Среда разработки IAR Embedded Workbench®

Правила MISRA C

Справочное руководство

IAR Embedded Workbench®
MISRA C
REFERENCE MANUAL

Перевод: Андрей Шлеенков
http://andromega.narod.ru
mailto:andromega@narod.ru

АВТОРСКИЕ ПРАВА

ОТКАЗ ОТ ОТВЕТСТВЕННОСТИ

Информация в данном документе не представляет никаких обязательств со стороны IAR Systems и может быть изменена без уведомления. IAR Systems не подразумевает своей ответственности за какие-либо ошибки или пропуски в данном документе.

Корпорация IAR Systems, ее сотрудники, подрядчики или авторы данного документа ни в каком случае не несут ответственности за умышленный, неумышленный, прямой, косвенный или последовавший ущерб, повлекший повреждения, убытки, потерю выгоды, претензии, требования или затраты любого вида.

ТОРГОВЫЕ МАРКИ

IAR Embedded Workbench, IAR visualSTATE, IAR MakeApp и IAR PreQual являются зарегистрированными торговыми марками IAR Systems. C-SPY является торговой маркой IAR Systems, зарегистрированной в Европейском Союзе. IAR, IAR XLINK Linker, IAR XAR Library Builder и IAR XLIB Librarian являются торговыми марками IAR Systems.

Microsoft и Windows являются зарегистрированными торговыми марками Microsoft Corporation. Adobe и Acrobat Reader являются зарегистрированными торговыми марками Adobe Systems Incorporated.

Все остальные наименования продуктов являются торговыми марками или зарегистрированными торговыми марками их соответствующих собственников.

ВЕРСИЯ ДОКУМЕНТА

Вторая редакция: сентябрь 2004

Код продукта: EWMISRAC-2

Данное руководство описывает версию 1.0 реализации компанией IAR Systems проверки соблюдения правил *Guidelines for the Use of the C Language in Vehicle Based Software (Руководящие материалы по применению языка Си в программных продуктах для автомобильной техники)*, составленных ассоциацией Motor Industry Software Reliability Association (Ассоциации надежности программного обеспечения автомобильной промышленности).

Содержание

Предисловие

Представляем справочное руководство IAR Embedded Workbench® MISRA C. Данное руководство содержит справочную информацию о реализации компанией IAR Systems проверки соблюдения правил *Guidelines for the Use of the C Language in Vehicle Based Software (Руководящие материалы по применению языка Си в программных продуктах для автомобильной техники)*, разработанных ассоциацией Motor Industry Software Reliability Association (Ассоциация надежности программного обеспечения автомобильной промышленности).

Для кого предназначено данное руководство

Данное руководство необходимо изучать при разработке программных продуктов с соблюдением правил MISRA C. Также необходимо иметь знания в следующих областях:

- язык программирования Си;
- подмножество MISRA C языка программирования Си;
- разработка приложений для встроенных систем, критичных к безопасности;
- архитектура и набор команд используемого микроконтроллера;
- операционная система инструментального компьютера.

За дополнительной информацией о других средствах разработки, включенных в среду IAR Embedded Workbench IDE, обращайтесь к руководствам *IAR C/EC++ Compiler Reference Guide*, *IAR Assembler Reference Guide* и *IAR Linker and Library Tools Reference Guide*.

Что содержит данное руководство

Данное руководство содержит следующие разделы.

- *Общие опции* – описание общих настроек MISRA C в среде IAR Embedded Workbench®.
- *Опции компилятора* – описание настроек MISRA C компилятора в оконной среде IAR Embedded Workbench®.
- *Опции командной строки* – описание установки опций компилятора из командной строки.
- *Справочник MISRA C* – описание того, как IAR Systems интерпретирует и реализует проверку соблюдения правил, описанных в документации *Guidelines for the Use of the C Language in Vehicle Based Software*.

Дополнительная документация

Полный комплект средств разработки IAR для микроконтроллеров описан в следующих руководствах.

- Среда IAR Embedded Workbench® и отладчик IAR C-SPY™ Debugger описаны в *IAR Embedded Workbench® IDE User Guide*
- Программирование на компиляторе IAR C/C++ Compiler описано в *IAR C/EC++ Compiler Reference Guide*

- Программирование на ассемблере IAR Assembler описано в *IAR Assembler Reference Guide*
- Использование компоновщика IAR XLINK Linker™, генератора библиотек IAR XAR Library Builder™ и библиотекаря IAR XLIB Librarian™ описано в *IAR Linker and Library Tools Reference Guide*
- Использование библиотеки времени исполнения описано в *Library Reference information*, доступном во встроенной справке среды IAR Embedded Workbench IDE.

Все данные руководства поставляются в гипертекстовом формате PDF или HTML на носителе с дистрибутивом. Некоторые из них поставляются также в печатном виде.

Рекомендуемые веб-сайты:

- веб-сайт MISRA www.misra.org.uk содержит информацию и новости о правилах MISRA C.
- веб-сайт IAR www.iar.com содержит информацию о приложениях и других продуктах.

Типографские соглашения

В данном документе используются типографские соглашения, показанные в таблице 1.

Таблица 1: Типографские соглашения данного руководства

Стиль	Использование
`computer`	Текст, вводимый с клавиатуры или выводимый на экран.
`parameter`	Текст, представляющий параметры, вводимые как часть команды.
`[option]`	Опциональная часть команды.
`{a | b | c}`	Альтернативный выбор в команде.
OK, Cancel	Имена кнопок, окон, панелей, вкладок, меню.
File>Save As...	Выбор в меню.
`Reference`	Ссылка на текст в данном или другом руководстве.
Замечание.	Замечание.
`#define TEST`	Исходный программный код.
	Инструкции оконного интерфейса IAR Embedded Workbench.
	Инструкции интерфейса командной строки IAR Embedded Workbench.

Общие опции

Данная глава описывает общие опции MISRA C среды IAR Embedded Workbench®.

Способ установки опций описан в руководстве *IAR Embedded Workbench® IDE User Guide*.

MISRA C

Опции на вкладке **MISRA C** управляют тем, как среда IAR Embedded Workbench проверяет исходный код на отклонения от правил MISRA C. Установки используются как компилятором, так и компоновщиком.

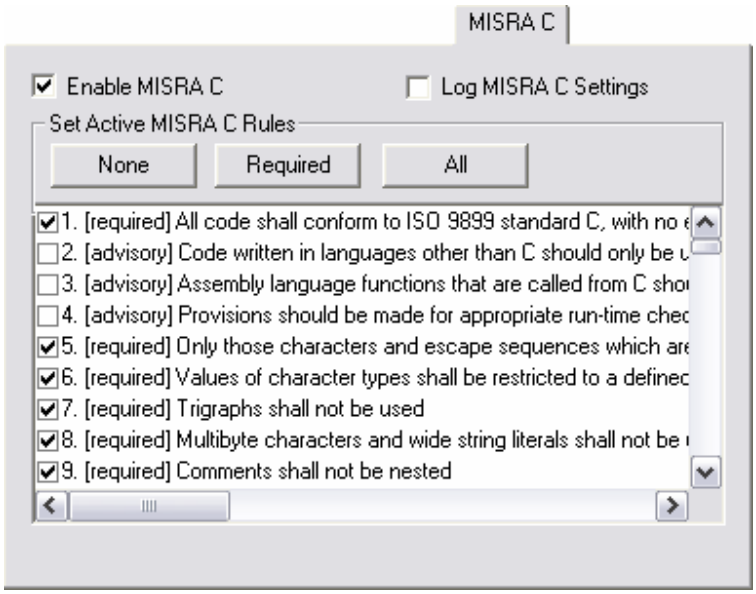

Рисунок 1: Общие опции MISRA C

Enable MISRA C (Разрешение MISRA C)

Выбор данной опции разрешает проверку исходного кода на отклонения от правил MISRA C при компиляции и компоновке. Будет проверяться соблюдение только выбранных в списке правил.

Log MISRA C Settings (Отчет об установках MISRA C)

Выбор данной опции генерирует отчет MISRA C при компиляции и компоновке. Отчет содержит список разрешенных, но не обязательно проверяемых, а также реально проверяемых правил.

Set active MISRA C Rules (Установка активных правил MISRA C)

При компиляции и компоновке будут проверяться только правила выбранные в прокручиваемом списке. Щелчок по кнопке **None**, **Required** или **All** выбирает или отменяет некоторую группу правил. Кнопка **Required** выбирает все 93 *обязательные* правила и отменяет все *рекомендуемые* правила в соответствии с их категоризацией в *Guidelines for the Use of the C Language in Vehicle Based Software*.

Опции компилятора

Данная глава описывает опции MISRA C компилятора среды IAR Embedded Workbench®.

Способ установки опций описан в руководстве *IAR Embedded Workbench® IDE User Guide*.

MISRA C

Если необходимо, чтобы компилятор проверял соблюдение другого набора правил, можно использовать установки в категории **C/C++ Compiler**. Данные опции заменяют набор общих опций, выбранных на вкладке **MISRA C** в категории **General Options**.

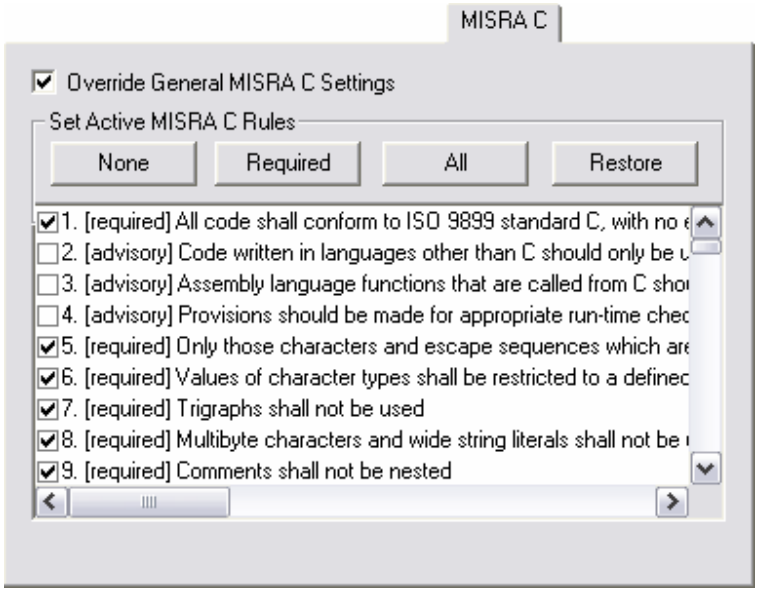

Рисунок 2: Опции MISRA C компилятора

Override General MISRA C Settings (Замена общих установок MISRA C)

Выбор данной опции заставляет компилятор проверять соблюдение другого набора правил, отличного от набора, установленного в категории **General Options**.

Set Active MISRA C Rules (Установка активных правил MISRA C)

При компиляции будут проверяться только правила выбранные в прокручиваемом списке. Щелчок по кнопке **None**, **Required** или **All** выбирает или отменяет некоторую группу правил. Кнопка **Required** выбирает все 93 *обязательные* правила и отменяет все *рекомендуемые* правила в соответствии с их категоризацией в *Guidelines for the Use of the C Language in Vehicle Based Software*. Кнопка **Restore** восстанавливает установки MISRA C, используемые в категории **General Options**.

Опции командной строки

Данная глава объясняет установку опций компилятора из командной строки и содержит детальную информацию о каждой опции.

Перечень опций

В таблице 2 перечислены опции командной строки компилятора.

Таблица 2: Перечень опций компилятора

Опции командной строки	Описание
`--misrac`	Разрешение сообщений об ошибках, специфичных для MISRA C.
`--misrac_verbose`	Разрешение подробного отчета о проверке правил MISRA C.

Описание опций

Данный раздел содержит детальную информацию о каждой опции компилятора.

--misrac

```
--misrac[={tag1,tag2-tag3,...|all|required}]
```

Использование данных опций разрешает проверку на отклонение от правил, описанных в документации MISRA *Guidelines for the Use of the C Language in Vehicle Based Software*. Применение аргументов данной опции ограничивает проверку соблюдения правил MISRA C определенным подмножеством правил. Допустимые аргументы перечислены в таблице 3.

Таблица 3: Разрешение правил MISRA C

Опции командной строки	Описание
`--misrac`	Разрешение проверки на все правила MISRA C.
`--misrac=n`	Разрешение проверки на правило MISRA C с номером *n*.
`--misrac=m,n`	Разрешение проверки на правила MISRA C с номерами *m* и *n*.
`--misrac=k-n`	Разрешение проверки на все правила MISRA C с номерами от *k* до *n*.
`--misrac=k,m,r-t`	Разрешение проверки на все правила MISRA C с номерами *k*, *m* и от *r* до *t*.
`--misrac=all`	Разрешение проверки на все правила MISRA C.
`--misrac=required`	Разрешение проверки на все правила MISRA C, характеризуемые, как обязательные.

Если компилятор не способен проверить некоторое правило, указание такого правила в опции не будет иметь никакого эффекта. Например, правило MISRA C с номером 15 является вопросом документации, и, следовательно, не может быть проверено компилятором. Как следствие, определение опции `--misrac=15` не даст никакого эффекта.

Замечание. Правила MISRA C поддерживаются не всеми продуктами IAR Systems. Если проверка правил MISRA C не поддерживается каким-либо конкретным компилятором, использование данной опции с таким компилятором вызовет сообщение об ошибке.

Для установки эквивалентных опций в оконной среде IAR Embedded Workbench необходимо вызвать меню *Project>Options>General Options>MISRA C* или *Project>Options>C/C++ Compiler>MISRA C*.

`--misrac_verbose`

`--misrac_verbose`

Данная опция используется для генерации отчета MISRA C при компиляции и компоновке. Отчет содержит список разрешенных, но не обязательно проверяемых, а также реально проверенных правил.

Если данная опция разрешена, компилятор выводит текст с указанием как разрешенных, так и реально проверенных правил MISRA C.

Для установки эквивалентных опций в оконной среде IAR Embedded Workbench необходимо вызвать меню *Project>Options>General Options>MISRA C*.

Справочник MISRA C

Данная глава описывает, как IAR Systems интерпретирует и реализует правила, данные в документах *Guidelines for the Use of the C Language in Vehicle Based Software*, для повышения безопасности применения языка программирования Си, соответствующего стандарту ISO [ISO/IEC 9899:1990].

Реализация IAR Systems основана на версии 1 правил MISRA C, датированной апрелем 1998 года.

Почему MISRA C?

Язык Си является наиболее популярным языком программирования высокого уровня для встроенных систем. Однако при разработке кода для систем, требовательных к безопасности, язык Си обнаруживает много недостатков. Существует несколько неопределенных или реализационно-зависимых аспектов языка Си, делающих его неподходящим для разработки систем, требовательных к безопасности.

Ассоциация надежности программного обеспечения автомобильной промышленности в документах *Guidelines for the Use of the C Language in Vehicle Based Software* описывает подмножество языка Си, подходящего для разработки систем, требовательных к безопасности.

ПОДТВЕРЖДЕНИЕ СООТВЕТСТВИЯ

Для подтверждения соответствия программного продукта правилам MISRA C необходимо выполнить следующее:

- должна быть создана и заполнена таблица соответствия, показывающая, как выполняется каждое правило;
- весь код Си должен соответствовать правилам MISRA C, либо все отклонения от правил должны быть задокументированы;
- должен быть создан список всех случаев несоответствия правилам и в каждом случае отклонение от правил должно быть соответствующим образом задокументировано;
- должны быть приняты соответствующие меры в вопросах обучения, применения стилей, выбора и верификации компилятора, верификации контрольных инструментов, метрик и полноты тестирования, как описано в части 5.2 документации *Guidelines for the Use of the C Language in Vehicle Based Software*.

Реализация и интерпретация правил MISRA C

Реализация правил MISRA C не влияет на генерацию кода и не оказывает значительного влияния на производительность среды IAR Embedded Workbench. Библиотеки времени исполнения IAR CLIB и IAR DLIB не подвергаются изменениям.

Замечание. Правила применяются к исходному коду приложения, написанному программистом, и не применяются к коду, сгенерированному компилятором. Например, правило 101 означает, что программист не должен явно использовать арифметику указателей, но арифметика, сгенерированная компилятором, например, из выражения `a[3]`, не считается нарушением правил.

ПРОВЕРКА СООТВЕТСТВИЯ ПРАВИЛАМ

Компилятор и компоновщик генерируют только сообщения об ошибках и не предохраняют от нарушения проверяемых правил. Разрешение или запрет проверки отдельных правил возможны как для проекта в целом, так и для отдельного файла. Отчет о проверке генерируется при компиляции и компоновке и выводится в окне **Build** среды IAR Embedded Workbench. Отчет может быть сохранен в файле в соответствии с настройками среды *IAR Embedded Workbench User Guide*.

Для каждого случая отклонения от обязательного или рекомендуемого правила, не запрещенного в настройках, выводится сообщение об ошибке. Каждое сообщение содержит ссылку на место нарушения правил MISRA C. Формат ссылки соответствует следующему примеру:

```
Error[Pm088]: pointer arithmetics should not be used
(MISRA C rule 101)
```

Замечание. Нумерация сообщений не имеет связи с нумерацией правил.

Для каждого файла, для которого разрешена проверка на правила MISRA C, возможно создание полного отчета, содержащего следующие разделы:

- список всех разрешенных для проверки правил MISRA C;
- список всех реально проверенных правил MISRA C.

Ручная проверка

Существует группа правил, требующих ручной проверки. Например, правила, требующие знаний особых намерений программиста, или правил, невозможных для статической проверки и требующих дополнительных действий.

Замечание. Фактически, правило 116 не может быть категорическим, потому что стандартные заголовочные файлы не проверяются на соответствие.

Документирование отклонений

Отклонение от правил MISRA C является случаем, когда исходный код приложения нарушает данные правила. Если отклонение от правил документируется, то допускается запрет вывода сообщений о нарушении какого-либо отдельного правила.

Замечание. Отклонение от правил допускается в случае, если причины отклонений четко документированы. Так как нарушения правил по соглашению MISRA C в общем случае отслеживаются, сообщения об ошибках могут быть явно запрещены директивой `#pragma diag`.

Каждое правило проверяется отдельно. Не делается никаких предположений относительно связи с какими либо другими правилами, запрещающими проверку каких-либо других частей кода.

Разрешение правил MISRA C

В среде IAR Embedded Workbench® IDE разрешение проверки на правила MISRA C в меню *Project>Options* выбирается опциями на вкладке **MISRA C** в категории **General Options**.

В командной строке необходимо использовать опцию компилятора `--misrac` для разрешения проверки на правила MISRA C.

Перечень правил

В таблице 4 перечислены все правила MISRA C. Обозначение (O) означает обязательное требование, обозначение (P) означает рекомендуемое требование.

Таблица 4: Перечень правил MISRA C

№	Правило	Тип	Вид
1	Исходный код должен соответствовать стандарту языка Си ISO 9899 и не должен использовать расширения языка.	Среда	(O)
2	Код на языке, отличном от Си, может быть использован только при наличии определенного стандарта интерфейса для объектного кода, которому следуют оба языка.	Среда	(P)
3	Функции на языке ассемблера, вызываемые из функций Си, должны быть написаны, как функции Си, содержащие только встроенный ассемблер, и встроенный ассемблер не должен встраиваться в обычный код.	Среда	(P)
4	В исходном коде должны быть предусмотрены средства для адекватного контроля времени исполнения.	Среда	(P)
5	В исходном коде могут применяться только символы и эскейп-последовательности, определенные в ISO-стандарте Си.	Наборы символов	(O)
6	Значения символьных типов должны быть ограничены определенным и документированным подмножеством стандарта ISO 10646-1.	Наборы символов	(O)
7	Трехзнаковые последовательности не должны применяться.	Наборы символов	(O)
8	Многобайтные символы и широкие строковые литералы не должны применяться.	Наборы символов	(O)
9	Вложенные комментарии не должны применяться.	Комментарии	(O)
10	Фрагменты кода не должны быть «закомментированы».	Комментарии	(P)
11	Идентификаторы (внутренние и внешние) различаются только по 31 значащему символу. Также компилятор и компоновщик должны быть настроены для поддержки во внешних идентификаторах 31 значащего символа и чувствительности к регистру символов.	Идентификаторы	(O)
12	В одном пространстве имен не должно быть идентификаторов, совпадающих с идентификаторами из другого пространства имен.	Идентификаторы	(P)

№	Правило	Тип	Вид
13	Базовые типы `char`, `int`, `short`, `long`, `float` и `double` не должны использоваться. Вместо них для специфического компилятора директивами `typedef` должны быть определены и использованы имена типов эквивалентной длины.	Типы	(P)
14	Тип `char` всегда должен объявляться как `unsigned char` или `signed char`.	Типы	(O)
15	Реализация плавающих типов должна соответствовать определенному стандарту формата с плавающей точкой.	Типы	(P)
16	Внутреннее битовое представление плавающих чисел никогда не должно использоваться программистом.	Типы	(O)
17	Имена типов, определенные директивой `typedef` не должны использоваться в определениях повторно.	Типы	(O)
18	Числовые константы должны иметь суффикс, обозначающий тип, если такой суффикс существует в стандарте.	Константы	(P)
19	Ненулевые восьмеричные константы не должны использоваться.	Константы	(O)
20	Любой идентификатор объекта или функции должен быть объявлен перед использованием.	Объявления и определения	(O)
21	Идентификаторы во внутренней области видимости не должны использовать имена, совпадающие с именами во внешней области видимости.	Объявления и определения	(O)
22	Объявление объекта должно быть в области видимости функции, если не требуется внешняя видимость.	Объявления и определения	(P)
23	Все объявления в области видимости файла должны быть по возможности статическими.	Объявления и определения	(P)
24	Идентификаторы не должны иметь одновременно и внешний и внутренний тип связи в одной единице трансляции.	Объявления и определения	(O)
25	Идентификатор с внешним типом связи должен иметь только одно внешнее определение.	Объявления и определения	(O)
26	Если объекты или функции объявляются более одного раза, они должны иметь совместимые объявления.	Объявления и определения	(O)
27	Внешние объекты не должны объявляться более чем в одном файле.	Объявления и определения	(P)
28	Класс хранения `register` не должен использоваться.	Объявления и определения	(P)
29	Использование тега должно быть согласовано с его объявлением.	Объявления и определения	(O)
30	Все автоматические переменные должны быть инициализированы перед использованием.	Инициализация	(O)
31	Ненулевой инициализатор массивов и структур должен быть заключен в скобки, и содержать значения для всех членов.	Инициализация	(O)

№	Правило	Тип	Вид
32	Список перечислимых констант не должен содержать явных инициализаторов для всех констант кроме первой, если не все константы инициализируются явно.	Инициализация	(O)
33	Правый операнд оператора `&&` или `||` не должен содержать побочного эффекта.	Операторы	(O)
34	Операнды логических операторов `&&` или `||` должны быть первичными выражениями.	Операторы	(O)
35	Операторы присваивания не должны использоваться в выражениях, возвращающих булево значение.	Операторы	(O)
36	Логические операторы не должны использоваться совместно с поразрядными операторами.	Операторы	(P)
37	Поразрядные операции не должны выполняться над знаковыми целыми типами.	Операторы	(O)
38	Правый операнд оператора сдвига должен быть в диапазоне от нуля до числа, на единицу меньше, чем ширина левого операнда в битах.	Операторы	(O)
39	Оператор унарный минус не должен применяться к беззнаковому выражению.	Операторы	(O)
40	Оператор `sizeof` не должен применяться к выражениям, имеющим побочный эффект.	Операторы	(P)
41	Реализация целочисленного деления в применяемом компиляторе должна быть определена, документирована и учитываться в расчетах.	Операторы	(P)
42	Оператор запятая должен использоваться только в управляющем выражении цикла `for`.	Операторы	(O)
43	Неявные преобразования, способные приводить к потере информации не должны использоваться.	Преобразования	(O)
44	Избыточные явные приведения типов не должны использоваться.	Преобразования	(P)
45	Приведение между указателем и любым другим типом не должно использоваться.	Преобразования	(O)
46	Значение выражения должно быть одним и тем же при любом порядке вычислений, допускаемом компилятором.	Выражения	(O)
47	В выражениях не должны использоваться зависимости от приоритета операторов Си.	Выражения	(P)
48	Арифметика смешанной точности должна использовать явное приведение типов для получения желаемого результата.	Выражения	(P)
49	Проверка значения на нуль должна быть явной, если операнд не является точно булевым.	Выражения	(P)
50	Плавающие переменные не должны проверяться на точное равенство или неравенство.	Выражения	(O)
51	Вычисление беззнакового целого константного выражения не должно приводить к появлению переноса.	Выражения	(P)
52	Код не должен содержать недостижимых фрагментов.	Управление	(O)

№	Правило	Тип	Вид
53	Все не нуль-операторы должны иметь побочный эффект.	Управление	(O)
54	Нуль-оператор должен располагаться на отдельной строке и не должен иметь на той же строке другого текста.	Управление	(O)
55	Метки допустимы только в операторах `switch`.	Управление	(P)
56	Оператор `goto` не должен использоваться.	Управление	(O)
57	Оператор `continue` не должен использоваться.	Управление	(O)
58	Оператор `break` должен использоваться только для завершения выбора в операторе `switch`.	Управление	(O)
59	Операторы, составляющие тело оператора `if`, `else if`, `else`, `while`, `do ... while` или `for` всегда должны быть заключены в фигурные скобки.	Управление	(O)
60	Все конструкции `if`, `else if` должны содержать заключительный элемент `else`.	Управление	(P)
61	Каждый непустой элемент `case` в операторе `switch` должен быть завершен оператором `break`.	Управление	(O)
62	Все операторы `switch` должны содержать заключительный пункт `default`.	Управление	(O)
63	Выражение `switch` не должно представлять булевых значений.	Управление	(P)
64	Каждый оператор `switch` должен иметь хотя бы один элемент `case`.	Управление	(O)
65	Плавающие переменные не должны использоваться как счетчики цикла.	Управление	(O)
66	В операторе `for` должны содержаться только выражения, относящиеся к управлению циклом.	Управление	(P)
67	Переменные, используемые в цикле `for` для счета циклов не должны модифицироваться в теле цикла.	Управление	(P)
68	Функции всегда должны объявляться на уровне области видимости файла.	Функции	(O)
69	Функции с переменным числом аргументов не должны использоваться.	Функции	(O)
70	Функции не должны вызывать сами себя прямо или косвенно.	Функции	(O)
71	Функции всегда должны иметь прототип, видимый как при определении функции, так и при ее вызове.	Функции	(O)
72	Типы возвращаемого значения и каждого параметра функции в объявлении и в определении должны совпадать.	Функции	(O)
73	Прототип функции должен либо иметь идентификаторы для всех параметров, либо не иметь их вообще.	Функции	(O)
74	Если для каких либо параметров заданы идентификаторы, то они должны быть идентичны как в объявлении, так и в определении.	Функции	(O)
75	Каждая функция должна иметь явный возвращаемый тип.	Функции	(O)

№	Правило	Тип	Вид
76	Функции без параметров должны быть объявлены с параметром типа `void`.	Функции	(O)
77	Параметр без квалификатора, передаваемый функции должен быть совместим с ожидаемым типом без квалификатора, определенным в прототипе функции.	Функции	(O)
78	Число параметров, передаваемых функции, должно соответствовать прототипу функции.	Функции	(O)
79	Функции типа `void` не должны возвращать параметров.	Функции	(O)
80	Выражения типа `void` не должны передаваться как параметры функции.	Функции	(O)
81	Параметры функции, передаваемые по ссылке должны иметь квалификатор `const`, если подразумевается, что функция не должна модифицировать параметры.	Функции	(P)
82	Функция должна иметь только одну точку выхода.	Функции	(P)
83	Правило для функций, возвращающих значение типа, отличного от типа `void`.	Функции	(O)
84	В функциях, возвращающих значение типа `void`, оператор `return` не должен иметь выражения.	Функции	(O)
85	Вызовы функций без параметров должны иметь пустые скобки.	Функции	(P)
86	Если функция возвращает информацию об ошибке, то эта информация должна быть проанализирована.	Функции	(P)
87	Директивам `#include` в исходном файле могут предшествовать только другие директивы препроцессора или комментарии.	Директивы препроцессора	(O)
88	В именах заголовочных файлов директивы `#include` не должны присутствовать нестандартные символы.	Директивы препроцессора	(O)
89	После директивы `#include` должна следовать последовательность `<filename>` или `"filename"`.	Директивы препроцессора	(O)
90	Макросы Си должны использоваться только для символьных констант, функция-подобных макросов, квалификаторов типа и спецификаторов класса хранения.	Директивы препроцессора	(O)
91	Макросы не должны определяться (`#define`) или уничтожаться (`#undef`) внутри блока.	Директивы препроцессора	(O)
92	Директива `#undef` не должна использоваться.	Директивы препроцессора	(P)
93	Использование функций должно пользоваться предпочтением перед использованием функция-подобных макросов.	Директивы препроцессора	(P)
94	Функция-подобный макрос должен вызываться со всеми своими аргументами.	Директивы препроцессора	(O)
95	Аргументы функция-подобных макросов не должны содержать лексем, выглядящих как директивы препроцессора.	Директивы препроцессора	(O)

№	Правило	Тип	Вид
96	В определении функция-подобного макроса все определение и каждый экземпляр параметра должны быть заключены в скобки.	Директивы препроцессора	(O)
97	Идентификаторы в директивах препроцессора должны быть определены перед использованием.	Директивы препроцессора	(P)
98	В одном макроопределении должно быть не более одного включения оператора препроцессора # или ##.	Директивы препроцессора	(O)
99	Любое использование директивы `#pragma` должно быть документировано и объяснено.	Директивы препроцессора	(O)
100	Оператор препроцессора `defined` должен использоваться только в одной из двух стандартных форм.	Директивы препроцессора	(O)
101	Арифметика указателей не должна использоваться.	Указатели и массивы	(P)
102	Допускается использование не более двух уровней косвенной адресации при помощи указателей.	Указатели и массивы	(P)
103	К типам указателей операторы отношения могут применяться только если оба операнда имеют один и тот же тип и указывают на один и тот же массив, структуру или объединение.	Указатели и массивы	(O)
104	Допускается использование только константных указателей на функции.	Указатели и массивы	(O)
105	Все функции, вызываемые одним указателем на функцию должны иметь одинаковое число и типы параметров, и тип возвращаемого значения.	Указатели и массивы	(O)
106	Адрес объекта с автоматическим классом хранения не должен присваиваться указателю, который может сохраняться после того, как объект перестает существовать.	Указатели и массивы	(O)
107	Ссылка по нуль-указателю не допускается.	Указатели и массивы	(O)
108	В описании типа структуры или объединения все члены должны быть специфицированы полностью.	Структуры и объединения	(O)
109	Перекрытие памяти не должно использоваться.	Структуры и объединения	(O)
110	Объединения не должны использоваться для доступа к фрагментам более крупных типов данных.	Структуры и объединения	(O)
111	Тип битовых полей должен определяться только как `unsigned int` или `signed int`.	Структуры и объединения	(O)
112	Битовые поля типа `signed int` должны иметь длину как минимум 2 бита.	Структуры и объединения	(O)
113	Все члены структуры или объединения должны иметь имена и получать доступ только по именам.	Структуры и объединения	(O)
114	Зарезервированные слова и имена функций стандартной библиотеки не могут быть переопределены или уничтожены.	Стандартные библиотеки	(O)
115	Имена функций стандартной библиотеки не должны использоваться повторно.	Стандартные библиотеки	(O)

№	Правило	Тип	Вид
116	Все библиотеки, используемые для получения результирующего кода, должны быть написаны в соответствии с положениями данного документа и соответственно аттестованы.	Стандартные библиотеки	(O)
117	Действительность значений, передаваемых библиотечным функциям, должна подвергаться проверке.	Стандартные библиотеки	(O)
118	Динамическое выделение памяти в куче не должно использоваться.	Стандартные библиотеки	(O)
119	Индикатор ошибки `errno` не должен использоваться.	Стандартные библиотеки	(O)
120	Макрос `offsetof` из библиотеки <`stddef.h`> не должен использоваться.	Стандартные библиотеки	(O)
121	Заголовочный файл <`locale.h`> и функция `setlocale` не должны использоваться.	Стандартные библиотеки	(O)
122	Макрос `setjmp` и функция `longjmp` не должны использоваться.	Стандартные библиотеки	(O)
123	Средства поддержки сигнала <`signal.h`> не должны использоваться.	Стандартные библиотеки	(O)
124	Библиотека ввода-вывода <`stdio.h`> не должна применяться в результирующем коде.	Стандартные библиотеки	(O)
125	Библиотечные функции `atof`, `atoi` и `atol` из библиотеки <`stdlib.h`> не должны использоваться.	Стандартные библиотеки	(O)
126	Библиотечные функции `abort`, `exit`, `getenv` и `system` из библиотеки <`stdlib.h`> не должны использоваться.	Стандартные библиотеки	(O)
127	Функции поддержки времени из библиотеки <time.h> не должны использоваться.	Стандартные библиотеки	(O)

Среда

Правила данного раздела относятся к языковой среде.

Правило 1 (O)

Исходный код должен соответствовать стандарту языка Си ISO 9899 и не должен использовать расширения языка.

Как проверяется правило

Компилятор выдает ошибку, если компиляция производится в любом из следующих режимов:

- компиляция с расширениями IAR;
- компиляция кода C++.

Замечание. Компилятор не выдает данную ошибку, если расширения IAR применяются при помощи директивы `#pragma`.

Пример нарушения правил

```
int __far my_far_variable;
int port @ 0xbeef;
```

Пример корректного кода

```
#pragma location=0xbeef
int port;
```

Правило 2 (R)

Код на языке, отличном от Си, может быть использован только при наличии определенного стандарта интерфейса для объектного кода, которому следуют оба языка.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила. Это правило требует ручной проверки.

Правило 3 (R)

Функции на языке ассемблера, вызываемые из функций Си, должны быть написаны, как функции Си, содержащие только встроенный ассемблер; встроенный ассемблер не должен использоваться вне тела функций.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 4 (R)

В исходном коде должны быть предусмотрены средства для адекватного контроля времени исполнения.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Наборы символов

Правила данного раздела относятся к использованию наборов символов.

Правило 5 (O)

В исходном коде могут применяться только символы и эскейп-последовательности, определенные в ISO-стандарте Си.

Как проверяется правило

Компилятор выдает ошибку, если внутри строкового или символьного литерала встречаются любые из следующих символов:

- символы ASCII-кода вне диапазонов 32–35, 37–63, 65–95 и 97–126;
- эскейп-последовательности, отличающиеся от `\a`, `\b`, `\f`, `\n`, `\r`, `\t`, `\v` или `\`*`octal`* (восьмеричная константа).

Замечание. Символы `$` (dollar), `@` (at), и `` ` `` (backquote) не входят в набор символов исходного кода.

Пример нарушения правил

```
"Just my $0.02"
"Just my £0.02"
```

Пример корректного кода

```
"Hello world!\n"
'\n'
```

Замечание. Это правило служит для ограничения неопределенного и реализационно-зависимого поведения. Реализационно-зависимое поведение имеет место только тогда, когда символы преобразуются для внутреннего представления, что применимо к символьным константам и строковым литералам. По этой причине реализация IAR Systems ограничивает использование символов только в строковых и символьных литералах; символы внутри комментариев не имеют ограничений.

Правило 6 (O)

Значения символьных типов должны быть ограничены определенным и документированным подмножеством стандарта ISO 10646-1.

Как проверяется правило

Ограничения реализованы согласно информации в разделе об использовании символов в главе *Implementation-defined behavior* руководства *IAR C/EC++ Compiler Reference Guide*.

Правило 7 (O)

Трехзнаковые последовательности не должны применяться.

Как проверяется правило

Компилятор выдает ошибку при использовании трехзнаковых последовательностей.

Пример нарушения правил

```
SI_16 a ??( 3 ??);
STRING sic = "??(sic??)";
```

Пример корректного кода

```
STRING str = "What???";
```

Правило 8 (O)

Многобайтные символы и широкие строковые литералы не должны применяться.

Как проверяется правило

Компилятор выдает ошибку в случаях, если:

- в символьном или строковом литерале, в комментарии или в имени заголовочного файла присутствует любой многобайтный символ;
- вызывается любая из функций: `mblen`, `mbtowc`, `wctomb`, `mbstowcs` или `wcstombs` (объявленных в файле `stdlib.h`);
- используется широкий строковый литерал.

Замечание. При использовании функций `mblen`, `mbtowc`, `wctomb`, `mbstowcs` или `wcstombs` компилятор выдает ошибку только при включении правильного заголовочного файла. Использование любых других функций с этими же именами ошибку не генерирует.

Комментарии

Правила данного раздела относятся к комментированию кода.

Правило 9 (O)

Вложенные комментарии не должны применяться.

Как проверяется правило

Компилятор выдает ошибку, если в комментарии используется последовательность `/*`.

Правило 10 (R)

Фрагменты кода не должны быть «закомментированы».

Как проверяется правило

Компилятор выдает ошибку, если комментарий завершается знаком `;`, `{` или `}`.

Замечание. Это правило проверяется таким способом, что примеры кода внутри комментариев допускаются и не генерируют ошибку.

Идентификаторы

Правила данного раздела относятся к идентификаторам.

Правило 11 (O)

Идентификаторы (внутренние и внешние) различаются только по 31 значащему символу. Также компилятор и компоновщик должны быть настроены для поддержки во внешних идентификаторах 31 значащего символа и чувствительности к регистру символов.

Как проверяется правило

Компоновщик выдает ошибку, если какие-либо идентификаторы имеют одинаковые первые 31 символ.

Компилятор выдает ошибку при объявлении или определении идентификатора, если он совпадает с ранее объявленным или определенным идентификатором по первым 31 символам.

Правило 12 (O)

В одном пространстве имен не должно быть идентификаторов, совпадающих с идентификаторами из другого пространства имен.

Как проверяется правило

Компилятор выдает ошибку, если в объявлении или определении написание идентификатора совпадает с написанием другого идентификатора в другом пространстве имен. Одноименные поля разных структур не генерируют ошибку.

Пример нарушения правила

```
struct an_ident { int an_ident; } an_ident;
```

Пример корректного кода

```
struct a_struct { int a_field; } a_variable;
```

Типы

Правила данного раздела относятся к объявлению типов данных.

Правило 13 (P)

Базовые типы `char`, `int`, `short`, `long`, `float` и `double` не должны использоваться. Вместо них для определенного компилятора директивами `typedef` должны быть определены и использованы имена типов эквивалентной длины.

Как проверяется правило

Компилятор выдает ошибку, если в объявлении или определении используется любой из базовых типов вместо типа, определенного директивой `typedef`.

Пример нарушения правила

```
int x;
```

Пример корректного кода

```
typedef int SI_16
SI_16 x;
```

Правило 14 (O)

Тип `char` всегда должен объявляться как `unsigned char` или `signed char`.

Как проверяется правило

Компилятор выдает ошибку, если базовый тип `char` объявляется без явного указания знаковости `signed` или беззнаковости `unsigned`.

Правило 15 (P)

Реализация плавающих типов должна соответствовать определенному стандарту формата с плавающей точкой.

Как проверяется правило

Стандарт плавающих чисел, используемый компилятором IAR C/C++ Compiler, описан в руководстве *IAR C/EC++ Compiler Reference Guide*.

Правило 16 (O)

Внутреннее битовое представление плавающих чисел никогда не должно использоваться программистом.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила. Это правило требует ручной проверки.

Правило 17 (O)

Имена типов, определенные директивой `typedef` не должны использоваться в определениях повторно.

Как проверяется правило

Компилятор выдает ошибку в следующих случаях:

- использование в объявлении или определении любого имени, ранее использованного в директиве `typedef`;
- использование в директиве `typedef` любого имени, ранее использованного в объявлении или определении.

Пример корректного кода

```
/* Для этой широко используемой идиомы ошибки нет */
typedef struct a_struct {
...
} a_struct;
```

Константы

Правила данного раздела относятся к использованию констант.

Правило 18 (Р)

Числовые константы должны иметь суффикс, обозначающий тип, если такой суффикс существует в стандарте.

Как проверяется правило

Компилятор выдает ошибку для любой целой константы, чей тип не соответствует типу значения по умолчанию в любой реализации, соответствующей стандарту.

Пример нарушения правила

```
100000
```

Пример корректного кода

```
30000
100000L
100000UL
```

Правило 19 (О)

Ненулевые восьмеричные константы не должны использоваться.

Как проверяется правило

Компилятор выдает ошибку, если ненулевая константа начинается с символа 0 (нуль).

Объявления и определения

Правила данного раздела относятся к объявлениям и определениям.

Правило 20 (О)

Любой идентификатор объекта или функции должен быть объявлен перед использованием.

Как проверяется правило

Компилятор выдает ошибку для любого неявного объявления функции.

Замечание. Данное правило, однако допускает функции в стиле Кернигана и Ритчи, если поведение функций определяется однозначно.

Правило 21 (O)

Идентификаторы во внутренней области видимости не должны использовать имена, совпадающие с именами во внешней области видимости.

Как проверяется правило

Компилятор выдает ошибку, если объявление или определение скрывает имя другого идентификатора.

Правило 22 (P)

Объявление объекта должно быть в области видимости функции, если не требуется внешняя видимость.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 23 (P)

Все объявления в области видимости файла должны быть по возможности статическими.

Как проверяется правило

Компоновщик выдает ошибку, если символ, используемый в модуле, объявлен глобальным, но не имеет ссылок из других модулей.

Правило 24 (O)

Идентификаторы не должны иметь одновременно и внешний и внутренний тип связи в одной единице трансляции.

Как проверяется правило

Компилятор выдает ошибку, если символ объявлен любым из следующих способов:

- с *внешним* типом связи, если уже имеется идентичный символ в текущей области видимости с *внутренним* типом связи;
- с *внутренним* типом связи, если уже имеется идентичный символ в текущей области видимости с *внешним* типом связи.

Правило 25 (O)

Идентификатор с внешним типом связи должен иметь только одно внешнее определение.

Как проверяется правило

Компоновщик всегда проверяет данное правило, даже если правила MISRA C запрещены.

Замечание. Множественное определение глобального символа компоновщик считает ошибкой.

Использование символа с недоступным определением также считается ошибкой компоновки.

Правило 26 (O)

Если объекты или функции объявляются более одного раза, они должны иметь совместимые объявления.

Как проверяется правило

Компоновщик всегда проверяет данное правило и выводит предупреждение, даже если правила MISRA C запрещены. Если правила MISRA C разрешены, то выводится сообщение об ошибке.

Компоновщик проверяет, чтобы объявления и определения имели совместимые типы за следующими исключениями:

- типы `bool` и `wchar_t` совместимы со всеми типами `int` того же размера;
- для параметров функций в стиле Кернигана и Ритчи –
  - типы `int` и `unsigned int` считаются совместимыми;
  - типы `long` и `unsigned long` считаются совместимыми;
- незавершенные типы считаются совместимыми, если они имеют то же самое имя;
- завершенные типы считаются совместимыми, если они имеют поля совместимых типов.

Правило 27 (Р)

Внешние объекты не должны объявляться более чем в одном файле.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 28 (Р)

Класс хранения `register` не должен использоваться.

Как проверяется правило

Компилятор выдает ошибку при использовании ключевого слова `register`.

Правило 29 (О)

Использование тега должно быть согласовано с его объявлением.

Как проверяется правило

Компилятор выдает ошибку, если перечислимая константа присваивается переменной неверного перечислимого типа.

Компоновщик выдает ошибку, если один и тот же тег структуры или перечисления используется в нескольких различных единицах трансляции.

Инициализация

Правила данного раздела относятся к инициализации переменных.

Правило 30 (О)

Все автоматические переменные должны быть инициализированы перед использованием.

Как проверяется правило

Реализована частичная поддержка проверки данного правила.

Компилятор выдает ошибку, если переменная используется без предварительной инициализации, за исключением случая, когда алгоритм содержит присваивание значения этой переменной.

Правило 31 (O)

Ненулевой инициализатор массивов и структур должен быть заключен в скобки, и содержать значения для всех членов.

Как проверяется правило

Компилятор выдает ошибку для любой инициализации, не имеющей корректной структуры скобок и числа элементов. Компилятор не генерирует ошибку, если используется нулевой инициализатор `{ 0 }`.

Пример нарушения правил

```
struct { int a, b; } a_struct = { 1 };
struct { int a[3]; } a_struct = { 1, 2 };
```

Пример корректного кода

```
struct { int a, b; } a_struct = { 1, 2 };
struct { int a, b; } a_struct = { 0 };
struct { int a[3]; } a_struct = { 0 };
```

Правило 32 (O)

Список перечислимых констант не должен содержать явных инициализаторов для всех констант кроме первой, если не все константы инициализируются явно.

Как проверяется правило

Компилятор выдает ошибку, если существуют инициализаторы хотя бы одной константы в случаях, если:

- первая перечислимая константа не имеет инициализатора;
- число инициализаторов больше одного и меньше числа перечислимых констант.

Операторы

Правила данного раздела относятся к поведению операторов и операндов.

Правило 33 (O)

Правый операнд оператора `&&` или `||` не должен содержать побочного эффекта.

Как проверяется правило

Компилятор выдает ошибку, если выражение справа от оператора `&&` или `||` содержит оператор `++`, `--`, присваивание или вызов функции.

Правило 34 (O)

Операнды логических операторов `&&` или `||` должны быть первичными выражениями.

Как проверяется правило

Компилятор выдает ошибку, если левый и правый операнды логического оператора не являются одиночными переменными, константами или выражениями в скобках.

Замечание. Существует исключение: ошибка не генерируется, если левое или правое выражение использует один и тот же логический оператор. Это безопасно в плане порядка вычислений и читабельности.

Пример нарушения правил

```
a && b || c
a || b && c
a == 3 || b > 5
```

Пример корректного кода

```
a && b && c
a || b || c
(a == 3) || (b > 5)
```

Правило 35 (O)

Операторы присваивания не должны использоваться в выражениях, возвращающих булево значение.

Как проверяется правило

Компилятор выдает ошибку при любом появлении оператора присваивания в булевом контексте, то есть:

- на верхнем уровне управляющего выражения в операторах `if`, `while` или `for`;
- в первой части тернарного оператора `?:`;
- на верхнем уровне левого или правого операнда оператора `&&` или `||`.

Пример нарушения правила

```
if (a = func()) {
...
}
```

Пример корректного кода

```
if ((a = func()) != 0) {
...
}
```

Правило 36 (R)

Логические операторы не должны использоваться совместно с поразрядными операторами.

Как проверяется правило

Компилятор выдает ошибку в следующих ситуациях:

- если поразрядный оператор используется в булевом контексте;
- если логический оператор используется в небулевом контексте.

Пример нарушения правил

```
d = ( c & a ) && b;
d = a && b << c;
if ( ga & 1 ) { ... }
```

Пример корректного кода

```
d = a && b ? a : c;
d = ~a & b;
if ( (ga & 1) == 0 ) { ... }
```

Замечание. Следующие случаи считаются булевым контекстом:
- верхний уровень управляющего выражения операторов `if`, `while`, `for`;
- верхний уровень первого выражения тернарного оператора `?:`;
- верхний уровень левого или правого операнда оператора `&&` или `||`.

Правило 37 (O)

Поразрядные операции не должны выполняться над знаковыми целыми типами.

Как проверяется правило

Компилятор выдает ошибку, если тип операции является знаковым целым, за исключением случаев, если выражение является:

- положительной константой;
- прямо конвертированным из целого типа, строго меньшего, чем `int`;
- логической операцией.

Правило 38 (O)

Правый операнд оператора сдвига должен быть в диапазоне от нуля до числа, на единицу меньше, чем ширина левого операнда в битах.

Как проверяется правило

Компилятор выдает ошибку, если правый операнд оператора сдвига является целой константой со значением, превышающим ширину левого операнда в битах после его расширения до целого типа.

В следующем примере для знаковой целой 8-битной переменной `i8` компилятор не выдает ошибку при сдвиге на `8` позиций, так как значение `i8` будет расширено до `int` перед операцией над левым операндом и, следовательно, будет обработано корректно.

Пример корректного кода

```
i8 = i8 >> 8; /* i8 расширяется до int */
```

Правило 39 (O)

Оператор унарный минус не должен применяться к беззнаковому выражению.

Как проверяется правило

Компилятор выдает ошибку, если унарный минус применяется к выражению беззнакового типа.

Правило 40 (R)

Оператор `sizeof` не должен применяться к выражениям, имеющим побочный эффект.

Как проверяется правило

Компилятор выдает ошибку, если оператор `sizeof` применяется к выражению, содержащему оператор `++`, `--`, оператор присваивания или вызов функции.

Правило 41 (R)

Реализация целочисленного деления в применяемом компиляторе должна быть определена, документирована, и учитываться в расчетах.

Как проверяется правило

Поведение данной операции является реализационно-зависимым. Для компилятора IAR C/C++ Compiler знак остатка при целочисленном делении совпадает со знаком делимого, что задокументировано в руководстве *IAR C/EC++ Compiler Reference Guide*.

Правило 42 (O)

Оператор запятая должен использоваться только в управляющем выражении цикла `for`.

Как проверяется правило

Компилятор выдает ошибку, если запятая используется в любом месте кроме первой или последней части заголовка цикла `for`.

Преобразования

Правила данного раздела относятся к преобразованию данных и приведению типов.

Правило 43 (O)

Неявные преобразования, способные приводить к потере информации не должны использоваться.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 44 (P)

Избыточные явные приведения типов не должны использоваться.

Как проверяется правило

Компилятор выдает ошибку, если явное приведение типа используется для идентичного типа.

Правило 45 (O)

Приведение между указателем и любым другим типом не должно использоваться.

Как проверяется правило

Компилятор выдает ошибку, если значение типа указатель на объект приводится к любому другому типу или если любое значение приводится к типу указатель на объект.

Замечание. Это включает явное и неявное приведение между указателем на объект любого типа и указателем `void`, разрешенное стандартом.

Выражения

Правила данного раздела относятся к выражениям.

Правило 46 (O)

Значение выражения должно быть одним и тем же при любом порядке вычислений, допускаемом компилятором.

Как проверяется правило

Компилятор выдает ошибку для выражений, содержащих:

- множественную запись переменной без промежуточной проверки;
- беспорядочное чтение и запись одной и той же переменной;
- беспорядочный доступ к переменной класса `volatile`.

Замечание. Реализация не генерирует ошибку для выражения `f() + f()`.

Правило 47 (Р)

В выражениях не должны использоваться зависимости от приоритета операторов Си.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Пример нарушения правила

```
x = 3 * a + b / c;
```

Пример корректного кода

```
x = (3 * a) + (b / c);
```

Правило 48 (Р)

Арифметика смешанной точности должна использовать явное приведение типов для получения желаемого результата.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 49 (Р)

Проверка значения на нуль должна быть явной, если операнд не является точно булевым.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 50 (О)

Плавающие переменные не должны проверяться на точное равенство или неравенство.

Как проверяется правило

Компилятор выдает ошибку, если к плавающим значениям применяется операция `==` или `!=`. Ошибка не генерируется при явном сравнении с плавающей константой `0.0`.

Правило 51 (Р)

Вычисление беззнакового целого константного выражения не должно приводить к появлению переноса.

Как проверяется правило

Компилятор выдает ошибку, если вычисление беззнакового целого константного выражения приводит к появлению переноса.

Управление

Правила данного раздела относятся к управлению потоком вычислений.

Правило 52 (О)

Код не должен содержать недостижимых фрагментов.

Как проверяется правило

Компилятор выдает ошибку при наличии любого из следующего:

- код после `goto` или `return`;
- код в теле `switch` до первой метки;

- код после бесконечного цикла (с константным управляющим выражением, вычисляющимся в значение `true`);
- код после вызова функции, заведомо не имеющей возврата;
- код после `break` в операторе `switch`;
- код после оператора `if`, всегда выбирающий один подчиненный оператор, делая недоступным другой, подчиненный оператор;
- код после оператора `if`, делающий недоступными оба подчиненных альтернативных оператора;
- код после оператора `switch`, в котором недоступны все метки подчиненных меток выбора.

Правило 53 (O)

Все не нуль-операторы должны иметь побочный эффект.

Как проверяется правило

Компилятор выдает ошибку, если оператор не содержит вызова функции, присваивания, оператора с побочным эффектом (`++`, `--`) или доступа к переменной класса `volatile`.

Пример нарушения правила

```
v; /* Если 'v' - не volatile */
```

Пример корректного кода

```
do_stuff();
;  /* Нуль-оператор       */
v; /* Если 'v' - volatile */
```

Правило 54 (O)

Нуль-оператор должен располагаться на отдельной строке и не должен иметь на той же строке другого текста.

Как проверяется правило

Компилятор выдает ошибку нуль-оператора, если физическая строка содержит что-либо, кроме одиночной точки с запятой в окружении пробельных символов.

Правило 55 (P)

Метки допустимы только в операторах `switch`.

Как проверяется правило

Компилятор выдает ошибку, если используются метки кроме `case` и `default`.

Правило 56 (O)

Оператор `goto` не должен использоваться.

Как проверяется правило

Компилятор выдает ошибку, если используется оператор `goto`.

Правило 57 (O)

Оператор `continue` не должен использоваться.

Как проверяется правило

Компилятор выдает ошибку, если используется оператор `continue`.

Правило 58 (O)

Оператор `break` должен использоваться только для завершения выбора в операторе `switch`.

Как проверяется правило

Компилятор выдает ошибку для любого оператора `break`, не являющегося частью оператора `switch`.

Правило 59 (O)

Операторы, составляющие тело оператора `if`, `else if`, `else`, `while`, `do ... while` или `for` всегда должны быть заключены в фигурные скобки.

Как проверяется правило

Компилятор выдает ошибку, если операторы, составляющие тело указанных конструкций не являются блоками.

Правило 60 (P)

Все конструкции `if`, `else if` должны содержать заключительный элемент `else`.

Как проверяется правило

Компилятор выдает ошибку, если оператор `if` не содержит элемента `else`.

Правило 61 (O)

Каждый непустой элемент `case` в операторе `switch` должен быть завершен оператором `break`.

Как проверяется правило

Компилятор выдает ошибку для каждого элемента `case`, не завершенного оператором `break`.

Замечание. Ошибка генерируется даже в том случае, если элемент `case` завершается оператором `return`.

Правило 62 (O)

Все операторы `switch` должны содержать заключительную метку `default`.

Как проверяется правило

Компилятор выдает ошибку, если оператор `switch` не имеет метки `default` или метка `default` не последняя в операторе `switch`.

Правило 63 (P)

Выражение `switch` не должно представлять булевых значений.

Как проверяется правило

Компилятор выдает ошибку в следующих двух случаях:

- управляющее выражение оператора `switch` является результатом операции сравнения (равенства или отношения) или логической операции (`&&`, `||` или `!`).
- тело `switch` содержит только один элемент `case`.

Правило 64 (O)

Каждый оператор `switch` должен иметь хотя бы один элемент `case`.

Как проверяется правило

Компилятор выдает ошибку, если оператор `switch` не содержит хотя бы одного элемента `case`.

Правило 65 (O)

Плавающие переменные не должны использоваться как счетчики цикла.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 66 (P)

В операторе `for` должны содержаться только выражения, относящиеся к управлению циклом.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 67 (P)

Переменные, используемые в цикле `for` для счета циклов не должны модифицироваться в теле цикла.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Функции

Правила данного раздела относятся к объявлению и использованию функций.

Правило 68 (O)

Функции всегда должны объявляться на уровне области видимости файла.

Как проверяется правило

Компилятор выдает ошибку, встречая объявление функции на уровне области видимости блока.

Правило 69 (O)

Функции с переменным числом аргументов не должны использоваться.

Как проверяется правило

Компилятор выдает ошибку при объявлении или определении функции с использованием аргумента в виде многоточия.

Замечание. При использовании макросов `va_start`, `va_end` или `va_arg` ошибки не возникает, потому что использовать их без аргумента в виде многоточия не имеет смысла.

Правило 70 (O)

Функции не должны вызывать сами себя прямо или косвенно.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 71 (O)

Функции всегда должны иметь прототип, видимый как при определении функции, так и при ее вызове.

Как проверяется правило

Компилятор выдает ошибку в случаях, если:

- определяется нестатическая функция с невидимым в месте определения функции прототипом;
- используется тип указатель на функцию без прототипа функции;
- объявляется функция без прототипа.

Пример нарушения правила

```
void func(); /* Если отсутствует прототип */
```

Пример корректного кода

```
void func(void);
void func(void) { ... }
```

Правило 72 (O)

Типы возвращаемого значения и каждого параметра функции в объявлении и в определении должны совпадать.

Как проверяется правило

Компилятор выдает ошибку для каждого определения функции, в котором типы возвращаемого значения и параметров не совпадают с типами в объявлении. Типы, определенные с помощью `typedef` и имеющие разные имена, считаются различными и генерируют ошибку.

Правило 73 (O)

Прототип функции должен либо иметь идентификаторы для всех параметров, либо не иметь их вообще.

Как проверяется правило

Компилятор выдает ошибку, если прототип имеет идентификаторы более чем для одного параметра, но не для всех.

Правило 74 (O)

Если для каких либо параметров заданы идентификаторы, то они должны быть идентичны как в объявлении, так и в определении.

Как проверяется правило

Компилятор выдает ошибку, если идентификатор в определении функции не совпадает с соответствующим идентификатором в прототипе.

Правило 75 (O)

Каждая функция должна иметь явный возвращаемый тип.

Как проверяется правило

Компилятор выдает ошибку, если функция имеет неявно объявленный тип возвращаемого значения.

Правило 76 (O)

Функции без параметров должны быть объявлены с параметром типа `void`.

Как проверяется правило

Компилятор выдает ошибку, если объявление или определение функции отличается от прототипа.

Правило 77 (O)

Параметр без квалификатора, передаваемый функции должен быть совместим с ожидаемым типом без квалификатора, определенным в прототипе функции.

Как проверяется правило

Компилятор выдает ошибку для вызовов функций, требующих неявного преобразования какого либо параметра.

Правило 78 (O)

Число параметров, передаваемых функции, должно соответствовать прототипу функции.

Как проверяется правило

Компилятор всегда проверяет это даже при запрещенных правилах MISRA C.

Правило 79 (O)

Функции типа `void` не должны возвращать параметров.

Как проверяется правило

Компилятор всегда проверяет это даже при запрещенных правилах MISRA C.

Правило 80 (O)

Выражения типа `void` не должны передаваться как параметры функции.

Как проверяется правило

Компилятор всегда проверяет это даже при запрещенных правилах MISRA C.

Правило 81 (P)

Параметры функции, передаваемые по ссылке должны иметь квалификатор `const`, если подразумевается, что функция не должна модифицировать параметры.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила. Это правило требует ручной проверки.

Правило 82 (P)

Функция должна иметь только одну точку выхода.

Как проверяется правило

Компилятор выдает ошибку для второй точки выхода из функции, образуется ли она оператором `return` или концом функции.

Для недостижимых точек выхода ошибка не генерируется.

Правило 83 (O)

Правило для функций, возвращающих значение типа, отличного от типа `void`.

- Для каждой ветви, заканчивающейся выходом (включая конец программы), должен быть один оператор `return`;
- каждый оператор `return` должен иметь выражение;
- выражение оператора `return` должно соответствовать объявленному типу возвращаемого значения.

Как проверяется правило

Компилятор выдает ошибку в следующих случаях:

- функция, возвращающая значение типа, отличного от `void` не имеет оператора `return` в самом конце функции;
- оператор `return` не имеет выражения;
- выражение любого оператора `return` неявно конвертируется для соответствия возвращаемому типу.

Правило 84 (O)

В функциях, возвращающих значение типа `void`, оператор `return` не должен иметь выражения.

Как проверяется правило

Компилятор всегда проверяет это даже при запрещенных правилах MISRA C.

Правило 85 (P)

Вызовы функций без параметров должны иметь пустые скобки.

Как проверяется правило

Компилятор выдает ошибку, если:

- обозначение функции (имя функции без скобок) используется в управляющем выражении оператора `if`, `while` или `for`;
- обозначение функции сравнивается с 0 при помощи оператора `==` или `!=`;
- обозначение функции используется в выражении типа `void`.

Пример нарушения правила

```
extern int func(void);
if ( func ) { ... }
```

Пример корректного кода

```
extern int func(void);
if ( func() ) { ... }
```

Правило 86 (P)

Если функция возвращает информацию об ошибке, то эта информация должна быть проанализирована.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила. Это правило требует ручной проверки.

Директивы препроцессора

Правила данного раздела относятся к включаемым файлам и директивам препроцессора.

Правило 87 (O)

Директивам `#include` в исходном файле могут предшествовать только другие директивы препроцессора или комментарии.

Как проверяется правило

Компилятор выводит ошибку, если директиве `#include` предшествует что-либо кроме директивы препроцессора или комментария.

Правило 88 (O)

В именах заголовочных файлов директивы `#include` не должны присутствовать нестандартные символы.

Как проверяется правило

Компилятор выводит ошибку, если имя заголовочного файла содержит какой либо нестандартный символ.

Правило 89 (O)

После директивы `#include` должна следовать последовательность `<filename>` или `"filename"`.

Как проверяется правило

Компилятор выводит ошибку, если за директивой `#include` не следует " или <.

Правило 90 (O)

Макросы Си должны использоваться только для символьных констант, функция-подобных макросов, квалификаторов типа и спецификаторов класса хранения.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 91 (O)

Макросы не должны определяться (`#define`) или уничтожаться (`#undef`) внутри блока.

Как проверяется правило

Компилятор выводит ошибку, если директива `#define` или `#undef` используется вне области видимости файла.

Правило 92 (P)

Директива `#undef` не должна использоваться.

Как проверяется правило

Компилятор выводит ошибку при использовании директивы `#undef`.

Правило 93 (P)

Использование функций должно пользоваться предпочтением перед использованием функция-подобных макросов.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила. Это правило требует ручной проверки.

Правило 94 (O)

Функция-подобный макрос должен вызываться со всеми своими аргументами.

Как проверяется правило

Компилятор выводит ошибку при вызове макроса, если один или более аргументов не содержат какой либо лексемы.

Пример нарушения правила

```
MACRO(,)
```

Пример корректного кода

```
#define EMPTY
MACRO(EMPTY, EMPTY)
```

Правило 95 (O)

Аргументы функция-подобных макросов не должны содержать лексем, выглядящих как директивы препроцессора.

Как проверяется правило

Компилятор выдает ошибку, если используется препроцессируемая лексема, начинающаяся с символа #.

Замечание. Ошибка не возникает для никогда не расширяющихся макросов.

Правило 96 (O)

В определении функция-подобного макроса все определение и каждый экземпляр параметра должны быть заключены в скобки.

Как проверяется правило

Компилятор выдает ошибку, если в определении функция-подобного макроса присутствует любое из следующего:

- макропараметр в замещающем тексте не заключен в скобки;
- замещающий текст не заключен в скобки.

Пример нарушения правил

```
#define FOO(x) x + 2
#define FOO(x) (x) + 2
```

Пример корректного кода

```
#define FOO(x) ((x) + 2)
```

Правило 97 (P)

Идентификаторы в директивах препроцессора должны быть определены перед использованием.

Как проверяется правило

Компилятор выдает ошибку, если используется неопределенный символ препроцессора в директивах `#if` или `#elif`.

Правило 98 (O)

В одном макроопределении должно быть не более одного включения оператора препроцессора # или ##.

Как проверяется правило

Компилятор выдает ошибку, если в комбинации используется более одного из символов # или ##. Например, появление обоих символов # и ## в одном макросе вызовет ошибку.

Пример нарушения правила

```
#define FOO(x) BAR(#x) ## _var
```

Пример корректного кода

```
#define FOO(x) #x
#define FOO(x) my_ ## x
```

Правило 99 (O)

Любое использование директивы `#pragma` должно быть документировано и объяснено.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила. Это правило требует ручной проверки.

Правило 100 (O)

Оператор препроцессора `defined` должен использоваться только в одной из двух стандартных форм.

Как проверяется правило

Компилятор выдает ошибку, если результат расширения макроса в выражении, управляющим условным включением, приводит к унарному оператору `defined`.

Указатели и массивы

Правила данного раздела относятся к указателям и массивам.

Правило 101 (P)

Арифметика указателей не должна использоваться.

Как проверяется правило

Компилятор выдает ошибку, если левый или правый операнд операторов `+`, `-`, `+=` или `-=` является выражением типа указателя.

Правило 102 (P)

Допускается использование не более двух уровней косвенной адресации при помощи указателей.

Как проверяется правило

Компилятор выдает ошибку, если какой либо тип использует более, чем два уровня косвенной адресации в объявлении или определении объекта или функции.

Правило 103 (О)

К типам указателей операторы отношения могут применяться только если оба операнда имеют один и тот же тип и указывают на один и тот же массив, структуру или объединение.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 104 (О)

Допускается использование только константных указателей на функции.

Как проверяется правило

Компилятор выдает ошибку, если производится явное приведение значения к указателю на функцию, за исключением приведения типов:

- константных значений;
- указателей на функции.

Правило 105 (О)

Все функции, вызываемые одним указателем на функцию должны иметь одинаковое число и типы параметров, и тип возвращаемого значения.

Как проверяется правило

Компилятор выдает ошибку, если производится явное или неявное приведение типа указателя на функцию одного типа к типу указателя на функцию иного типа.

Правило 106 (О)

Адрес объекта с автоматическим классом хранения не должен присваиваться указателю, который может сохраняться после того, как объект перестает существовать.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 107 (О)

Ссылка по нуль-указателю не допускается.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Структуры и объединения

Правила данного раздела относятся к описанию и использованию структур и объединений.

Правило 108 (О)

В описании типа структуры или объединения все члены должны быть специфицированы полностью.

Как проверяется правило

Компилятор выдает ошибку, если поле объявляется массивом без размера.

Правило 109 (O)

Перекрытие памяти не должно использоваться.

Как проверяется правило

Компилятор выдает ошибку для определения или объявления объединения.

Правило 110 (O)

Объединения не должны использоваться для доступа к фрагментам более крупных типов данных.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 111 (O)

Типы битовых полей должны определяться только как `unsigned int` или `signed int`.

Как проверяется правило

Компилятор выдает ошибку, если битовое поле объявляется с типом, отличным от `unsigned int` или `signed int`.

Замечание. Ошибка происходит, если битовое поле объявляется с типом `int` без использования спецификатора `signed` или `unsigned`.

Правило 112 (O)

Битовые поля типа `signed int` должны иметь длину не менее 2 бит.

Как проверяется правило

Компилятор выдает ошибку, если битовое поле типа `signed int` объявлено с размером 0 или 1 бит.

Правило 113 (O)

Все члены структуры или объединения должны иметь имена и получать доступ только по именам.

Как проверяется правило

Компилятор выдает ошибку, если битовое поле объявляется без имени или если используется адрес поля структуры.

Стандартные библиотеки

Правила данного раздела относятся к использованию функций стандартной библиотеки.

Правило 114 (O)

Зарезервированные слова и имена функций стандартной библиотеки не могут быть переопределены или уничтожены.

Как проверяется правило

Компилятор выдает ошибку для любого определения (`#define`) или уничтожения (`#undef`) объекта или функция-подобного макроса с использованием имени, являющимся одним из следующих:

- предопределенный макрос компилятора;
- объект, функция или функция-подобный макрос, определенные в каком либо стандартном заголовочном файле.

Правило 115 (O)

Имена функций стандартной библиотеки не должны использоваться повторно.

Как проверяется правило

Компилятор выдает ошибку для любого определения функции, использующего имя уже объявленное в стандартном заголовочном файле, не зависимо от того, включается ли этот файл или нет.

Правило 116 (O)

Все библиотеки, используемые для получения результирующего кода, должны быть написаны в соответствии с положениями данного документа и аттестованы соответствующим образом.

Как проверяется правило

Данное правило не навязывается.

Правило 117 (O)

Действительность значений, передаваемых библиотечным функциям, должна подвергаться проверке.

Как проверяется правило

Компилятор и компоновщик не проверяют нарушение этого правила.

Правило 118 (O)

Динамическое выделение памяти в куче не должно использоваться.

Как проверяется правило

Компилятор выдает ошибку для любых ссылок на функции с именами `malloc`, `realloc`, `calloc` или `free`, даже если заголовочный файл `stdlib.h` был включен.

Правило 119 (O)

Индикатор ошибки `errno` не должен использоваться.

Как проверяется правило

Компилятор выдает ошибку для любой ссылки на объекты с именем `errno`, даже если заголовочный файл `errno.h` был включен.

Правило 120 (O)

Макрос `offsetof` из библиотеки `<stddef.h>` не должен использоваться.

Как проверяется правило

Компилятор выдает ошибку, если макрос `offsetof` подвергается расширению.

Замечание. Включение заголовочного файла `stddef.h` само по себе не генерирует ошибку.

Правило 121 (O)

Заголовочный файл `<locale.h>` и функция `setlocale` не должны использоваться.

Как проверяется правило

Компилятор выдает ошибку, если заголовочный файл `locale.h` включен.

Правило 122 (O)

Макрос `setjmp` и функция `longjmp` не должны использоваться.

Как проверяется правило

Компилятор выдает ошибку, для любой ссылки на функции с именами `setjmp` или `longjmp`; независимо от включения заголовочного файла `setjmp.h`.

Правило 123 (O)

Средства поддержки сигнала <`signal.h`> не должны использоваться.

Как проверяется правило

Компилятор выдает ошибку, если заголовочный файл `signal.h` включен.

Правило 124 (O)

Библиотека ввода-вывода <`stdio.h`> не должна использоваться в результирующем коде.

Как проверяется правило

Компилятор выдает ошибку, если заголовочный файл `stdio.h` включен и определен макрос `NDEBUG`.

Правило 125 (O)

Библиотечные функции `atof`, `atoi` и `atol` из библиотеки <`stdlib.h`> не должны использоваться.

Как проверяется правило

Компилятор выдает ошибку для любой ссылки на функции `atof`, `atoi` или `atol` независимо от включения заголовочного файла `stdlib.h`.

Правило 126 (O)

Библиотечные функции `abort`, `exit`, `getenv` и `system` из библиотеки <`stdlib.h`> не должны использоваться.

Как проверяется правило

Компилятор выдает ошибку для любой ссылки на функции `abort`, `exit`, `getenv` и `system` независимо от включения заголовочного файла `stdlib.h`.

Правило 127 (O)

Функции поддержки времени из библиотеки <`time.h`> не должны использоваться.

Как проверяется правило

Компилятор выдает ошибку, если включен заголовочный файл `time.h`.