# Анализ изменений стандарта MISRA-C 2023

**К. А. Костюхин[1], Г. Л. Левченкова[2]**

[1]НИЦ "Курчатовский институт" - НИИСИ, Москва, kost@niisi.ras.ru;

[2]НИЦ "Курчатовский институт" - НИИСИ, Москва, galka@niisi.ras.ru

**Аннотация.** Работа содержит краткий обзор основных изменений стандарта MISRA C 2023 по сравнению с его предыдущей редакцией 2012 года. рассматриваются принципиальные изменения в структуре стандарта, новые требования и рекомендации, а также практические последствия для разработчиков встроенных и критичных к безопасности систем.

**Ключевые слова:** язык Си, стандарт, MISRA C 2023

## 1. Введение

Стандарт MISRA C (Motor Industry Software Reliability Association) [1] возник как ответ на потребности автомобильной индустрии в предписывающих руководящих принципах для безопасного, переносимого и проверяемого использования языка C. С течением времени он превратился в де-факто эталон для множества отраслей, где надежность программного обеспечения критична: авиация, медицина, железнодорожный транспорт, промышленные контроллеры и т.п.

Версия MISRA C 2012 [2] установила базу для строгой дисциплины программирования на языке C с четкой классификацией правил, директив и механизмов доказательства соответствия. Однако развитие языка C (включая стандарты C11, C17 и C23), рост числа многопоточных/параллельных программ, а также накопившийся опыт применения MISRA C в промышленности требовали пересмотра и обновления набора правил. В результате появился MISRA C 2023 [3], который, при соблюдении преемственности, вносит ряд существенных изменений: обновленную структуру правил, новые или переработанные требования, уточнения в терминологии и механизмах доказательства соответствия, а также усиление внимания к современным аспектам, таким как порядок вычислений, многопоточность и взаимодействие с компиляторными расширениями.

Цель данной статьи - выполнить систематическую декомпозицию отличий между MISRA C 2012 (AMD1+AMD2) и MISRA C 2023, предоставить объяснение существенных новых или измененных правил и проиллюстрировать их примерами.

## 2. Обзор изменений в структуре стандарта

### 2.1. Переработка организационной структуры и классификации правил

**Ключевое изменение.** MISRA C 2023 переосмысливает и уточняет структуру правил: происходит более строгая сегрегация требований по уровням обязательности, четкая фиксация зависимостей между правилами и директивами, а также улучшенное разделение между строго нормативными требованиями и поясняющими рекомендациями. Это отражено в переработанных заголовках правил и их аннотациях.

**Почему это важно.** В прошлой версии (MISRA C 2012) некоторые правила и директивы могли трактоваться неоднозначно при реализации процессов соответствия. Новая структура снимает неоднозначности, облегчая реализацию и аудит: инструменты статического анализа теперь могут точнее отображать причину нарушения, уровень строгости и возможную стратегию соответствия.

**Пример (иллюстрация структурной перестройки).**

*Исходный контекст:* правило X (в MISRA C 2012) могло быть классифицировано как рекомендация, однако его нарушение на практике приводило к критическим ошибкам. MISRA C 2023 перенесло это положение в разряд обязательных или ввело сопутствующую директиву для уточнения области применения.

*Код:*

```
/* Пример: использование динамического приведения типов для работы с бинарными пакетами */
```

```
void process (void *data) {
   uint32_t *p = (uint32_t *)data;
/* предположение: data выровнено и указывает на 32-битный блок */
   uint32_t v = *p;
   /* ... */
}
```

*Что иллюстрирует пример.* Данный фрагмент иллюстрирует ситуацию, где код полагается на предположения о выравнивании и представлении данных, что может быть небезопасно на разных архитектурах. В MISRA C 2023 такие конструкции получили более жесткую классификацию и дополнились директивой по обязательной верификации условий (alignment/representation) при использовании подобных конструкций.

*Действие для приведения к стандарту.* Явная проверка выравнивания, использование memcpy для копирования вместо безопасного приведения, или структурированное чтение байтов с учетом их порядка. Все эти меры иллюстрируют практическую адаптацию.

## 2.2. Расширение охвата языка C и явный учет современных языковых конструкций

**Ключевое изменение.** MISRA C 2023 официально учитывает развитие языка C после C11 (включая особенности, включенные в C17/C23) и уточняет совместимость правил с конструкциями современных компиляторов, в том числе с механизмами атомарных операций, ограничениями на порядок вычислений и новыми типовыми возможностями. Это зафиксировано в ряде новых заголовков и пояснений.

**Почему это важно.** Современные проекты чаще используют элементы стандарта C, появившиеся после C99, включая атомарные типы <stdatomic.h>, условную компиляцию для разных стандартов и новые правила оптимизации компиляторов. Для поддержания релевантности MISRA C должен явным образом описывать поведение правил для этих конструкций.

**Пример (атомарные операции и порядок выполнения).**

*Код:*

```
#include <stdatomic.h>

atomic_int flag = 0;
int shared = 0;

void writer (void) {
   shared = 42;              /* (1) */
   atomic_store_explicit (&flag, 1, memory_order_release); /* (2) */
}

void reader (void) {
   if (atomic_load_explicit (&flag, memory_order_acquire) == 1) { /* (3) */
      /* гарантируется видимость записи shared */
      int v = shared;       /* (4) */
   }
}
```

*Что иллюстрирует пример.* Использование атомарных операций с семантикой release/acquire для синхронизации записи и чтения некоторых переменных. MISRA C 2023 более явно рассматривает такие поведенческие шаблоны и дает рекомендации по корректному использованию атомарных операций, чтобы избежать неопределенного поведения и несогласованных состояний в многопоточной среде.

*Действие для приведения к стандарту.* Правила стандарта требуют однозначности - если код использует атомарные операции, эти операции должны иметь явно заданный порядок доступа к памяти, и их применение должно сопровождаться комментариями/доказательствами, почему выбранный порядок корректен в контексте проекта.

## 2.3. Уточнения терминологии и механизма доказательства соответствия (compliance)

**Ключевое изменение.** MISRA C 2023 уточняет термины: что имеется в виду под «нарушением», «директивой», «рекомендацией», «правилом обязательного характера», а также отношения между «доказательством соответствия» и «допустимыми отклонениями (deviations)». Эти уточнения помогают аудиторам и инженерам точно интерпретировать выявленные нарушениях и корректно оформлять отклонения.

**Почему это важно.** Четкое понимание классификации нарушений – это основа для корректных процедур аудита, тестирования и документирования решений об отклонениях. Без этого разработчики могут неправильно относить предупреждения компилятора/анализатора к категории «рекомендация» вместо «обязательное требование», и наоборот.

**Пример (доказательство соответствия).**

*Сценарий:* Проект использует низкоуровневую операцию ввода-вывода,

которая по дизайну нарушает правило X (например, прямой доступ к регистрам через приведенный указатель). В MISRA C 2012 это могло быть признано допустимым при оформлении отклонения. MISRA C 2023 требует более формального доказательства: документального анализа, тестов и статического/динамического обоснования безопасности.

*Что иллюстрирует пример.* Необходимость формальной документации при отклонениях - не только запись «отклонение разрешено», но и конкретные ссылки на тесты, статический/динамический анализ и условия эксплуатации. Это повышает надежность процедур соответствия.

# 3. Новые и существенно измененные требования MISRA C 2023

Ниже приведен разбор ключевых новых требований и переработок. Для удобства они сгруппированы по тематике: управление неопределенным поведением, типовая безопасность и преобразования, обработка указателей и выравнивание, порядок вычислений и побочные эффекты, многопоточность и параллелизм, взаимодействие с компиляторными расширениями, проверяемость и инструментальная поддержка, процесс отклонений.

## 3.1. Управление неопределенным поведением (Undefined Behavior, UB)

**Изменение.** MISRA C 2023 усиливает требования по избеганию ситуаций, приводящих к неопределенному поведению в языке C. В тексте правил уделено повышенное внимание таким ситуациям, как: переполнение знаковых целых типов, некорректный доступ по выровненному указателю, чтение неинициализированной памяти, использование освобожденной памяти и некорректные преобразования типов.

**Почему это важно.** UB может приводить к непредсказуемым последствиям: от ошибочных результатов до аварийной остановки или эксплуатации уязвимостей. Современные компиляторы активно используют UB для оптимизаций, следовательно, код, полагающийся на UB, может вести себя по-разному в разных версиях компилятора или с разными параметрами оптимизации.

**Пример 1 (переполнение знакового типа).**

*Код:*

```
#include <stdint.h>

int32_t sum (int32_t a, int32_t b) {
  return a + b; /* потенциальное переполнение */
}
```

*Что происходит.* При сложении двух знаковых 32-битных чисел может произойти переполнение, которое в стандарте C является неопределенным поведением. Компилятор может предполагать, что такого переполнения не бывает, и оптимизировать код исходя из этого предположения, что приведет к неожиданным результатам в случае переполнения.

*MISRA C 2023.* Указывает на необходимость либо использовать беззнаковую арифметику с проверкой переполнения, либо производить проверки до операции, либо использовать безопасные функции/макросы, которые явно документируют поведение при переполнении.

*Как исправлять.* Один из подходов - приведение к 64-битному типу с проверкой:

```
int32_t sum_safe (int32_t a, int32_t b) {
  int64_t tmp = (int64_t)a + (int64_t)b;
  if (tmp > INT32_MAX) {
    /* обработка ошибки */
  }
  if (tmp < INT32_MIN) {
    /* обработка ошибки */
  }
  return (int32_t)tmp;
}
```

**Пример 2 (чтение неинициализированной памяти).**

*Код:*

```
int f (void) {
  int x; /* неинициализирован */
  return x + 1;
}
```

*Что происходит.* Чтение x до инициализации, классический пример неопределенного поведения. MISRA C 2023 ужесточает требования по обнаружению подобных операций и рекомендует явно инициализировать переменные или использовать статический анализ для доказательства инициализации.

*Практика.* Использование инструментов статического анализа, добавление правил кодирования, обязательные обзоры и тесты покрывают такие проблемы.

## 3.2. Порядок вычислений и побочные эффекты

**Изменение.** MISRA C 2023 сильнее фокусируется на проблемах, связанных с неопределенным порядком вычислений выражений, в результате чего пересматриваются и уточняются правила, запрещающие конструкции, зависящие от порядка вычислений подвыражений (например, модификация объекта более одного раза между последовательными точками наблюдения).

**Почему это важно.** Распространенной категорией ошибок является использование выражений вроде i = i++ + 1; или a[i++] = i; где результат зависит от порядка вычислений. Компиляторы и стандарты языка допускают вариативное поведение, следовательно, такие конструкции приводят к UB.

**Пример (выражение с побочными эффектами).**

*Код:*

```
int i = 0;
int a[2];
a[i] = ++i;
```

*Что происходит.* Поведение не определено: одновременно читается и модифицируется i без промежуточной последовательной точки наблюдения. MISRA C 2023 требует запрета таких конструкций и рекомендует разделять операции:

```
int temp = ++i;
a[i] = temp;
```

**Особое внимание следует уделять выражениям с функциями и порядком вычислений их аргументов.** MISRA C 2023 вводит пояснения о том, что порядок вычисления аргументов функций является зависимым от реализации, поэтому код должен не допускать зависимостей между аргументами, где один аргумент модифицирует объект, читаемый в другом аргументе.

## 3.3. Указатели, выравнивание и приведенные типы

**Изменение.** В MISRA C 2023 усилена позиция по наглядности и безопасности операций с указателями: переработаны требования к приведению указателей и обеспечению корректного выравнивания.

**Почему это важно.** Неправильное выравнивание может привести к аппаратным исключениям на некоторых архитектурах, а неявные преобразования типов указателей могут скрывать ошибки. Это особенно критично в низкоуровневом коде (драйверы, работа с регистровыми картами).

**Пример (приведение void* к типу с более строгим выравниванием).**

*Код:*

```
void *buffer = get_unaligned_buffer ();
uint32_t *p = (uint32_t *)buffer;
uint32_t v = *p; /* потенциальное нарушение выравнивания */
```

*Что происходит.* Если buffer не выровнен по 4-байтовой границе, то чтение *p может вызывать аппаратную ошибку. MISRA C 2023 требует либо проверки выравнивания перед такими операциями, либо использование memcpy для безопасного копирования.

*Исправление:*

```
uint32_t v;
memcpy (&v, buffer, sizeof v);
```

**Пример (strict aliasing).**

*Код:*

```
float f = 1.0f;
int *p = (int *)&f;
int i = *p; /* может нарушать strict aliasing */
```

*Что происходит.* Чтение float через int* нарушает правило strict aliasing и может вызвать неопределенное поведение при оптимизациях. MISRA C 2023 более явным образом рекомендует избегать подобных приведений и использовать, например, memcpy.

## 3.4. Типовая безопасность и преобразования

**Изменение.** Стандарт усиливает требования к контролю за неявными преобразованиями, особенно между знаковыми и беззнаковыми типами, при преобразовании целочисленных типов различной ширины, а также при использовании литералов и макросов, которые могут приводить к неожиданным расширениям типа.

**Почему это важно.** Неправильные преобразования приводят к логическим ошибкам и уязвимостям, в частности когда отрицательные значения трактуются как большие положительные после преобразования в беззнаковый тип.

**Пример (смешивание знаковых и беззнаковых типов).**

*Код:*

```
int32_t a = -1;
uint32_t b = 1;
if (a < b) { /* сравнение int32_t и uint32_t */
   /* ... */
}
```

*Что происходит.* В выражении a < b значение a будет преобразовано к uint32_t, что дает большое положительное число, и поэтому условие, вероятно, окажется ложным, что не соответствует интуитивному ожиданию. MISRA

C 2023 заставляет избегать таких неочевидных сравнений и требует явного приведения типов или использования временных переменных.

*Исправление:*

```
if ((int32_t)a < (int32_t)b) { /* явное приведение и проверка */
    /* ... */
}
```

## 3.5. Многопоточность, атомарные операции и взаимодействие с памятью

**Изменение.** В MISRA C 2023 появляется более формализованное обсуждение многопоточности, атомарности и порядка доступа к памяти. Это включает рекомендации по использованию <stdatomic.h>, корректной последовательности вызовов для обращения к памяти и необходимости документирования инвариантов при многопоточном доступе к общим объектам.

**Почему это важно.** Современные встраиваемые системы все чаще полагаются на многопоточность и аппаратный параллелизм. Неправильное использование неблокирующих конструкций и атомарных операций может приводить к трудноуловимым ситуациям гонки и UB.

**Пример (неправильное использование атомарности).**

*Код:*

```
#include <stdatomic.h>

int counter = 0; /* неатомарная переменная */

void inc (void) {
  counter++; /* гонка при параллельном доступе */
}
```

*Что происходит.* Инкремент counter++ не является атомарной операцией, следовательно, при конкурирующем доступе возможны потерянные значения counter. MISRA C 2023 рекомендует использовать атомарные типы:

```
atomic_int counter = ATOMIC_VAR_INIT(0);

void inc (void) {
  atomic_fetch_add_explicit (&counter, 1, memory_order_relaxed);
}
```

*Дополнение.* Стандарт требует также документировать ожидаемую семантику памяти и показать, почему выбранный порядок доступа к памяти (в примере - memory_order_relaxed) является достаточным для корректности.

## 3.6. Взаимодействие с расширениями компилятора и платформозависимыми конструкциями

**Изменение.** MISRA C 2023 дает более формализованные указания по использованию компиляторных расширений, встроенных ассемблерных вставок, специфичных платформенных API и пр. Новые объяснения направлены на то, чтобы разработчик явно документировал причину и контекст использования расширения, а также включал доказательства того, что поведение, выходящее за рамки стандарта C, находится под контролем и безопасно.

**Почему это важно.** Расширения компилятора часто используются для оптимизации или доступа к аппаратуре; при этом они нарушают переносимость и могут скрывать UB. Четкая документация и ограничения предотвращают непреднамеренные последствия.

**Пример (встроенный ассемблер).**

*Код:*

```
int read_hw (void) {
  int val;
  __asm__ volatile ("in %0, 0x60" : "=r"(val));
  return val;
}
```

*Что происходит.* Встроенный ассемблер выходит за рамки стандарта C. MISRA C 2023 допускает его использование при условии оформления отклонения/пояснения: указать причину, влияние на переносимость, предусмотреть альтернативные реализации и тесты.

*Практика.* Добавление макроса-обвязки, тестов эмуляции и документирование вызовов обеспечивает соответствие требованиям.

## 3.7. Процесс оформления отклонений (deviations) и требования к документации

**Изменение.** MISRA C 2023 уточняет и формализует процесс оформления отклонений: какие шаги должны присутствовать в документе об отклонении, какие доказательства требуются, и какие тесты/проверки должны проводиться.

**Почему это важно.** Отклонения - это не «дыра» в стандарте, а формализованный процесс управления исключениями. Повышенные требования к доказательствам и тестам делают отклонения менее рискованными и более прозрачными для аудита.

**Пример (ориентировочная структура документа отклонения).**

Документ должен содержать:

- Идентификатор отклонения и ссылку на правило.
- Обоснование (архитектурное, аппаратное, производительное).
- Варианты минимизации рисков от использования отклонения.
- Список тестов и результаты статического анализа, которые подтверждают безопасность.
- Ответственное лицо и срок пересмотра.

# 4. Выборочный разбор новых правил и пояснения с примерами

Следующий раздел содержит последовательный разбор наиболее значимых новых или существенно измененных правил, как они отражены в заголовках MISRA C 2023, их практическое значение и развернутые примеры.

Важное замечание. Ниже приведены тематически независимые пункты. Каждый из них ориентирован на отдельную проблему безопасности или надежности кода на языке C и иллюстрирован примером.

## 4.1. Требования по выявлению операций чтения неинициализированных объектов

MISRA C 2023 усиливает требование по обнаружению ситуаций, где читается неинициализированная память. Инструменты должны обнаруживать такие чтения на этапе статического анализа, а код должен либо явно инициализировать объекты, либо иметь доказательство инициализации до чтения.

**Пример (локальная переменная, используемая без инициализации).**

*Код:*

```
int compute(void) {
   int x;          /* неинициализированная переменная */
   if (some_cond ()) {
      x = 10;
   }
   return x;          /* чтение может быть неопределенным */
}
```

*Анализ.* Если some_cond() иногда ложно, то x останется неинициализированным.

MISRA C 2023 требует либо предоставить инициализацию по умолчанию (int x = 0;), либо изменить код, чтобы гарантировать присваивание перед возвратом.

*Рекомендация.* Предпочтительно инициализировать переменные при объявлении, особенно для скалярных локальных переменных, либо использовать явные проверяемые пути присваивания.

## 4.2. Уточнение правил работы с switch и case

MISRA C 2023 требует ясности при использовании switch/case: все допустимые варианты должны быть явно обработаны, а «проваливание» (fallthrough) из одного case в другой допустимо только при явном указании и обосновании. Это уменьшает вероятность ошибок при добавлении новых case-ветвей.

**Пример (неявное проваливание).**

*Код:*

```
switch (x) {
case 1:
   do_a();
case 2:  /* неявное проваливание */
   do_b();
   break;
}
```

*Что происходит.* Если разработчик явно полагался на поведение «проваливания», это может быть трудно заметно при анализе кода. MISRA C 2023 требует явной пометки и обоснования:

```
switch (x) {
case 1:
   do_a();
   /* fallthrough */ /* обоснование: Intentional fallthrough */
case 2:
   do_b();
   break;
}
```

## 4.3. Новые требования по явной обработке ошибок и возвратных кодов

Повышены требования к проверке возвратных кодов функций, особенно тех, которые могут указывать на ошибку (например, функции ввода-вывода, системные вызовы). MISRA C 2023 предписывает явную обработку таких кодов или документированное обоснование, почему их можно игнорировать.

**Пример (игнорирование кода возврата).**

*Код:*

```
int res = write (fd, buf, n);
/* Далее res нигде не используется, игнорирование */
```

*Что происходит.* Игнорирование результата операции может привести к потере данных. Новая политика требует либо обработки res, либо оформленного отклонения с указанием причин.

### 4.4. Расширенные требования по анализу времени выполнения (timing) и влиянию на безопасность

MISRA C 2023 акцентирует внимание на коде, где временные характеристики (латентность, дедлайн) критичны для безопасности. Новые аннотации правил требуют, чтобы код, влияющий на временную надежность, был документирован и прошел соответствующие измерения/тесты.

**Пример (цикл с потенциально неопределенной длительностью).**

*Код:*

```
while (!hardware_ready ()) {
   /* пустой цикл ожидания */
}
```

*Что происходит.* Если hardware_ready () никогда не станет истинной, цикл будет бесконечным, что может иметь критические последствия. MISRA C 2023 требует установки явного таймаута и наличия стратегий выхода из цикла:

```
unsigned timeout = 1000u;
while (!hardware_ready () && timeout--) {
   /* ожидание с тайм-аутом */
}
if (timeout == 0u) {
   /* обработка ошибки */
}
```

### 4.5. Требования по фиксации и аудиту операций со смещениями и индексами массивов

Уточнены правила про доступ к массивам: индексы должны быть проверены, значения границ явно документированы, и использование вычисляемых индексов без проверки должно быть обосновано.

**Пример (индексирование с риском выхода за границы).**

*Код:*

```
int get_value (int *arr, size_t idx) {
   return arr[idx]; /* нет проверки границ */
}
```

*Что происходит.* Без гарантии на размер arr доступ к arr[idx] может выйти за пределы массива. MISRA C 2023 рекомендует либо передавать размер массива вместе с указателем, либо использовать API, где длина известна, либо выполнять проверки.

### 4.6. Новые положения, касающиеся использования макросов и генерации кода через препроцессор

MISRA C 2023 ужесточает требования к макросам: сложные макросы, содержащие инструкции, влияющие на ход выполнения потока, многострочные выражения или побочные эффекты, должны быть сокращены и заменены inline функциями или тщательно документированы. Макрос, меняющий семантику исходного кода, требует отдельного обоснования и тестов.

**Пример (макрос с побочными эффектами).**

*Код:*

```
#define MAX(a,b) ((a) > (b) ? (a) : (b))
int x = MAX(i++, j++);
```

*Что происходит.* Макрос приводит к двойному выполнению i++ или j++, что является непредвиденным поведением. MISRA C 2023 требует избегать таких макросов или использовать inline функции:

```
static inline int max_int(int a, int b) {
   return (a > b) ? a : b;
}
```

## 5. Инструменты и практическая интеграция (рекомендации)

MISRA публикует упрощенные тексты правил (headlines) для интеграции с анализаторами кода - это позволяет инструментам сопоставлять предупреждение с правилом MISRA и облегчает отчетность.

Ниже приведены рекомендации по внедрению в проект.

**Обновление процесса анализа кода.** Включить проверки на соответствие MISRA C 2023 в обязательный процесс анализа кода. Ошибки высокого уровня должны блокировать процесс слияния веток кода или сборки.

**Обучение команды.** Провести семинары по новым правилам и их примерам; обратить внимание на тонкие места: порядок вычислений, строгую семантику, атомарность.

**Стратегия обработки отклонений.** Разработать шаблон для документа, описывающего отклонения в соответствии с требованиями MISRA C 2023.

**Миграция кода.** Постепенно производить рефакторинг проблемных областей: макросы → inline-функции; неинициализированные переменные → явная инициализация; неатомарные глобальные переменные → атомарные типы/мьютексы там, где требуется.

**Инструментальная автоматизация.** Настроить процедуру интеграции для регулярного запуска статического анализа с отчетностью по тенденциям (новые нарушения, устраненные, продолжающиеся).

## 6. Заключение

MISRA C 2023 представляет собой значимое эволюционное обновление набора правил, нацеленное на повышение надежности, предсказуемости и проверяемости кода на языке C, особенно в задачах критичных к безопасности. Основные направления изменений: ужесточение требований по управлению неопределенным поведением, уточнение правил работы с указателями и выравниванием, более явное рассмотрение современной семантики языка (атомарность, порядок вычислений), усиление требований к документированию отклонений и улучшение интеграции с инструментами статического анализа. Эти изменения отражают опыт применения MISRA в промышленности и учитывают современные практики разработки ПО.

Практическая польза от миграции на MISRA C 2023 очевидна: снижение числа трудноуловимых дефектов, улучшение переносимости и предсказуемости поведения на разных компиляторах и архитектурах, а также повышение качества процедур аудита и управления отклонениями. Однако внедрение потребует усилий: обновления инструментов, обучения кадров и рефакторинга унаследованного кода. Поэтому организациям рекомендуется планировать поэтапную миграцию с выделенными ресурсами на адаптацию инструментов и обучение.

MISRA C 2023 - это современный и прагматичный шаг вперед в развитии правил надежного программирования на языке C. Он способствует снижению рисков, связанных с неопределенным поведением и контекстно зависимыми ошибками, делая управление широким набором уязвимостей и дефектов более формализованным и предсказуемым.

«Публикация выполнена в рамках государственного задания НИЦ «Курчатовский институт» - НИИСИ» по теме FNEF-2024-0001».

# MISRA C 2023 brief overview

**K. A. Kostiukhin, G. L. Levchenkova**

**Abstract.** The paper provides a brief overview of the main changes in the MISRA C 2023 standard compared to its previous version in 2012. Fundamental changes in the structure of the standard, new requirements and recommendations, as well as practical implications for developers of embedded and security-critical systems are considered.

**Keywords:** C language, standard, MISRA C 2023

## Литература

1. A. Homann, L. Grabinger, F. Hauser, J. Mottok. An Eye Tracking Study on MISRA C Coding Guidelines. In Proceedings of the 5th European Conference on Software Engineering Education (ECSEE '23). Association for Computing Machinery, 2023, New York, NY, USA, pp. 130–137. https://doi.org/10.1145/3593663.3593671
2. MISRA C C 2012 recommendations, https://gitlab.com/MISRA/MISRA-C/MISRA-C-2012/tools/-/blob/main/misra_c_2012__headlines_for_cppcheck%20-%20AMD1+AMD2.txt
3. MISRA C C 2023 recommendations, https://gitlab.com/MISRA/MISRA-C/MISRA-C-2012/tools/-/blob/main/misra_c_2023__headlines_for_cppcheck.txt