# MISRA C:2012

Tháng 8 năm 2024

# TUYÊN BỐ SỨ MỆNH CỦA MISRA

Chúng tôi cung cấp các hướng dẫn về thực hành tốt nhất hàng đầu thế giới để ứng dụng an toàn cho cả hệ thống điều khiển nhúng và phần mềm độc lập.

MISRA, Hiệp hội Độ tin cậy Phần mềm Công nghiệp Ô tô, là sự hợp tác giữa các nhà sản xuất, nhà cung cấp linh kiện và các công ty tư vấn kỹ thuật nhằm thúc đẩy thực hành tốt nhất trong việc phát triển các hệ thống nhúng liên quan đến an toàn và các ứng dụng phần mềm khác đòi hỏi sự phức tạp. Để đạt được điều này, MISRA xuất bản các tài liệu cung cấp thông tin dễ tiếp cận cho các kỹ sư và nhà quản lý, và tổ chức các sự kiện nhằm cho phép trao đổi kinh nghiệm giữa các nhà thực hành.

[www.misra.org.uk](http://www.misra.org.uk)

# TUYÊN BỐ TỪ CHỐI TRÁCH NHIỆM

Tuân thủ các yêu cầu của tài liệu này không tự đảm bảo phần mềm không có lỗi, đáng tin cậy hoặc đảm bảo khả năng di động và tái sử dụng.

Tuân thủ các yêu cầu của tài liệu này, hoặc bất kỳ tiêu chuẩn nào khác, tự nó không miễn trừ các nghĩa vụ pháp lý.

# LỜI NÓI ĐẦU

Lúc đầu, phiên bản thứ ba của Hướng dẫn MISRA C này có thể trông khá đáng sợ. Vì có kích thước gấp đôi phiên bản trước, người ta có thể nghĩ rằng nó chứa gấp đôi số lượng hướng dẫn, và việc tuân thủ những hướng dẫn đó có thể tốn gấp đôi công sức.

Thực tế, số lượng hướng dẫn tăng không đáng kể, chỉ khoảng 10%. Phần còn lại của sự gia tăng kích thước là do những cải tiến trong hướng dẫn, chẳng hạn như:

* Lý do thuyết phục hơn cho các hướng dẫn;
* Mô tả chính xác hơn;
* Ví dụ mã, thể hiện sự tuân thủ và không tuân thủ, cho hầu hết các hướng dẫn;
* Hướng dẫn chi tiết hơn về việc kiểm tra tuân thủ và quy trình chấp nhận sai lệch;
* Các bảng kiểm tra có thể được sử dụng để hỗ trợ một tuyên bố tuân thủ.

Cuối cùng, tôi muốn nhấn mạnh đến các phần giới thiệu của tài liệu này. Chúng không chỉ chứa các hướng dẫn thực tế về cách sử dụng MISRA C, mà đồng thời, cũng đã được làm rõ hơn so với các phiên bản trước. Tôi khuyến khích tất cả người dùng làm quen với tài liệu này.

Steve Montgomery MA (Cantab), PhD Chủ tịch, Nhóm Công tác MISRA C.

# MỤC LỤC

[MISRA C:2012 1](#_Toc184515503)

[TUYÊN BỐ SỨ MỆNH CỦA MISRA 2](#_Toc184515504)

[TUYÊN BỐ TỪ CHỐI TRÁCH NHIỆM 2](#_Toc184515505)

[LỜI NÓI ĐẦU 3](#_Toc184515506)

[NỘI DUNG 6](#_Toc184515507)

[1. Tổng quan 6](#_Toc184515508)

[1.1. Mục tiêu của MISRA C 6](#_Toc184515509)

[1.2. Hướng dẫn của MISRA C 6](#_Toc184515510)

[1.3. Phiên bản trước của MISRA C 6](#_Toc184515511)

[1.4. Mục tiêu của phiên bản MISRA C 2012 6](#_Toc184515512)

[1.5. Lợi ích của phiên bản MISRA C 2012 7](#_Toc184515513)

[2. Nền tảng của MISRA C 8](#_Toc184515514)

[2.1. Sự phổ biến của ngôn ngữ C 8](#_Toc184515515)

[2.2. Nhược điểm của ngôn ngữ C 8](#_Toc184515516)

[2.2.1. Định nghĩa ngôn ngữ 9](#_Toc184515517)

[2.2.2. Lạm dụng ngôn ngữ 9](#_Toc184515518)

[2.2.3. Hiểu lầm về ngôn ngữ 10](#_Toc184515519)

[2.2.4. Kiểm tra lỗi thời gian chạy (Runtime Error Checking) 10](#_Toc184515520)

[3. Lựa chọn Tiêu chuẩn C và MISRA C 12](#_Toc184515521)

[3.1. Các phiên bản của tiêu chuẩn C 12](#_Toc184515522)

[3.2. Lựa chọn giữa Tiêu chuẩn C90 và C99 12](#_Toc184515523)

[3.3. Cách lựa chọn trình biên dịch 12](#_Toc184515524)

[3.4. Công cụ phân tích tĩnh 13](#_Toc184515525)

[3.4.1. Các tiêu chí lựa chọn công cụ phân tích tĩnh 13](#_Toc184515526)

[3.4.2. Cân bằng giữa tốc độ và độ chính xác 13](#_Toc184515527)

[3.4.3. Công cụ phân tích tĩnh và yêu cầu an toàn 14](#_Toc184515528)

[4. Kiến thức cần thiết 15](#_Toc184515529)

[4.1. Kiến thức nền tảng 15](#_Toc184515530)

[4.2. Hiểu về trình biên dịch 15](#_Toc184515531)

[4.3. Hiểu về các công cụ phân tích tĩnh 16](#_Toc184515532)

[5. Áp dụng và sử dụng MISRA C 18](#_Toc184515533)

[5.1. Áp dụng 18](#_Toc184515534)

[5.2. Quy trình phát triển phần mềm 18](#_Toc184515535)

[5.2.1. Cách hoạt động quy trình yêu cầu tới MISRA C 19](#_Toc184515536)

[5.2.2. Cách hoạt động và quy trình được mong đợi MISRA C 19](#_Toc184515537)

[5.3. Tuân thủ 20](#_Toc184515538)

[5.3.1. Cấu hình trình biên dịch 22](#_Toc184515539)

[5.3.2. Cấu hình công cụ phân tích tatic 22](#_Toc184515540)

[5.3.3. Điều tra thông báo 23](#_Toc184515541)

[5.4. Thủ tục sai lệch 23](#_Toc184515542)

[5.5. Tuyên bố tuân thủ 24](#_Toc184515543)

# NỘI DUNG

## Tổng quan

MISRA C 2012 là một bộ quy tắc giúp lập trình viên viết mã C an toàn và ít lỗi hơn, đặc biệt trong các hệ thống đòi hỏi độ an toàn cao.

### Mục tiêu của MISRA C

MISRA C ra đời nhằm giảm thiểu các lỗi trong lập trình C bằng cách cung cấp các quy tắc và hướng dẫn cụ thể. Nó đặc biệt cần thiết trong các hệ thống đòi hỏi độ an toàn cao, ví dụ như trong lĩnh vực ô tô, hàng không, y tế,... Những hệ thống này yêu cầu mã nguồn phải được kiểm tra rất kỹ để tránh lỗi gây hậu quả nghiêm trọng. MISRA C không chỉ hữu ích cho các hệ thống an toàn, mà còn cho các ứng dụng cần độ chính xác và độ tin cậy cao.

### Hướng dẫn của MISRA C

* **Tài liệu hướng dẫn**: Đây là tài liệu nhằm giúp các lập trình viên hiểu rõ hơn về các quy tắc của MISRA C và cách áp dụng chúng. Nó giúp lập trình viên phát triển phần mềm an toàn hơn, hiệu quả hơn.
* **Tài liệu tham khảo**: Dành cho những người phát triển các công cụ phân tích mã nguồn, giúp tự động kiểm tra và xác minh mã có tuân theo các quy tắc MISRA C hay không.

### Phiên bản trước của MISRA C

Các phiên bản MISRA C trước đây dựa trên chuẩn ISO C năm 1990 (C90). Nhưng khi chuẩn ISO C năm 1999 (C99) ra đời và được sử dụng rộng rãi, đặc biệt trong hệ thống nhúng (embedded systems), vì vậy MISRA C 2012 ra đời và cập nhật để phù hợp với các chuẩn mới này.

### Mục tiêu của phiên bản MISRA C 2012

* Hỗ trợ cả hai chuẩn ISO C 1999 và 1990: Phiên bản mới vẫn duy trì hỗ trợ cho các dự án đang sử dụng chuẩn C cũ (1990), đồng thời hỗ trợ các tính năng mới của chuẩn C năm 1999.
* Sửa lỗi từ các phiên bản trước: Các vấn đề trong các phiên bản trước được xem xét và sửa chữa nếu cần thiết.
* Thêm hướng dẫn mới: Các hướng dẫn mới được bổ sung dựa trên những lý do hợp lý, tức là các lỗi hoặc vấn đề phát sinh khi lập trình sẽ được cân nhắc để đưa ra các quy tắc mới.
* Cải tiến quy tắc hiện có: Các quy tắc cũ được viết lại hoặc điều chỉnh để rõ ràng và dễ hiểu hơn.
* Loại bỏ các quy tắc lỗi thời: Nếu một quy tắc nào đó không còn phù hợp hoặc không còn lý do hợp lý để áp dụng, nó sẽ bị loại bỏ.
* Tăng số lượng quy tắc có thể kiểm tra tự động: MISRA C 2012 cố gắng tạo ra nhiều quy tắc có thể được kiểm tra bằng các công cụ phân tích tĩnh (static analysis tools). Điều này giúp lập trình viên phát hiện lỗi sớm và dễ dàng hơn trong quá trình phát triển phần mềm.

### Lợi ích của phiên bản MISRA C 2012

* Lập trình viên có thể viết mã C an toàn hơn, ít lỗi hơn.
* Các quy tắc được viết rõ ràng và dễ hiểu, giúp lập trình viên dễ dàng tuân thủ.
* Việc sử dụng các công cụ phân tích tĩnh để kiểm tra mã giúp tăng tốc độ phát hiện lỗi.

## Nền tảng của MISRA C

### Sự phổ biến của ngôn ngữ C

Ngôn ngữ lập trình C rất phổ biến trong phát triển phần mềm, đặc biệt là các hệ thống nhúng và hệ thống quan trọng, do những ưu điểm sau:

* **Sự linh hoạt về trình biên dịch**: Trình biên dịch C có mặt trên rất nhiều nền tảng, từ máy tính cá nhân đến các vi xử lý nhúng. Điều này làm cho C trở thành ngôn ngữ có tính tương thích cao, có thể chạy trên nhiều thiết bị và kiến trúc phần cứng khác nhau.
* **Hiệu suất cao**: Chương trình viết bằng C có thể được biên dịch thành mã máy rất hiệu quả. Điều này giúp tối ưu hóa tốc độ chạy của chương trình và sử dụng tài nguyên hệ thống (CPU, bộ nhớ) một cách tiết kiệm nhất.
* **Tiêu chuẩn quốc tế**: Ngôn ngữ C tuân theo các tiêu chuẩn quốc tế như ISO C, giúp đảm bảo tính nhất quán giữa các nền tảng. Điều này nghĩa là một chương trình viết bằng C sẽ có cơ hội chạy trên nhiều hệ thống khác nhau mà không cần thay đổi nhiều về mã nguồn.
* **Truy cập phần cứng trực tiếp**: C cung cấp cơ chế cho phép lập trình viên truy cập trực tiếp vào phần cứng của hệ thống, ví dụ như địa chỉ bộ nhớ hoặc các thanh ghi của vi xử lý. Điều này rất quan trọng trong lập trình nhúng và các ứng dụng điều khiển phần cứng.
* **Sự phổ biến trong các hệ thống quan trọng**: Ngôn ngữ C đã được sử dụng rộng rãi trong các hệ thống đòi hỏi tính an toàn và độ tin cậy cao như hàng không, ô tô, y tế,... Do đó, đã có rất nhiều kinh nghiệm và kiến thức tích lũy trong việc sử dụng C để xây dựng các ứng dụng quan trọng.
* **Hỗ trợ công cụ phân tích và kiểm tra phần mềm**: Ngôn ngữ C có nhiều công cụ mạnh mẽ hỗ trợ việc phân tích mã nguồn và kiểm tra phần mềm, như các công cụ phân tích tĩnh. Các công cụ này giúp phát hiện lỗi sớm trong quá trình phát triển và cải thiện chất lượng mã nguồn.

### Nhược điểm của ngôn ngữ C

Dù ngôn ngữ C có nhiều ưu điểm, nhưng nó cũng có một số nhược điểm khiến việc lập trình với C trở nên phức tạp hơn, đặc biệt là trong các hệ thống đòi hỏi an toàn cao. MISRA C ra đời để giúp giảm thiểu những nhược điểm này.

#### Định nghĩa ngôn ngữ

Tiêu chuẩn ISO C không định nghĩa tất cả các chi tiết của ngôn ngữ mà để lại một số khía cạnh cho từng trình biên dịch hoặc nền tảng triển khai tự quyết định. Điều này nhằm giúp C tương thích với nhiều loại bộ vi xử lý khác nhau, nhưng lại dẫn đến một số vấn đề:

* Hành vi không được định nghĩa rõ ràng (undefined behavior): Đây là những trường hợp mà chuẩn C không quy định cách hệ thống phải xử lý. Ví dụ, nếu chương trình sử dụng một biến chưa được khởi tạo, hành vi của chương trình sẽ không đoán trước được.
* Hành vi không được xác định cụ thể (unspecified behavior): Đây là những trường hợp mà cách hành xử có thể thay đổi tùy thuộc vào trình biên dịch hoặc môi trường cụ thể, nhưng không nhất thiết phải gây lỗi. Ví dụ, thứ tự đánh giá các toán tử có thể thay đổi giữa các trình biên dịch.
* Hành vi phụ thuộc vào cách triển khai (implementation-defined behavior): Trong trường hợp này, trình biên dịch hoặc môi trường triển khai có thể tự quyết định cách thực hiện, miễn là hành vi đó được tài liệu hóa rõ ràng. Điều này có thể dẫn đến việc chương trình hoạt động khác nhau trên các hệ thống khác nhau.

Những vấn đề trên có thể làm cho mã nguồn khó dự đoán và khó chuyển giữa các hệ thống. Ngoài ra, nếu mã phụ thuộc quá nhiều vào hành vi không được định nghĩa hoặc do triển khai quyết định, việc sử dụng công cụ phân tích tĩnh để kiểm tra mã cũng sẽ gặp khó khăn.

#### Lạm dụng ngôn ngữ

Ngôn ngữ C tuy mạnh mẽ và linh hoạt, nhưng cũng dễ bị lạm dụng, dẫn đến mã nguồn khó đọc và dễ gây nhầm lẫn. Một số ví dụ về lạm dụng trong C:

* Toán tử dễ gây lỗi: Một trong những lỗi phổ biến nhất trong C là nhầm lẫn giữa phép gán (=) và phép so sánh (==).

Ví dụ:

|  |
| --- |
| if ( a == b )  // // So sánh a và b có bằng nhau không  if ( a = b )    // Gán b cho a, sau đó kiểm tra xem a có khác 0 không |

Cả hai câu lệnh này đều hợp lệ với trình biên dịch, nhưng chúng có ý nghĩa hoàn toàn khác nhau. Lỗi này thường khó phát hiện vì không gây ra lỗi biên dịch, nhưng có thể dẫn đến lỗi logic rất nghiêm trọng.

#### Hiểu lầm về ngôn ngữ

Ngôn ngữ C có một số đặc điểm khiến lập trình viên dễ hiểu lầm hoặc gặp khó khăn, đặc biệt khi họ chưa quen với cách thức hoạt động của nó:

* Toán tử và mức độ ưu tiên: Ngôn ngữ C có rất nhiều toán tử với các mức độ ưu tiên khác nhau. Điều này có thể gây khó khăn cho lập trình viên trong việc hiểu rõ các quy tắc kết hợp và thứ tự thực thi các toán tử. Ví dụ, trong một biểu thức phức tạp với nhiều toán tử, việc xác định toán tử nào được thực thi trước có thể không luôn trực quan, đặc biệt đối với những người mới làm quen với ngôn ngữ. Việc hiểu nhầm về mức độ ưu tiên có thể dẫn đến các lỗi logic khó phát hiện.
* Quy tắc về kiểu dữ liệu: Trong C, các phép toán có thể tự động "nâng cấp" các toán hạng lên kiểu dữ liệu rộng hơn. Ví dụ, khi thực hiện một phép toán giữa một số nguyên (int) và một số dấu phẩy động (float), số nguyên có thể tự động được chuyển đổi thành số dấu phẩy động trước khi phép toán diễn ra. Điều này đôi khi không rõ ràng và dễ gây nhầm lẫn, đặc biệt đối với các lập trình viên quen thuộc với các ngôn ngữ lập trình có kiểu dữ liệu chặt chẽ và không cho phép chuyển đổi kiểu ngầm định.

Một ví dụ điển hình về hiểu nhầm này là khi làm việc với các giá trị nhỏ, như kiểu char, có thể tự động được nâng cấp lên kiểu int khi tham gia vào các phép toán, điều này dẫn đến kết quả không như mong đợi. Các lỗi kiểu dữ liệu này có thể không được trình biên dịch cảnh báo, khiến chúng khó phát hiện trong quá trình lập trình.

#### Kiểm tra lỗi thời gian chạy (Runtime Error Checking)

Ngôn ngữ C được thiết kế để tối ưu hóa cho hiệu suất và kích thước mã biên dịch. Tuy nhiên, điều này cũng dẫn đến một số hạn chế trong việc kiểm tra lỗi trong quá trình chạy chương trình. Không giống như một số ngôn ngữ hiện đại có cơ chế kiểm tra lỗi runtime mạnh mẽ, C yêu cầu lập trình viên phải chủ động thực hiện các kiểm tra này. Dưới đây là một số loại lỗi thời gian chạy mà ngôn ngữ C không tự động kiểm tra:

* Ngoại lệ số học: Ví dụ, khi xảy ra phép chia cho 0, chương trình có thể gây ra lỗi, nhưng C không có cơ chế tự động bắt và xử lý lỗi này như một số ngôn ngữ khác.
* Tràn số (integer overflow): Khi giá trị của một biến vượt quá giới hạn của kiểu dữ liệu, C sẽ không đưa ra cảnh báo mà tiếp tục thực hiện phép toán. Kết quả có thể không chính xác và gây ra các lỗi logic nghiêm trọng trong chương trình. Ví dụ, với một biến kiểu unsigned int khi tràn số, nó sẽ quay trở về giá trị nhỏ nhất của kiểu đó thay vì cảnh báo lỗi.
* Lỗi con trỏ: Con trỏ là một phần quan trọng của ngôn ngữ C, nhưng chúng cũng là một nguồn lớn gây lỗi. C không tự động kiểm tra tính hợp lệ của con trỏ, ví dụ khi truy cập một vùng nhớ không hợp lệ hoặc sử dụng con trỏ chưa được khởi tạo. Điều này có thể dẫn đến lỗi "segmentation fault" (vi phạm bộ nhớ), khiến chương trình dừng đột ngột.
* Lỗi phạm vi mảng (Array bounds): C cho phép lập trình viên truy cập các phần tử của mảng mà không kiểm tra xem chỉ số có nằm trong phạm vi hợp lệ của mảng hay không. Điều này có thể dẫn đến truy cập ngoài phạm vi (out-of-bounds), gây ra các lỗi khó lường, từ việc làm hỏng dữ liệu đến sụp đổ hệ thống.

Trong các ngôn ngữ lập trình có cơ chế quản lý lỗi runtime tốt hơn, chẳng hạn như Java, nhiều lỗi này sẽ được phát hiện và xử lý ngay trong quá trình chạy chương trình, giúp bảo vệ chương trình khỏi các lỗi tiềm ẩn. Tuy nhiên, trong C, lập trình viên cần phải tự thực hiện các kiểm tra này, điều này đòi hỏi sự cẩn thận và kỹ năng. Trong các hệ thống yêu cầu độ an toàn cao như hệ thống nhúng hoặc hệ thống quan trọng (critical systems), việc không xử lý tốt các lỗi runtime có thể dẫn đến những hậu quả nghiêm trọng.

## Lựa chọn Tiêu chuẩn C và MISRA C

### Các phiên bản của tiêu chuẩn C

Ngôn ngữ lập trình C có nhiều phiên bản tiêu chuẩn khác nhau, mỗi phiên bản được quy định bởi các tổ chức tiêu chuẩn quốc tế và thường có sự cải tiến để phù hợp với các yêu cầu mới của lập trình và phần cứng. Dưới đây là một số phiên bản tiêu chuẩn C phổ biến:

* C90 (ISO/IEC 9899:1990): Đây là một trong những phiên bản đầu tiên của ngôn ngữ C được tiêu chuẩn hóa và vẫn còn được sử dụng rộng rãi. Nhiều trình biên dịch hiện nay vẫn hỗ trợ tiêu chuẩn này vì có một lượng lớn mã nguồn cũ được viết dựa trên C90, và các dự án lớn, lâu đời thường cần duy trì tính tương thích với mã cũ.
* C99 (ISO/IEC 9899:1999): Tiêu chuẩn C99 mang lại nhiều cải tiến đáng kể so với C90, bao gồm hỗ trợ thêm các tính năng như biến cục bộ trong khối lệnh, khai báo biến động thời gian chạy (VLA), và nhiều cải tiến khác về cú pháp và tính năng ngôn ngữ.
* C11 (ISO/IEC 9899:2011): Đây là phiên bản cải tiến tiếp theo của C. Tuy nhiên, Hướng dẫn MISRA C chưa hỗ trợ trực tiếp tiêu chuẩn này, do C11 ra đời sau khi các hướng dẫn MISRA đã hoàn thiện.

### Lựa chọn giữa Tiêu chuẩn C90 và C99

Khi lựa chọn giữa C90 và C99, bạn cần cân nhắc các yếu tố sau:

* Khối lượng mã nguồn cũ cần tái sử dụng: Nếu dự án của bạn có nhiều mã nguồn cũ dựa trên C90, việc chọn C90 có thể giúp bạn duy trì tính tương thích và giảm thiểu việc viết lại hoặc chỉnh sửa mã.
* Trình biên dịch hỗ trợ bộ vi xử lý mục tiêu: Tùy vào bộ vi xử lý bạn sử dụng, bạn cần kiểm tra xem trình biên dịch có hỗ trợ phiên bản tiêu chuẩn nào (C90, C99, hay C11). Một số trình biên dịch có thể chỉ hỗ trợ một trong hai tiêu chuẩn này, hoặc cung cấp tùy chọn để lập trình viên chọn tiêu chuẩn phù hợp với dự án.

Ngoài ra, trình biên dịch lý tưởng cần tuân thủ đầy đủ tiêu chuẩn ISO cho phiên bản C mà bạn chọn, và nhà cung cấp trình biên dịch nên cung cấp thông tin về các thử nghiệm tuân thủ để đảm bảo tính tương thích.

### Cách lựa chọn trình biên dịch

Khi gặp khó khăn trong việc chọn trình biên dịch, bạn có thể làm theo các bước sau:

* Kiểm tra quy trình phát triển phần mềm của nhà phát triển trình biên dịch: Ví dụ, xem xét liệu nhà phát triển có tuân thủ các quy trình đạt chuẩn ISO 9001, đảm bảo chất lượng sản phẩm.
* Đọc đánh giá từ cộng đồng: Tham khảo các phản hồi và kinh nghiệm sử dụng từ những lập trình viên khác trong cộng đồng.
* Tự kiểm thử: Bạn có thể kiểm tra trình biên dịch bằng cách biên dịch các ứng dụng mẫu hoặc sử dụng các bộ kiểm thử tuân thủ độc lập để đảm bảo chất lượng và tính tuân thủ.

### Công cụ phân tích tĩnh

Công cụ phân tích tĩnh là các công cụ tự động kiểm tra mã nguồn để đảm bảo rằng mã tuân thủ các hướng dẫn của MISRA C và không có lỗi tiềm ẩn. Mặc dù có thể kiểm tra mã nguồn theo cách thủ công, điều này rất tốn thời gian và dễ mắc lỗi. Do đó, việc sử dụng công cụ phân tích tĩnh là rất quan trọng và cần thiết trong bất kỳ quy trình kiểm tra mã nào.

#### Các tiêu chí lựa chọn công cụ phân tích tĩnh

Một công cụ phân tích tĩnh tốt cần đảm bảo các yêu cầu sau:

* Phát hiện tất cả các vi phạm MISRA C: Công cụ phải có khả năng nhận diện tất cả các vi phạm của hướng dẫn MISRA C trong mã nguồn.
* Giảm thiểu cảnh báo sai: Một trong những vấn đề với công cụ phân tích tĩnh là có thể tạo ra cảnh báo sai (false positives), tức là cảnh báo về các lỗi không thực sự tồn tại. Một công cụ lý tưởng nên chỉ báo cáo các lỗi thực sự và giảm thiểu tối đa các cảnh báo sai.

#### Cân bằng giữa tốc độ và độ chính xác

* Công cụ nhanh: Các công cụ phân tích tĩnh nhanh thường chỉ mất vài giây để chạy, nhưng có thể tạo ra nhiều cảnh báo sai hơn do quá trình phân tích không sâu.
* Công cụ kỹ lưỡng: Ngược lại, các công cụ phân tích kỹ lưỡng hơn có thể mất nhiều thời gian (thậm chí vài ngày) để phân tích toàn bộ mã nguồn, nhưng chúng có độ chính xác cao hơn và ít cảnh báo sai hơn.

Do đó, bạn cần phải cân nhắc giữa tốc độ phân tích và độ chính xác của kết quả khi chọn công cụ phân tích tĩnh. Đôi khi, việc kết hợp sử dụng nhiều công cụ phân tích khác nhau cũng là một giải pháp hợp lý để đảm bảo bao phủ toàn diện các vấn đề tiềm ẩn trong mã nguồn.

#### Công cụ phân tích tĩnh và yêu cầu an toàn

Trong các hệ thống yêu cầu độ an toàn cao, các tiêu chuẩn như IEC 61508, ISO 26262, và DO-178C cũng yêu cầu các công cụ phân tích tĩnh phải đáp ứng một số tiêu chí về trình độ chuyên môn và tuân thủ các quy trình an toàn. Điều này đảm bảo rằng công cụ phân tích có đủ khả năng kiểm tra và xác nhận rằng mã nguồn đáp ứng các yêu cầu an toàn nghiêm ngặt của hệ thống.

## Kiến thức cần thiết

### Kiến thức nền tảng

Để đảm bảo lập trình viên C có đủ kỹ năng và kiến thức cần thiết, các khóa đào tạo chuyên biệt là rất quan trọng. Một số lĩnh vực chính cần đào tạo bao gồm:

* Sử dụng C cho các ứng dụng nhúng: Trong các hệ thống nhúng, lập trình viên cần hiểu cách mã C tương tác trực tiếp với phần cứng. Điều này bao gồm làm việc với các thanh ghi, quản lý tài nguyên hệ thống như bộ nhớ và bộ đếm thời gian, cũng như hiểu cách trình biên dịch tối ưu mã cho các hệ thống tài nguyên giới hạn.
* Sử dụng C trong các hệ thống có độ an toàn và toàn vẹn cao: Các hệ thống đòi hỏi tính an toàn cao như hàng không, y tế, hoặc ô tô cần lập trình viên hiểu rõ về các phương pháp lập trình đảm bảo tính ổn định, đáng tin cậy, và an toàn. MISRA C thường được sử dụng trong các hệ thống này, nên lập trình viên cần nắm vững các quy tắc của nó.

Ngoài ra, với các trình biên dịch và công cụ phân tích tĩnh là những phần mềm phức tạp, việc đào tạo chuyên sâu về cách sử dụng chúng là cần thiết. Các khóa học về cách kiểm tra mã theo tiêu chuẩn MISRA C sẽ giúp lập trình viên dễ dàng nhận diện và khắc phục vi phạm trong mã nguồn.

### Hiểu về trình biên dịch

Trong ngữ cảnh của MISRA C, "Trình biên dịch" không chỉ đơn giản là phần mềm biên dịch mà còn bao gồm nhiều công cụ liên quan khác như:

* Trình liên kết (linker): Kết nối các đoạn mã và thư viện lại thành một tệp thực thi duy nhất.
* Trình quản lý thư viện: Quản lý các thư viện mã dùng chung, cho phép lập trình viên tái sử dụng mã mà không cần viết lại.
* Công cụ chuyển đổi định dạng tệp thực thi: Chuyển đổi mã sau khi biên dịch thành định dạng tệp có thể thực thi trên nền tảng mục tiêu.

Trình biên dịch thường cung cấp các tùy chọn để kiểm soát quá trình biên dịch mã, và lập trình viên cần hiểu rõ các tùy chọn này vì chúng có thể ảnh hưởng đến:

* Khả năng mở rộng: Mã có thể dễ dàng chuyển đổi và chạy trên các nền tảng và môi trường khác nhau.
* Tuân thủ tiêu chuẩn ISO C: Đảm bảo rằng mã nguồn tuân thủ các quy định của tiêu chuẩn ISO C, như C90, C99 hoặc C11.
* Tài nguyên hệ thống: Tối ưu hóa hiệu suất chương trình, bao gồm cả tốc độ xử lý và dung lượng bộ nhớ.
* Rủi ro lỗi từ trình biên dịch: Đặc biệt khi sử dụng các tùy chọn tối ưu hóa phức tạp, trình biên dịch có thể gây ra lỗi mà lập trình viên khó phát hiện.

Các yếu tố cần hiểu về trình biên dịch:

* Tính năng "xác định bởi việc triển khai": Một số tính năng trong ngôn ngữ C phụ thuộc vào cách triển khai cụ thể của trình biên dịch. Lập trình viên cần nắm rõ những tính năng này vì chúng có thể thay đổi khi chạy trên các trình biên dịch khác nhau hoặc các nền tảng khác nhau.
* Mở rộng ngôn ngữ: Một số trình biên dịch có thể hỗ trợ các tính năng mở rộng ngoài tiêu chuẩn ISO. Hiểu về những mở rộng này rất quan trọng nếu bạn đang làm việc trong một môi trường mà chúng được sử dụng.

Ngoài ra, nhà phát triển trình biên dịch thường cung cấp danh sách các lỗi đã biết và giải pháp khắc phục. Lập trình viên cần nắm rõ danh sách này để tránh các vấn đề tiềm ẩn trong quá trình phát triển. Nếu không có danh sách chính thức từ nhà phát triển, việc duy trì danh sách lỗi phát hiện được và báo cáo lại cho nhà phát triển là một thói quen tốt để tránh lặp lại lỗi.

### Hiểu về các công cụ phân tích tĩnh

Công cụ phân tích tĩnh đóng vai trò quan trọng trong việc kiểm tra mã nguồn tuân thủ tiêu chuẩn MISRA C. Để sử dụng hiệu quả các công cụ này, lập trình viên cần hiểu rõ cách chúng hoạt động thông qua việc tham khảo tài liệu hướng dẫn. Một số điểm chính cần xem xét bao gồm:

* Cấu hình công cụ phù hợp với trình biên dịch: Một số cài đặt của trình biên dịch, chẳng hạn như kích thước kiểu số nguyên (int, long), có thể ảnh hưởng đến cách mã nguồn được phân tích. Việc hiểu rõ cách cấu hình công cụ phân tích tĩnh cho phù hợp với trình biên dịch đang sử dụng là rất quan trọng.
* Phạm vi hỗ trợ của công cụ: Không phải tất cả các công cụ phân tích tĩnh đều hỗ trợ toàn bộ hướng dẫn của MISRA C. Lập trình viên cần biết công cụ có thể kiểm tra những quy tắc nào để tránh bỏ sót vi phạm.
* Hỗ trợ mở rộng ngôn ngữ và tính năng không chuẩn: Nếu trình biên dịch của bạn sử dụng các tính năng mở rộng không nằm trong chuẩn ISO, cần kiểm tra xem công cụ phân tích tĩnh có thể xử lý được các tính năng này không.

Hiểu rõ những yếu tố này giúp lập trình viên sử dụng công cụ phân tích tĩnh một cách hiệu quả hơn, từ đó đảm bảo rằng mã nguồn tuân thủ đúng tiêu chuẩn và giảm thiểu lỗi trong quá trình phát triển phần mềm.

## Áp dụng và sử dụng MISRA C

### Áp dụng

MISRA C nên được **áp dụng ngay từ đầu dự án** để đảm bảo mã nguồn tuân thủ tiêu chuẩn an toàn và đáng tin cậy.

Tuy nhiên, nếu dự án đang xây dựng dựa trên **mã nguồn hiện có**, đã được kiểm chứng là ổn định, thì việc áp dụng MISRA C có thể tiềm ẩn rủi ro. Điều này là do trong quá trình điều chỉnh mã để tuân thủ, có thể vô tình tạo ra lỗi mới. Trong những trường hợp như vậy, **quyết định áp dụng MISRA C** nên dựa trên việc cân nhắc kỹ lưỡng về **lợi ích ròng** mang lại, tức là liệu lợi ích của việc tuân thủ có lớn hơn rủi ro không.

### Quy trình phát triển phần mềm

MISRA C được thiết kế để hoạt động tốt nhất trong một quy trình phát triển phần mềm có tài liệu đầy đủ và được tổ chức cẩn thận. Mặc dù có thể sử dụng MISRA C một cách độc lập, nó sẽ mang lại hiệu quả cao hơn khi được tích hợp vào một quy trình phát triển phần mềm chuẩn, bao gồm các bước như:

* Yêu cầu phần mềm: Đảm bảo rằng các yêu cầu phần mềm, đặc biệt là yêu cầu về an toàn, được xác định đầy đủ, rõ ràng và chính xác.
* Đặc tả thiết kế: Đảm bảo các thiết kế phần mềm chính xác và phù hợp với các yêu cầu, không có các chức năng ngoài yêu cầu.
* Mã hóa và biên dịch: Đảm bảo rằng các mô-đun được biên dịch hoạt động chính xác như đã được thiết kế.
* Kiểm tra mô-đun: Kiểm tra các mô-đun riêng lẻ và cùng nhau để phát hiện và loại bỏ lỗi.

Việc áp dụng MISRA C ngay từ giai đoạn đầu của quy trình phát triển là rất quan trọng. Nếu kiểm tra tuân thủ MISRA C diễn ra muộn trong quá trình phát triển, dự án có thể phải mất nhiều thời gian để sửa đổi, viết lại và kiểm tra lại mã. Vì vậy, quy trình phát triển phần mềm nên yêu cầu lập trình viên tuân thủ các nguyên tắc của MISRA C ngay từ giai đoạn đầu để tránh rủi ro và tiết kiệm thời gian.

MISRA C thường được áp dụng trong các quy trình phát triển phần mềm liên quan đến an toàn, nhưng yêu cầu chi tiết về các quy trình này nằm ngoài phạm vi của tài liệu MISRA. Tuy nhiên, các tiêu chuẩn và hướng dẫn phát triển phần mềm liên quan đến an toàn như IEC 61508, ISO 26262, DO-178C, EN 50128, và IEC 62304 cung cấp các ví dụ cụ thể về quy trình phát triển.

Phần tiếp theo của tài liệu sẽ bàn về sự tương tác giữa MISRA C và quy trình phát triển phần mềm để đảm bảo tuân thủ và tính an toàn.

#### Cách hoạt động quy trình yêu cầu tới MISRA C

Để áp dụng MISRA C một cách hiệu quả, cần phải phát triển và ghi lại những yếu tố quan trọng sau:

1. Ma trận tuân thủ: Đây là một tài liệu theo dõi quá trình tuân thủ từng hướng dẫn của MISRA C. Nó chỉ ra cách mỗi quy tắc của MISRA C sẽ được kiểm tra và chứng minh trong mã nguồn. Ma trận này giúp đảm bảo mọi quy tắc đều được đánh giá kỹ lưỡng.
2. Quy trình xử lý các trường hợp không tuân thủ: Không phải lúc nào cũng có thể tuân thủ hoàn toàn tất cả các hướng dẫn của MISRA C. Trong trường hợp đó, cần có quy trình biện minh cho việc không tuân thủ, nhận ủy quyền, và ghi lại lý do. Quy trình này phải được quản lý chặt chẽ để đảm bảo sự nhất quán và tính minh bạch.

Ngoài ra, quy trình phát triển phần mềm cần ghi lại các bước để tránh lỗi thời gian chạy và chứng minh rằng các lỗi này đã được phòng ngừa. Ví dụ:

* Đảm bảo môi trường thực thi cung cấp đủ tài nguyên cho chương trình (như thời gian xử lý và dung lượng ngăn xếp).
* Đảm bảo các lỗi thời gian chạy như tràn số học không xảy ra. Điều này có thể được thực hiện thông qua mã kiểm tra giới hạn giá trị đầu vào, nhằm đảm bảo các phép tính không gây ra lỗi.

#### Cách hoạt động và quy trình được mong đợi MISRA C

MISRA C nhận ra rằng một **phong cách lập trình nhất quán** sẽ giúp lập trình viên dễ dàng hiểu mã nguồn do người khác viết. Tuy nhiên, vì phong cách lập trình có thể thay đổi theo từng tổ chức, MISRA C không đưa ra khuyến nghị cụ thể về phong cách lập trình. Thay vào đó, các tổ chức nên phát triển **các quy tắc phong cách lập trình nội bộ** phù hợp với quy trình phát triển của mình và áp dụng chúng một cách nhất quán.

Một số **chỉ số phần mềm** thường được các tiêu chuẩn quy trình phát triển phần mềm khuyến nghị sử dụng để xác định các khu vực mã cần xem xét kỹ lưỡng hơn hoặc cần được kiểm tra thêm. Tuy nhiên, tính chất của các chỉ số này và các ngưỡng tương ứng của chúng sẽ được xác định dựa trên ngành công nghiệp, tổ chức và yêu cầu dự án cụ thể. Do đó, tài liệu MISRA C không đưa ra hướng dẫn cụ thể nào về việc thu thập các chỉ số này, mà để các tổ chức tự quyết định dựa trên nhu cầu và đặc điểm của dự án.

### Tuân thủ

Để đảm bảo mã nguồn tuân thủ đầy đủ các hướng dẫn MISRA C, việc tạo ra một ma trận tuân thủ là rất cần thiết. Ma trận này sẽ liệt kê từng hướng dẫn của MISRA C và chỉ ra cách mà mỗi hướng dẫn được kiểm tra và xác minh. Cách kiểm tra hiệu quả nhất thường bao gồm:

* Sử dụng công cụ phân tích tĩnh: Đây là phương pháp nhanh chóng, đáng tin cậy và hiệu quả về chi phí để kiểm tra nhiều hướng dẫn MISRA C.
* Trình biên dịch và các công cụ khác: Ngoài công cụ phân tích tĩnh, có thể sử dụng trình biên dịch và các công cụ hỗ trợ khác để kiểm tra sự tuân thủ của mã.
* Kiểm tra thủ công: Đối với những hướng dẫn mà các công cụ không thể kiểm tra hoàn toàn, việc xem xét thủ công là cần thiết. Điều này đòi hỏi sự thận trọng và kỹ lưỡng từ lập trình viên hoặc người kiểm tra.
* Tuyên bố tuân thủ từ các công cụ tạo mã tự động

Một số công cụ tạo mã tự động có khả năng cung cấp tuyên bố tuân thủ kèm theo. Tuyên bố này đảm bảo rằng mã được tạo ra bởi công cụ sẽ không vi phạm các hướng dẫn MISRA C, nếu công cụ được cấu hình đúng cách. Các tuyên bố này có thể:

* Liệt kê các hướng dẫn mà mã tạo ra không vi phạm.
* Đảm bảo rằng tất cả các hướng dẫn bắt buộc và yêu cầu đều được tuân thủ.

Việc sử dụng các tuyên bố tuân thủ này có thể giảm đáng kể khối lượng công việc cần thiết để kiểm tra các hướng dẫn MISRA C theo cách thủ công hoặc bằng các phương pháp khác.

* Tài liệu tham khảo hỗ trợ việc tạo ma trận tuân thủ

Để hỗ trợ quá trình tạo ma trận tuân thủ đầy đủ, bạn có thể tham khảo Bảng 1 trong tài liệu để xem ví dụ minh họa. Ngoài ra, Phụ lục A cung cấp một bản tóm tắt các hướng dẫn MISRA C, giúp tổ chức và theo dõi quá trình kiểm tra tuân thủ dễ dàng hơn.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Guideline | Compilers | | Checking tools | | Manual review |
| ‘A’ | ‘B’ | ‘A’ | ‘B’ |
| Dir 1.1 |  |  |  |  | Procedure x |
| Dir 2.1 | no errors | no errors |  |  |  |
| … |  |  |  |  |  |
| Rule 4.1 |  |  | message 38 |  |  |
| Rule 4.2 |  |  |  | warning 97 |  |
| Rule 5.1 | warning 347 |  |  |  |  |
| … |  |  |  |  |  |

**Bảng 1:** Giới thiệu về ma trận tuân thủ

**Giải thích:**

|  |
| --- |
| Ma trận này minh họa cách kiểm tra các hướng dẫn MISRA C qua việc sử dụng các trình biên dịch, công cụ kiểm tra và xem xét thủ công. Dưới đây là các phần chính trong ma trận:   * Cột Guideline: Liệt kê các hướng dẫn của MISRA C (ví dụ: Dir 1.1, Rule 4.1). * Cột Compilers: Chỉ ra thông tin kiểm tra từ hai trình biên dịch được gọi là 'A' và 'B'. Ví dụ:   + Đối với Dir 2.1, cả hai trình biên dịch 'A' và 'B' không báo cáo lỗi.   + Đối với Rule 5.1, trình biên dịch 'A' báo cảnh báo số 347. * Cột Checking tools: Chỉ ra thông tin kiểm tra từ hai công cụ phân tích tĩnh 'A' và 'B'. Ví dụ:   + Đối với Rule 4.1, công cụ 'A' báo cáo "message 38".   + Đối với Rule 4.2, công cụ 'B' báo cáo "warning 97". * Cột Manual review: Liệt kê các thủ tục hoặc quy trình xem xét thủ công. Ví dụ, đối với Dir 1.1, "Procedure x" được thực hiện để kiểm tra tuân thủ.   Ma trận này giúp tổ chức và theo dõi việc tuân thủ các quy tắc MISRA C bằng cách sử dụng các công cụ tự động và thủ tục kiểm tra thủ công. |

Khi đã có ma trận tuân thủ, trình biên dịch và trình phân tích có thể được cấu hình và sử dụng để tạo danh sách các thông báo cần điều tra.

Các thông tin sau đây cần được ghi lại cho từng công cụ được sử dụng trong quá trình kiểm tra:

* Số phiên bản;
* Các tùy chọn được sử dụng khi gọi công cụ;
* Mọi dữ liệu cấu hình mà công cụ này sử dụng.

#### Cấu hình trình biên dịch

Khi sử dụng trình biên dịch, cấu hình của nó cần phải phù hợp với các yêu cầu của dự án. Đặc biệt:

* Lựa chọn giữa C90 và C99: Nếu trình biên dịch hỗ trợ cả hai phiên bản C90 và C99, cần chọn đúng phiên bản mà dự án sử dụng.
* Cấu hình theo mục tiêu bộ xử lý: Nếu trình biên dịch cho phép chọn mục tiêu phần cứng cụ thể (ví dụ, loại bộ vi xử lý), phải đảm bảo rằng trình biên dịch được cấu hình chính xác cho phần cứng mục tiêu của dự án.

Khi lựa chọn tùy chọn tối ưu hóa, cần xem xét cẩn thận để đạt được sự cân bằng tốt nhất giữa tốc độ thực thi và kích thước mã. Cần lưu ý rằng tối ưu hóa mạnh mẽ có thể tăng nguy cơ gặp lỗi từ trình biên dịch, do đó, cần kiểm tra kỹ lưỡng khi sử dụng các tùy chọn này.

Ngay cả khi không dùng trình biên dịch để kiểm tra tuân thủ MISRA C, nó vẫn có thể tạo ra các cảnh báo hữu ích trong quá trình biên dịch, cho thấy các lỗi tiềm ẩn. Các thông báo này cần được xem xét, và nếu trình biên dịch cho phép kiểm soát số lượng và loại cảnh báo, cần chọn mức độ phù hợp để xem xét các cảnh báo liên quan.

#### Cấu hình công cụ phân tích tatic

Khác với trình biên dịch, công cụ phân tích tĩnh thường phục vụ nhiều mục đích và không phụ thuộc vào một loại bộ xử lý cụ thể. Vì vậy, cần cấu hình công cụ này sao cho phù hợp với các quyết định triển khai của trình biên dịch. Ví dụ:

* Kích thước kiểu dữ liệu: Công cụ phân tích cần biết kích thước các kiểu dữ liệu (như int, long) mà trình biên dịch đang sử dụng.
* Hỗ trợ C90 hoặc C99: Nếu công cụ hỗ trợ cả C90 và C99, cần cấu hình để nó phù hợp với phiên bản C mà dự án đang dùng.

Nếu công cụ phân tích tĩnh hỗ trợ các mở rộng ngôn ngữ (ví dụ, mở rộng cụ thể cho trình biên dịch), nó nên được cấu hình để hỗ trợ các phần mở rộng này. Nếu không thể cấu hình được, cần có quy trình thay thế để kiểm tra liệu mã nguồn có tuân thủ đúng ngôn ngữ mở rộng hay không.

Ngoài ra, công cụ phân tích tĩnh có thể cần cấu hình để kiểm tra các nguyên tắc nhất định của MISRA C. Ví dụ, nếu dự án không sử dụng \_Bool cho dữ liệu Boolean, thì công cụ cần được biết về kiểu dữ liệu Boolean cơ bản mà dự án đang sử dụng để kiểm tra tuân thủ.

#### Điều tra thông báo

Các thông báo được tạo ra trong quá trình kiểm tra tuân thủ hoặc dịch thuật sẽ thuộc một trong ba loại sau:

1. Chẩn đoán chính xác hành vi vi phạm: Công cụ phát hiện đúng các vi phạm hướng dẫn của MISRA C.
2. Chẩn đoán khả năng vi phạm: Công cụ cho thấy có khả năng vi phạm nguyên tắc MISRA C, nhưng cần xác minh thêm.
3. Chẩn đoán sai: Cảnh báo không chính xác về vi phạm hướng dẫn MISRA C (cảnh báo giả).

Mọi thông báo nào không rõ ràng cần được điều tra để xác định tính chính xác. Đôi khi, giải pháp đơn giản nhất là chỉnh sửa mã nguồn để loại bỏ thông báo, nhưng điều này không phải lúc nào cũng có thể hoặc mong muốn. Trong trường hợp không thể sửa mã, cần lưu giữ hồ sơ điều tra để giải thích lý do tại sao cảnh báo không được xử lý hoặc không cần sửa mã.

Mục đích của việc ghi lại quá trình điều tra là để đảm bảo:

* Mọi vi phạm đã được xác minh và xem xét kỹ lưỡng.
* Có tài liệu lưu trữ để tham khảo trong các lần kiểm tra hoặc rà soát mã nguồn sau này.

### Thủ tục sai lệch

Trong một số trường hợp, bạn có thể cần sai lệch khỏi các hướng dẫn của MISRA C. Ví dụ, việc truy cập vào các cổng I/O được ánh xạ bộ nhớ tại các địa chỉ cố định không tuân thủ hoàn toàn MISRA C, vì nó liên quan đến việc chuyển đổi số nguyên thành con trỏ, như trong đoạn mã sau:

|  |
| --- |
| #define PORT (\*(volatile unsigned char \*)0x0002)  PORT = 0x10u; |

Những sai lệch như vậy cần được ghi lại và phê duyệt chính thức. Các lập trình viên không nên tự ý thực hiện sai lệch mà phải tuân theo quy trình ủy quyền chính thức. Việc lưu giữ hồ sơ chính thức giúp hỗ trợ bất kỳ luận điểm nào liên quan đến an toàn phần mềm và đảm bảo tính minh bạch trong quá trình phát triển.

Quy trình sai lệch nên được thiết lập rõ ràng ngay từ đầu trong quá trình phát triển phần mềm. MISRA C không yêu cầu một quy trình cụ thể, vì các tổ chức có thể sử dụng các phương pháp khác nhau để quản lý sai lệch, chẳng hạn như:

* Cách thức nêu lên, xem xét và phê duyệt sai lệch.
* Mức độ xem xét và bằng chứng cần thiết, có thể thay đổi tùy thuộc vào mức độ quan trọng của quy tắc bị sai lệch.

MISRA C phân loại các quy tắc theo mức độ quan trọng, điều này ảnh hưởng đến cách áp dụng sai lệch (xem Phần 6.2).

Sai lệch được chia thành hai loại:

* Sai lệch dự án: Áp dụng cho nhiều trường hợp trong dự án. Ví dụ, nếu bạn sử dụng I/O được ánh xạ bộ nhớ trong nhiều tệp trong trình điều khiển, bạn có thể cấp sai lệch dự án cho toàn bộ hệ thống đó.
* Sai lệch cụ thể: Áp dụng cho một trường hợp cụ thể trong một tệp mã duy nhất.

### Tuyên bố tuân thủ

Tuyên bố tuân thủ chỉ có thể được thực hiện cho một dự án cụ thể, không phải cho toàn bộ tổ chức. Khi đưa ra tuyên bố tuân thủ theo MISRA C cho một dự án, bạn đang khẳng định rằng:

* Ma trận tuân thủ đã được hoàn thành, mô tả cách kiểm tra tuân thủ cho từng quy tắc MISRA C.
* Tất cả mã C trong dự án tuân thủ các hướng dẫn của MISRA C, hoặc có sai lệch đã được phê duyệt.
* Hồ sơ về tất cả các sai lệch đã được phê duyệt đều được lưu giữ.
* Các hướng dẫn từ Phần 3 và Phần 4 của tài liệu này đã được thực hiện.
* Nhân sự có đủ kỹ năng và kinh nghiệm để đảm bảo tuân thủ.

**Lưu ý**: Khi bạn tuyên bố rằng mã tuân thủ MISRA C, điều này giả định rằng tất cả các công cụ kiểm tra hoặc người đánh giá đã xác định được tất cả các vi phạm. Tuy nhiên, các công cụ hoặc người đánh giá khác nhau có thể phát hiện ra các vi phạm khác, vì vậy, tuyên bố tuân thủ không phải là tuyệt đối mà phụ thuộc vào quy trình kiểm tra được sử dụng.

Một danh sách kiểm tra tóm tắt các hướng dẫn được nêu ở mục (4) có thể được tìm thấy trong Phụ lục F của tài liệu.