Особенности применения языков программирования Си и C++ при кодировании для систем ответственного назначения

С.Н. Зыль (ООО «СВД Встраиваемые Системы»)

Разработчики ПО реального времени хорошо знают, насколько эффективны языки Си и C++. Общеизвестна также и обратная сторона этой эффективности: наличие конструкций, которые при синтаксической корректности могут вызвать некорректное поведение программы. Мировой промышленностью накоплен колоссальный опыт по применению этих языков, который отражен в международных стандартах по функциональной безопасности. Рассмотрен подход, который предлагается этими стандартами и состоит в применении безопасных подмножеств языков программирования.

Ключевые слова: языки программирования, функциональная безопасность, управление рисками.

Введение

Крис Хоббс в своей фундаментальной работе «Embedded Software Development for Safety-Critical Systems» [1] приводит распространенное среди программистов мнение о том, что накладывать ограничения на языки программирования, это как заказывать Пикассо создание картины, при этом запрещать ему использовать желтый цвет. Тем не менее, сложно представить себе предприятие, которое серьезно занимается разработкой программного обеспечения для систем ответственного назначения, у которого в писанных или неписанных стандартах не было бы указаний о том, какой язык программирования применять и, мало того, как его применять.

Рассмотрение предмета стоит начать с указаний ключевого международного стандарта по функциональной безопасности — IEC 61508. Этот стандарт имеет семь частей и официальный перевод на русский язык, имеющий статус государственного стандарта РФ. Для обсуждаемой темы нас, в первую очередь, интересуют часть 3 «Требования к ПО», которая содержит необходимые базовые принципы, и часть 7 «Методы и средства», которая предлагает конкретные рекомендации. Далее мы рассмотрим стандарты MISRA C:2012 и MISRA C++:2008, которые имеют для программистов самое непосредственное практическое значение.

Программирование как инструмент управления рисками

Ключевые принципы базового международного стандарта по функциональной безопасности IEC 61508, в частности, V-модель разработки ПО, хорошо известны российским программистам. Этот документ содержит структурированные и систематизированные рекомендации о применении различных средств и методов на различных стадиях жизненного цикла разработки. Написание программного кода, то есть кодирование, является одной из стадий. Впрочем, эту стадию физически невозможно исключить даже при самом недетерминированном процессе разработки ПО.

Прежде всего, нужно помнить, что при всей насыщенности технической информацией IEC 61508 относится к области *управления рисками*. Любой разработчик систем, критичных с точки зрения функциональной безопасности, должен осознавать риск, создаваемый разрабатываемой системой обществу, — *исходный риск*. Также необходимо знать, какой уровень риска приемлем для общества, то есть каков *допустимый риск*. Допустимый риск формулируется обществом с помощью нормативных документов (законов, приказов, стандартов и т.д.), издаваемых уполномоченными органами. Разница между исходным и допустимым риском составляет минимальное значение *необходимого снижения риска*. Стандарт IEC 61508 является инструментом, позволяющим определить, что именно специалисты в вычислительной технике должны сделать для получения определенного коэффициента снижения риска.

Разумеется, меры по снижению риска имеют свою цену, выражаемую в затратах рабочего времени на их реализацию. А рабочее время каждого сотрудника имеет вполне конкретное денежное выражение.

Например, на одной из конференций по ПО для авионики была озвучена история про некоторую систему управления, разработанную крупной международной компанией. Продукт должен был соответствовать требованиям регуляторов 66 стран, поэтому только в 2015 г. аудиты предприятия-разработчика и продукта проводили свыше 100 организаций. За год на обеспечение проведения этих аудитов разработчик затратил 3500 человеко-дней.

Разумеется, чем выше требования к снижению риска, тем выше затраты на выполнение соответствующих мероприятий. Поэтому IEC 61508 делит системы, обеспечивающие функциональную безопасность, на четыре иерархические категории, именуемые *уровнями полноты безопасности*. Самый высокий уровень —

УПБ4, самый низкий — УПБ1. В английском оригинале используется термин *safety integrity level* (SIL).

Критерии выбора языка программирования

Приложение А седьмого раздела IEC 61508 содержит весьма информативное руководство по выбору методов и средств, а приложение С, в свою очередь, содержит обзор методов и средств с их кратким описанием и ссылками на источники информации о них.

Указания по выбору формулируются просто: для каждого уровня и средства задается обозначение. Эти обозначения называют "рекомендациями". Самыми жесткими рекомендациями являются HR (настоятельно рекомендуется) и NR (категорически не рекомендуется).

Разумеется, для разработки ПО систем ответственного назначения существует ряд специализированных языков программирования, таких как, например, Ada, D, FORTRAN 77 или RUST, в которых обеспечивается строгая типизация данных и решены другие важные вопросы, связанные с кодированием. Все они имеют свои ниши и интересны с инженерной точки зрения. Но в данной статье будет рассмотрен язык, на котором на самом деле чаще всего пишут программы для систем ответственного назначения, — Си. А коль скоро такие объектно-ориентированные технологии, как Qt от компании Digia и Mesa3D, завоевывают все большую популярность среди разработчиков систем реального времени, то нельзя обойти вниманием и вопросы применения C++.

Что же об этих языках говорит МЭК 61508?

Для высоких уровней полноты безопасности — УПБ3 и УПБ4 — для языков C, C++ и Java задана рекомендация NR. Однако для C и C++, как сказано в стандарте, «с подмножеством и стандартом кодирования, а также использование инструментов статического анализа», задана рекомендация HR. В то же время, например, для Java, даже для подмножества с выключенной или детерминированной сборкой мусора, для уровней УПБ3 и УПБ4 указана рекомендация NR.

Отметим, что на современном этапе развития средств и методов анализа и верификации вычислительной техники максимальным уровнем полноты безопасности, достижимым программными методами, является УПБ3. Уровень УПБ4 не достижим без аппаратных средств.

«Подмножества языка» нужны для исключения потенциально опасных конструкций, обеспечения читабельности кода и его пригодности для обработки программами статического анализа.

Давайте разберемся, о каких подмножествах и стандартах кодирования для языков C и C++ настоятельно рекомендуемых для наиболее ответственных систем, говорит МЭК 61508.

Язык Си

Наиболее авторитетным документом, определяющим подмножество языка Си является стандарт, известный как MISRA C. На основе него или, по крайней мере, под его влиянием разрабатываются стандарты безопасного кодирования различных организаций, например, стандарт кодирования компании JPL [2].

Во вводной части MISRA C приводит причины популярности языка Си. Назовем некоторые из них:

— программы, написанные на языке Си, могут быть скомпилированы в эффективный машинный код;

— существует готовые компиляторы для самых разных аппаратных архитектур;

— язык Си поддерживается значительным количеством инструментов статического анализа и тестирования;

— существует международный стандарт на язык Си;

— накоплен большой опыт применения языка Си при разработке систем ответственного назначения.

В то же время разработчики MISRA C констатируют недостатки языка Си. В их числе недостаточно точные определения языка, позволяющие разработчикам компиляторов по-разному реализовывать некоторый функционал, а также возможность внесения ошибок, которые не являются таковыми с точки зрения компилятора — классическая проблема замены знака присвоения на знак равенства или наоборот. Также в качестве недостатка, являющегося обратной стороной эффективной компиляции кода, указывается отсутствие контроля ошибок времени исполнения, что требует от программиста явного контроля ошибок в программе.

MISRA C допускает использование как C99, так и C90. Рекомендации касательно стандарта C11 отсутствуют только по причине того, что на момент завершения работы над текстом MISRA C рабочая группа ISO/IEC еще не успела одобрить новую версию языка Си, а ссылаться на неутвержденный стандарт было бы моветоном. Поэтому, разумеется, C11 вполне может применяться.

Большое внимание стандарт придает выбору непосредственно инструментальных средств: компиляторов и статических анализаторов. При этом весьма неслучайно упоминаются стандарты IEC 61508, ISO 26262 и DO-178C. Если в этот список добавить ещё упоминаемые в MISRA C стандарты EN 50128 и IEC 62304, которые касаются жизненного цикла разработки ПО, то круг замкнется. Все остальные стандарты, относящиеся к функциональной безопасности ПО, содержат уточнения или специализированные варианты тех принципов и подходов, которые описаны в перечисленных документах. При этом автор не отрицает необходимости изучать другие документы — безусловно, такие стандарты, как, например, IEC 60880 очень важны с точки зрения применения тех или иных средств и методов к решению определенного класса задач.

Указания MISRA C бывают двух типов: правила и директивы.

Науки - это хорошо организованные языки в той же мере, в какой языки - это еще не разработанные науки.
Мишель Фуко

Правилами называют такие указания, соответствие которым можно проверить путем анализа исходного текста без каких-либо иных источников информации. Статические анализаторы должны уметь строго проверять выполнение правил. К директивам же относятся такие указания, которые не могут быть сформулированы так строго, как правила. Для анализа соответствия кода директиве одного только кода недостаточно: может потребоваться изучение проектной документации. Статические анализаторы могут помогать в ходе проверок, но результаты их работы нужно корректно интерпретировать.

Все указания — правила или директивы — категорируются по одной из трех степеней важности: обязательно, требуется и рекомендовано.

Если код, написанный на языке Си, заявлен как соответствующий стандарту MISRA C, то он должен без каких-либо оговорок или исключений удовлетворять всем обязательным указаниям стандарта.

Аналогичным образом Си-код, заявленный как соответствующий MISRA C, должен удовлетворять всем «требуемым» указаниям стандарта. Разница состоит в том, что для указаний этой степени допускаются отклонения, каждое из которых должно быть четко задокументировано. Пример документального оформления отклонения приведен в одном из приложений стандарта. Для некоторых проектов «требуемые» указания по решению заказчика или разработчика могут приравниваться к «обязательным».

И, наконец, «рекомендованные» указания могут нарушаться без документально оформленных объяснений. Однако следует помнить, что эти указания были выработаны на основе колоссального опыта, поэтому игнорировать их не стоит ни в коем случае. Стандарт *советует* оформлять документ, объясняющий причину невыполнения рекомендаций. Мало того, для некоторых проектов «рекомендуемые» указания по решению заказчика или разработчика могут приравниваться к «требуемым» или даже «обязательным».

Правила, независимо от степени их важности, могут быть разрешимыми или не разрешимыми. К разрешимым правилам относятся такие, для которых статический анализатор может точно сказать, соответствует код правилу или нет. Но бывают конструкции, для которых, чтобы сделать такой вывод, нужно иметь данные, которые могут быть известны только во время исполнения кода. В таком случае правило считается неразрешимым, а мнение статического анализатора о соответствии или несоответствии кода тому или иному неразрешимому правилу — не достоверным. Поэтому для определения соответствия понадобится динамический анализатор.

Ну и, наконец, последняя классификация: по масштабу анализа правила делятся на модульные и системные. Модульными называют такие правили, для соответствия которым достаточно проверять единицу компиляции, то есть отдельный файл. Как не сложно догадаться, для проверки соответствия правилу масштаба системы необходимо анализировать две и более единицы компиляции, а чаще всего весь исходный текст проекта.

Правила по области их применения разбиты на 22 группы, а директивы на четыре группы. Есть указания, которые относятся к типам данных, к работе с указателями, к операторам управления, к директивам препроцессора и т. д.

Пример «требуемого» правила, входящего в группу «Стандартные библиотеки», разрешимого, масштаба единицы компиляции: функции выделения и освобождения памяти, определенные в заголовочном файле <stdlib.h> стандартной библиотеки Си, использоваться не должны. Назначение это правила достаточно очевидное: сократить такие распространенные ошибки программирования, как утечка памяти, освобождение невыделенной памяти, доступ к выделенной памяти до инициализации и т. п. Допустим, все-таки необходимо использовать «кучу». Поскольку правило «требуемое», а не «обязательное», то его можно «нарушить», задокументировав обоснование, почему это сделано. Но в таком случае вступают в силу обязательные правила из группы «Ресурсы», нерешаемые, масштаба системы:

— все ресурсы, выделенные динамически с помощью стандартной библиотеки языка Си, должны быть явно освобождены;

— область памяти должна освобождаться, только если она выделена посредством функций стандартной библиотеки Си.

Таким образом, второе из указанных правил в какой-то мере сужает применение функции *free()* по сравнению, например с функцией *malloc()*.

В заключение этого краткого обзора стандарта MISRA C подчеркнем, что изучение этого документа обязательно для любого программиста, который использует язык Си при разработке приложений ответственного назначения. Причем независимо от того, внедрен этот стандарт в организации или нет. Как минимум он позволит углубить знания Си и разобраться во многих нюансах этого языка. Поскольку для чтения этого документа нужна персональная или корпоративная лицензия, можно обратить внимание на разработанные под его влиянием правила кодирования компании JPL [2], и краткий, но эффективный и достаточно знаменитый набор десяти правил Жерара Хольцмана [3].

Язык C++

Стандарт ГОСТ Р МЭК 61508-7-2012 в приложении G коротко определяет (рекомендация HR для УПБ3 и УПБ4) базовые принципы написания объ-

ектно-ориентированного кода независимо от языка программирования, например:

— наследование допускается только с целью уточнения базового класса;

— глубина наследования должна быть ограничена;

— множественное наследование допускается только для интерфейсных классов;

— должен осуществляться контроль переопределения методов и операций.

Стандарт настоятельно рекомендует (HR) использовать апробированные паттерны (шаблоны) проектирования и программные каркасы (фреймворки). И действительно, каркасы и автоматически сгенерированный код, вероятно, являются основными путями применения языка C++ в системах ответственного назначения. С конца 1990-х гг. росла популярность фреймворков с генерацией кода из UML-моделей, которые удобно использовать для реализации алгоритмов, основанных на конечных автоматах. Затем, когда мощность встраиваемых систем позволила использовать богатые UX, передовые коммуникационные и другие технологии, широкое распространение получил также фреймворк Qt. Паттернам для разных этапов проектирования посвящена, например работа [4]. В качестве примеров фреймворков можно привести OXF от IBM Rational и RXF от Willert Software Tools.

Как бы то ни было, The Motor Industry Software Reliability Association не осталась в стороне от процесса и внесла свой вклад в разработку подходов к безопасному кодированию на языке C++, выпустив стандарт MISRA C++:2008. Несмотря на то, что уже несколько лет назад был принят стандарт C++11 (ISO/IEC 14882:2011 Information technology — Programming languages — C++.), MISRA C++ все еще основан на C++03, поскольку на момент утверждения это была действующая редакция. MISRA C++ прямо заявляет, что разработчики документа не имеют намерения продвигать язык C++. Их целью является рекомендованное IEC 61508 определение подмножества C++, требуемого для различных УПБ при разработке ПО ответственного назначения.

Еще раз отметим, что C++ особенно актуален при использовании фреймворков совместно с автоматической генерацией кода из моделей того или иного типа. Основными целями модельно-ориентированной разработки является повышение уровня абстракции (а следовательно — продуктивности) разработки сложных приложений, а также сокращение ошибок за счет автоматизации генерации и анализа кода.

Заключение

И наконец необходимо сказать несколько слов о статическом анализе кода, написанного на языках Си и C++. Собственно говоря, одной из задач соответствующих стандартов MISRA было предоставление таких правил, которые были бы пригодны для статического анализа в автоматическом режиме. Выбор подходящих анализаторов, с точки зрения стандартов MISRA, практически не отделим от выбора компиляторов.

Существует немалое число как свободно распространяемых, так и коммерческих статических анализаторов. Мало того, некоторые компиляторы имеют встроенные возможности по анализу выполнения ограничительных правил.

Среди некоммерческих продуктов выделим популярную утилиту cppcheck, по применению которой можно найти много полезной информации в открытом доступе, например статья на ресурсе Хабрахабр. Что касается анализа кода на соответствие MISRA C и MISRA C++, то список возможных инструментов можно найти в соответствующих статьях Википедии.

Список литературы

1. *Chris Hobbs.* Embedded Software Development for Safety-Critical Systems. CRC Press, 2016.
2. JPL Institutional Coding Standard for the C Programming Language, 2009. URL: http://lars-lab.jpl.nasa.gov/JPL_Coding_Standard_C.pdf (дата обращения 07.07.2016).
3. *Gerard J. Holzmann.* The Power of 10: Rules for Developing Safety-Critical Code. IEEE Computer Society, 39 (6): p.p. 95–99, 2006. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1642624 (дата обращения 07.07.2016).
4. *Bruce Powel Douglass.* Real-Time Design Patterns: robust scalable architecture for Real-time systems. Addison-Wesley, 2003.

Зыль Сергей Николаевич *— канд. техн. наук, технический директор ООО «СВД Встраиваемые Системы».*
Контактный телефон (812) 346-89-56.
E-mail: s.zyl@kpda.ru