Развитие языка Си и обзор будущего стандарта C23

В. А. Галатенко[1], Г. Л. Левченкова[2], С. В. Самборский[3]

[1]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, galat@niisi.ras.ru;

[2]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, galka@niisi.ras.ru;

[3]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, sambor@niisi.ras.ru

Аннотация. Работа содержит краткий обзор основных этапов развития языка программирования Си с момента его создания. Рассматриваются существующие стандарты этого языка. Пристальное внимание уделяется новому стандарту языка Си – C23. Выделяются его достоинства и недостатки.

Ключевые слова: язык Си, стандарт, C23

1. Введение

Язык Си был создан в 1969-1973 годах [1] – ровно полвека назад. За первые 20 лет своего существования Си стал основным языком системного программирования. Язык C++, созданный в середине 80-х годов 20-го века на основе Си, смог потеснить Си во многих сферах, но остаются области, в которых конкурировать с языком Си очень сложно.

Язык Си незаменим при написании ядер операционных систем и драйверов самых разных устройств. На этом языке реализованы сетевые протоколы, ставшие фундаментом современных компьютерных сетей. На нем запрограммированы всевозможные микроконтроллеры, которые окружают нас повсюду: в микроволновых печах и автомобилях, пылесосах и стиральных машинах.

Компиляторы языка Си существуют для всевозможных процессорных архитектур и практически для всех операционных систем. Таким образом, сложно представить ситуацию, когда прикладному программисту доступен, например, язык Java или C++, но недоступен Си.

Обратное возможно, и это необходимо учитывать, выбирая язык, если предполагается в будущем переносить программу на новые платформы.

То, что язык Си продолжает широко использоваться, заставляет этот язык развиваться. С другой стороны, то, что на языке Си пишутся программы, которые используются многие десятилетия, естественно приводит к значительному консерватизму в его развитии. Таким образом, каждый новый стандарт языка Си заслуживает внимательного изучения.

2. История возникновения языка Си

Как известно, язык Си был создан [1] для того, чтобы сделать операционную систему Unix переносимой. Существующие в конце 60-х годов языки программирования высокого уровня (FORTRAN, ALGOL, PL/I) плохо подходили для написания ядра операционной системы и драйверов устройств (есть исключения: например, операционная система Multics реализована на EPL - диалекте PL/I).

Утилиты операционной системы, такие как ls, cat или wc, можно было реализовывать на языке высокого уровня, но это было неудобно и непрактично. ОС Unix изначально создавалась для миникомпьютеров с довольно ограниченными ресурсами, поэтому было желательно, чтобы утилиты занимали как можно меньше оперативной памяти. Языки программирования высокого уровня не всегда позволяли программисту получить достаточно компактный исполняемый код. Кроме того, компиляторы «серьезных» языков высокого уровня сами требовали слишком много ресурсов (в первую очередь, памяти). На слабом компьютере такие компиляторы было сложно реализовать и работали они слишком медленно.

Альтернативой языкам высокого уровня были языки ассемблера, но использование для создания операционной системы ассемблера не только очень трудоемко, но и сразу лишает эту операционную систему переносимости. Получается операционная система для одной компьютерной архитектуры. Таких операционных систем было написано много, и все они перестали

использоваться вместе с компьютерами, для которых были написаны. Собственно, Unix изначально была написана на ассемблере PDP-7, и только четвертая версия была переписана на Си в 1973 году.

Проблемы с трудоемкостью и непереносимостью ассемблерного программирования частично решаются использованием ассемблера с развитым макропроцессором. Подобные средства были популярны при программировании на миникомпьютерах - макропроцессирование позволяло замаскировать специфические особенности разных моделей компьютеров и несколько снизить трудоемкость программирования. При помощи макросов можно было даже имитировать в ассемблере структурное программирование - использовать управляющие конструкции (условные, циклы разных видов, и т.п.) реализованные в виде макросов. Эти макросы получались весьма сложными, так как управляющие конструкции (например, циклы) могут быть вложенными и требуется не перепутать, где какая конструкция заканчивается.

В целом, подход, основанный на макроассемблерах, лишь частично снижал трудоемкость ассемблерного программирования, и не позволял добиться настоящей переносимости программ. Также затруднялась отладка, так как отлаживать приходилось уже препроцессированный код, а не исходный.

Таким образом, была потребность в языке низкого уровня, который должен быть близок к аппаратуре, чтобы программы на этом языке могли оперировать всеми ресурсами компьютера. Но поскольку язык предназначен для реализации переносимых программ, он не должен быть связан с какой-либо конкретной компьютерной архитектурой.

Кроме того, программировать должно быть намного легче, чем на ассемблере. В идеале предполагалось, что язык должен сочетать мощь языка ассемблера (в смысле доступа к архитектуре и полного контроля со стороны программиста к генерируемому коду) с удобством языков высокого уровня. Наконец, для этого языка должно быть возможно реализовать быстрый, однопроходный, и нетребовательный к ресурсам компилятор.

Понятно, что эти требования противоречивы, но именно из них и получился язык Си. Этапы раннего развития Си описаны Деннисом Ритчи (Dennis Ritchie) в статье [1].

3. Противоречивая природа языка Си

Изначально с языком Си было связано еще одно противоречие: с одной стороны Си – «переносимый макроассемблер», средство для реализации операционной системы, инструмент, используемый в системном программировании. При этом не было потребности в формальных определениях, так как, если программист не знаком ни с одной компьютерной архитектурой (адресуемая память, регистры, машинные инструкции), то он все равно никак не сможет программировать на Си. Если же программист хорошо знаком с устройством компьютера (любого) и умеет программировать на ассемблере, то ему довольно просто изучить язык Си и для этого не потребуется никаких формальных определений.

Подобный «прагматически-технический» подход доминировал в отношении языка Си первые 10-15 лет и сейчас популярен среди системных программистов и тех, кто пишет программы, которые должны работать на «голом железе» без использования операционной системы. То есть среди именно тех программистов, для которых Си и предназначен.

С другой стороны, Си можно рассматривать как полноценный язык программирования, а значит, у него должны быть точно определены синтаксис и семантика. Кроме того, должно быть описано, как именно устроен ввод/вывод и прочее взаимодействие с операционной системой. И должны быть стандартные библиотеки, а значит должны быть стандарты для языка и его библиотеки.

Можно также заметить, что если язык Си создан для написания переносимых программ, то только соответствие стандарту (точнее той части стандарта, где поведение точно определено) гарантирует эту переносимость. Иначе работоспособность программы на новом компьютере остается вопросом везения. Точное определение семантики языка необходимо также для верификации программ. В конце концов, стандарт языка необходим хотя бы для того, чтобы язык не распался на множество диалектов.

Эти два подхода, «прагматически-технический» и «научно-формализованный», неизбежно конфликтуют между собой. Как только стандарт языка точно описывает семантику языка, у компилятора появляется возможность глубоко проанализировать программу. После этого компилятор может ее агрессивно оптимизировать, и не всегда результат окажется тем, что ожидал программист-практик.

Подробнее эти проблемы рассматриваются в статье [2]. Автор замечает, что в процессе стандартизации Си приобрел много свойств обычных языков программирования и стал менее пригоден для системного программирования. Причем иногда это сделано ради того, чтобы оп-

тимизирующий компилятор мог улучшить оптимизацию на несколько процентов.

4. Краткая история стандартов языка Си

Можно считать, что язык Си сформировался в 1973 году: в нем появились структуры (тип struct), препроцессор (частично), название поменялось на «Си» (C), до этого он назывался «Би» (B), затем «Энби» (NB). Наконец, в 1973 году на Си была успешно переписана ОС Unix, что можно рассматривать как доказательство полноценности языка: в языке присутствуют все необходимые для программирования средства.

4.1. Керниган-Ритчи (K&R)

После 1973 года развитие языка продолжалось, исправлялись ошибки, в том числе довольно серьезные, например, обнаружились программы с неоднозначным синтаксическим разбором. Скажем, выражение «n=-1» можно было интерпретировать как «n = -1», или как «n = n-1». После того, как это обнаружилось, синтаксис был изменен: бинарные операторы =+, =-, и т.д. приняли современный вид +=, -=, *=, и пр. Но даже 15 лет спустя, некоторые компиляторы выдавали предупреждение на конструкции, вроде «n=-1», «x=+0.3» или даже «h=*p» (где p - указатель).

Кроме того, язык развивался, добавлялись новые возможности. Наконец, в 1978 году Брайан Керниган (Brian Kernighan) и один из основных разработчиков языка - Деннис Ритчи (Dennis Ritchie) опубликовали учебник «The C Programming Language» [3]. Эта книга стала de facto первым стандартом языка Си. Описанный в ней диалект языка Си кратко обозначается как K&R C или C78 (подчеркивая статус неофициального стандарта).

В K&R C были исправлены ошибки и включены новые возможности: добавлены новые типы данных «long int» и «unsigned int», добавлена стандартная библиотека ввода/вывода, усовершенствован препроцессор.

4.2. C89 или C90

После 1978 язык Си продолжал развиваться, многие важные возможности были добавлены разработчиками компиляторов: перечислимые типы, функции типа void (не возвращающие никакого результата), присваивание структур и функции, возвращающие структуры.

В начале 80-х годов американский национальный институт стандартизации (ANSI) начал разработку стандарта языка Си, и, после нескольких черновых вариантов, в 1989 году вышел стандарт ANSI X3.159-1989 «Programming Language C» [4], на который обычно ссылаются как на ANSI C или C89.

На следующий год вышел международный стандарт ISO/IEC 9899:1990, полностью совпадающий с C89. Таким образом, этот стандарт получил еще обозначение C90. В дальнейшем ANSI отказался от разработки собственных стандартов языка Си, так что все последующие стандарты Си - международные стандарты ISO.

В стандарт ANSI C были включены расширения, добавленные в популярных компиляторах, поддержка наборов символов, нужных для разных языков, прототипы функций, указатели типа void* (абстрактные указатели), усовершенствования препроцессора и многое другое. Тем не менее, главное отличие ANSI C от K&R C заключается в том, что ANSI C позволил указывать типы аргументов функции прямо в списке аргументов (как уже было сделано в C++), а не между списком аргументов и телом функции (как в K&R C).

Керниган и Ритчи в 1988 году опубликовали второе издание своего учебника [5], переработав его под стандарт ANSI C и добавив материал по стандартной библиотеке. Эта книга также стала классическим учебником, возможно лучшим учебником по языку Си, и была переведена на десятки языков. Русский перевод вышел в 1992 году [6].

В 1995 году был опубликовано расширение стандарта C90, получившее обозначение ISO/IEC 9899:1990/AMD1:1995, это расширение часто называют C95. Существенных дополнений немного, можно упомянуть добавления расширенной поддержки интернациональных наборов символов.

4.3. C99

Следующий за C89 значительный стандарт вышел в 1999 и называется ISO/IEC 9899:1999 [7], сокращенно C99. В этот стандарт вошло очень много полезных новинок, а именно:

- объявление переменных больше не нужно собирать в начале блока;
- новые целочисленные типы long long int и unsigned long long int;
- типы для комплексных чисел;
- массивы переменной длины;
- макросы с переменным числом аргументов;
- встраиваемые функции (inline);
- ключевое слово restrict;
- комментарии начинающиеся с // и до конца строки (заимствовано из C++);
- выборочная инициализация элементов массивов и полей структур.

Возможно самое важное дополнение в C99 - это поддержка стандарта на арифметику с плавающей точкой (IEEE 754-1985 также известен как IEC 60559). Стало возможным управлять тем, как происходит вычисление, без чего было очень

сложно, например, правильно реализовать интервальные вычисления.

4.4. C11

Следующий существенный стандарт был выпущен в 2011 году под названием ISO/IEC 9899:2011 [8], неформально C11. В этом стандарте включено много добавлений и усовершенствований, таких как:

- управление выравниванием данных: спецификатор _Alignas, оператор alignof, функция aligned_alloc;
- спецификатор функции _Noreturn;
- улучшенная поддержка кодировок UTF-16/UTF-32, типы char16_t и char32_t, функции преобразования, запись строковых констант в этих кодировках;
- ключевое слово _Generic для «type-generic expressions», можно рассматривать как условный оператор, но не по значению, а по типу, например:

```
#define cbrt(x) _Generic((x), long double: \
cbrtl, \
default: cbrt, \
float: cbrtf)(x)
```

- статические проверки (assert), т.е. проверки времени компиляции.

Тем не менее, намного важнее, то, что в C11 впервые описано стандартное поведение многопоточной программы, для чего потребовалось детально описать модель памяти.

Также были стандартизованы средства создания и управления потоками, механизмы синхронизации параллельных вычислений, в том числе мьютексы. Для создания собственных переменных потока потребовался новый спецификатор хранения _Thread_local. Для всего вышеперечисленного добавлен новый заголовочный файл <threads.h>.

В связи с добавлением многопоточности, также добавлены атомарные операции доступа к памяти, новый модификатор типа _Atomic и заголовочный файл <stdatomic.h>.

Тем самым было, наконец, исправлена странная ситуация, когда стандарт языка педантично описывает поведение программы, но только однопоточной. Тогда как множество ответственных приложений, например, в сфере телекоммуникации, - давно многопоточные.

Вслед за C11 в 2018 году вышел стандарт ISO/IEC 9899:2018 [9], который ожидался в 2017 году и, поэтому, получил неформальное имя C17 (хотя иногда его называют C18). В этом стандарте были исправлены мелкие ошибки C11 и не добавлены никакие новые возможности.

5. Новый стандарт C23 (C24)

Как легко заметить, значительные стандарты языка Си (C78, C89, C99, C11) выходили примерно с интервалом 10-12 лет. Таким образом мы вправе рассчитывать на выход нового стандарта в районе 2021-2023 годов. Действительно, в следующем 2024 ожидается выход нового стандарта языка C: ISO/IEC 9899:2024, уже давно получившего неформальное обозначение C23 [10]. Доступна окончательная черновая версия стандарта C23, и можно оценить, что именно предлагает новый стандарт.

Полный перечень того, что содержит C23, занимает 3 страницы по одной строке на каждое изменение. Следовательно, необходимо как-то систематизировать этот список.

Сразу можно заметить, что C23 отменяет или объявляет устаревшими многое из предыдущих версий Си. Здесь можно выделить два случая.

Во-первых, те особенности языка Си, которыми перестали пользоваться 30 лет назад:

- запрещено описание аргументов в стиле K&R (между списком параметров и телом функции);
- не поддерживаются представления целых чисел со знаком отличные от дополнительного кода;
- отменены триграфы (trigraphs), впрочем, и так редко используемые.

В-вторых, некоторые средства, реализованные в предыдущих версиях как макросы (например, true, false, bool, thread_local) стали ключевыми словами языка.

Очевидно, что удаление устаревших или реализованных по-новому средств не несет никакой пользы для Си-программиста. В то время, как некоторые неудобства, связанные с поддержкой старых проектов, вполне возможны.

Существенно изменилось поведение библиотечной функции realloc: раньше вызов realloc с новым нулевым размером приводил к освобождению памяти, такому же, как при использовании функции free. Теперь поведение в этой ситуации не определено (undefined).

В стандарте C23 легализованы многие возможности, которые давно стали de facto стандартными, например, поддерживаемый компилятором GCC [11] более 30 лет оператор typeof. Теперь можно пользоваться typeof не нарушая стандарт, а еще (что очень полезно) появился оператор typeof_unqual.

Также теперь позволено ставить метки внутри блока не только перед инструкциями, но и перед декларациями или в конце блока. Фактически, компиляторы это и раньше позволяли.

Кроме этого, в стандарт добавлен целый набор функций и макросов для поиска и подсчета количества бит в целых числах разного размера (например, stdc_count_ones) и включаемый файл stdbit.h. Многие подобные функции

уже были включены в стандарт POSIX, например, ffs (найти первый поднятый бит) - функция POSIX.1-2001. Но полного комплекта подобных функций не было.

Некоторые новые возможности представляют собой просто изменения, направленные на улучшение читаемости текста программы, например, запись константы теперь можно разбивать разделителем «'»: например, 0xAB'CD'EF или 123'456'789.

К такого рода возможностям отнести и поддержку двоичного форматного ввода/вывода (по аналогии с десятичным, восьмеричным и шестнадцатеричным). Теперь можно записывать десятичное число 19 как 0b10011.

Еще одно полезное нововведение – возможность не задавать имя аргумента функции, если его не предполагается использовать. Тем самым, можно лаконично указать компилятору, на то, что аргумент не используется, и это не ошибка.

Многие изменения или расширения заимствованы в C23 из C++, например, упомянутая выше свобода ставить метку в произвольное место в блоке. Некоторые возможности выглядят как несущественные украшения, например, директивы препроцессора #elifdef и #elifndef, другие представляют интерес, например, возможность, указав ключевое слово auto, не указывать явно тип инициализированной переменной.

Вероятно, самое интересное заимствование из C++ - добавление атрибутов в двойных квадратных скобках. Как минимум, данный стиль компактнее того, который использует служебное слово __attribute__. Кроме того, сразу были добавлены несколько полезных атрибутов из C++11, а также разрешено дублирование атрибутов (как в C++23).

Очевидно полезное нововведение - макросы ckd_add(), ckd_sub() и ckd_mul() позволяющие гарантировать правильный математический результат целочисленных арифметических операций. Эти макросы определены в новом включаемом файле stdckdint.h.

Остались некоторые новшества, которые пока сложно оценить. Среди них можно привести новый целочисленный тип (типы) _BitInt(N) и unsigned _BitInt(N). Будет ли это использоваться, станет ясно со временем. Пока поддержка этих типов отсутствует в последней версии компилятора GCC.

Интересное нововведение – директива препроцессора #embed, позволяющая включать в текст программы бинарные данные, автоматически преобразованные в текстовый вид. Это полезно, если требуется инициализировать массив данными содержащими изображение в формате jpeg или музыку в формате mp3. Конечно, это легко проделать внешними средствами: собственным конвертором бинарных данных в текстовые, утилитой make и обычной директивой #include. Однако, #embed удобнее тем, что не создается промежуточный текстовый файл, размер которого может в разы превышать размер исходного бинарного файла. Пока оценить пользу от директивы #embed сложно, так как ее поддержка еще отсутствует во всех реально используемых компиляторах. Также остаются нереализованными некоторые вошедшие в C23 атрибуты.

6. Критика стандарта C23

Появление черновика стандарта C23 вызвало много критики. Можно отметить статью [12]. В этой работе авторы отмечают большую пользу от макросов ckd_add(), ckd_sub() и ckd_mul(), от того что, наконец, легализован оператор typeof, и добавлена библиотека битовой обработки (stdc_count_ones и другие).

Однако затем авторы переходят к тем особенностям нового стандарта, с которыми категорически не согласны, и приходят к выводу, что риски перевешивают выгоду. Таким образом они предлагают продолжать использовать C99 или C11.

Основные претензии в этой статье выдвигаются к изменению поведения realloc при вызове с нулевым размером. Отмечается, что функция realloc являлась «швейцарским ножом», поскольку работала как функция malloc при вызове от нулевого указателя, и как free при вызове с нулевым размером. Таким образом, можно было организовать все управление памятью через realloc, и это действительно использовалось во многих старых программах.

С одной стороны, возмущение критиков понятно, но с другой стороны для старых программ можно использовать старые компиляторы, или новые в режиме совместимости со старыми стандартами, а использование подобного стиля в новых разработках является не очень хорошей идеей.

Другие проблемы, которые отмечаются в упомянутой работе, связаны со сравнением указателей, но сложно согласиться с выводами об опасности C23, так как приводимые конструкции сами довольно провокационны, например:

```
free(p);...;if(p==q){....
```

Конечно, функция free освобождает память, на которую указывает указатель p, и никак не должна модифицировать сам указатель p, поэтому последующее сравнение с q должно быть корректно. Но это однозначно очень плохой стиль, так как программист, который видит, что p используется, легко может забыть, что память уже освобождена, и заменить сравнение p==q на

p[0]==q[0], сделав, тем самым, грубую ошибку.

7. Заключение

Новый стандарт языка Си оставляет несколько противоречивое впечатление. С одной стороны, в C23 попало множество безусловно полезных новшеств, таких как надежные арифметические операции ckd_add() и другие из stdckdint.h. Кроме того, появилась масса удобных новшеств: auto вместо явного указания типа, возможность не указывать имя неиспользуемого аргумента, атрибуты в двойных квадратных скобках. Многие новшества имеют скорее декоративный характер, как сепаратор «'» в записи констант.

С другой стороны, новое поведение функции realloc() легко может привести к неработоспособности старых программ.

Опять же совершенно непонятно, будут ли использоваться новые типы _BitInt(N) и unsigned _BitInt(N). И, наконец, директива препроцессора #embed, которая выглядит довольно чужеродно в препроцессоре языка Си.

В целом создается впечатление, что авторы нового стандарта в своем стремлении узаконить практически все известные расширения, несколько перестарались, из-за чего сам стандарт выглядит довольно разнородно. Также можно отметить, что среди новинок стандарта C23 трудно выделить какую-то основную - какой в C11 была поддержка многопоточности и всего с ней связанного. Не видно в C23 и масштабных изменений, после которых можно утверждать, что появился новый язык программирования (как C++20).

Как ни странно, возможно, C23 похож на C89: также в стандарт собрано много всего разного, и также заимствованы самые заметные синтаксические новшества из C++. Но заметим, что C89 был исключительно успешный стандарт, ставший основой всех современных реализаций и стандартов языка Си.

Остается открытым еще один важный вопрос: связь между развитием программного обеспечения, в частности, языков программирования и развитием аппаратных средств. Влияние в одну сторону очевидно: при использовании современных процессоров очень важно, чтобы программа эффективно использовала кэш, предсказуемо обращалась в память и передавала управление, а также, чтобы среди исполняемых машинных команд было больше независимых друг от друга, ради их параллельного исполнения. Эти требования влияют на использование различных техник и приемов программирования, а также на использование языков программирования. Те возможности языков программирования, которые не удается эффективно реализовать, отмирают.

Однако в случае языка Си, который, (как отмечалось ранее) незаменим и вездесущ, можно заметить и обратное влияние. Если в настоящем или ближайшем будущем времени планируется разработать микропроцессор с новой оригинальной архитектурой, то несовместимость этой архитектуры с языком Си становится сильным аргументом против подобного проекта.

В качестве гипотетического примера можно привести архитектуры с неоднородной памятью, размещенной непосредственно в процессоре, или архитектуры, подобные транспьютерным, с очень небольшим объемом памяти на один вычислительный узел. В любом случае реализация языка Си для таких архитектур оказывается сложна и неэффективна. Поэтому то, что в C23 остался только дополнительный код в качестве представления знаковых целых чисел, ограничивает разработку экзотических компьютерных архитектур.

«Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).»

The Brief History of the C Language and an Overview of the Future C23 Standard

Vladimir Galatenko, Galina Levchenkova, Sergei Samborskii

Abstract. The article contains a brief overview of the main stages of the development of the C programming language since its creation. The existing standards of this language are considered. Close attention is paid to the new standard of the C language – C23. Its advantages and disadvantages are highlighted.

Keywords: C language, standard, C23

Литература

1. Ritchie Dennis M. The Development of the C Language // The Second ACM SIGPLAN Conference on History of Programming Languages (HOPL-II). 1993. ACM. pp. 201-208.

2. Yodaiken V. How ISO C became unusable for operating systems development // 11th Workshop on Programming Languages and Operating Systems (PLOS '21). 2021.

3. Kernighan Brian W., Ritchie Dennis M. The C Programming Language (1st ed.) // Englewood Cliffs, NJ: Prentice Hall. 1978.

4. C89 Standard. https://web.archive.org/web/20161223125339/http://flash-gordon.me.uk/ansi.c.txt

5. Kernighan Brian W., Ritchie Dennis M. The C Programming Language (2nd ed.) // Prentice Hall. 1988.

6. Керниган Б., Ритчи Д. Язык программирования Си: Пер. с англ. // Под ред. и с предисл. Вс.С.Штаркмана. - 2-е изд., перераб. - М.: Финансы и статистика, 1992. - 272 с.

7. C99 Standard (draft n1256). https://www.open-std.org/jtc1/sc22/WG14/www/docs/n1256.pdf

8. C11 Standard (draft n1570). http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf

9. C17 Standard (draft n2176). https://web.archive.org/web/20181230041359/http:/www.open-std.org/jtc1/sc22/wg14/www/abq/c17_updated_proposed_fdis.pdf

10. WG14-N3096: Draft for ISO/IEC 9899:2023, April 1, 2023. https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3096.pdf

11. GNU Compiler collection. https://www.gnu.org/software/gcc

12. Terence Kelly, Yekai Pan. Catch-23: The New C Standard Sets the World on Fire // ACM Queue, March 29, 2023, Volume 21, issue 1. https://queue.acm.org/detail.cfm?id=3588242