* [Вступление](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%B2%D1%81%D1%82%D1%83%D0%BF%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5)
* [Назначение документа](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BD%D0%B0%D0%B7%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0)
* [Цели соглашения](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%86%D0%B5%D0%BB%D0%B8_%D1%81%D0%BE%D0%B3%D0%BB%D0%B0%D1%88%D0%B5%D0%BD%D0%B8%D1%8F)
* [Абсолютные запреты](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%B0%D0%B1%D1%81%D0%BE%D0%BB%D1%8E%D1%82%D0%BD%D1%8B%D0%B5_%D0%B7%D0%B0%D0%BF%D1%80%D0%B5%D1%82%D1%8B)
* [Под вопросом](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BF%D0%BE%D0%B4_%D0%B2%D0%BE%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%BC)
* [Базовые принципы](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B5_%D0%BF%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF%D1%8B)
* [Исходные файлы](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B5_%D1%84%D0%B0%D0%B9%D0%BB%D1%8B)
  + [Ширина строки](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%88%D0%B8%D1%80%D0%B8%D0%BD%D0%B0_%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B8)
  + [Отступы](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BE%D1%82%D1%81%D1%82%D1%83%D0%BF%D1%8B)
  + [Расширения](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%80%D0%B0%D1%81%D1%88%D0%B8%D1%80%D0%B5%D0%BD%D0%B8%D1%8F)
  + [Правила написания инклудов](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BF%D1%80%D0%B0%D0%B2%D0%B8%D0%BB%D0%B0_%D0%BD%D0%B0%D0%BF%D0%B8%D1%81%D0%B0%D0%BD%D0%B8%D1%8F_%D0%B8%D0%BD%D0%BA%D0%BB%D1%83%D0%B4%D0%BE%D0%B2)
  + [Заголовок файла](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%B7%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BE%D0%BA_%D1%84%D0%B0%D0%B9%D0%BB%D0%B0)
  + [Соответствие классов и файлов](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%81%D0%BE%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D1%81%D1%82%D0%B2%D0%B8%D0%B5_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2_%D0%B8_%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2)
  + [Порядок кода в файлах](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%B0_%D0%B2_%D1%84%D0%B0%D0%B9%D0%BB%D0%B0%D1%85)
* [Соглашение об именах](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%81%D0%BE%D0%B3%D0%BB%D0%B0%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BE%D0%B1_%D0%B8%D0%BC%D0%B5%D0%BD%D0%B0%D1%85)
  + [Запреты](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%B7%D0%B0%D0%BF%D1%80%D0%B5%D1%82%D1%8B)
  + [Имена папок](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%B8%D0%BC%D0%B5%D0%BD%D0%B0_%D0%BF%D0%B0%D0%BF%D0%BE%D0%BA)
  + [Имена файлов](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%B8%D0%BC%D0%B5%D0%BD%D0%B0_%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2)
  + [Макросы и элементы enum](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BC%D0%B0%D0%BA%D1%80%D0%BE%D1%81%D1%8B_%D0%B8_%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D1%8B_enum)
  + [Пользовательские типы данных - typedef, типы структур, классы, объединения, enum](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B5_%D1%82%D0%B8%D0%BF%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_-_typedef_%D1%82%D0%B8%D0%BF%D1%8B_%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D1%8B_%D0%BE%D0%B1%D1%8A%D0%B5%D0%B4%D0%B8%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F_enum)
  + [Интерфейсы](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81%D1%8B)
  + [Обычные переменные и объекты](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BE%D0%B1%D1%8B%D1%87%D0%BD%D1%8B%D0%B5_%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B8_%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D1%8B)
  + [Булевские переменные](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%B1%D1%83%D0%BB%D0%B5%D0%B2%D1%81%D0%BA%D0%B8%D0%B5_%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5)
  + [Приватные переменные в классах](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BF%D1%80%D0%B8%D0%B2%D0%B0%D1%82%D0%BD%D1%8B%D0%B5_%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B2_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0%D1%85)
  + [const-константы](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#const-%D0%BA%D0%BE%D0%BD%D1%81%D1%82%D0%B0%D0%BD%D1%82%D1%8B)
  + [Функции и методы](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8_%D0%B8_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B)
  + [Функции и методы, возвращающие bool](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8_%D0%B8_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B_%D0%B2%D0%BE%D0%B7%D0%B2%D1%80%D0%B0%D1%89%D0%B0%D1%8E%D1%89%D0%B8%D0%B5_bool)
  + [Пространства имен](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%B0_%D0%B8%D0%BC%D0%B5%D0%BD)
  + [Шаблонные функции и классы](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8_%D0%B8_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D1%8B)
  + [Параметры шаблонов](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D1%8B_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%BE%D0%B2)
* [Типы данных](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%82%D0%B8%D0%BF%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)
  + [Стандартные типы данных](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D1%8B%D0%B5_%D1%82%D0%B8%D0%BF%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)
  + [class vs struct](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#class_vs_struct)
  + [Объявления структур, enum и union](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BE%D0%B1%D1%8A%D1%8F%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80_enum_%D0%B8_union)
  + [Указатели vs ссылки](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%83%D0%BA%D0%B0%D0%B7%D0%B0%D1%82%D0%B5%D0%BB%D0%B8_vs_%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B8)
  + [Приведение типа](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BF%D1%80%D0%B8%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5_%D1%82%D0%B8%D0%BF%D0%B0)
* [Пространства имен](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%B0_%D0%B8%D0%BC%D0%B5%D0%BD1)
  + [Анонимные пространства имен или static?](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%B0%D0%BD%D0%BE%D0%BD%D0%B8%D0%BC%D0%BD%D1%8B%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%B0_%D0%B8%D0%BC%D0%B5%D0%BD_%D0%B8%D0%BB%D0%B8_static)
  + [Глобальное пространство имен](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%B3%D0%BB%D0%BE%D0%B1%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%B8%D0%BC%D0%B5%D0%BD)
  + [using namespace](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#using_namespace)
  + [Функция в пространстве имен или статический метод класса?](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%B2_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%B5_%D0%B8%D0%BC%D0%B5%D0%BD_%D0%B8%D0%BB%D0%B8_%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0)
* [Классы](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D1%8B)
  + [Конструкторы](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BA%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D0%BE%D1%80%D1%8B)
  + [Деструкторы](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%B4%D0%B5%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D0%BE%D1%80%D1%8B)
  + [Наследование](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BD%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)
* [Юнит-тесты](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%8E%D0%BD%D0%B8%D1%82-%D1%82%D0%B5%D1%81%D1%82%D1%8B)
* [Общие рекомендации](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BE%D0%B1%D1%89%D0%B8%D0%B5_%D1%80%D0%B5%D0%BA%D0%BE%D0%BC%D0%B5%D0%BD%D0%B4%D0%B0%D1%86%D0%B8%D0%B8)
* [Форматирование](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)
  + [Классы](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D1%8B1)
  + [Строки кода](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B8_%D0%BA%D0%BE%D0%B4%D0%B0)
  + [Объявление переменных](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BE%D0%B1%D1%8A%D1%8F%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D1%85)
  + [Секции кода](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%81%D0%B5%D0%BA%D1%86%D0%B8%D0%B8_%D0%BA%D0%BE%D0%B4%D0%B0)
  + [Пробелы](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BF%D1%80%D0%BE%D0%B1%D0%B5%D0%BB%D1%8B)
  + [Конструкции](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BA%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D0%B8%D0%B8)
  + [Циклы без тела](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%86%D0%B8%D0%BA%D0%BB%D1%8B_%D0%B1%D0%B5%D0%B7_%D1%82%D0%B5%D0%BB%D0%B0)
  + [Стиль расстановки скобок](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D1%81%D1%82%D0%B8%D0%BB%D1%8C_%D1%80%D0%B0%D1%81%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B8_%D1%81%D0%BA%D0%BE%D0%B1%D0%BE%D0%BA)
  + [Комментарии](http://wiki.dep111.rtc.local/code_conventions:embedded_cpp#%D0%BA%D0%BE%D0%BC%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%80%D0%B8%D0%B8)

**Вступление**

Самое главное, что вам нужно помнить - С++ это **не С**. Если вы хорошо знаете С, то помните:

* Поведение языка может отличаться от привычного вам. Будьте готовы к сюрпризам со всех сторон!
* Писать на С++ в стиле С можно, но неэффективно.
* Вас никто не заставляет использовать сразу все возможности С++. Не хотите писать шаблоны - не пишите.
* Не забывайте, что узнавать что-то новое - это здорово!

**Назначение документа**

Документ предназначен для стандартизации основных приемов оформления кода и использования элементов языка С++.

Помните, этот документ носит предварительный характер; изменения весьма вероятны!

**Цели соглашения**

1. Повышение читаемости кода для облегчения поддержки и передачи кода между разработчиками.
2. Уменьшение вероятности ошибок при разработке.
3. Облегчение написания unit-тестов.

**Абсолютные запреты**

* Оператор GOTO
* Объявление переменных и структур как extern в **глобальном пространстве имен**
* Использование malloc (и т.п.), массивов переменной длины, new и delete в продакшен-коде
* Элементы стандартной библиотеки C++ (STL), которые используют кучу
* Исключения (exceptions)
* Множественное наследование (кроме как от интерфейсов)
* Символ табуляции
* Транслит в названиях
* using namespace в заголовочных файлах
* Стражи включения заголовочных файлов (#ifndef SUPER\_HEADER\_FILE\_H…)
* Тела циклов и ветвлений без фигурных скобок
* Извращения и неуточненное поведение (триграфы, многосимвольные литералы и т.д.)

**Под вопросом**

С++11 на данный момент (июнь 2015) поддерживается почти в полном объеме Кейлом 5. К сожалению, компиляция на лету при этом подчеркивает каждое новое ключевое слово и рисует желтый значок справа.

С другой стороны, С++11 дает несколько плюшечек, среди которых:

* constexpr
* auto
* лямбда-функции
* строгий enum
* std::array и std::function
* и еще много чего

Часть из этого можно реализовать и в С++03.

**Базовые принципы**

Фундаментальная цель этих стандартов состоит в том, чтобы поддерживать сопровождение кода. Это означает, что код должен быть читабельным, понятным, тестируемым и переносимым. Для достижения этого, необходимо следовать следующим принципам:

* Не используйте возможности языка, которых не понимаете!
* Следуйте духу стандарта. Старайтесь делать похожие вещи одинаково, даже если это не оговорено в стандарте.
* KISS
* [Python Zen](https://ru.wikipedia.org/wiki/Python#.D0.A4.D0.B8.D0.BB.D0.BE.D1.81.D0.BE.D1.84.D0.B8.D1.8F)
* Стремитесь к простоте кода. Пишите точно и ясно. Избегайте неявных и малоизвестных возможностей язык.
* Будьте последовательны. Используйте одни и те же правила везде, где можно.
* Избегайте сложных выражений. Выражения, включающие много подвыражений, трудно изучать и тестировать.
* Всякий раз при изменении существующего кода пытайтесь его модифицировать, соблюдая соглашения, описанные в данном документе. Это создает уверенность, что код может быть расширен снова через какое-то время.
* Прочтите “Чистый код” Роберт Мартин
* Спрашивайте, если есть сомнения. Если вы столкнулись с какой-то проблемой, велика вероятность, что ее уже кто-то решил. Не спешите изобретать велосипед.

**Исходные файлы**

**Ширина строки**

Вы должны стремится ограничивать ширину исходного текста 100-а символами.

**Отступы**

Символ табуляции к использованию запрещен. Выравнивание кода и текста в комментариях должно осуществляться только пробелами! Длина отступа для блока кода - 4 пробела. Препроцессорные директивы #if, #ifdef, #elif и #else тоже порождают отступ!

int test( void )

{

int a;

int b;

if ( a - b < 0 )

{

return a;

}

return b;

}

**Расширения**

Для файлов с исходным кодом на С++ используйте расширение .cpp Для файлов с исходным кодом на С - расширение .с Для заголовочных файлов - расширение .h

Keil, опираясь на расширение, самостоятельно определит язык. При необходимости, для каждого файла (или для группы файлов) можно указать опцию компилятора (например –с99 или –cpp).

**Правила написания инклудов**

Одна папка с файлами образует модуль. Все файлы .с и .срр внутри модуля должны инклудить файлы .h внутри модуля с помощью #include “file.h” - в кавычках и без путей. В таком случае модуль становится переносимым, его код не нужно менять, если вы назовете папку по-другому или передвините ее в дереве проекта.

Все системные инклуды (т.е. которые НЕ лежат в папке с проектом) инклудятся через угловые скобки - #include <stdint.h>

Все остальные инклуды (за исключением косячных стронних) инклудятся через кавычки с указанием пути, начиная от папки src - #include “UART/uart.h”. В проекте должно быть прописано как можно меньше путей (в идеале - только к src). Подъем на уровень выше, т.е. инклуды вида #include “../../blahblah.h” допускаются только в пределах одного модуля.

Обратный слеш '\' использовать запрещается, потому что он работает только в Windows. Используйте прямой слеш '/', который работает везде.

**Заголовок файла**

В начале каждого файла включайте блок комментариев на **русском** языке содержащий в себе краткое описание и назначение содержимого файла, имя и фамилию основного разработчика.

Пример:

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*Описание*

*Это пример.*

*Разработчик: Пупкин Василий*

*Заметки*

*Компилируется только под C99.*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

Текст заголовка в свободной форме.

**Соответствие классов и файлов**

Объявления публично-доступных классов располагаются в заголовочных файлах. Старайтесь следовать правилу “один класс - один заголовочный файл”; имя файла должно совпадать с именем класса. Например, объявление класса RegisterViewer должно лежать в файле register\_viewer.h.

Если класс входит в какое-то пространство имен, припишите его в название файла: enotic\_register\_viewer.h.

**Порядок кода в файлах**

**Файлы языка С**

см. [embedded\_c](http://wiki.dep111.rtc.local/code_conventions:embedded_c)

**Файлы языка С++**

Общая рекомендация: чем больше область видимости чего-то - тем ближе оно должно быть к началу файла.

Для файлов .срр порядок пока что не определен.

Для объявления классов (в заголовочных файлах): сначала public секция, потом protected (если она есть), потом private.

Соблюдайте единый порядок внутри каждой секции, например:

1. Конструктор (если он в этой секции есть)
2. Типы данных (typedef, enum, встроенные классы/структуры)
3. const-константы
4. Методы
5. Переменные

**Не допускайте многократного включения заголовочного файла**

Используйте конструкцию:

#pragma once

Не используйте стражи включений через ifdef.

**Соглашение об именах**

* Там, где это возможно, это соглашение повторяет нашу нотацию для языка С.
* Если для данного типа имен запрещены несколько заглавных букв подряд, аббревиатуры должны подстраиваться: UartGetter, а не UARTGetter.
* Избегайте транслита, используйте английский язык.
* Не используйте “наивную венгерскую нотацию”, т.е. не приписывайте ТИП переменной к ее имени. Имя переменной должно отражать ее смысл.

Неправильно:

uint32\_t ui32Counter;

Правильно:

uint32\_t msgCounter;

Исключения допустимы, например, для указателей.

**Запреты**

Стандарт языка С++ запрещает имена, которые:

* Начинаются на нижнее подчеркивание и заглавную букву - \_Variable
* Начинаются на два нижних подчеркивания - \_\_variable

Такие имена запрещены вообще, в любых пространствах имен и областях видимости.

В глобальном пространстве имен запрещены имена, начинающиеся на нижнее подчеркивание.

Во избежание оказий, не рекомендуется начинать имена с нижнего подчеркивания вообще.

**Имена папок**

Camel\_Case или snaking\_case. КАПС\_ЛОК запрещен. Имена директорий в проекте для Keil должны совпадать с названиями соответствующих папок, насколько это возможно.

**Имена файлов**

С маленькой буквы. С использованием нижнего подчеркивания.

test\_my\_mind.c

break\_my\_heart.h

Имена заголовочных файлов, содержащих объявления класса, должны повторять имя класса с учетом его пространства имен.

**namespace** animals { **class** BigRacoon{...}; }

должно лежать в файле

animals\_big\_racoon.h

**namespace** animals { **class** ISmallBehemoth{...}; }

должно лежать в файле

animals\_i\_small\_behemoth.h

**Макросы и элементы enum**

БОЛЬШИМИ\_БУКВАМИ\_С\_ИСПОЛЬЗОВАНИЕМ\_НИЖНЕГО\_ПОДЧЕРКИВАНИЯ.

#define BUFFER\_SIZE 20

**enum** Color { RED, GREEN, BLUE };

**Пользовательские типы данных - typedef, типы структур, классы, объединения, enum**

Начинаются с заглавной буквы. Каждое следующее слово в название - с заглавной (CamelCase).

typedef uint32\_t MySuperType;

typedef struct

{

...

} MySuperStruct;

union MySuperUnion

{

...

};

**Интерфейсы**

Начинаются с заглавной буквы I (ай), каждое следующее слово в название - с заглавной (CamelCase).

**class** IRegTable;

**class** IGanjubusSender;

**Обычные переменные и объекты**

Начинаются со строчной буквы. Каждое следующее слово в название - с заглавной (camelCase). В названиях желательно использовать существительные и прилагательные (но не глаголы).

int bufferSize;

Borsh borsh;

**Булевские переменные**

Должны начинаться на is или are.

bool isBufferReady;

**class** Badger

{

**private**:

bool m\_areLegsBroken;

};

**Приватные переменные в классах**

Начинаются с префикса m\_, дальше camelCase.

**class** Racoon

{

**private**:

int m\_legs;

bool m\_isReady;

}

Это позволяет избежать странностей в сеттерах и конструкторах, а так же облегчает работу с автоподстановкой.

**const-константы**

строчными\_буквами\_через\_подчеркивание.

const uint32\_t buffer\_size = 42;

**Функции и методы**

Начинаются со строчной буквы. Каждое следующее слово в название - с заглавной (camelCase).

int bufferSize;

Borsh borsh;

**Функции и методы, возвращающие bool**

Должны начинаться на is или are.

bool isBufferReady(void);

**Пространства имен**

строчными\_буквами\_через\_подчеркивание.

**namespace** ganjubus\_tasks{ ... }

**Шаблонные функции и классы**

Называются так же, как обычные

**Параметры шаблонов**

Параметры шаблонов могут быть:

* не-типы
* типы
* другие шаблоны

**template** < **typename** T, int reg\_num >

**class** Table

{};

Рекомендуемое число параметров - не больше трех.

**Параметры-типы**

Параметры-типы следует называть:

* Заглавными буквами T, U, V, если шаблон короткий или вы не хотите указать каких-то особых ограничений на тип

**template** < **typename** T >

bool signum(T val)

{

if(val > 0)

return 1;

else if(val < 0)

return -1;

else

return 0;

}

* Начиная с заглавной буквы T, далее CamelCase, если вы хотите “конкретизировать” тип

**template** < **typename** TReg >

**class** Table

{};

**Параметры не-типы**

Параметры не-типы могут быть численными арифметическими константами, указателями или ссылками. Их можно называть как обычные численные константы или переменные (маленькими\_буквами\_через\_подчеркивание или camelCase).

**Параметры шаблоны**

Никак специально не выделяются.

**Типы данных**

**Стандартные типы данных**

Используются переопределения из стандартной библиотеки (файл stdint.h свой для каждой платформы/среды/компилятора). Они имеют вид: uint32\_t - беззнаковый целый размером 32 бита int8\_t - знаковый целый размеров 8 бит

**Тип bool**

В С++ есть “родной” тип bool. Используйте его.

**Область видимости типов данных**

Если тип данных должен использоваться только в файле реализации, то он должен быть объявлен в файле реализации. Если тип данных глобален, то он должен быть помещен в заголовочный файл модуля.

Cтандарт С++ запрещает объявлять классы с одинаковыми именами (но разными телами) даже локально в разных срр-файлах! В Кейле это вызывает ошибку линковки. Что делать? Использовать разные имена или использовать анонимные пространства имен.

**class vs struct**

Для компилятора единственная разница - в структуре все поля по-умолчанию public, в классе - private.

Рекомендации: используйте структуры для простых наборов переменных, желательно, без методов. Например, набор коэффициентов ПИД-регулятора, значения широты и долготы и т.п.

Для типов со сложным поведением используйте классы.

**Объявления структур, enum и union**

Нет необходимости писать как в С:

typedef struct {} Name;

Можно писать просто

struct Name {};

**Указатели vs ссылки**

При передаче параметров в функции и классы, старайтесь использовать ссылки - это делает код чище (не нужно писать &). Что вы храните внутри класса (указатель или ссылку) - не так важно.

Если размер массива известен на этапе компиляции, его можно указать в типе ссылки и тем самым сделать объявление более строгим. При возможности, используйте этот прием:

void foo(int (& arr)[5])

**Приведение типа**

“C-style cast” - это когда вы пишете тип в скобках. Например:

int a = (int)5.5;

Таким образом можно привести **любой тип в любой другой**.

В С++ существует целых 4 оператора приведения типа, в дополнении к C-style cast:

* static\_cast
* const\_cast
* dynamic\_cast
* reinterpret\_cast

Эти операторы имеют более узкую функциональность и, теоретически, позволяют писать более конкретный код. А еще, их проще искать поиском или глазами. Однако писать их дольше, читать их труднее, в некоторых случаях требуется делать больше одного каста за раз.

Т.к. в embedded требуются частые приведения между примитивными типами, наш стандарт не запрещает и не поощрает использование какого-то определенного способа кастования.

Будьте аккуратны, если приводите типы между классом-родителем и классом-потомком! При работе с составными типами лучше использовать С++-ные операторы приведения типа.

**Пространства имен**

**Анонимные пространства имен или static?**

Тип данных нельзя объявить как static, в остальном функционал аналогичен (вроде бы). Используйте то или другое по собственному желанию.

Отметим, что namespace нужно написать один раз, а static придется приписывать к каждой переменной или функции.

**Глобальное пространство имен**

Его следует использовать как можно меньше. Функции в глобальном пространстве имен можно вызывать явно, чтобы исключить конфликты и неоднозначности:

::foo();

**using namespace**

Старайтесь не использовать using namespace, т.к. оно провоцирует трудноразрешимые конфликты имен. Максимально ограничиваете область видимости этой директивы.

Не используйте using namespace в заголовочных файлах!

**Функция в пространстве имен или статический метод класса?**

Общая рекомендация: классы предназначены для создания объектов. Если вам нужно просто несколько функций, объединенных некой общей целью, используйте пространство имен. Если у набора функций могут появиться общие данные, используйте класс.

**Классы**

Лучше не вызывайте виртуальные функции в конструкторах и деструкторах, т.к. при этом вызывается версия функции из родительского класса. Плохо понятно, зачем нам это может быть нужно.

**Конструкторы**

**Конструктор с одним параметром**

**Важно**: конструктор с одним параметром может быть использован для неявного приведения типа! Например:

**class** BigBuffer

{

**public**:

BigBuffer(int initialValue)

{

;

}

};

void Foo(const BigBuffer & o);

Foo(3); *// <-- 3 неявно приводится к типу BigBuffer, т.к. у него есть конструктор с одним параметром типа int*

Всегда объявляйте конструктор с одним параметром как exlicit! Тогда будет ошибка компиляции при попытке неявного приведения типа.

**Как правильно инициализировать члены класса в конструкторе**

Используйте “initializer list”, а не код в теле конструктора. Правильно:

MyClass::MyClass( float f, bool b ) :

m\_float( f ),

m\_bool( b )

{}

Неправильно:

MyClass::MyClass( float f, bool b)

{

m\_float = f;

m\_bool = b;

}

Причины:

* Константы, статические переменные класса, элементы базовых классов, элементы без конструктора по-умолчанию и ссылки можно инициализировать только первым способом
* При втором способе может происходит лишняя запись нуля в каждый член класса

**Деструкторы**

**Ремарка**: даже если вы не используете динамическую память и ваш деструктор ничего не делает, если объявить его как virtual, Кейл сгенирирует код для сбора статистики. Этот код вызовется только после выхода из main (т.е. никогда), но он захочет использовать кучу.

Если в вашем проекте размер кучи установлен в 0 - программа зависнет до main.

Не забывайте объявлять деструктор как virtual, если:

* Вы используете виртуальные функции в данном классе
* Хотя бы один объект этого класса уничтожается во время работы программы

**Наследование**

* Множественное наследование от не-интерфейсов запрещено! И скорее всего, оно вам не нужно.
* Старайтесь не делать слишком сложную иерархию наследования; двух поколений достаточно в подавляющем большинстве случаев.
* Старайтесь наследоваться от интерфейсов, а не от классов. Это облегчает написание юнит-тестов и снижает связность кода, хотя и делает вызовы методов медленнее.
* Если вы используете шаблонный класс, пусть он реализует какой-то интерфейс и храните ссылку на интерфейс. Не пытайтесь хранить ссылку на конкретный инстанс шаблона, если речь не идет о стандартной коллекции.
* Три раза подумайте, прежде чем делать приведение типа к базовому классу, если это не интерфейс.
* Не используйте наследование времени компиляции (CRTP).
* Не переопределяйте невиртуальные методы базовых классов.

**Юнит-тесты**

Рекомендованный фреймворк - наш велосипедный umbaCppTest. Но вы вольны использовать любой другой фреймворк, если он работает.

Самый простой способ писать тестируемый код - это использовать TDD, т.е. сначала писать тест, а потом тестируемый код.

Как сделать код тестопригодным:

* Сначала продумайте интерфейс для модуля в целом, не кидайтесь писать функции сразу.
* Приватный код тестировать напрямую не нужно, модуль должен работать как черный ящик.
* Чтобы протестировать модуль, тесту нужно будет подать ему какое-то воздействие на вход и проверить реакцию на выходе. Подумайте, насколько легко это сделать?
* Самый низкий уровень кода, который непосредственно работает с периферией микроконтроллера, допускается не покрывать юнит-тестами, а проверять вручную. Иначе тесту придется, фактически, эмулировать поведение периферии.

**Общие рекомендации**

* Старайтесь не переопределять операторы для ваших классов или хотя бы не меняйте их семантику.
* Не делайте переменные класса публичными; используйте геттер/сеттер. Публичные константы допустимы.
* Старайтесь не писать определения длинных и сложных методов в заголовочных файлах. Допускаются геттеры-сеттеры, простые конструкторы и т.п.
* Будьте очень осторожны с ключевым словом friend
* Если метод ничего не меняет в классе - объявите его как const.
* Вообще, все, что может быть const, должно быть const!
* Будьте осторожны при объявлении static переменных. Если вы объявляете такую переменную в методе, то переменная будет *одна* для всех объектов этого класса!
* Старайтесь не использовать макросы для определения простых констант, используйте строгий enum или const.
* Область видимости должна быть настолько мала, насколько это возможно (но без перегибов).

**Форматирование**

**Классы**

Спецификаторы доступа (public, protected и private) внутри класса можно выделять отсутствием отступа:

**class** SuperClass

{

**public**:

const int a;

const int size;

...

**private**:

static const int size\_max = 5;

void killAllHumans(void);

};

Так секции лучше выделяются визуально.

**Строки кода**

Используйте одну точку с запятой на одной строке (кроме for, lol)

Пример:

Правильно

firstStep = 0;

setPosition();

Неправильно

firstStep = 0; setPosition();

**Объявление переменных**

Не объявляйте несколько переменных на одной строке, даже если у них одинаковый тип. И особенно, если у них разный тип!

Правильно

int8\_t a = 0;

int8\_t b = 3;

Неправильно

int8\_t a, b;

Очень неправильно

int8\_t \* a, b; *// внезапно, тут у переменных тип будет разный*

**Секции кода**

Отделяйте логические секции кода пустыми строками или комментариями

**Пробелы**

**Между именем функции и круглыми скобками после него не должно быть пробелов при вызове функции**

dispInit();

**После открывающей скобки (квадратной и круглой) и перед закрывающей пробелы - по желанию**

setRobotPar( bar );

buffer[ 100 ];

**Один пробел необходим после каждой запятой для разделения аргументов**

setRobotPar( speed, position );

**Унарные операторы пишутся без пробела между ними и их операндом**

~bits;

++i;

\*ptr; *// разыменование*

sizeof( x );

**Унарные операторы \*, & и && при объявлении указателя или ссылки пишутся через два пробела**

int \* a;

char & b = \*a;

**Бинарные операторы и тернарный оператор пишутся как минимум с одним пробелом между ними и операндом**

c1 = c2;

a + b = c;

a < b;

x << 8;

**Один пробел после точки с запятой**

for ( i = 0; i < 69; ++i )

{

}

**Ключевые слова if, else, while, for, switch, return сопровождаются одним пробелом**

if ( a > b )

while ( x > 0 )

for ( i = 0; i < 10; ++i )

switch ( x )

return ( y )

**Каждая функция состоящая более чем из 3-х строк снабжается заголовком на русском языке**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*Описание:*

*Аргументы:*

*Возврат:*

*Замечания:*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

**Конструкции**

Использовать фигурные скобки – всегда

if ( a > b )

{

a = b;

}

**Циклы без тела**

Всегда снабжаются фигурными скобками (вид - на вкус пользователя)

while ( 1 )

{

}

или

while(1)

{;}

**Стиль расстановки скобок**

**Фигурные скобки**

*Не используем стиль оформления K&R*

Не правильно

if ( ){

}

*Правильно - стиль Allman*

if ( )

{

}

**Круглые скобки**

При необходимости (т.е. при превышении предельной длины строки) делаем вот так, дабы визуально отличить это от блока в фигурных скобках:

TaskHandle\_t createTask( const char \* const name,

unsigned short stackDepth,

void \* params,

UBaseType\_t prio )

**Круглые скобки в STRONG\_ENUM**

Оба варианта допустимы:

STRONG\_ENUM(Ind3cRequestId, GET\_VALUES,

GET\_VER,

GET\_ZERO,

SET\_ZERO,

SET\_ADR,

SET\_BAUD,

REQ\_MAX );

STRONG\_ENUM

(

Ind3cRequestId,

GET\_VALUES,

GET\_VER,

GET\_ZERO,

SET\_ZERO,

SET\_ADR,

SET\_BAUD,

REQ\_MAX

);

**Угловые скобки в шаблонах**

Старайтесь все-таки укладываться в длину строки! Но если нет, то как с фигурными:

**template**

<

**typename** T,

**typename** U

>

void foo( T

<

U

> );

**Комментарии**

Для комментариев внутри кода используйте однострочные комментарии. Блочные комментарии используются только для блоков перед функциями и т.п. Комментарии размещаются непосредственно над комментируемым участком кода или справа от него. Язык – русский.

*// массив для приема сообщения*

uint8\_t buffer[5];

int8\_t \* ptr; *//указатель на массив для передачи данных*