стандартный нестандартный приплюснутый си

крис касперски аргентинский болотный бобер nezumi el raton ака жирный хочемяк ибн мыщъх

любой дурак создаст аппарат, которым может управлять только гений, но только гений создает аппарат, с которым управится любой дурак

реверанс в сторону плюсов

введение

Священные войны вокруг языков программирования идут давно, но все как-то мимо писсуара и совсем не в тему. Обычный тезис: "покажите мне пример, который нельзя реализовать на моем любимом XYZ и тогда я съем свою тюбетейку". Ага! Еще козырек намазать маслом, чтобы было легче глотать!

Теоретически, если задачу можно решить на Машине Тьюринга, ее можно запрограммировать на любом существующем языке. Весь вопрос в том: за какое время, какой ценой и с какой эффективностью. Чистый язык сам по себе никому не интересен. Для полноценной работы, программисту требуется средства разработки: трансляторы, линкеры, верификаторы, IDE, отладчики, библиотеки и т. д. и т. п., причем желательно, чтобы бы трансляторов было более одного.

Основная разница между Си++ и Бейском вовсе не в том, что трансляторы Бейсика генерируют тормозной код. Си++ в первую очередь отличается от Бейсика тем, что он никому не принадлежит. Есть открытый Стандарт и десятки аттестованных компиляторов, генерирующих стандартный промежуточный код, легко интегрируемый в любой проект, написанный, например, на Паскале или Ассемблере. Это не только упрощает перенос на другие платформы (LINUX, Palm OS), но и стабилизирует обстановку на рынке.

Язык Си++ нельзя просто взять и "свернуть", как Microsoft свернула Visual Basic, забив на совместимость с ранее написанным кодом. Десятки тысяч программистов оказались буквально выброшенными на улицу. Переделывать отлаженный код под .NET – муторно, сложно и нет никакой гарантии, что через несколько лет Microsoft не повторит свой маразм, а переучиваться под что-то другое слишком поздно. Как показывает практика, Бейсик необратимо калечит образ мышления программиста, особенно если это Visual. Кстати говоря, у Бейсика тоже есть свой Стандарт, как Европейский, так и Американский, однако, ее возможности настолько скромны, что даже самая непривередливая девушка не возьмет его замуж.

две стороны одной монеты — стандарты и реализации

Выбирая язык, вы выбираете судьбу и чтобы эта судьба не зависела от воли левой пятки Microsoft или Borland необходимо писать так, чтобы программа транслировалась любым (или хотя бы несколькими) независимыми компиляторами. Сказать намного проще, чем сделать! Чистые компиляторы сейчас не в моде. Молодое племя программистов с трудом отличает язык от IDE, прочно подсаживаясь на иглу "Мастеров" и прочих растительных заграничных штучек. Можно долго спорить: что больше в Мастерах: пользы или вреда, несомненно одно — человек, привыкший к Microsoft Visual C++, переходит на "правильные" трансляторы типа GCC с большим трудом, если переходит вообще. А ведь, казалось бы, и то, и другое — компиляторы одного и того же языка — языка Си++.

Вот тут-то мы и подходим к самому интересному. Чистый компилятор стандартного языка не интересен так же, как неинтересен "чистый" язык, поскольку возможности ввода/вывода (если это можно так назвать) очень ограничены, а стандартные библиотеки совершенно непригодны для создания программ с графическим интерфейсом. Механизмы взаимодействия с операционной системой отсутствуют как класс и даже такую простую операцию как выдвижение CD-ROM каретки стандартными средствами осуществить невозможно! Вот и приходится использовать готовые компоненты, заточенные под конкретный компилятор, и нестандартные языковые расширения, привязывающие программиста к поставщику. И хотя существует множество открытых библиотек, написанных на стандартном Си++, и позволяющих создавать переносимые графические приложения, компилируемые любым компилятором, но... как же они тормозят! Взять хотя бы Горящего Лиса, к примеру, и сравнить его с Оперой.

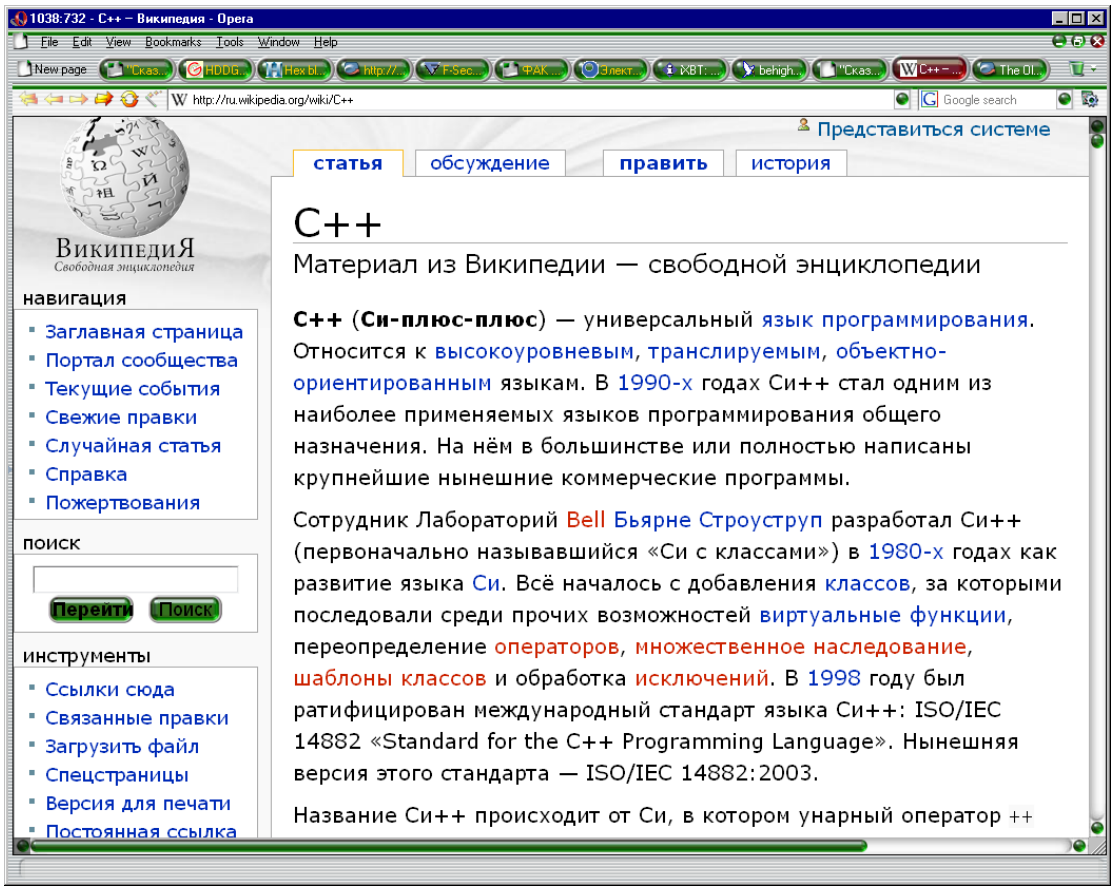

Рисунок 1 Opera – самый правильный браузер (не считая моего любимого Рыся, конечно), написанный на смеси оптимизированного Си/Си++ и заметно обгоняющего Лиса, реализующего графический интерфейс через переносимые библиотеки

Решение проблемы в общем виде практически всегда проигрывает частному случаю. Это — инженерный закон. Нестандартные языковые средства ускоряют процесс разработки в несколько раз и отказываться от них там, где переносимость не требуется — глупо и невежественно. Подавляющее большинство программистов руководствуется отнюдь не Стандартом (о существовании которого многие из них даже и не догадываются), а документацией на конкретный компилятор или даже популярными книжками из серии "Microsoft Visual C++ для полный дэбилов" и это, в общем-то, правильно.

Вопреки распространенному заблуждению, Стандарт пишется не для программистов, а для разработчиков компиляторов. Изучать приплюснутый Си по Стандарту ни в коем случае нельзя. И не потому, что он написан заумным языком, в котором путаются даже профессионалы, и даже не из-за обилия фраз "неопределенно, зависит от конкретной реализации". Камень преткновения в том, что ни один из компиляторов не поддерживает стандарт на 100%! Реально можно использовать только базовые языковые средства, составляющие ядро Си++, а все остальное это сплошное "шаг влево и компилятор разваливает программу без предупреждения".

Проходит много лет, прежде чем принятые новой версией Стандарта возможности приобретают реальную поддержку среди компиляторов. Трансляция программы — очень сложная, можно сказать, магическая штука, намного более сложная, чем это кажется со стороны. Некоторое представление о глубине проблемы дает статья "Редкая профессия" Евгения Зуева (http://www.pcmag.ru/archive/9705s/05s979.asp), рассказывающая о том, как три российских программиста разрабатывали Си++ компилятор и с каким грабли они воевали.

Пожалейте компилятор! Не насилуйте его навороченными конструкциями, почерпнутыми из новейшей редакции Стандарта и до сих пор еще неотработанные. Этим вы существенно ограничиваете круг компиляторов, пережевывающих вашу программу и создаете все условия для появления трудноуловимых ошибок, генерируемых самим компилятором. Правило номер один гласит — не греши на компилятор и прежде всего ищи ошибку у себя,

однако, из этого вовсе не следует, что компилятор никогда не ошибается. Компиляторы содержат поистине гигантское количество ошибок и buglist'ы обычно бывают очень и очень внушительны.

Процесс программирования на Си++ нередко сравнивают с хождением по минному полю. Виною тому — чрезмерная сложность языка, который с возрастом становится все сложнее и сложнее. Поговаривают даже о скорой кончине Си++ и хотя слухи о его смерти сильно преувеличены, симптомы загнивания и деградации на лице. Си++ не решил тех проблем, которые на него возлагали, программирование не стало ни проще, ни эффективнее, количество ошибок ничуть не уменьшилась, удачные примеры повторного использования кода (о котором трубят все поклонники Си++) можно пересчитать по пальцам одной руки… При этом, совокупная себестоимость программирования существенно возросла: достаточно просто взглянуть на зарплату Си++ программистов, цену на средства разработки и затраты на процесс обучения и освоения языка.

Программисты, одинакового хорошо владеющими двумя языками (Си Си++), неоднократно замечали, что для 99% проектов 99% возможностей Си++ попросту не нужны! Взять хотя бы классический пример. Начинающие программисты убеждены, что форма записи "a = b + c" лучше, элегантнее и выразительнее, чем "a = add(b,c)", но это всего лишь заблуждение (по молодости и не такое случается)! Первая запись скрывает логику программы, делая алгоритм неочевидным и заставляя программиста постоянно вспоминать: был или не был перекрыт оператор сложения, какие побочные эффекты он имеет, как реализован и т. д. Стоп! Тут кто неожиданного говорит, что "необходимо программировать так, чтобы не было побочных эффектов". Ха! И как это так можно запрограммировать?! Даже если нужно "сложить" всего две строки, то без побочных эффектов дело уже не обходится и приходится выделять память, а выделить ее можно разными путями, которые должны быть описаны в документации на перекрытый оператор сложения. Следовательно, удобство — чисто внешнее, к тому же в ряде случаев намного полезнее функция, которая не выделяет память, а берет ее из первой строки, что делает реализацию оператора "+" либо невозможной, либо нелогичной.

Это совсем не призыв к отказу от плюсов! Это призыв к осмотрительности, осмысленности и осторожности. Программирование — это инженерная дисциплина, а всякий инженер должен руководствоваться принципом целесообразности. Вот только один пример из личной жизни. Пишем с другом программу. Друг говорит: а давай здесь используем вот такую возможность. Я — а зачем? Друг — она сократит программу на пять строк и сделает ее более "наглядной". Я — а ты уверен, что другие компиляторы ее поддерживают? По мне так лучше написать пять лишних строк сейчас и никогда потом не возвращаться к этому коду, чем править твой "элегантный" код при всяком переносе на другой компилятор, мучительно вспоминая как он работает.

Замечено, что программисты, незнакомые со стандартом, зачастую пишут более переносимый код, чем те, кто излазил его вдоль и поперек и теперь горит желанием применить полученные знания на практике. Тем не менее, без карты минного поля далеко не уйти, а без знания стандарта — ничего не запрограммировать. Знание пунктов стандарта и номеров статей уголовного кодекса придает программисту шарм профессиональном солидности, значительно упрощающий трудоустройство, особенно на руководящие места, так что близкое знакомство со Стандартом обещает быть отнюдь не бесполезным.

Новый и Старый Стандарты

Стандартизацией языка Си++ занимается множество различных "инициативных" групп, основной "костяк" которых составляют **ISO**, **IEC**, **JTC1**, **SC22** и **WG21**. Все вместе они образуют единый Комитет, который так и называется "ISO/IEC/JTC1/SC22/WG21 The C++ Standards Committee", формальной главой которого является ISO — псеводобщественная организация, продвигающая в качестве международных стандартов коммерческие решения своих создателей. Короче говоря, международная мафия в чистом виде. Покруче всяких крестных отцов будет. Но это ладно, это все лирика. Переходим к главной странице комитета: http://www.open-std.org/jtc1/sc22/wg21/.

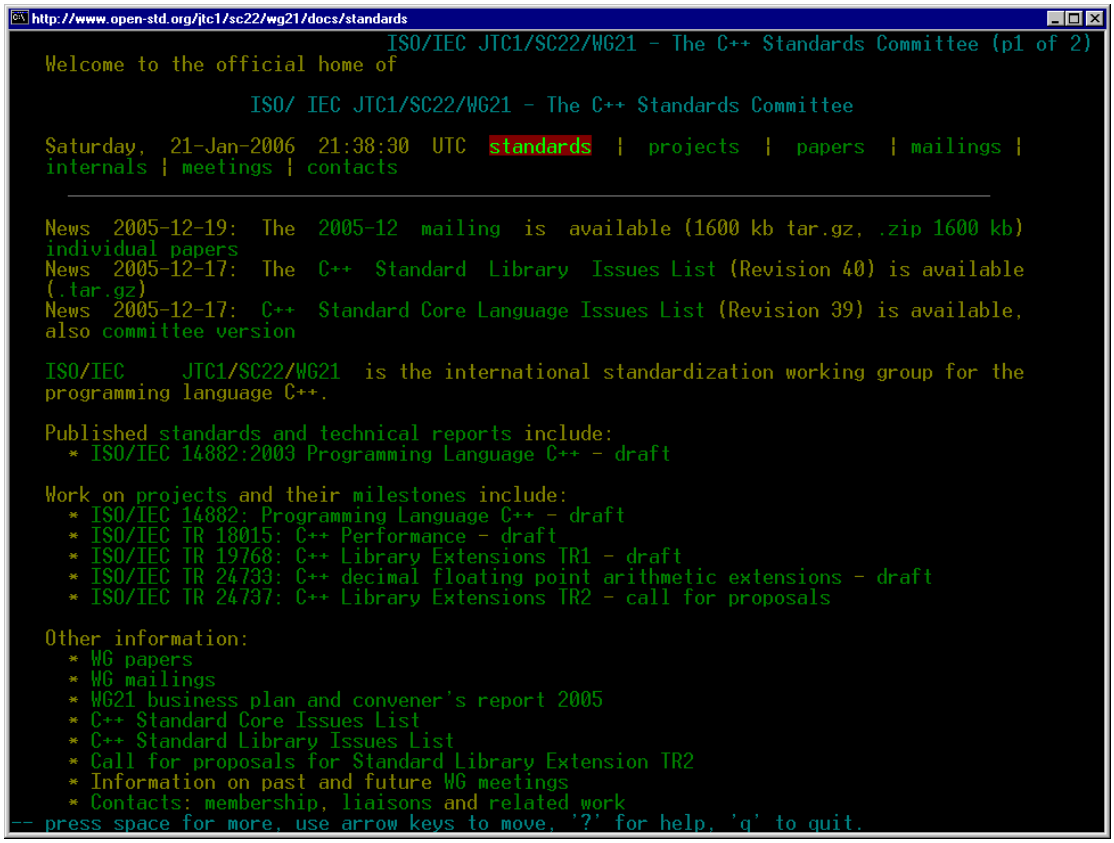

Рисунок 2 главная страница Комитета по Стандартизации Си++ в браузере Lynx

Здесь можно подписаться на рассылку, узнать новости, поживиться различными сопроводительные материалы, а вот текста самого стандарта нет — он распространяется только на платой основе в печатном виде, причем бумага делается отнюдь не из конопли, а из деревьев, которых мыщъх'у искренне жаль. Бесплатно можно заточить только черновую версию, так называемый draft: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2001/n1316/body.pdf. Для "простых смертных" программистов, не озабоченных сертификацией своего компилятора, она вполне пригодна.

Как вариант, можно воспользоваться Европейским ECMA Стандартом на язык C++/CLI, практически слово в слово копирующим полную версию Стандарта на Си++ (и заслужившим за это множество упреков и нападок). Стандарт за номером 372 (http://www.ecma-international.org/publications/standards/Ecma-372.htm) — наш. В отличии от буржуазной ISO, Европа еще не совсем охрена и раздает полные версии Стандартов с одной хапки, не требуя за это ни денег ни даже традиционной регистрации, однако, пользуясь ECMA-372 необходимо быть готовым к любых неожиданностям и несовпадениям с ISO/IEC Си++, поэтому дальше по тексту упоминается только ISO/IEC, а ECMA используется как заначка.

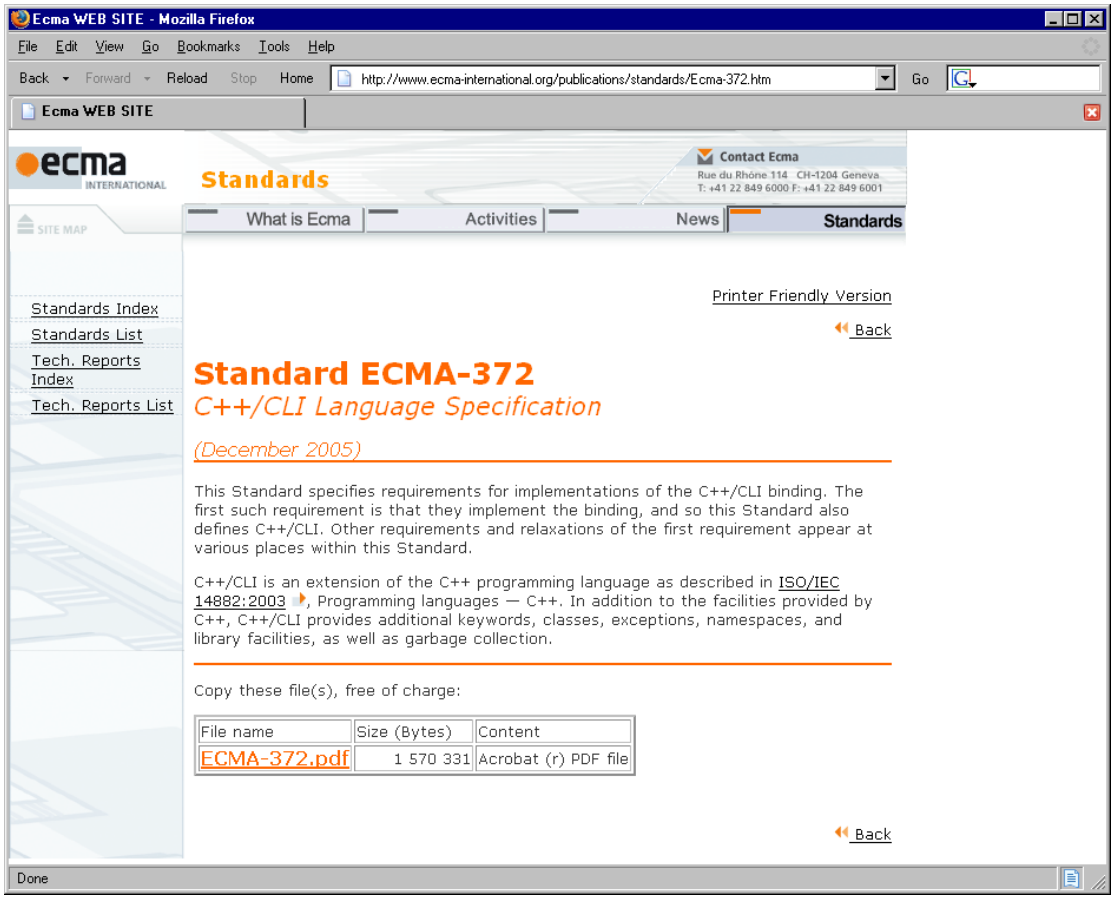

Рисунок 3 Главная страница ECMA-372 – стандарта языка C++/CLI

Согласно ISO/IEC, стандарту, описывающему приплюснутый Си, присвоен номер 14882 за которым идет год, в котором этот стандарт был принят. В настоящее время самой ходовой версией является стандарт от 1998 года, обозначаемый как ISO/IEC 14882:1998 и далее по тексту именуемый "Старым" Стандартом. Последняя редакция была принята в 2003 году ("Новый" Стандарт) и реально поддерживается только несколькими компиляторами, да и то криво. Следующий стандарт выйдет где-то в 2007-2010 году и будет содержать кучу нововведений, описанных в соответствующей врезке.

Списка изменений комитет не ведет, вынуждая нас сравнивать различные версии стандарта самостоятельно вычитывая и сверяя порядка 750 листов на буржуинском. Правда, в Сети можно найти неофициальный перечень изменений, подготовленный третьими лицами: http://www.acceleratedcpp.com/authors/koenig/c++std/revisions.pdf, но там этих изменений целых 300 страниц, большую часть из которых составляет чисто "редакторская" правка, устраняющая разночтение в формулировках. Ковыряться в этой навозной куче занятие неинтересное и неблагодарное. Где бы найти квинтэссенцию: что нового появилось в Стандарте по существу? А что если обратиться к компиляторам? Берем какой-нибудь компилятор, наиболее полно поддерживающий Новый Стандарт (например, EDG C++ Front End) и читаем what's new, который можно слить с http://www.edg.com/cpp_ftrs.html. На самом деле, там каша отличий Старых и Новых Стандартов, сдобренная исправленными багами самого компилятора EDG, однако, с чисто практической точки зрения — один хрен разница.

В основном, изменения касаются шаблонов, причем многие из них носят "отвоевательный" характер. Большой победой стало утверждение экспортируемых (**export**) шаблонов. Формально, эта возможность присутствовала еще в Старом Стандарте, однако мало кем поддерживалось в силу невостребованости и технических сложностей реализации. Примеры, приведенные в учебниках по Си++, не транслировались ни GCC, ни Microsoft Visual C++ и долгое время их переваривал только уже упомянутый компилятор переднего плана EDG, на котором, кстати говоря, основан популярный Borland C++ Builder и малоизвестный Comeau C++. Секретарь Комитета по стандартизации Эрб Саттер (Herb Sutter), выступил с переложением убрать экспортируемые шаблоны из Нового Стандарта на фиг, (см. дискуссию под лозунгом "Why We Can't Afford Export" в

http://std.dkuug.dk/jtc1/sc22/wg21/docs/papers/2003/n1459.html), однако это предложение не прошло: 8 участников проголосовали за удаление export'a и 28 были за то, чтобы его оставить, и, в конечном счете, export был оставлен, что высадило разработчиков компиляторов на полную измену. Еще бы им не высаживаться! Ведь поддержка экспортируемых шаблонов требует значительных переделок не только компилятора, но и линкера. Многие куски кода вообще придется переписывать заново… А не поддерживать export нельзя — перестанут уважать. Полнота поддержки Стандарта стала вполне весомым критерием при выборе компилятора.

Другим немаловажным достижением можно считать проработку шаблонов с частичной специализацией (partial specialisation), которые в Старом Стандарте описывались весьма туманно и кроме EDG никем не поддерживались, а зря. Как известно, шаблоны представляют собой механизм абстрактной обработки данных, "переваривающий" целочисленные переменные наряду с векторными типами и еще хрен (хвост?) знает чем. Это удобно, но непроизводительно и для достижения наивысшей производительности необходимо создать специализированный шаблон (specialize the template), обрабатывающий "свой" тип данных. Таким образом, у нас будет уже два шаблона: общий шаблон (general/genetic template) обрабатывающий все типы данных и специализированный шаблон, обрабатывающий какой-то один конкретный тип с наибольшей эффективностью. Это и есть специализация, худо-бедно поддерживаемая большинством компиляторов еще со времен Старого Стандарта.

Теперь возьмем шаблон класса с несколькими параметрами. Полная специализация требует специфицировать либо все параметры шаблона, либо ни одного, что есть саксь. При частичной же специализации, мы можем специфицировать любое подмножество параметров, а остальные — обрабатывать в общем виде. Но это — по стандарту. В реальной жизни, большинство компиляторов либо совсем не поддерживают частичной специализации, выдавая ошибку трансляции, либо молчаливо игнорируют ее, высаживая программиста на измену возле напрочь убитой программы, которая работает совсем не так, как задумывалось. Подробнее об этом можно прочитать в статье "Partial template specialisation" http://www.absoluteastronomy.com/reference/partial_template_specialisation/.

В целом, ситуация с шаблонами выглядит как откат к старым парадигмам и знаменует приближающийся провал. Большинство задач, решаемых шаблонами, легко разрешимы в рамках классического процедурного программирования и ассемблерных макросов. Если бы не убогость Сишного препроцессора, шаблоны могли вообще не возникнуть — в них просто не было бы потребности. Специализированные шаблоны это возврат к прежним способом обработки данных (своему типу — свой метод), но на "качественно новом уровне", который можно проиллюстрировать так: было у нас отдельное сверло по дереву и отдельное — по металлу и решили мы, что два сверла, дублирующих друг друга, это не только экономически невыгодно, но и вообще беспонтово. Вот и изобрели универсальное сверло, берущее и дерево, и металл. Быстро выяснилось, что режет оно хреново, ведь каждый материал имеет свои особенности обработки, но вместо того, чтобы честно признать свое поражение, мы усовершенствовали сверлильный механизм так, чтобы он самостоятельно распознавал тип материала, и автоматически изменял геометрию профиля сверла!!! Конструкторы крякнули и послали всю эту раджу на хрен, даже не пытаясь ее реализовать! Ничего не напоминает?

Возможности метапрограммирования в Новом Стандарте так же усилились. Опять-таки это возврат к древнему самомодифицирующемуся коду, только под другим углом. В отличии от функционального программирования, метапрограммирование ориентировано на создание программ, манипулирующих другими программами или самим собой, что на Си++ опять-таки реализуется посредством шаблонов. В частности, пример мета-реализации факториала выглядит так: template struct Factorial; template <> struct Factorial<1> ; Красиво, конечно, но, увы, не эффективно!

Шаблоны представляют одну из тех областей языка, агрессивного использования которых по возможности следует избегать, поскольку качество реализации компиляторов оставляет желать лучшего и наибольшая часть багов сосредоточена именно здесь. С другой стороны, шаблоны значительно упрощают программирование, ускоряя процесс разработки программы в несколько раз, а ошибки трансляторов исправляются по мере обнаружения. Нельзя просто сидеть и ждать. Чтобы разработчики компиляторов довели поддержку шаблонов до ума, их нужно пинать.

В стороне от шаблонов идет возня по стандартизации механизма обработки структурных исключений и "декорации" (decoration) имен, так же называемое "мангляжом". До тех пор, пока это не будет сделано, объектные файлы, сгенерированные различными компиляторами, останутся несовместимыми между собой, препятствуя созданию "смешанных" проектов. Впрочем, уже сейчас существуют линкеры, поддерживающие несколько

компиляторов, например, Microsoft Visual C++ и Borland Builder (например, профессиональный линкер ulink, созданный Юрием Хароном, бесплатную копию которого можно найти на сайте www.doswin32.com).

Остальные "инновации" Нового Стандарта носят сугубо "косметический" характер, к которому относится появление типа long long или возможность записи "list<vector<string>>" вместо "list<vector<string> >".

Компилятор GCC, начиная с версии 4.0.2, так же поддерживают новый Стандарт, однако не в полной мере. В архиве с исходными кодами находится директория gcc/testsuite/g++.dg/tc1 с тестовыми примерами и текущим статусом. Проваленные тесты отмечаются ключевым словом "xfail" в комментариях, что означает: данный тест еще не реализован. Ко всем остальным прилагаются "дефектные рапорты" (defect report), по одному рапорту на файл.

Microsoft Visual C++ 8, известный под кодовым именем Whidbey (остров в штате Вашингтон), ака Visual C++ 2005, так же поддерживает Новый Стандарт, но… в очень незначительной мере. Основные усилия группы разработчиков направлены в сторону выдвижения C++/CLI и на устранение ранее обнаруженных ошибок предыдущих версий, а ошибок там… короче, у Microsoft ошибок просто тьма. Впрочем, у других производителей дела обстоят не сильно лучше и чтобы написать портабельную программу, компилируемую более чем одним компилятором, необходимо ограничиться лишь базовыми языковыми функциями, да и то с кучей предосторожностей.

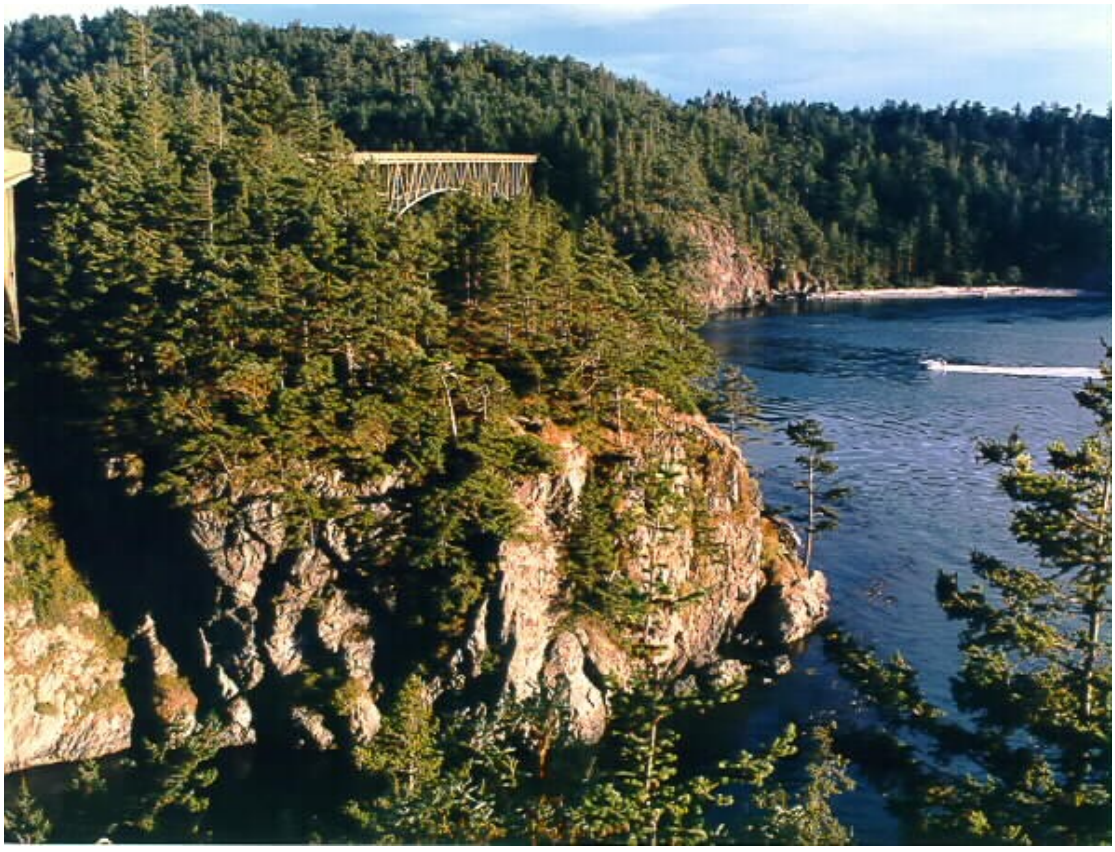

Рисунок 4 живописный остров Whidbey в штате Вашингтон, именем которого названа очередная версия компилятора Microsoft Visual C++

На сайте Mozill'ы лежит руководство по созданию переносимого кода, перечисляющее основные "разногласия" приплюснутых компиляторов, которое так и называется: "C++ portability guide" (http://www.mozilla.org/hacking/portable-cpp.html). Полчаса увлекательного чтения, сопровождаемого отборным матом и истерикой — гарантированы. Правда, не всему написанному можно верить. Несмотря на то, что последняя доступная на данный момент версия — версия 0.8 датируется 2001 годом, ситуация вовсе не так плачевна и многие из упомянутых ошибок давно исправлены. Тем не менее, при переносе программы на другие платформы, далеко не всегда удается найти свежий компилятор, поэтому, осторожность и осмотрительность не помешает.

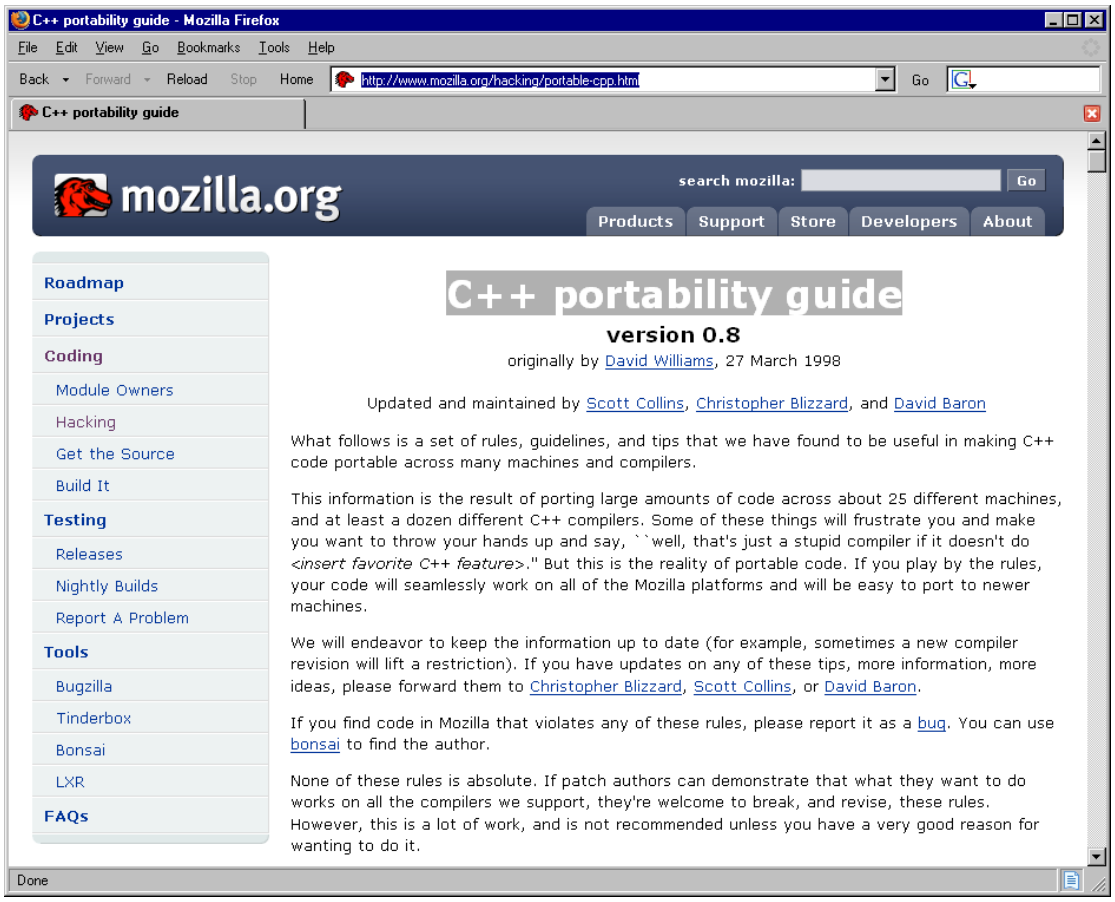

Рисунок 5 руководство по написанию портабельного кода на приплюснутом Си от Mozill'ы

заключение

Хотим ли мы этого или нет, но Си++ неотвратимо эволюционирует от плохого к еще более худшему. Впрочем, на этот счет имеются различные мнения. Некоторые хотят видеть язык предельно простым, каким был и остается классический Си, другим требуется навороченный монстр, который в совершенстве может освоить только эксперт. Какой из этих путей "правильный"? Обратимся к естественным языкам типа русского и английского.

Язык аристократов это сплошное нагромождение условностей и противоестественных сложностей. В нем преобладают длинные слова, сложные грамматические правила и т. д. Язык трущоб обычно бывает намного более выразителен и в то же время незамысловатым, а просочившиеся в него аристократические слова со временем теряют все лишнее и усекаются, сокращаясь по длине в несколько раз. Естественно, аристократам это не нравится, ведь главный признак образованности с их точки зрения заключен в языке, точнее в умении им "владеть". Но что стоит за этой "образованностью" кроме знания дутых конструкций?

Или вот музыка. Сначала было бум-бум, потом — во времена Баха и Моцарта — целая симфония чувств. Эволюция? А вот и нет! С приходом попа все вернулась к прежнему бум-бум. Народ устал от сложной музыки, захотелось простых мотивов, которые тоже не стоят на месте, а с каждым годом усложняются прямо на наших глазах и в 21 век мы въезжаем с оркестровым пением и группами типа Sirenia и Penumbra. Вопрос — надолго ли?

Вот так точно и в программировании. Первые машинные языки были очень простыми, но они все усложнялись и усложнялись до тех пор, пока не появился Си, который многие воспринимали как "студенческую подделку", "варварский откат назад" и т. д.

Вполне логично ожидать, что на смену приплюснотому Си придет язык с предельно простым синтаксисом, который можно выучить буквально за ночь! Поэтому, не спешите тратить время на углубленное изучение тонкостей Си++, возможно, они исчезнут прежде, чем успеют понадобиться.

>>> врезка киты и титаны или кто занимается стандартизацией Си++

Комитеты, занимающиеся стандартизаций Си++, прячутся за малопонятными аббревиатурами. Прячутся, бля, и прячутся. Вот суки! Пиздец, если их не расшифровать!

- ISO: International Standards Organization — Международная Организация Стандартизации, членом которой может стать только представитель Национального комитета по стандартизации;
- JTC1: Joint Technical Committee – Совместный технический консультационный совет по информационным технологиям;
- SC22: Subcommittee for Programming Languages — Подкомитет по правам Языков Программирования, их Окружению и Системным Интерфейсам;
- WG21: Working Group for C++ — Рабочая Группа по стандартизации Си++;
- ANSI: American National Standards Institute – Американский Национальный Институт Стандартов;
- NCITS: National Committee for Information Technology Standards – Национальный Комитет по Стандартизации Информационных Технологий (Бывший X3 – что расшифровывается вовсе не как хрен его знает, а как Аккредитованный Комитет Стандартизации);
- J16: Technical Committee for Programming Language C++ — Технический Комитет по Языку Программирования Си++;
- Fermilab — Лаборатория Ферми, член J16

>>> врезка что день грядущий нам готовит или основные направления развития Си++

Язык Си++ не стоит на месте, а неуклонно развивается. Свежие редакции Стандарта уже не за горами. Каких изменений нам ждать? Сотрудники подразделения Лаборатория ускорительной техники имени Ферми (Fermi National Accelerator Laboratory), базирующейся в американском городе Батавия (штат Иллиноис) и по совместительствую члены Комитета по Стандартизации, Walter E. Brown и Marc F. Paterno любезно предоставили перечень основных направлений, перевод которого в сокращенном виде приводится ниже:

- **ядро языка:**
  - динамические библиотеки;
  - расширенная семантика операций присвоения данных в векторных типах;
  - сокращение времени компиляции;
  - новые концепции и парадигмы;
  - статические assert'ы;
  - автоматические и decltype типы данных;
  - forwarding-конструкторы;
  - локальные классы как параметры шаблонов;
  - литералы, определяемые пользователем;
  - null-pointer константы;
  - алиасы шаблонов;
- **стандартная библиотека:**
  - улучшенный датчик случайных чисел
  - поддержка специальных математических функций;
  - "умные" указатели;
  - расширенная связка функций;
  - разупорядоченные и хэш контейнеры;
  - регулярные выражения;
  - полиморфные обертки вокруг библиотечных функций;
  - поддержка типов tuple и traits;
  - "адаптеры" для указателей-членов;

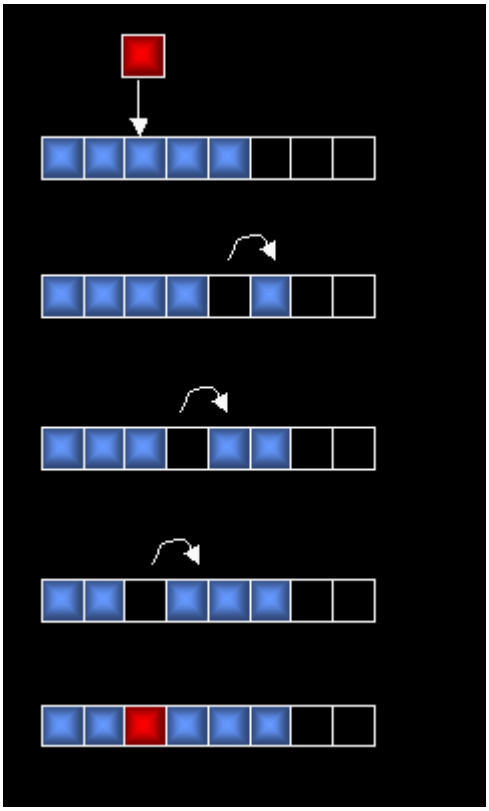

Рисунок 6 пример, иллюстрирующий расширенную семантику операций присвоения данных на векторах

>>> врезка ссылки по теме

- **JTC1/SC22/WG21 - The C++ Standards Committee**:
  - главная страница Комитета по стандартизации Си++ (на английском языке): http://www.open-std.org/jtc1/sc22/wg21/
- **ISO/IEC 14882: Programming Language C++ - draft**:
  - черновая версия Новой редакции Стандарта (на английском языке): http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2001/n1316/body.pdf;
- **C + + standard-unofficial list of revisions**:
  - неофициальный список изменений (не diff) между Старой и Новой редакции Стандарта языка Си++, составленный энтузиастами (на английском языке) http://www.acceleratedcpp.com/authors/koenig/c++std/revisions.pdf;
- **Standard ECMA-372 C++/CLI Language Specification**:
  - **полная** версия Стандарта языка C++/CLI, распространяющаяся на бесплатной основе (даже без традиционной регистрации!!!) и практически слово в слово копирующая Стандарт на Си++ — спасибо Microsoft (на английском языке): http://www.ecma-international.org/publications/standards/Ecma-372.htm;
- Supported C++ and C Language Features
  - перечень новых возможностей языка Си++, поддерживаемых компилятором переднего плана EDG (на английском языке): http://www.edg.com/cpp_ftrs.html;
- Minutes of J16 Meeting No. 36/WG21 Meeting No. 31, April 7-11, 2003
  - текст одной из дискуссий Комитета по Стандартизации, обсуждающий экспортируемые шаблоны и другие вопросы (на английском языке): http://std.dkuug.dk/jtc1/sc22/wg21/docs/papers/2003/n1459.html;
- MSDN Product Feedback Center:
  - центр управления багами в Visual C++ и других продуктов фирмы Microsoft: http://lab.msdn.microsoft.com/productfeedback/;
- Standard Features Missing From VC++ 7.1:

- статья, подробно описывающая некоторые из дефектов поддержки Стандарта компилятором Microsoft Visual C++ 7.1 (на английском языке) https://secure.codeproject.com/cpp/stdexport.asp;

- C++ portability guide:
  - подробное руководство по созданию переносимого кода на Си++, компилируемого всеми компиляторами (на английском языке): http://www.mozilla.org/hacking/portable-cpp.html;
- Incompatibilities Between ISO C and ISO C++:
  - несовместимости между классическим Си и Си++ (на английском языке): http://david.tribble.com/text/cdiffs.htm;
- A Brief Look at C++0x:
  - дохлый труп страуса делиться своим имхо о будущем языка Си++ — забавная и сильно водянистая статья пропагандистского типа (на английском языке): http://www.artima.com/cppsource/cpp0x.html;
- The C++ Standards Committee: Progress & Plans
  - презентация в ppt и pdf, подготовленная Комитетом по Стандартизации Си++ и рассказывающая, что нас ждет в грядущих редакциях (на английском языке): http://www-cdserver.fnal.gov/cd_public/sag/J16/J16.ppt