu hiện bất kỳ hành vi không xác định nào, mã không thể truy cập không thể được thực thi và không thể có bất kỳ ảnh hưởng nào đến đầu ra của chương trình. Do đó, sự hiện diện của mã không thể truy cập có thể chỉ ra một lỗi trong logic của chương trình.

Dưới đây là bản dịch của phần nội dung thêm từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

Một trình biên dịch được phép loại bỏ bất kỳ mã không thể truy cập nào mặc dù nó không bắt buộc phải làm như vậy. Mã không thể truy cập không bị loại bỏ bởi trình biên dịch gây lãng phí tài nguyên, ví dụ:

- Nó chiếm không gian trong bộ nhớ của máy mục tiêu;

- Sự hiện diện của nó có thể khiến trình biên dịch chọn các lệnh nhảy dài hơn và chậm hơn khi chuyển điều khiển xung quanh mã không thể truy cập;

- Trong một vòng lặp, nó có thể ngăn toàn bộ vòng lặp cư trú trong bộ nhớ đệm lệnh.

Đôi khi, cần thiết chèn mã dường như không thể truy cập để xử lý các trường hợp ngoại lệ. Ví dụ, trong một câu lệnh switch trong đó mọi giá trị có thể của biểu thức điều khiển đều được bao phủ bởi một case cụ thể, một mệnh đề default phải có mặt theo Quy tắc 16.4. Mục đích của mệnh đề default là để bẫy một giá trị không nên xuất hiện bình thường nhưng có thể đã được tạo ra do:

- Hành vi không xác định có mặt trong chương trình;

- Lỗi của phần cứng bộ vi xử lý.

Nếu một trình biên dịch có thể chứng minh rằng một mệnh đề default là không thể truy cập, nó có thể loại bỏ nó, do đó loại bỏ hành động phòng thủ. Với giả định rằng hành động phòng thủ là quan trọng, cần phải chứng minh rằng trình biên dịch không loại bỏ mã mặc dù nó không thể truy cập, hoặc thực hiện các bước để làm cho mã phòng thủ có thể truy cập. Hành động đầu tiên yêu cầu một sự lệch khỏi quy tắc này, có lẽ bằng cách xem xét mã đối tượng hoặc kiểm tra đơn vị để hỗ trợ sự lệch như vậy. Hành động thứ hai thường có thể được thực hiện bằng cách truy cập thông qua một lvalue đủ điều kiện volatile. Ví dụ, một trình biên dịch có thể xác định rằng phạm vi các giá trị được giữ bởi x được bao phủ bởi các mệnh đề case trong một câu lệnh switch như sau:

```c

uint16\_t x;

switch ( x )

```

Bằng cách buộc x được truy cập bằng một lvalue đủ điều kiện volatile, trình biên dịch phải giả định rằng biểu thức điều khiển có thể lấy bất kỳ giá trị nào:

```c

switch ( \*( volatile uint16\_t \* ) &x )

```

Lưu ý: mã đã bị loại trừ có điều kiện bởi các chỉ thị tiền xử lý không thuộc quy tắc này vì nó không được trình bày cho các giai đoạn dịch sau.

#### Ví dụ

```c

enum light { red, amber, red\_amber, green };

enum light next\_light ( enum light c )

{

enum light res;

switch ( c )

{

case red:

res = red\_amber;

break;

case red\_amber:

res = green;

break;

case green:

res = amber;

break;

case amber:

res = red;

break;

default:

// Mã xử lý phòng thủ cho giá trị không xác định

res = red; // Hoặc một hành động phù hợp khác

break;

}

return res;

}

```

Dưới đây là phần tiếp theo của bản dịch tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

```c

default:

{

/\*

\* Mệnh đề default này sẽ chỉ có thể truy cập nếu tham số c

\* giữ một giá trị không phải là thành viên của enum light.

\*/

error\_handler();

break;

}

}

return res;

res = c; /\* Không tuân thủ - câu lệnh này chắc chắn không thể truy cập \*/

}

```

#### Xem thêm

- Quy tắc 14.3, Quy tắc 16.4

#### Quy tắc 2.2: Không được có mã chết

- [IEC 61508-7 Phần C.5.10], [ISO 26262-6 Phần 9.4.5], [DO-178C Phần 6.4.4.3.c]

- Phân loại: Bắt buộc

- Phân tích: Không quyết định được, Hệ thống

- Áp dụng cho: C90, C99

##### Mở rộng

Bất kỳ thao tác nào được thực thi nhưng việc loại bỏ nó không ảnh hưởng đến hành vi của chương trình đều cấu thành mã chết. Các thao tác được giới thiệu bởi các phần mở rộng ngôn ngữ luôn được giả định có ảnh hưởng đến hành vi của chương trình.

Lưu ý: Hành vi của một hệ thống nhúng thường được xác định không chỉ bởi bản chất của các hành động của nó, mà còn bởi thời điểm chúng xảy ra.

Lưu ý: Mã không thể truy cập không phải là mã chết vì nó không thể được thực thi.

##### Lý do

Sự hiện diện của mã chết có thể chỉ ra một lỗi trong logic của chương trình. Vì mã chết có thể bị loại bỏ bởi trình biên dịch, sự hiện diện của nó có thể gây ra sự nhầm lẫn.

##### Ngoại lệ

Một ép kiểu sang void được giả định là để chỉ ra một giá trị không được sử dụng một cách có chủ đích. Do đó, ép kiểu không phải là mã chết. Nó được coi là sử dụng toán hạng của nó và do đó cũng không phải là mã chết.

Dưới đây là phần dịch của các quy tắc tiếp theo từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

#### Quy tắc 2.3: Một dự án không nên chứa các khai báo kiểu không sử dụng

- Phân loại: Tư vấn

- Phân tích: Quyết định được, Hệ thống

- Áp dụng cho: C90, C99

##### Lý do

Nếu một kiểu được khai báo nhưng không được sử dụng, thì người đánh giá sẽ không rõ liệu kiểu đó là dư thừa hay đã bị bỏ qua một cách nhầm lẫn.

##### Ví dụ

```c

int16\_t unusedtype ( void )

{

typedef int16\_t local\_Type; /\* Không tuân thủ \*/

return 67;

}

```

#### Quy tắc 2.4: Một dự án không nên chứa các khai báo nhãn không sử dụng

- Phân loại: Tư vấn

- Phân tích: Quyết định được, Hệ thống

- Áp dụng cho: C90, C99

##### Lý do

Nếu một nhãn được khai báo nhưng không sử dụng, thì người đánh giá sẽ không rõ liệu nhãn đó là dư thừa hay đã bị bỏ qua một cách nhầm lẫn.

##### Ví dụ

Trong ví dụ sau, nhãn `state` không được sử dụng và khai báo có thể được viết mà không cần nhãn này.

```c

void unusedtag ( void )

{

enum state { S\_init, S\_run, S\_sleep }; /\* Không tuân thủ \*/

}

```

Trong ví dụ sau, nhãn `record\_t` chỉ được sử dụng trong typedef của `record1\_t` mà được sử dụng trong phần còn lại của đơn vị dịch bất cứ khi nào cần kiểu này. Typedef này có thể được viết một cách tuân thủ bằng cách bỏ qua nhãn như được hiển thị trong định nghĩa của `record2\_t`.

```c

typedef struct record\_t /\* Không tuân thủ \*/

{

uint16\_t key;

uint16\_t val;

} record1\_t;

typedef struct /\* Tuân thủ \*/

{

uint16\_t key;

uint16\_t val;

} record2\_t;

```

Dưới đây là bản dịch các quy tắc tiếp theo từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

#### Quy tắc 2.5: Một dự án không nên chứa các khai báo macro không sử dụng

- Phân loại: Tư vấn

- Phân tích: Quyết định được, Hệ thống

- Áp dụng cho: C90, C99

##### Lý do

Nếu một macro được khai báo nhưng không được sử dụng, thì người đánh giá sẽ không rõ liệu macro đó là dư thừa hay đã bị bỏ qua một cách nhầm lẫn.

##### Ví dụ

```c

void use\_macro ( void )

{

#define SIZE 4

/\* Không tuân thủ - DATA không được sử dụng \*/

#define DATA 3

use\_int16 ( SIZE );

}

```

#### Quy tắc 2.6: Một hàm không nên chứa các khai báo nhãn không sử dụng

- Phân loại: Tư vấn

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Lý do

Nếu một nhãn được khai báo nhưng không được sử dụng, thì người đánh giá sẽ không rõ liệu nhãn đó là dư thừa hay đã bị bỏ qua một cách nhầm lẫn.

##### Ví dụ

```c

void unused\_label ( void )

{

int16\_t x = 6;

label1: /\* Không tuân thủ \*/

use\_int16 ( x );

}

```

#### Quy tắc 2.7: Không nên có các tham số không sử dụng trong các hàm

- Phân loại: Tư vấn

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Lý do

Hầu hết các hàm sẽ được chỉ định là sử dụng từng tham số của chúng. Nếu một tham số hàm không được sử dụng, có thể việc triển khai hàm không khớp với đặc tả của nó. Quy tắc này nhấn mạnh những sự không khớp tiềm năng như vậy.

Dưới đây là phần dịch các quy tắc và ví dụ tiếp theo từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

#### Ví dụ

```c

void withunusedpara ( uint16\_t \*para1, int16\_t unusedpara ) /\* Không tuân thủ - không sử dụng \*/

{

\*para1 = 42U;

}

```

### 8.3 Bình luận (Comments)

#### Quy tắc 3.1: Chuỗi ký tự /\* và // không được sử dụng trong một bình luận

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Lý do

Nếu một chuỗi bắt đầu bình luận, /\* hoặc //, xuất hiện trong một bình luận /\*, rất có thể là do thiếu chuỗi kết thúc bình luận \*/.

Nếu một chuỗi bắt đầu bình luận xuất hiện trong một bình luận //, có thể là vì một vùng mã đã được bình luận ra sử dụng //.

##### Ngoại lệ

Chuỗi // được phép trong một bình luận //.

##### Ví dụ

Xem xét đoạn mã sau:

```c

/\* một số bình luận, dấu kết thúc bình luận vô tình bị bỏ qua

<<New Page>>

Perform\_Critical\_Safety\_Function( X );

/\* bình luận này không tuân thủ \*/

```

Khi xem xét trang chứa lời gọi đến hàm, giả định là nó được thực thi. Vì vô tình bỏ sót dấu kết thúc bình luận, lời gọi đến hàm an toàn quan trọng sẽ không được thực thi.

Trong ví dụ C99 sau, sự hiện diện của các bình luận // thay đổi ý nghĩa của chương trình:

```c

x = y // /\*

+ z

// \*/

;

```

Điều này cho x = y + z; nhưng sẽ là x = y; nếu không có hai chuỗi bắt đầu bình luận //.

#### Xem thêm

- Dir 4.4

Dưới đây là bản dịch của các quy tắc tiếp theo từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

#### Quy tắc 3.2: Không được sử dụng nối dòng trong các bình luận //

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C99

##### Mở rộng

Nối dòng xảy ra khi ký tự \ được theo sau ngay lập tức bởi ký tự xuống dòng. Nếu tệp nguồn chứa các ký tự đa byte, chúng sẽ được chuyển đổi sang bộ ký tự nguồn trước khi xảy ra nối dòng.

##### Lý do

Nếu dòng nguồn chứa một bình luận // kết thúc bằng ký tự \ trong bộ ký tự nguồn, dòng tiếp theo trở thành một phần của bình luận. Điều này có thể dẫn đến việc loại bỏ mã không mong muốn.

Lưu ý: nối dòng được mô tả trong Mục 5.1.1.2(2) của cả C90 và C99.

##### Ví dụ

Trong ví dụ không tuân thủ sau, dòng vật lý chứa từ khóa if về mặt logic là một phần của dòng trước đó và do đó là một bình luận.

```c

extern bool\_t b;

void f ( void )

{

uint16\_t x = 0; // bình luận \

if ( b )

{

++x; /\* Điều này luôn được thực thi \*/

}

}

```

#### Xem thêm

- Dir 4.4

### 8.4 Bộ ký tự và quy ước từ vựng

#### Quy tắc 4.1: Các chuỗi thoát bát phân và thập lục phân phải được kết thúc

- C90 [Triển khai 11], C99 [Triển khai J.3.4(7, 8)]

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Mở rộng

Một chuỗi thoát bát phân hoặc thập lục phân phải được kết thúc bởi một trong các điều sau:

- Bắt đầu của một chuỗi thoát khác, hoặc

- Kết thúc của hằng ký tự hoặc kết thúc của chuỗi ký tự.

Dưới đây là bản dịch phần Lý do và ví dụ cho các quy tắc tiếp theo từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

##### Lý do

Có khả năng gây nhầm lẫn nếu một chuỗi thoát bát phân hoặc thập lục phân được theo sau bởi các ký tự khác. Ví dụ, hằng ký tự '\x1f' bao gồm một ký tự duy nhất trong khi hằng ký tự '\x1g' bao gồm hai ký tự '\x1' và 'g'. Cách mà các hằng ký tự đa ký tự được biểu diễn dưới dạng số nguyên là do triển khai định nghĩa.

Khả năng gây nhầm lẫn sẽ giảm nếu mọi chuỗi thoát bát phân hoặc thập lục phân trong hằng ký tự hoặc chuỗi ký tự đều được kết thúc.

##### Ví dụ

Trong ví dụ này, mỗi chuỗi được trỏ tới bởi s1, s2 và s3 đều tương đương với “Ag”.

```c

const char \*s1 = "\x41g"; /\* Không tuân thủ \*/

const char \*s2 = "\x41" "g"; /\* Tuân thủ - kết thúc bởi kết thúc của chuỗi \*/

const char \*s3 = "\x41\x67"; /\* Tuân thủ - kết thúc bởi một chuỗi thoát khác \*/

int c1 = '\141t'; /\* Không tuân thủ \*/

int c2 = '\141\t'; /\* Tuân thủ - kết thúc bởi một chuỗi thoát khác \*/

```

#### Xem thêm

- C90: Phần 6.1.3.4, C99: Phần 6.4.4.4

#### Quy tắc 4.2: Không nên sử dụng các trigraf

- Phân loại: Tư vấn

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Lý do

Các trigraf được biểu thị bằng một chuỗi hai dấu hỏi theo sau bởi một ký tự thứ ba được chỉ định (ví dụ: ??- biểu thị một ký tự ~ (dấu ngã) và ??) biểu thị một ] ). Chúng có thể gây nhầm lẫn tình cờ với các cách sử dụng hai dấu hỏi khác.

Lưu ý: các digraf gọi là:

- <: :> <% %> %: %:%:

được phép vì chúng là các token. Các trigraf được thay thế bất cứ nơi nào chúng xuất hiện trong chương trình trước khi tiền xử lý.

##### Ví dụ

Ví dụ chuỗi

```c

"(Date should be in the form ??-??-??)"

```

sẽ không hoạt động như mong đợi, thực sự được trình biên dịch hiểu là

```c

"(Date should be in the form ~~]"

```

---

Dưới đây là bản dịch các quy tắc tiếp theo từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

### 8.5 Định danh

#### Quy tắc 5.1: Các định danh bên ngoài phải khác biệt

- C90 [Không xác định 7], C99 [Không xác định 7; Không xác định 28]

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Hệ thống

- Áp dụng cho: C90, C99

##### Mở rộng

Quy tắc này yêu cầu các định danh bên ngoài khác nhau phải khác biệt trong giới hạn do triển khai áp đặt.

Định nghĩa của sự khác biệt phụ thuộc vào triển khai và phiên bản của ngôn ngữ C đang được sử dụng:

- Trong C90, yêu cầu tối thiểu là 6 ký tự đầu tiên của các định danh bên ngoài là quan trọng nhưng không yêu cầu phân biệt chữ hoa chữ thường;

- Trong C99, yêu cầu tối thiểu là 31 ký tự đầu tiên của các định danh bên ngoài là quan trọng, với mỗi ký tự toàn cầu hoặc ký tự nguồn mở rộng tương ứng chiếm từ 6 đến 10 ký tự.

Trên thực tế, nhiều triển khai cung cấp các giới hạn lớn hơn. Ví dụ, thường thấy các định danh bên ngoài trong C90 là phân biệt chữ hoa chữ thường và ít nhất 31 ký tự đầu tiên là quan trọng.

##### Lý do

Nếu hai định danh chỉ khác nhau ở các ký tự không quan trọng, hành vi là không xác định.

Nếu khả năng di chuyển là một mối quan tâm, nên áp dụng quy tắc này bằng cách sử dụng các giới hạn tối thiểu được chỉ định trong Tiêu chuẩn.

Các định danh dài có thể làm giảm khả năng đọc của mã. Mặc dù nhiều hệ thống tạo mã tự động tạo ra các định danh dài, có lý do tốt để giữ độ dài định danh dưới mức giới hạn này.

Lưu ý: Trong C99, nếu một ký tự nguồn mở rộng xuất hiện trong một định danh bên ngoài và ký tự đó không có ký tự toàn cầu tương ứng, Tiêu chuẩn không chỉ định bao nhiêu ký tự nó chiếm.

##### Ví dụ

Trong ví dụ sau, tất cả các định nghĩa đều xảy ra trong cùng một đơn vị dịch. Triển khai liên quan hỗ trợ 31 ký tự phân biệt chữ hoa chữ thường trong các định danh bên ngoài.

```c

/\* 1234567890123456789012345678901\*\*\*\*\*\*\*\*\* Characters \*/

int32\_t engine\_exhaust\_gas\_temperature\_raw;

int32\_t engine\_exhaust\_gas\_temperature\_scaled; /\* Không tuân thủ \*/

/\* 1234567890123456789012345678901\*\*\*\*\*\*\*\*\* Characters \*/

int32\_t engine\_exhaust\_gas\_temp\_raw;

int32\_t engine\_exhaust\_gas\_temp\_scaled; /\* Tuân thủ \*/

```

Dưới đây là phần dịch các quy tắc và ví dụ tiếp theo từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

Trong ví dụ không tuân thủ sau, triển khai hỗ trợ 6 ký tự phân biệt chữ hoa chữ thường trong các định danh bên ngoài. Các định danh trong hai đơn vị dịch khác nhau nhưng không khác biệt trong các ký tự quan trọng.

```c

/\* file1.c \*/

int32\_t abc = 0;

/\* file2.c \*/

int32\_t ABC = 0;

```

#### Xem thêm

- Dir 1.1, Quy tắc 5.2, Quy tắc 5.4, Quy tắc 5.5

#### Quy tắc 5.2: Các định danh được khai báo trong cùng phạm vi và không gian tên phải khác biệt

- C90 [Không xác định 7], C99 [Không xác định 28]

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Mở rộng

Quy tắc này không áp dụng nếu cả hai định danh đều là định danh bên ngoài vì trường hợp này được bao phủ bởi Quy tắc 5.1.

Quy tắc này không áp dụng nếu một trong hai định danh là định danh macro vì trường hợp này được bao phủ bởi Quy tắc 5.4 và Quy tắc 5.5.

Định nghĩa của sự khác biệt phụ thuộc vào triển khai và phiên bản của ngôn ngữ C đang được sử dụng:

- Trong C90, yêu cầu tối thiểu là 31 ký tự đầu tiên là quan trọng;

- Trong C99, yêu cầu tối thiểu là 63 ký tự đầu tiên là quan trọng, với mỗi ký tự toàn cầu hoặc ký tự nguồn mở rộng tính là một ký tự.

##### Lý do

Nếu hai định danh chỉ khác nhau ở các ký tự không quan trọng, hành vi là không xác định.

Nếu khả năng di chuyển là một mối quan tâm, nên áp dụng quy tắc này bằng cách sử dụng các giới hạn tối thiểu được chỉ định trong Tiêu chuẩn.

Các định danh dài có thể làm giảm khả năng đọc của mã. Mặc dù nhiều hệ thống tạo mã tự động tạo ra các định danh dài, có lý do tốt để giữ độ dài định danh dưới mức giới hạn này.

##### Ví dụ

Trong ví dụ sau, triển khai liên quan hỗ trợ 31 ký tự phân biệt chữ hoa chữ thường trong các định danh không có liên kết bên ngoài. Định danh `engine\_exhaust\_gas\_temperature\_local` tuân thủ quy tắc này. Mặc dù nó không khác biệt so với định danh `engine\_exhaust\_gas\_temperature\_raw`, nó nằm trong một phạm vi khác. Tuy nhiên, nó không tuân thủ Quy tắc 5.3.

```c

int32\_t engine\_exhaust\_gas\_temperature\_raw;

void f(void)

{

int32\_t engine\_exhaust\_gas\_temperature\_local; /\* Tuân thủ quy tắc 5.2 \*/

/\* Không tuân thủ quy tắc 5.3 nếu được sử dụng cùng với định danh trên \*/

}

```

Dưới đây là phần dịch tiếp theo của các quy tắc và ví dụ từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

```c

/\* 1234567890123456789012345678901\*\*\*\*\*\*\*\*\* Characters \*/

extern int32\_t engine\_exhaust\_gas\_temperature\_raw;

static int32\_t engine\_exhaust\_gas\_temperature\_scaled; /\* Không tuân thủ \*/

void f ( void )

{

/\* 1234567890123456789012345678901\*\*\*\*\*\*\*\*\* Characters \*/

int32\_t engine\_exhaust\_gas\_temperature\_local; /\* Tuân thủ \*/

}

/\* 1234567890123456789012345678901\*\*\*\*\*\*\*\*\* Characters \*/

static int32\_t engine\_exhaust\_gas\_temp\_raw;

static int32\_t engine\_exhaust\_gas\_temp\_scaled; /\* Tuân thủ \*/

```

#### Xem thêm

- Dir 1.1, Quy tắc 5.1, Quy tắc 5.3, Quy tắc 5.4, Quy tắc 5.5

#### Quy tắc 5.3: Một định danh được khai báo trong một phạm vi bên trong không được che khuất một định danh được khai báo trong một phạm vi bên ngoài

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Mở rộng

Một định danh được khai báo trong một phạm vi bên trong phải khác biệt so với bất kỳ định danh nào được khai báo trong một phạm vi bên ngoài. Định nghĩa của sự khác biệt phụ thuộc vào triển khai và phiên bản của ngôn ngữ C đang được sử dụng:

- Trong C90, yêu cầu tối thiểu là 31 ký tự đầu tiên là quan trọng;

- Trong C99, yêu cầu tối thiểu là 63 ký tự đầu tiên là quan trọng, với mỗi ký tự toàn cầu hoặc ký tự nguồn mở rộng tính là một ký tự.

##### Lý do

Nếu một định danh được khai báo trong một phạm vi bên trong nhưng không khác biệt so với một định danh đã tồn tại trong một phạm vi bên ngoài, thì khai báo bên trong nhất sẽ "che khuất" cái bên ngoài. Điều này có thể dẫn đến sự nhầm lẫn cho nhà phát triển.

Lưu ý: Một định danh được khai báo trong một không gian tên không che khuất một định danh được khai báo trong một không gian tên khác.

Các thuật ngữ phạm vi bên ngoài và phạm vi bên trong được định nghĩa như sau:

- Các định danh có phạm vi tệp có thể được coi là có phạm vi ngoài cùng;

- Các định danh có phạm vi khối có phạm vi bên trong hơn;

- Các khối lồng nhau kế tiếp nhau, giới thiệu các phạm vi bên trong hơn.

##### Ví dụ

```c

extern int32\_t engine\_exhaust\_gas\_temperature\_raw;

void f ( void )

{

int32\_t engine\_exhaust\_gas\_temperature\_local; /\* Không tuân thủ \*/

}

```

Trong ví dụ này, định danh `engine\_exhaust\_gas\_temperature\_local` được khai báo trong phạm vi bên trong của hàm `f`, nhưng không khác biệt so với định danh `engine\_exhaust\_gas\_temperature\_raw` được khai báo trong phạm vi bên ngoài, do đó vi phạm quy tắc này.

Dưới đây là bản dịch tiếp theo của các quy tắc và ví dụ từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

#### Ví dụ

```c

void fn1 ( void )

{

int16\_t i; /\* Khai báo một đối tượng "i" \*/

{

int16\_t i; /\* Không tuân thủ - che khuất "i" trước đó \*/

i = 3; /\* Có thể gây nhầm lẫn là "i" nào được tham chiếu \*/

}

}

struct astruct

{

int16\_t m;

};

extern void g ( struct astruct \*p );

int16\_t xyz = 0; /\* Khai báo một đối tượng "xyz" \*/

void fn2 ( struct astruct xyz ) /\* Không tuân thủ - "xyz" ngoài bị che khuất bởi tên tham số \*/

{

g ( &xyz );

}

uint16\_t speed;

void fn3 ( void )

{

typedef float32\_t speed; /\* Không tuân thủ - kiểu che khuất đối tượng \*/

}

```

#### Xem thêm

- Quy tắc 5.2, Quy tắc 5.8

#### Quy tắc 5.4: Các định danh macro phải khác biệt

- C90 [Không xác định 7], C99 [Không xác định 7; Không xác định 28]

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Mở rộng

Quy tắc này yêu cầu rằng, khi một macro được định nghĩa, tên của nó phải khác biệt với:

- Tên của các macro khác hiện đang được định nghĩa; và

- Tên của các tham số của chúng.

Nó cũng yêu cầu rằng các tên của các tham số của một macro nhất định phải khác biệt với nhau nhưng không yêu cầu rằng các tên tham số macro phải khác biệt giữa hai macro khác nhau.

Dưới đây là phần dịch tiếp theo của các quy tắc và ví dụ từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

Định nghĩa của sự khác biệt phụ thuộc vào triển khai và phiên bản của ngôn ngữ C đang được sử dụng:

- Trong C90, yêu cầu tối thiểu là 31 ký tự đầu tiên của các định danh macro là quan trọng;

- Trong C99, yêu cầu tối thiểu là 63 ký tự đầu tiên của các định danh macro là quan trọng.

Trên thực tế, các triển khai có thể cung cấp giới hạn lớn hơn. Quy tắc này yêu cầu các định danh macro phải khác biệt trong giới hạn do triển khai áp đặt.

##### Lý do

Nếu hai định danh macro chỉ khác nhau ở các ký tự không quan trọng, hành vi là không xác định. Vì các tham số macro chỉ hoạt động trong quá trình mở rộng của macro, không có vấn đề với các tham số trong một macro bị nhầm lẫn với các tham số trong một macro khác.

Nếu khả năng di chuyển là một mối quan tâm, nên áp dụng quy tắc này bằng cách sử dụng các giới hạn tối thiểu được chỉ định trong Tiêu chuẩn.

Các định danh macro dài có thể làm giảm khả năng đọc của mã. Mặc dù nhiều hệ thống tạo mã tự động tạo ra các định danh macro dài, có lý do tốt để giữ độ dài định danh macro dưới mức giới hạn này.

Lưu ý: Trong C99, nếu một ký tự nguồn mở rộng xuất hiện trong tên macro và ký tự đó không có ký tự toàn cầu tương ứng, Tiêu chuẩn không chỉ định bao nhiêu ký tự nó chiếm.

##### Ví dụ

Trong ví dụ sau, triển khai liên quan hỗ trợ 31 ký tự phân biệt chữ hoa chữ thường trong các định danh macro.

```c

/\* 1234567890123456789012345678901\*\*\*\*\*\*\*\*\* Characters \*/

#define engine\_exhaust\_gas\_temperature\_raw egt\_r

#define engine\_exhaust\_gas\_temperature\_scaled egt\_s /\* Không tuân thủ \*/

/\* 1234567890123456789012345678901\*\*\*\*\*\*\*\*\* Characters \*/

#define engine\_exhaust\_gas\_temp\_raw egt\_r

#define engine\_exhaust\_gas\_temp\_scaled egt\_s /\* Tuân thủ \*/

```

#### Xem thêm

- Quy tắc 5.1, Quy tắc 5.2, Quy tắc 5.5

Dưới đây là phần dịch tiếp theo của các quy tắc và ví dụ từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

#### Quy tắc 5.5: Các định danh phải khác biệt so với tên macro

- C90 [Không xác định 7], C99 [Không xác định 7; Không xác định 28]

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Mở rộng

Quy tắc này yêu cầu rằng tên của các macro tồn tại trước khi tiền xử lý phải khác biệt so với các định danh tồn tại sau khi tiền xử lý. Nó áp dụng cho các định danh, bất kể phạm vi hoặc không gian tên, và cho bất kỳ macro nào đã được định nghĩa bất kể định nghĩa đó còn hiệu lực khi định danh được khai báo.

Định nghĩa của sự khác biệt phụ thuộc vào triển khai và phiên bản của ngôn ngữ C đang được sử dụng:

- Trong C90, yêu cầu tối thiểu là 31 ký tự đầu tiên là quan trọng;

- Trong C99, yêu cầu tối thiểu là 63 ký tự đầu tiên là quan trọng, với mỗi ký tự toàn cầu hoặc ký tự nguồn mở rộng tính là một ký tự.

##### Lý do

Giữ cho tên macro và định danh khác biệt có thể giúp tránh sự nhầm lẫn của nhà phát triển.

##### Ví dụ

Trong ví dụ không tuân thủ sau, tên của macro dạng hàm Sum cũng được sử dụng như một định danh. Khai báo của đối tượng Sum không phải chịu sự mở rộng macro vì nó không được theo sau bởi một ký tự (. Do đó, định danh tồn tại sau khi tiền xử lý đã được thực hiện.

```c

#define Sum(x, y) ( ( x ) + ( y ) )

int16\_t Sum;

```

Ví dụ sau đây tuân thủ vì không có trường hợp nào của định danh Sum sau khi tiền xử lý.

```c

#define Sum(x, y) ( ( x ) + ( y ) )

int16\_t x = Sum ( 1, 2 );

```

Trong ví dụ sau, triển khai liên quan hỗ trợ 31 ký tự phân biệt chữ hoa chữ thường trong các định danh không có liên kết bên ngoài. Ví dụ này không tuân thủ vì tên macro không khác biệt so với tên định danh có liên kết nội bộ trong 31 ký tự đầu tiên.

```c

/\* 1234567890123456789012345678901\*\*\*\*\*\*\*\*\* Characters \*/

#define low\_pressure\_turbine\_temperature\_1 lp\_tb\_temp\_1

static int32\_t low\_pressure\_turbine\_temperature\_2;

```

#### Xem thêm

- Quy tắc 5.1, Quy tắc 5.2, Quy tắc 5.4

Dưới đây là phần dịch tiếp theo của các quy tắc và ví dụ từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

#### Quy tắc 5.6: Tên typedef phải là định danh duy nhất

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Hệ thống

- Áp dụng cho: C90, C99

##### Mở rộng

Tên typedef phải duy nhất trên tất cả các không gian tên và đơn vị dịch. Nhiều khai báo cùng một tên typedef chỉ được phép theo quy tắc này nếu định nghĩa kiểu được thực hiện trong một tệp tiêu đề và tệp tiêu đề đó được bao gồm trong nhiều tệp nguồn.

##### Lý do

Sử dụng lại tên typedef như một tên typedef khác hoặc là tên của một hàm, đối tượng hoặc hằng số liệt kê có thể dẫn đến sự nhầm lẫn cho nhà phát triển.

##### Ngoại lệ

Tên typedef có thể giống với tên thẻ cấu trúc, liên hiệp hoặc liệt kê liên quan đến typedef đó.

##### Ví dụ

```c

void func ( void )

{

{

typedef unsigned char u8\_t;

}

{

typedef unsigned char u8\_t; /\* Không tuân thủ - sử dụng lại \*/

}

}

typedef float mass;

void func1 ( void )

{

float32\_t mass = 0.0f; /\* Không tuân thủ - sử dụng lại \*/

}

typedef struct list

{

struct list \*next;

uint16\_t element;

} list; /\* Tuân thủ - ngoại lệ \*/

typedef struct

{

struct chain

{

struct chain \*list;

uint16\_t element;

} s1;

uint16\_t length;

} chain; /\* Không tuân thủ - thẻ "chain" không liên quan đến typedef \*/

```

#### Xem thêm

- Quy tắc 5.7

Dưới đây là phần dịch tiếp theo của các quy tắc và ví dụ từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

#### Quy tắc 5.7: Tên thẻ phải là định danh duy nhất

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Hệ thống

- Áp dụng cho: C90, C99

##### Mở rộng

Tên thẻ phải duy nhất trên tất cả các không gian tên và đơn vị dịch. Tất cả các khai báo của thẻ phải chỉ định cùng một kiểu. Nhiều khai báo hoàn chỉnh của cùng một thẻ chỉ được phép theo quy tắc này nếu thẻ được khai báo trong một tệp tiêu đề và tệp tiêu đề đó được bao gồm trong nhiều tệp nguồn.

##### Lý do

Sử dụng lại tên thẻ có thể dẫn đến sự nhầm lẫn cho nhà phát triển. Có hành vi không xác định liên quan đến việc sử dụng lại tên thẻ trong C90 mặc dù điều này không được liệt kê trong Phụ lục của Tiêu chuẩn. Hành vi không xác định này đã được công nhận trong C99 như một ràng buộc trong Mục 6.7.2.3.

##### Ngoại lệ

Tên thẻ có thể giống với tên typedef mà nó liên quan.

##### Ví dụ

```c

struct stag

{

uint16\_t a;

uint16\_t b;

};

struct stag a1 = { 0, 0 }; /\* Tuân thủ - tương thích với trên \*/

union stag a2 = { 0, 0 }; /\* Không tuân thủ - khai báo kiểu khác so với struct stag. Vi phạm ràng buộc trong C99 \*/

```

Ví dụ sau đây cũng vi phạm Quy tắc 5.3

```c

struct deer

{

uint16\_t a;

uint16\_t b;

};

void foo ( void )

{

struct deer

{

uint16\_t a;

}; /\* Không tuân thủ - tên thẻ "deer" được sử dụng lại \*/

}

```

```c

typedef struct coord

{

uint16\_t x;

uint16\_t y;

} coord; /\* Tuân thủ theo ngoại lệ \*/

```

Dưới đây là phần dịch tiếp theo của các quy tắc và ví dụ từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

```c

struct elk

{

uint16\_t x;

};

struct elk /\* Không tuân thủ - khai báo kiểu khác Vi phạm ràng buộc trong C99 \*/

{

uint32\_t x;

};

```

#### Xem thêm

- Quy tắc 5.6

#### Quy tắc 5.8: Các định danh định nghĩa đối tượng hoặc hàm với liên kết bên ngoài phải là duy nhất

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Hệ thống

- Áp dụng cho: C90, C99

##### Mở rộng

Một định danh được sử dụng làm định danh bên ngoài không được sử dụng cho bất kỳ mục đích nào khác trong bất kỳ không gian tên hoặc đơn vị dịch nào, ngay cả khi nó biểu thị một đối tượng không có liên kết.

##### Lý do

Đảm bảo tính duy nhất của tên định danh theo cách này giúp tránh sự nhầm lẫn. Các định danh của các đối tượng không có liên kết không cần phải duy nhất vì có rất ít nguy cơ nhầm lẫn.

##### Ví dụ

Trong ví dụ sau, file1.c và file2.c đều là một phần của cùng một dự án.

```c

/\* file1.c \*/

int32\_t count; /\* "count" có liên kết bên ngoài \*/

void foo ( void ) /\* "foo" có liên kết bên ngoài \*/

{

int16\_t index; /\* "index" không có liên kết \*/

}

/\* file2.c \*/

static void foo ( void ) /\* Không tuân thủ - "foo" không duy nhất vì đã được định nghĩa với liên kết bên ngoài trong file1.c \*/

{

int16\_t count; /\* Không tuân thủ - "count" không có liên kết nhưng xung đột với định danh có liên kết bên ngoài \*/

int32\_t index; /\* Tuân thủ - "index" không có liên kết \*/

}

```

#### Xem thêm

- Quy tắc 5.3

#### Quy tắc 5.9: Các định danh định nghĩa đối tượng hoặc hàm với liên kết nội bộ nên là duy nhất

- Phân loại: Tư vấn

- Phân tích: Quyết định được, Hệ thống

- Áp dụng cho: C90, C99

##### Mở rộng

Tên định danh nên là duy nhất trên tất cả các không gian tên và đơn vị dịch. Bất kỳ định danh nào được sử dụng theo cách này không nên có cùng tên với bất kỳ định danh nào khác, ngay cả khi định danh đó biểu thị một đối tượng không có liên kết.

##### Lý do

Đảm bảo tính duy nhất của tên định danh theo cách này giúp tránh sự nhầm lẫn.

##### Ngoại lệ

Một hàm nội tuyến với liên kết nội bộ có thể được định nghĩa trong nhiều đơn vị dịch nếu tất cả các định nghĩa đó được thực hiện trong cùng một tệp tiêu đề được bao gồm trong mỗi đơn vị dịch.

##### Ví dụ

Trong ví dụ sau, file1.c và file2.c đều là một phần của cùng một dự án.

```c

/\* file1.c \*/

static int32\_t count; /\* "count" có liên kết nội bộ \*/

static void foo ( void ) /\* "foo" có liên kết nội bộ \*/

{

int16\_t count; /\* Không tuân thủ - "count" không có liên kết nhưng xung đột với định danh có liên kết nội bộ \*/

int16\_t index; /\* "index" không có liên kết \*/

}

void bar1 ( void )

{

static int16\_t count; /\* Không tuân thủ - "count" không có liên kết nhưng xung đột với định danh có liên kết nội bộ \*/

int16\_t index; /\* Tuân thủ - "index" không duy nhất nhưng không có liên kết \*/

foo ( );

}

/\* End of file1.c \*/

/\* file2.c \*/

static int8\_t count; /\* Không tuân thủ - "count" có liên kết nội bộ nhưng xung đột với các định danh khác cùng tên \*/

static void foo ( void ) /\* Không tuân thủ - "foo" có liên kết nội bộ nhưng xung đột với một hàm cùng tên \*/

{

int32\_t index; /\* Tuân thủ - cả "index" và "nbytes" đều không duy nhất nhưng không có liên kết \*/

int16\_t nbytes; /\* không có liên kết \*/

}

void bar2 ( void )

{

static uint8\_t nbytes; /\* Tuân thủ - "nbytes" không duy nhất nhưng không có liên kết và lớp lưu trữ không liên quan \*/

}

/\* End of file2.c \*/

```

#### Xem thêm

- Quy tắc 8.10

### 8.6 Các loại

#### Quy tắc 6.1: Các trường bit chỉ nên được khai báo với loại thích hợp

- C90 [Không xác định 38; Triển khai 29], C99 [Triển khai J.3.9(1, 2)]

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Mở rộng

Các loại trường bit thích hợp là:

- C90: hoặc unsigned int hoặc signed int;

- C99: một trong các loại:

- hoặc unsigned int hoặc signed int;

- một loại số nguyên khác được ký rõ ràng hoặc không ký rõ ràng được cho phép bởi triển khai;

- \_Bool.

Lưu ý: Sử dụng typedef để chỉ định một loại thích hợp được cho phép.

##### Lý do

Sử dụng int là do triển khai định nghĩa vì các trường bit kiểu int có thể là kiểu signed hoặc unsigned. Sử dụng enum, short, char hoặc bất kỳ loại nào khác cho các trường bit không được phép trong C90 vì hành vi là không xác định.

Trong C99, triển khai có thể xác định các loại số nguyên khác được cho phép trong các khai báo trường bit.

Dưới đây là bản dịch tiếp theo của các quy tắc và ví dụ từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

#### Ví dụ

Ví dụ sau đây áp dụng cho C90 và các triển khai C99 không cung cấp bất kỳ loại trường bit bổ sung nào. Giả sử rằng loại int là 16-bit.

```c

typedef unsigned int UINT\_16;

struct s {

unsigned int b1:2; /\* Tuân thủ \*/

int b2:2; /\* Không tuân thủ - không cho phép sử dụng int \*/

UINT\_16 b3:2; /\* Tuân thủ - typedef chỉ định unsigned int \*/

signed long b4:2; /\* Không tuân thủ ngay cả khi long và int có cùng kích thước \*/

};

```

#### Quy tắc 6.2: Các trường bit được đặt tên chỉ có một bit không được là loại có dấu

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Lý do

Theo Tiêu chuẩn C99 Mục 6.2.6.2, một trường bit có dấu chỉ có một bit có một (một) bit dấu và không (không) bit giá trị. Trong bất kỳ cách biểu diễn số nguyên nào, 0 bit giá trị không thể chỉ định một giá trị có ý nghĩa.

Do đó, một trường bit có dấu chỉ có một bit có khả năng không hoạt động theo cách hữu ích và sự hiện diện của nó có thể chỉ ra sự nhầm lẫn của lập trình viên.

Mặc dù Tiêu chuẩn C90 không cung cấp nhiều chi tiết về cách biểu diễn các loại, nhưng các cân nhắc tương tự áp dụng như đối với C99.

Lưu ý: quy tắc này không áp dụng cho các trường bit không được đặt tên vì giá trị của chúng không thể được truy cập.

### 8.7 Hằng số và hằng số số học

#### Quy tắc 7.1: Không được sử dụng hằng bát phân

- [Koenig 9]

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Lý do

Các nhà phát triển viết các hằng số có chữ số 0 đứng đầu có thể mong đợi chúng được hiểu là các hằng số thập phân.

Lưu ý: quy tắc này không áp dụng cho các chuỗi thoát bát phân vì việc sử dụng ký tự \ đứng đầu có nghĩa là có ít khả năng gây nhầm lẫn hơn.

##### Ngoại lệ

Hằng số nguyên 0 (viết dưới dạng một chữ số số), là một hằng bát phân nhưng được cho phép ngoại lệ với quy tắc này.

##### Ví dụ

```c

extern uint16\_t code[ 10 ];

code[ 1 ] = 109; /\* Tuân thủ - thập phân 109 \*/

code[ 2 ] = 100; /\* Tuân thủ - thập phân 100 \*/

code[ 3 ] = 052; /\* Không tuân thủ - thập phân 42 \*/

code[ 4 ] = 071; /\* Không tuân thủ - thập phân 57 \*/

```

#### Quy tắc 7.2: Một hậu tố “u” hoặc “U” phải được áp dụng cho tất cả các hằng số nguyên được biểu diễn ở loại không dấu

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Mở rộng

Quy tắc này áp dụng cho:

- Các hằng số nguyên xuất hiện trong các biểu thức điều khiển của các chỉ thị tiền xử lý #if và #elif;

- Bất kỳ hằng số nguyên nào khác tồn tại sau khi tiền xử lý.

Lưu ý: trong quá trình tiền xử lý, loại của một hằng số nguyên được xác định theo cách tương tự như sau khi tiền xử lý ngoại trừ:

- Tất cả các loại số nguyên có dấu hoạt động như thể chúng là long (C90) hoặc intmax\_t (C99);

- Tất cả các loại số nguyên không dấu hoạt động như thể chúng là unsigned long (C90) hoặc uintmax\_t (C99).

##### Lý do

Loại của một hằng số nguyên là một nguồn tiềm ẩn của sự nhầm lẫn, vì nó phụ thuộc vào một tổ hợp phức tạp của các yếu tố bao gồm:

- Độ lớn của hằng số;

- Kích thước được triển khai của các loại số nguyên;

- Sự hiện diện của bất kỳ hậu tố nào;

- Cơ số mà giá trị được biểu diễn (ví dụ: thập phân, bát phân hoặc thập lục phân).

Ví dụ, hằng số nguyên 40000 là loại signed int trong một môi trường 32-bit nhưng là loại signed long trong một môi trường 16-bit. Giá trị 0x8000 là loại unsigned int trong một môi trường 16-bit, nhưng là loại signed int trong một môi trường 32-bit.

##### Lưu ý

- Bất kỳ giá trị nào có hậu tố “U” là loại không dấu;

- Một giá trị thập phân không có hậu tố nhỏ hơn 2^31 là loại có dấu.

Dưới đây là bản dịch tiếp theo của các quy tắc và ví dụ từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

Nhưng:

- Một giá trị thập lục phân không có hậu tố lớn hơn hoặc bằng 2^15 có thể là loại có dấu hoặc không dấu;

- Đối với C90, một giá trị thập phân không có hậu tố lớn hơn hoặc bằng 2^31 có thể là loại có dấu hoặc không dấu.

Tính có dấu của các hằng số nên rõ ràng. Nếu một hằng số là loại không dấu, việc áp dụng hậu tố "U" làm rõ rằng lập trình viên hiểu rằng hằng số này là không dấu.

Lưu ý: quy tắc này không phụ thuộc vào ngữ cảnh mà hằng số được sử dụng; sự nâng cấp và các chuyển đổi khác có thể được áp dụng cho hằng số không liên quan đến việc xác định tuân thủ quy tắc này.

##### Ví dụ

Ví dụ sau đây giả định một máy có loại int 16-bit và loại long 32-bit. Nó cho thấy loại của mỗi hằng số nguyên được xác định theo Tiêu chuẩn. Hằng số nguyên 0x8000 không tuân thủ vì nó là loại không dấu nhưng không có hậu tố "U".

| Hằng số | Loại | Tuân thủ |

|---------|------|----------|

| 32767 | signed int | Tuân thủ |

| 0x7fff | signed int | Tuân thủ |

| 32768 | signed long | Tuân thủ |

| 32768u | unsigned int | Tuân thủ |

| 0x8000 | unsigned int | Không tuân thủ |

| 0x8000u | unsigned int | Tuân thủ |

#### Quy tắc 7.3: Ký tự chữ thường “l” không được sử dụng trong hậu tố của hằng số

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Lý do

Sử dụng hậu tố chữ hoa “L” loại bỏ khả năng nhầm lẫn giữa “1” (chữ số 1) và “l” (chữ cái “el”) khi khai báo các hằng số.

##### Ví dụ

Lưu ý: các ví dụ chứa hậu tố long long chỉ áp dụng cho C99.

```c

const int64\_t a = 0L;

const int64\_t b = 0l; /\* Không tuân thủ \*/

const uint64\_t c = 0Lu;

const uint64\_t d = 0lU; /\* Không tuân thủ \*/

const uint64\_t e = 0ULL;

const uint64\_t f = 0Ull; /\* Không tuân thủ \*/

const int128\_t g = 0LL;

const int128\_t h = 0ll; /\* Không tuân thủ \*/

const float128\_t m = 1.2L;

const float128\_t n = 2.4l; /\* Không tuân thủ \*/

```

#### Quy tắc 7.4: Một chuỗi ký tự không được gán cho một đối tượng trừ khi loại của đối tượng là “pointer to const-qualified char”

- C90 [Không xác định 12], C99 [Không xác định 14; Không xác định 30]

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Mở rộng

Không được cố gắng sửa đổi một chuỗi ký tự hoặc chuỗi ký tự rộng trực tiếp. Kết quả của toán tử địa chỉ &, áp dụng cho một chuỗi ký tự không được gán cho một đối tượng trừ khi loại của đối tượng đó là “pointer to array of const-qualified char”.

Các cân nhắc tương tự áp dụng cho các chuỗi ký tự rộng. Một chuỗi ký tự rộng không được gán cho một đối tượng trừ khi loại của đối tượng đó là “pointer to const-qualified wchar\_t”. Kết quả của toán tử địa chỉ &, áp dụng cho một chuỗi ký tự rộng không được gán cho một đối tượng trừ khi loại của đối tượng đó là “pointer to array of const-qualified wchar\_t”.

##### Lý do

Bất kỳ cố gắng nào để sửa đổi một chuỗi ký tự dẫn đến hành vi không xác định. Ví dụ, một số triển khai có thể lưu trữ các chuỗi ký tự trong bộ nhớ chỉ đọc trong trường hợp đó một cố gắng sửa đổi chuỗi ký tự sẽ thất bại và có thể dẫn đến một ngoại lệ hoặc sự cố.

Quy tắc này, khi được áp dụng cùng với các quy tắc khác, ngăn chặn một chuỗi ký tự bị sửa đổi.

C99 không chỉ rõ liệu các chuỗi ký tự chia sẻ một kết thúc chung có được lưu trữ ở các vị trí bộ nhớ riêng biệt hay không. Do đó, ngay cả khi cố gắng sửa đổi một chuỗi ký tự dường như thành công, có thể một chuỗi ký tự khác có thể bị thay đổi vô tình.

##### Ví dụ

Ví dụ sau cho thấy một cố gắng sửa đổi trực tiếp một chuỗi ký tự.

```c

"0123456789"[0] = '\*'; /\* Không tuân thủ \*/

```

Các ví dụ này cho thấy cách ngăn chặn sửa đổi các chuỗi ký tự gián tiếp.

```c

/\* Không tuân thủ - s không có const-qualified \*/

char \*s = "string";

/\* Tuân thủ - p có const-qualified; các định danh bổ sung được phép \*/

const volatile char \*p = "string";

extern void f1 ( char \*s1 );

extern void f2 ( const char \*s2 );

void g ( void )

{

f1 ( "string" ); /\* Không tuân thủ - tham số s1 không có const-qualified \*/

f2 ( "string" ); /\* Tuân thủ \*/

}

char \*name1 ( void )

{

return ( "MISRA" ); /\* Không tuân thủ - kiểu trả về không có const-qualified \*/

}

const char \*name2 ( void )

{

return ( "MISRA" ); /\* Tuân thủ \*/

}

```

#### Xem thêm

- Quy tắc 11.4, Quy tắc 11.8

### 8.8 Khai báo và định nghĩa

#### Quy tắc 8.1: Các loại phải được chỉ định rõ ràng

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90

##### Lý do

Tiêu chuẩn C90 cho phép các loại được bỏ qua trong một số trường hợp, trong trường hợp đó loại int được chỉ định ngầm định. Các trường hợp mà một int ngầm định có thể được sử dụng bao gồm:

- Khai báo đối tượng;

- Khai báo tham số;

- Khai báo thành viên;

- Khai báo typedef;

- Các kiểu trả về của hàm.

Việc bỏ sót một loại rõ ràng có thể dẫn đến sự nhầm lẫn. Ví dụ, trong khai báo:

```c

extern void g ( char c, const k );

```

loại của k là const int trong khi const char có thể đã được mong đợi.

Dưới đây là bản dịch tiếp theo của các quy tắc và ví dụ từ tài liệu MISRA Autosar từ tiếng Anh sang tiếng Việt:

---

##### Ví dụ

Các ví dụ sau đây cho thấy các khai báo đối tượng tuân thủ và không tuân thủ:

```c

extern x; /\* Không tuân thủ - loại int ngầm định \*/

extern int16\_t x; /\* Tuân thủ - loại rõ ràng \*/

const y; /\* Không tuân thủ - loại int ngầm định \*/

const int16\_t y; /\* Tuân thủ - loại rõ ràng \*/

```

Các ví dụ sau đây cho thấy các khai báo loại hàm tuân thủ và không tuân thủ:

```c

extern f ( void ); /\* Không tuân thủ - loại trả về int ngầm định \*/

extern int16\_t f ( void ); /\* Tuân thủ \*/

extern void g ( char c, const k ); /\* Không tuân thủ - loại int ngầm định cho tham số k \*/

extern void g ( char c, const int16\_t k ); /\* Tuân thủ \*/

```

Các ví dụ sau đây cho thấy các định nghĩa loại tuân thủ và không tuân thủ:

```c

typedef ( \*pfi ) ( void ); /\* Không tuân thủ - loại trả về int ngầm định \*/

typedef int16\_t ( \*pfi ) ( void ); /\* Tuân thủ \*/

typedef void ( \*pfv ) ( const x ); /\* Không tuân thủ - loại int ngầm định cho tham số x \*/

typedef void ( \*pfv ) ( int16\_t x ); /\* Tuân thủ \*/

```

Các ví dụ sau đây cho thấy các khai báo thành viên tuân thủ và không tuân thủ:

```c

struct str

{

int16\_t x; /\* Tuân thủ \*/

const y; /\* Không tuân thủ - loại int ngầm định cho thành viên y \*/

} s;

```

#### Xem thêm

- Quy tắc 8.2

#### Quy tắc 8.2: Các loại hàm phải ở dạng nguyên mẫu với các tham số có tên

- C90 [Không xác định 22-25], C99 [Không xác định 36-39, 73, 79]

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Đơn vị Dịch thuật Đơn

- Áp dụng cho: C90, C99

##### Lý do

Phiên bản đầu của C, thường được gọi là K&R C, không cung cấp cơ chế để kiểm tra số lượng đối số hoặc loại của chúng so với các tham số tương ứng. Loại của một đối tượng hoặc hàm không phải được khai báo trong K&R C vì loại mặc định của một đối tượng và loại trả về mặc định của một hàm là int.

Tiêu chuẩn C90 đã giới thiệu các nguyên mẫu hàm, một dạng khai báo hàm trong đó các loại tham số được khai báo. Điều này cho phép kiểm tra các loại đối số so với các loại tham số. Nó cũng cho phép kiểm tra số lượng đối số ngoại trừ khi một nguyên mẫu hàm chỉ định rằng mong đợi một số lượng đối số thay đổi. Tiêu chuẩn C90 không yêu cầu sử dụng các nguyên mẫu hàm vì lý do tương thích ngược với mã hiện có. Vì cùng lý do, nó tiếp tục cho phép bỏ qua các loại trong trường hợp đó loại sẽ mặc định là int.

Tiêu chuẩn C99 đã loại bỏ loại int mặc định khỏi ngôn ngữ nhưng tiếp tục cho phép các loại hàm kiểu K&R trong đó không có phương tiện để cung cấp thông tin loại tham số trong khai báo và việc cung cấp thông tin loại tham số trong định nghĩa là tùy chọn.

Sự không khớp giữa số lượng đối số và tham số, các loại của chúng và loại trả về dự kiến và thực tế của một hàm cung cấp tiềm năng cho hành vi không xác định. Mục đích của quy tắc này cùng với Quy tắc 8.1 và Quy tắc 8.4 là để tránh hành vi không xác định này bằng cách yêu cầu các loại tham số và loại trả về của hàm được chỉ định rõ ràng. Quy tắc 17.3 đảm bảo rằng thông tin này có sẵn tại thời điểm gọi hàm, do đó yêu cầu trình biên dịch chẩn đoán bất kỳ sự không khớp nào được phát hiện.

Quy tắc này cũng yêu cầu rằng các tên được chỉ định cho tất cả các tham số trong khai báo. Các tên tham số có thể cung cấp thông tin hữu ích về giao diện hàm và một sự không khớp giữa khai báo và định nghĩa có thể chỉ ra một lỗi lập trình.

Lưu ý: Một danh sách tham số trống không hợp lệ trong một nguyên mẫu. Nếu một loại hàm không có tham số, dạng nguyên mẫu của nó sử dụng từ khóa void.

##### Ví dụ

Ví dụ đầu tiên cho thấy các khai báo của một số hàm và các định nghĩa tương ứng cho một số hàm đó.

```c

/\* Tuân thủ \*/

extern int16\_t func1 ( int16\_t n );

/\* Không tuân thủ - tên tham số không được chỉ định \*/

extern void func2 ( int16\_t );

/\* Không tuân thủ - không ở dạng nguyên mẫu \*/

static int16\_t func3 ( );

/\* Tuân thủ - nguyên mẫu chỉ định 0 tham số \*/

static int16\_t func4 ( void );

/\* Tuân thủ \*/

int16\_t func1 ( int16\_t n )

{

return n;

}

/\* Không tuân thủ - danh sách định danh và khai báo kiểu cũ \*/

static int16\_t func3 ( vec, n )

int16\_t \*vec;

int16\_t n;

{

return vec[ n - 1 ];

}

```

Phần ví dụ này cho thấy việc áp dụng quy tắc cho các loại hàm khác ngoài các khai báo và định nghĩa hàm.

```c

/\* Không tuân thủ - không có nguyên mẫu \*/

int16\_t ( \*pf1 ) ( );

/\* Tuân thủ - nguyên mẫu chỉ định 0 tham số \*/

int16\_t ( \*pf1 ) ( void );

/\* Không tuân thủ - tên tham số không được chỉ định \*/

typedef int16\_t ( \*pf2\_t ) ( int16\_t );

/\* Tuân thủ \*/

typedef int16\_t ( \*pf3\_t ) ( int16\_t n );

```

#### Xem thêm

- Quy tắc 8.1, Quy tắc 8.4, Quy tắc 17.3

#### Quy tắc 8.3: Tất cả các khai báo của một đối tượng hoặc hàm phải sử dụng cùng tên và các định danh loại

- C90 [Không xác định 10], C99 [Không xác định 14], [Koenig 59-62]

- Phân loại: Bắt buộc

- Phân tích: Quyết định được, Hệ thống

- Áp dụng cho: C90, C99

##### Mở rộng

Các chỉ định lớp lưu trữ không nằm trong phạm vi của quy tắc này.

##### Lý do

Sử dụng các loại và định danh nhất quán trên các khai báo của cùng một đối tượng hoặc hàm khuyến khích gõ mạnh hơn.

Chỉ định tên tham số trong các nguyên mẫu hàm cho phép định nghĩa hàm được kiểm tra tính nhất quán giao diện với các khai báo của nó.

##### Ngoại lệ

Các phiên bản tương thích của cùng một loại cơ bản có thể được sử dụng thay thế lẫn nhau. Ví dụ, int, signed và signed int đều tương đương.

##### Ví dụ

```c

extern void f ( signed int );

void f ( int ); /\* Tuân thủ - Ngoại lệ \*/

extern void g ( int \* const );

void g ( int \* ); /\* Không tuân thủ - định danh loại \*/

```

Lưu ý: tất cả các ví dụ trên đều không tuân thủ với Dir 4.6.

66666666