

Оглавление

Предисловие от издательства	12
Отзывы и пожелания.....	12
Список опечаток.....	12
Нарушение авторских прав	12
Предисловие от главного редактора русского перевода	13
Истории читателей.....	17
Введение	18
Соглашения.....	18
Специальные шрифты	18
Специальные блоки.....	19
Исходный код.....	19
Компиляция программ	19
Как вам следует читать эту книгу?	21
Личные замечания	21
Благодарности	21
Сиппи	21
Редакторы русского перевода.....	22
Обо мне	23
О C++.....	24
1. Исторический контекст	25
1.1 C++98.....	25
1.2 C++03.....	26
1.3 TR1	26
1.4 C++11.....	26
1.5 C++14.....	26
1.6 C++17.....	26

2. Стандартизация	27
2.1 Стадия 3.....	28
2.2 Стадия 2.....	28
2.3 Стадия 1.....	28

КРАТКИЙ ОБЗОР C++20..... 30

3. C++20	31
3.1 Большая четверка.....	32
3.1.1 Концепты.....	32
3.1.2 Модули.....	33
3.1.3 Библиотека диапазонов (ranges).....	34
3.1.4 Корутины.....	35
3.2 Ядро языка.....	37
3.2.1 Оператор трехстороннего сравнения.....	37
3.2.2 Назначенная инициализация.....	38
3.2.3 <code>constexpr</code> и <code>constinit</code>	40
3.2.4 Улучшения работы с шаблонами.....	41
3.2.5 Улучшения лямбд.....	42
3.2.6 Новые атрибуты.....	42
3.3 Стандартная библиотека.....	43
3.3.1 <code>std::span</code>	43
3.3.2 Улучшения контейнеров.....	44
3.3.3 Арифметические утилиты.....	44
3.3.4 Календарь и временные зоны (часовые пояса).....	44
3.3.5 Библиотека для форматированного вывода.....	45
3.4 Параллельность.....	46
3.4.1 Атомарные операции.....	46
3.4.2 Семафоры.....	47
3.4.3 Защелки и барьеры.....	47
3.4.4 Кооперативное прерывание.....	48
3.4.5 <code>std::jthread</code>	50
3.4.6 Синхронные выходные потоки.....	51

Детали..... 54

4. Ядро языка	55
4.1 Концепты.....	55
4.1.1 Два неправильных подхода.....	56
4.1.2 Преимущества концептов.....	62
4.1.3 Длинная, длинная история.....	62
4.1.4 Исползования концептов.....	63
4.1.5 Ограниченные или неограниченные заполнители (Constrained and Unconstrained Placeholders).....	75
4.1.6 Сокращенные шаблонные функции.....	78
4.1.7 Предопределенные концепты.....	82

4.1.8	Определение концептов.....	88
4.1.9	Применение концептов.....	96
4.2	Модули.....	108
4.2.1	Зачем нам нужны модули?.....	108
4.2.2	Преимущества использования модулей	114
4.2.3	Первый пример.....	115
4.2.5	Экспорт	120
4.2.6	Рекомендации по структуре модуля.....	121
4.2.7	Блок интерфейса модуля и блок реализации модуля	122
4.2.8	Подмодули и разделы модулей	125
4.2.9	Шаблоны в модулях	129
4.2.10	Линковка на уровне модулей	132
4.2.11	Заголовочные блоки (header units)	134
4.3	Оператор трехстороннего сравнения.....	136
4.3.1	Упорядочение до C++20	136
4.3.2	Упорядочение начиная со стандарта C++20.....	138
4.3.3	Категории сравнения.....	141
4.3.4	Создаваемый компилятором оператор трехстороннего сравнения.....	143
4.3.5	Переписывание выражений (rewriting expressions)	148
4.3.6	Задаваемые пользователем и создаваемые автоматически операторы сравнения.....	151
4.4	Назначенная инициализация	154
4.4.1	Агрегированная инициализация	154
4.4.2	Именованная инициализация членов класса.....	156
4.5	consteval и constinit.....	161
4.5.1	consteval.....	161
4.5.2	constinit.....	163
4.5.3	Выполнение функций.....	164
4.5.4	Инициализация переменных.....	166
4.5.5	Исправляем проблему порядка статической инициализации	167
4.6	Улучшение работы с шаблонами	173
4.6.1	Условный явный конструктор.....	173
4.6.2	Нетипизированные параметры шаблона	176
4.7	Улучшения лямбд.....	180
4.7.1	Шаблонные параметры для лямбд.....	180
4.7.2	Определение неявного копирования указателя this	184
4.7.3	Лямбды в контекстах без выполнения. Использование конструктора по умолчанию и копирования для лямбд без состояния	186
4.8	Новые атрибуты	190
4.8.1	[[nodiscard("reason")]].....	191
4.8.2	[[likely]] и [[unlikely]]	195
4.8.3	[[no_unique_address]]	196
4.9	Дополнительные улучшения.....	199
4.9.1	volatile	199
4.9.2	Оператор цикла for с инициализацией на основе диапазона	201

4.9.3 Виртуальная функция с <code>constexpr</code>	202
4.9.4 Новый символьный тип для utf8-строк: <code>char8_t</code>	204
4.9.5 Использование <code>using enum</code> в локальной области видимости	205
4.9.6 Инициализаторы по умолчанию для битовых полей	206

5. СТАНДАРТНАЯ БИБЛИОТЕКА.....209

5.1 Библиотека диапазонов	210
5.1.1 Концепты <code>ranges</code> и <code>views</code>	211
5.1.2 Работа алгоритмов непосредственно со всем контейнером	212
5.1.3 Композиция функций	216
5.1.4 Отложенное выполнение	218
5.1.5 Определение видов	221
5.1.6 Аромат Python	224
5.2 <code>std::span</code>	230
5.2.1 Статическая и динамическая длина	230
5.2.2 Автоматический вывод размера непрерывной последовательности объектов	232
5.2.3 Создание <code>std::span</code> из указателя и размера	233
5.2.4 Изменение объектов, к которым происходит обращение через ссылку	235
5.2.5 Обращение к элементам <code>std::span</code>	236
5.2.6 Постоянный диапазон изменяемых элементов	238
5.3 Улучшения контейнеров	241
5.3.1 Контейнеры и алгоритмы со спецификатором <code>constexpr</code>	241
5.3.2 <code>std::array</code>	242
5.3.3 Последовательное удаление из контейнеров	244
5.3.4 <code>contains</code> для ассоциативных контейнеров	249
5.3.5 Проверка строки на наличие префикса и суффикса	252
5.4 Арифметические функции	255
5.4.1 Безопасное сравнение целых чисел	255
5.4.2 Математические константы	260
5.4.3 Вычисление середины отрезка и линейная интерполяция	262
5.4.4 Работа с битами	263
5.5 Календарные зоны и часовые пояса	269
5.5.1 Время дня	270
5.5.2 Календарные даты	273
5.5.3 Часовые пояса	289
5.6 Библиотека форматирования	296
5.6.0.1 Функции форматирования	296
5.6.1 Форматная строка	298
5.6.2 Задаваемые пользователем типы	306
5.7 Дальнейшие улучшения	312
5.7.1 <code>std::bind_front</code>	312
5.7.2 <code>std::is_constant_evaluated</code>	314
5.7.3 <code>std::source_location</code>	316

6. ПАРАЛЛЕЛЬНОСТЬ	318
6.1 Корутины	319
6.1.1 Функция-генератор	320
6.1.2 Характеристики	323
6.1.3 Фреймворк	325
6.1.4 Ожидаемые и ожидающие объекты	328
6.1.5 Исполняемый поток процессов	330
6.1.6 <code>co_return</code>	334
6.1.7 <code>co_yield</code>	336
6.1.8 <code>co_await</code>	339
6.2 Атомарные переменные.....	349
6.2.1 <code>std::atomic_ref</code>	349
6.2.2 Атомарный умный указатель.....	358
6.2.3 Расширения <code>std::atomic_flag</code>	362
6.2.4 Расширения <code>std::atomic</code>	370
6.3 Семафоры	374
6.4 Защелки и барьеры	379
6.4.1 <code>std::latch</code>	379
6.4.2 <code>std::barrier</code>	385
6.5 Координированное прерывание.....	389
6.5.1 <code>std::stop_source</code>	390
6.5.2 <code>std::stop_token</code>	391
6.5.3 <code>std::stop_callback</code>	391
6.6 <code>std::jthread</code>	398
6.6.1 Автоматическое присоединение	399
6.6.2 Кооперативное прерывание <code>std::jthread</code>	401
6.7 Синхронизированные потоки вывода	404
7. ПРАКТИЧЕСКИЕ ПРИМЕРЫ.....	413
7.1 Быстрая синхронизация потоков.....	414
7.1.1 Условные переменные	415
7.1.2 <code>std::atomic_flag</code>	417
7.1.3 <code>std::atomic<bool></code>	420
7.1.4 Семафоры	422
7.1.5 Общая статистика	424
7.2 Вариации объектов <code>future</code>	425
7.2.1 Ленивый объект <code>future</code>	428
7.2.2 Выполнение на другом потоке	431
7.3 Модификация и обобщение генератора.....	436
7.3.1 Изменения	440
7.3.2 Обобщение.....	443
7.4 Различные потоковые архитектуры, основанные на задачах	447
7.4.1 Прозрачная архитектура ожидающего потока задач	447
7.4.2 Автоматическое возобновление ожидающей задачи	450
7.4.3 Автоматическое возобновление ожидающего объекта на отдельном потоке	453

ЭПИЛОГ	458
ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ	460
8. C++23 и не только	461
8.1 C++23	462
8.1.1 Библиотека сопрограмм	462
8.1.2 Модуляризированная стандартная библиотека для модулей	476
8.1.3 Исполнители	479
8.1.4 Сетевая библиотека	483
8.2 C++23 или позже	485
8.2.1 Контракты	485
8.2.2 Рефлексия	488
8.2.3 Сопоставление с образцом	492
8.3 Дополнительная информация о стандарте C++23	496
9. Дополнительное тестирование	497
10. Глоссарий	510
10.1 Вызываемый объект, Callable	510
10.2 Вызываемый объект, Callable Unit	510
10.3 Одновременность, concurrency	510
10.4 Критическая секция	510
10.5 Конфликт по данным, Data race	510
10.6 Тупик, deadlock	511
10.7 Жадное выполнение, eager evaluation	511
10.8 Исполнитель, executor	511
10.9 Функциональные объекты, function objects	511
10.10 Лямбда-выражение, лямбда, lambda	512
10.11 Отложенное (ленивое) выполнение, lazy evaluation	512
10.12 Отсутствие блокировки, lock-free	512
10.13 Потерянное пробуждение, lost wakeups	512
10.14 Математические законы	512
10.15 Расположение в памяти, memory location	512
10.16 Модель памяти, memory model	513
10.17 Неблокирующий, non-blocking	513
10.18 Объект, object	513
10.19 Параллелизм, parallelism	513
10.20 Предикат, predicate	513
10.21 RAII	513
10.22 Состояние гонки, race conditions	513
10.23 Регулярный, regular	514
10.24 Скалярный, scalar	514
10.25 Полурегулярный, SemiRegular	514
10.26 Ложное пробуждение, Spurious Wakeup	514

10.27 Большая четверка, Big Four	514
10.28 Большая шестерка, Big Six	515
10.29 Поток, Thread.....	515
10.30 Сложность по времени, Time Complexity	515
10.31 Единица трансляции, Translation Unit.....	515
10.32 Неопределенное поведение, Undefined Behavior	516

Предисловие от главного редактора русского перевода

Дорогие друзья!

Книга, которую вы держите в руках, открывает серию переводов зарубежных изданий по высокоуровневым языкам программирования, которую издает Московский институт электроники и математики им. А. Н. Тихонова НИУ ВШЭ при поддержке группы компаний YADRO (yadro.com) совместно с издательством «ДМК Пресс».

Если вы интересуетесь цифровой электроникой, разработкой на ПЛИС, проектированием на языках описания аппаратуры Verilog или VHDL, то вы, скорее всего, уже знакомы с книгами серии «Цифровой синтез», такими как: Д. Харрис и С. Л. Харрис «Цифровая схемотехника и архитектура компьютера»; «Цифровой синтез: практический курс» (под. ред. А. Ю. Романова и Ю. В. Панчула), Ф. Бруно «Программирование FPGA для начинающих» и др. Эта книга несколько отличается по тематике и открывает новую серию книг, в первую очередь ориентированную на программистов, работающих с языками высокого уровня.

Почему я взялся редактировать русский перевод книги Р. Гримм «С++20 в деталях»?

На самом деле я сильно сомневался. Но меня убедила моя коллега Ирина Романова, она же и выполнила первую вычитку и редактуру этой книги. Без ее помощи я бы не согласился редактировать данный материал. Остальные мотивы описаны далее.

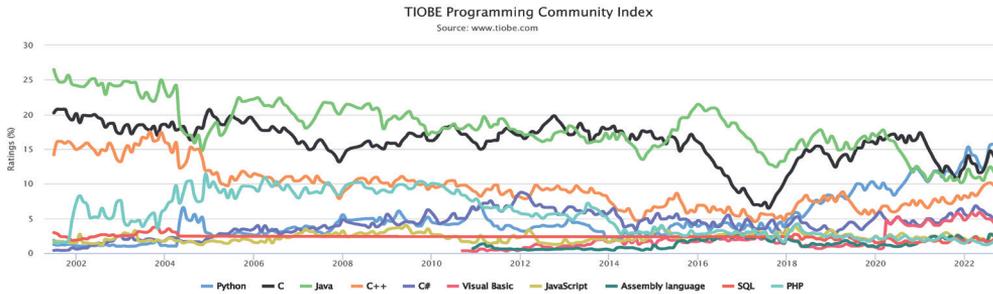
Как я использую С++?

Хотя я применяю С++ в своей практике постоянно, использую я его обычно как язык системного программирования, а также для программирования встраиваемых систем (таких как микроконтроллеры), т. е. вполне могу обходиться подмножеством С; а если нужна объектно-ориентированность, то вполне достаточно тех возможностей, которые предоставляет стандарт С++98.

Зачем изучать С++20?

Зачем же тогда изучать все эти С++11, потом С++14 и т. д., а теперь и С++20? (Ведь на подходе уже и следующий стандарт!)

Ответ: это очень интересно! С++ – это один из ведущих языков. По рейтингу ТЮВ С/С++ – все еще самый популярный язык в мире. Популярнее, чем Python!



C++ – довольно «старый» язык, который развивается, меняется, создает новые методы и принципы программирования прямо у нас на глазах. Это (во многом) отражение прошлого и будущего всех классических высокоуровневых языков программирования. Каждый новый стандарт – это новые парадигмы и «фишки» программирования. При этом предлагают, описывают и внедряют их обычные программисты в открытом комьюнити.

Поэтому даже если вы заядлый «питонист» или никогда не планируете использовать новшества C++, все равно стоит делать как я: следить за выходом новых стандартов и хотя бы пролистывать релизы о нововведениях в язык и другую литературу по теме – потому что понимать ход современной передовой мысли разработчиков программного кода, знать новые приемы и возможности ключевого языка общения всех программистов очень важно для саморазвития любого, кто любит и практикует разработку программ.

Зачем нужен русский перевод этой книги?

Очень часто в среде программистов встречаю мнение, что изучать специализированную литературу по программированию надо на языке оригинала: английском. В целом я с этим согласен – сам большинство информации по теме получаю на английском. Но это не потому, что читать на английском удобней, а потому, что на русском почти всегда нужной информации нет. Если хотите читать оригинал, пожалуйста, читайте его. Но если кто-то хочет быстро познакомиться с новшествами C++20 или разобраться более детально с какими-то тонкостями этих новшеств на примерах – эта книга для вас.

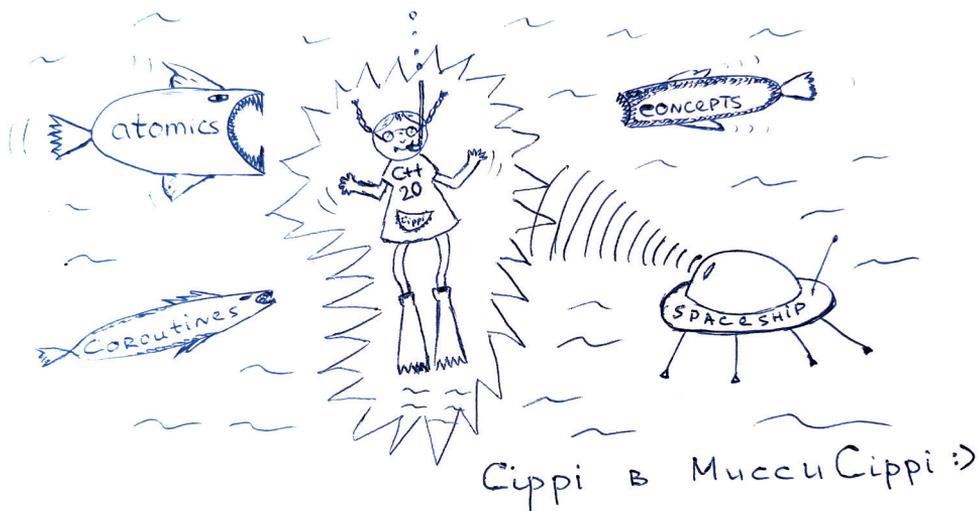
Почему именно книга Р. Гримма?

Потому что это признанный автор, а его книги очень популярны. Потому что Р. Гримм сам из «тусовки» разработчиков новых стандартов C++, и у него сравнительно легкая и интересная подача. Надеюсь, нам удалось это сохранить и в русском переводе. А еще в этой книге довольно информативные и показательные примеры со сравнительно небольшим количеством ошибок. Этим могут похвастаться далеко не все издания в данной области, даже издания именитых авторов!

Еще для себя я выработал один неформальный принцип оценки зарубежных книг, рассчитанных на большую аудиторию: это то, как она оформлена. Чем больше вложено в оформление издания, тем выше вероятность того (хоть

и не всегда!), что содержание книги будет на уровне. Дело в том, что западное общество довольно инфантильно (чего только стоят «График Шму» или «Принцип печенья Орео!»), поэтому книге без «картинок» довольно сложно добиться популярности. Объединение «картинок» и хорошего содержания является фактором, обеспечивающим успех книги.

Данная книга соответствует этому критерию. Хотя, на мой взгляд, персонаж книги, Сиппи, довольно безобразна, а подписи к рисункам катастрофически неинформативны. Такое представление персонажа я (не художник) тоже выдать могу! Но не включаю его в серьезную книгу. Хотя... а почему бы и нет?



Было желание убрать все эти бессмысленные рисунки из книги, но, в конце концов, было решено смириться с этим.

Для кого эта книга?

Давайте вернемся к серьезному тону и, наконец, ответим на вопрос: кому будет интересна эта книга? Программирование C++ по ней вы не изучите – для этого есть классические книги Х. Дейтела, Б. Страуструпа, С. Прата и др. Для того чтобы воспринимать материал, нужно уже знать концепцию объектно-ориентированного программирования и иметь хотя бы небольшой опыт разработки программ на C++ или «близких по духу» высокоуровневых языках – вроде C#, Java, Python. Неплохо бы знать такие концепции, как лямбда-функции и многопоточные/параллельные программы. Стандарт C++20 глубоко вы по этой книге тоже не изучите. Она нужна для первого быстрого и легкого знакомства с новшествами, которые появились в новом стандарте языка. Заодно читатели смогут узнать немного о причинах появления этих новшеств, принципах и идеях, которые в них заложены. Также в книге очень неплохие и понятные примеры программ. Остальные тонкости вы при необходимости узнаете на практике или изучив стандарт.

Ошибки, терминология и другие замечания

Проблема перевода такого рода книг состоит в том, что довольно часто в русском языке нет устоявшегося аналога английскому термину или прямой русский перевод выглядит глупо и неестественно. В основном это так, потому что сами английские термины являются «новоделами». Например, тот же *coroutine*. Все уже смирились говорить и писать «корутин», притом что в русском языке есть тоже иностранное, но более привычное слово «сопрограмма», которое имеет приблизительно то же значение. В русском переводе осуществлена попытка соблюсти баланс, обеспечивающий читабельность и понятность текста: какие-то термины используются в форме, более привычной русскоязычному читателю, а какие-то вообще не переведены. По той же причине сознательно не переведены надписи на рисунках и комментарии в листингах.

Эту книгу прочитали два научных редактора, один корректор и еще два студента (огромная благодарность Александру Богомолу и Руслану Нуржанову), но, безусловно, она не идеальна. Я буду очень признателен тем внимательным читателям, которые обнаружат в данном издании какие-либо ошибки или опечатки и сообщат о них по адресу a.romanov@hse.ru или dmkpress@gmail.com (книги постоянно перепечатываются, и в каждом новом тираже все найденные ошибки и недочеты исправляются).

Также вы можете присылать свои рисунки, изображающие Сиппи. Я пока еще не придумал, что с ними делать, но (возможно) получится импровизированная выставка 😊

Истории читателей



Сандор Дарго, старший программист в Amadeus

«С++ 20 в деталях» – это именно та книга, которая вам сейчас нужна, если вы хотите погрузиться в последнюю версию С++. Это полное руководство, в котором Рейнер рассматривает не просто передовые возможности С++20, но и небольшие добавления в язык. Эта книга содержит множество кода примеров, так что даже если у вас еще нет доступа к последним компиляторам, у вас все равно будет хорошее представление о том, чего вы можете ожидать от различных возможностей языка. Крайне рекомендую прочитать.



Адриан Там, директор по Data Science, Synechron Inc

С++ очень сильно развился с момента его появления на свет. С++20 – это практически новый язык. Конечно, это книга не предназначена для того, чтобы научить вас наследованию или перегрузке, но если вы хотите привести свои знания С++ к современному уровню, то это очень подходящая книга. Вы удивитесь тому, сколько всего нового из С++20 вошло в базовый язык С++. Эта книга дает ясные объяснения и краткие примеры. Структура книги позволит вам использовать ее как справочник в дальнейшем. Она поможет вам узнать все современные возможности языка.

Введение

Моя книга по C++ – это и учебник, и справочное руководство. Она научит вас C++20 и даст вам детальную информацию о новом стандарте C++. Наиболее выдающимися возможностями C++20 являются следующие:

- **концепты** – изменяют способ, которым мы думаем о программах с шаблонами. Они являются семантическими категориями для параметров шаблонов. Они позволяют вам ясно выражать свои намерения через систему типов. Если что-то пойдет не так, то компилятор даст вам понятное сообщение об ошибке;
- **модули** – позволяют обойти ограничения заголовочных файлов. Они очень многообещающи. Например, разделение заголовочного файла и исходного файла становится таким уже устаревшим, как и препроцессор. В результате вы получаете более быструю компиляцию и легкий способ создания пакетов;
- новая **библиотека диапазонов** (ranges library) поддерживает применение алгоритмов к контейнерам, включающим алгоритмы, организованные как конвейер (pipe), и «ленивое» (lazy) применение алгоритмов к бесконечным потокам данных;
- благодаря **корутинам** (coroutines) асинхронное программирование в C++ становится широко распространенным. Корутины образуют основу для кооперативных задач, циклов обработки событий, бесконечных потоков данных или конвейеров.

Конечно, это далеко не все. Вот еще некоторые новые возможности C++20:

- генерируемые автоматически операторы сравнения;
- библиотеки для работы с датами и временными зонами;
- библиотека format;
- «виды» (view) непрерывных блоков памяти;
- улучшенные, прерываемые потоки;
- атомарные умные указатели;
- семафоры;
- примитивы управления, такие как защелки (latch) и барьеры (barrier).

Соглашения

Есть всего несколько соглашений по форматированию этой книги:

Специальные шрифты

Курсив

Используется *курсив* для выделения цитат.

Жирный

Используется **жирный** шрифт для выделения имен.

Моноширинный

Используется моноширинный шрифт для кода, инструкций, ключевых слов и имен типов, переменных, функций и классов.

Специальные блоки

Блоки содержат подсказки, предупреждения и дополнительную информацию.



Подсказка

Этот блок предоставляет дополнительную информацию по текущему материалу и подсказки по компиляции программ.



Предупреждения

Блоки предупреждений должны помочь вам избегать ошибок.



Собранная информация

Этот блок в конце каждого раздела содержит важную информацию, которую стоит запомнить.

Исходный код

Примеры исходного кода, приведенные в книге, являются завершенными. Это значит, что если у вас есть подходящий компилятор, то вы можете их откомпилировать и выполнить. Имя исходного файла находится в заголовке каждого примера кода. Исходный код использует четыре пробела для табуляции (отступов). Иногда используется два пробела из соображений размещения на странице.

Кроме того, я не большой любитель директив namespace вроде `using namespace std`, поскольку они делают код более сложным для чтения и могут засорять пространства имен. Соответственно, я применяю их только тогда, когда это улучшает читаемость кода (например, `using namespace std::chrono_literals`). Иногда для более удобного размещения кода на странице я использую `using`, например `using std::chrono::system_clock`.

Таким образом, только в случае необходимости иногда делаются следующие отступления от общих правил оформления кода:

- табуляция на два пробела, а не на четыре;
- используется директива `using namespace std`.

Компиляция программ

Поскольку C++20 – это новый стандарт, то многие примеры могут быть откомпилированы только при помощи подходящих компиляторов. Я использую последние версии GCC¹, Clang² и MSVC³. При компиляции программы вы должны указать используемый стандарт C++. Это значит, что при использовании GCC

¹ <https://gcc.gnu.org>.

² <https://clang.llvm.org>.

³ https://en.wikipedia.org/Microsoft_Visual_C%2B%2B.

или Clang вы должны указать флаг `-std=c++20` и для MSVC флаг `/std:c++latest`. При применении параллельности (concurrency) для GCC и Clang необходимо указать, что вам требуется линковка с библиотекой `pthread` при помощи опции `-pthread`.

Если у вас нет в распоряжении подходящего компилятора C++, то вы можете использовать онлайн-компилятор вроде Wandbox¹ или Compiler Explorer². При применении Compiler Explorer с выбором компилятора GCC или Clang вы также можете выполнить откомпилированную программу. Для этого, во-первых, вам нужно выбрать `Run the compiled output` и, во-вторых, открыть окно `Output`.

```

1
2
3
4  Run the compiled output
5  Intel asm syntax
6  Demangle identifiers
7
8   call    std::basic_ostream<char, std::char_traits<char> >& std
9   mov     esi, OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13ba
10  mov     rdi, rax
11  call    std::basic_ostream<char, std::char_traits<char> >::ope
12  mov     eax, 0
13  pop     rbp
14  ret
15
16  static_initialization_and_destruction_0(int, int):
17  push   rbp
18  mov    rbp, rsp
19  sub    rsp, 16
20  mov    DWORD PTR [rbp-4], edi
21  mov    DWORD PTR [rbp-8], esi
22  cmp    DWORD PTR [rbp-4], 1
23  jne    .L5
24  cmp    DWORD PTR [rbp-8], 65535
25  jne    .L5
26  mov    edi, OFFSET FLAT:_ZStL8_ioinit
27  call   std::ios_base::Init::Init() [complete object construct
28  mov    edx, OFFSET FLAT:_dso_handle
29  mov    esi, OFFSET FLAT:_ZStL8_ioinit
30  mov    edi, OFFSET FLAT:_ZNSt8ios_base4InitD1Ev
31  call   __cxa_atexit
32
33  .L5:
34  nop
35  leave
36  ret
37
38  GLOBAL __sub_I_main:
39  push   rbp
40  mov    rbp, rsp
41  mov    esi, 65535
42  mov    edi, 1
43  call   __static_initialization_and_destruction_0(int, int)
44  pop    rbp
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Output (0/0) x86_64 gcc 10.2.1 - 1142ms (96472B)

Run code in the Compiler Explorer

Вы можете получить дополнительную информацию о совместимости с C++20 для различных компиляторов на cppreference.com³.

¹ <https://wandbox.org>.

² <https://godbolt.org>.

³ https://en.cppreference.com/w/cpp/compiler_support.

Как вам следует читать эту книгу?

Если вы незнакомы со стандартом C++20, то начните с краткого обзора для получения общей картины.

После получения общей картины вы можете переходить к ключевым возможностям языка (core language). Рассказ о каждой новой возможности языка является самодостаточным, но наилучшим вариантом будет прочтение всей книги от начала до конца. При первом прочтении вы можете пропустить то, что не упомянуто в кратком обзоре.

Личные замечания

Благодарности

Я просил читателей проверить на ошибки мой англоязычный блог Modernescpp¹ и получил гораздо больше откликов, чем ожидал. Огромное спасибо вам всем. Вот имена тех, кто внес в это свой вклад: Bob Bird, Nicola Bombace, Dave Burchill, Sandor Dargo, James Drobina, Frank Grimm, Kilian Henneberger, Ivan «espkk» Kondakov, Péter Kardos, Rakesh Mane, Jonathan O'Connor, John Plaice, Iwan Smith, Paul Targosz, Steve Vinoski и Greg Wagner.

Особенное спасибо моей дочери Джульетте и моей жене Беатрис. Джульетта поправила мой язык и исправила множество опечаток. Беатрис создала персонажа Сиппи (Cippi, девочка на рисунках) и иллюстрации к этой книге.

Сиппи



Позвольте представить вам Сиппи, которая будет сопровождать вас в этой книге. Надеюсь, она вам понравится.

Я Сиппи (по аналогии с Пеппи Длинныйчулок), любознательная, умная и женственная!

¹ <http://modernescpp.com>.

Редакторы русского перевода



Романов Александр Юрьевич (<https://www.hse.ru/staff/a.romanov>) – главный научный редактор русского перевода данной книги, доцент Московского института электроники и математики им. А. Н. Тихонова Национального исследовательского университета «Высшая школа экономики» (МИЭМ НИУ ВШЭ). С 2014 г. работает в МИЭМ НИУ ВШЭ, где возглавляет лабораторию САПР (<https://miem.hse.ru/edu/ce/cadsystem>), специализирующуюся на проектной деятельности, а также разработку цифровых систем на ПЛИС/микроконтроллерах, робототехнических комплексов, аппаратных реализаций систем искусственного интеллекта, многопроцессорных систем, систем удаленного доступа к ла-

бораторному оборудованию и т. д. В 2015 г. защитил диссертацию в Институте проблем проектирования в микроэлектронике РАН (г. Зеленоград), является автором более 150 научных статей, патентов и книг. А. Ю. Романов преподает C++ с 2009 г. в качестве лекционного и практического курса для студентов и использует данный язык постоянно в практической работе.



Романова Ирина Ивановна (<https://www.hse.ru/staff/iromanova>) – научный редактор русского перевода – занимается преподаванием компьютерных и инженерных дисциплин с 2010 г. В настоящее время работает старшим преподавателем в Московском институте электроники и математики им. А. Н. Тихонова Национального исследовательского университета «Высшая школа экономики» (МИЭМ НИУ ВШЭ), ведущий преподаватель и лектор дисциплины «Информатика» для студентов 1-го курса, а также соавтор известной книги «Цифровой синтез: практический курс» (М.: ДМК Пресс, 2020). Автор более 30 научных статей. Специализируется на методике преподавания компьютерных дисциплин студентам младших курсов.

Обо мне

Я работал в качестве архитектора программных систем, руководителя группы (team lead) и лектора начиная с 1999-го. В 2002 году я создавал тренинги для обучения. Я начал вести занятия с 2002 года. Моими первыми тренингами были занятия по управлению проприетарным программным обеспечением, но вскоре я начал обучать языкам Python и C++. В мое свободное время мне нравится писать статьи о C++, Python и Haskell. Также мне нравится выступать на различных конференциях. Каждую неделю я пишу что-то в моем англоязычном блоге ModernesCpp и в блоге на немецком языке¹, размещенном у Heise Developer.

С 2016 года я выступал в качестве независимого инструктора с семинарами по современному C++ и Python . Я написал несколько книг на разных языках о современном C++ и, в частности, по параллелизму. В связи с моей профессией я всегда ищу лучшие пути обучения современному C++.



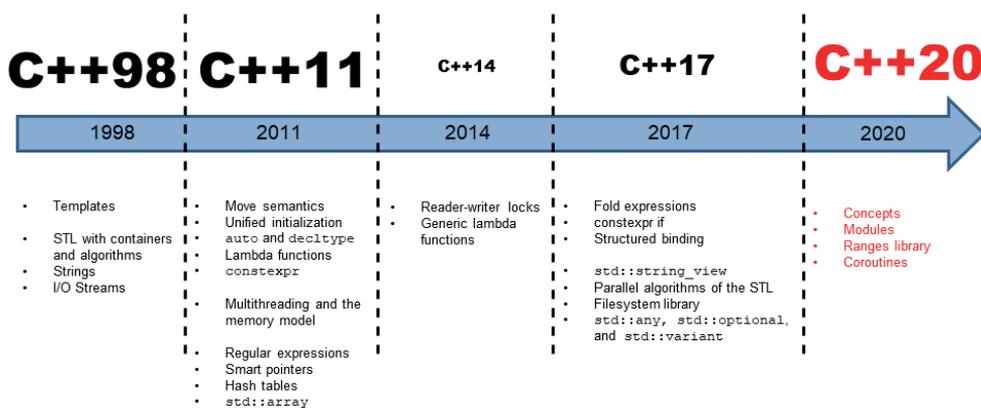
Рейнер Гримм

¹ <https://www.grimm-jaud.de/index.php/blog>.

0 C++

1. Исторический контекст

C++20 – это следующий большой шаг после стандарта C++11. Как и C++11, C++20 меняет способ, как мы программируем, используя современный C++. Эти изменения произошли в основном вследствие добавления в язык таких понятий, как концепты (Concepts), модули (Modules), диапазоны (Ranges) и корутины (Coroutines). Для понимания данного шага в эволюции C++ позвольте мне сказать несколько слов об историческом контексте C++20.



История C++

1.1 C++98

В конце 80-х Бьярн Страуструп и Маргарет А. Эллис написали свою знаменитую книгу *Annotated C++ Reference*¹ (ARM). Данная книга выполняла две функции – определяла функциональность C++ в мире, в котором существуют различные реализации C++, и стала основой для первого стандарта C++98 (ISO/IEC 14882). В число важных возможностей языка вошли шаблоны и стандартная библиотека шаблонов (Standard Template Library, STL) со своими контейнерами, алгоритмами и строками, а также потоки ввода/вывода.

¹ <https://stroustrup.com/arm.html>.

1.2 C++03

Со стандартом C++03 (14882:2003) C++98 получил небольшое уточнение, настолько небольшое, что ему даже не выделено места на таймлайне. В сообществе C++02, включающий C++98, обычно называется **legacy C++**.

1.3 TR1

В 2005 году произошло очень важное событие. Был опубликован документ под названием Technical Report 1. TR1 был огромным шагом к C++11 и, соответственно, современному C++. TR1 (TR 19768) был основан на проекте Boost¹, который был создан членами комитета по стандартизации C++. TR1 содержит 13 библиотек, которые должны были стать частью C++11. В их число входят такие, как библиотека регулярных выражений, библиотека для работы со случайными числами, умные указатели и хеш-таблицы. Только специальным математическим функциям пришлось ждать до C++17.

1.4 C++11

Мы называем C++11 *современным C++*. Это же название *современный C++* используется и для C++14 и C++17. C++11 внес много новых возможностей, которые принципиально изменили наше программирование на C++. Например, в C++11 вошли добавления из TR1, а также семантика перемещения (move semantics), форвардинг (perfect forwarding), вариадические шаблоны (variadic templates) и constexpr. Но это еще не все. С C++11 мы также получили, причем в первый раз, модель памяти как основу работы с нитями и стандартизированный API для работы с нитями.

1.5 C++14

C++14 – это довольно небольшой стандарт. Он привнес read-write блокировки, обобщенные лямбда-функции и расширенные constexpr-функции.

1.6 C++17

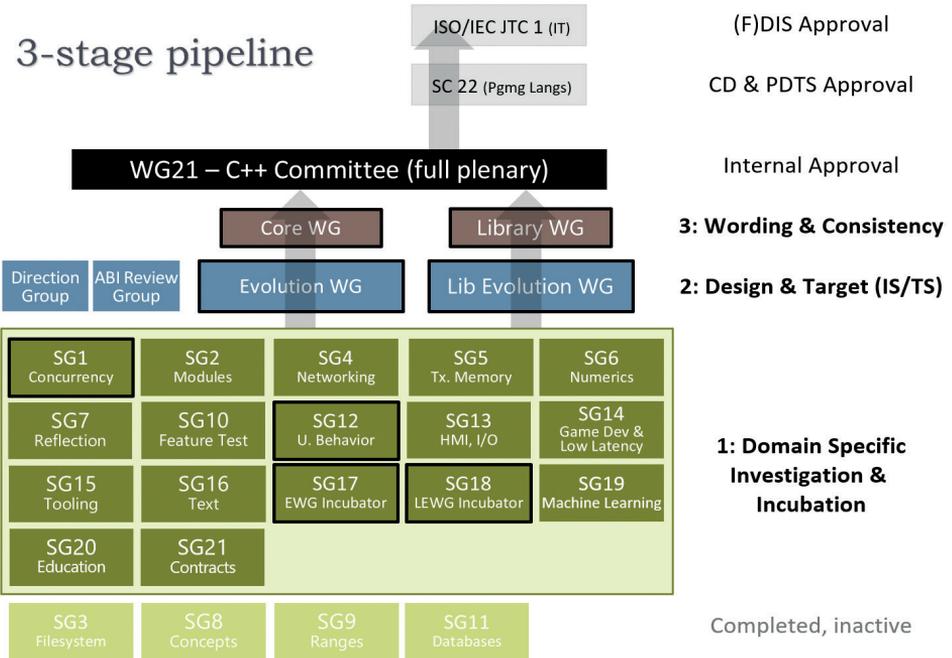
C++ не является ни большим, ни маленьким. Он привнес две выдающиеся возможности: параллельный STL и стандартный API для работы с файловой системой. Порядка 80 алгоритмов из стандартной библиотеки шаблонов могут выполняться параллельно или быть векторизованы. Как и с C++11, библиотеки boost оказали огромное влияние на C++17. Библиотеки boost предоставили библиотеку для работы с файловой системой и новые типы: `std::string_view`, `std::optional`, `std::variant` и `std::any`.

¹ <https://www.boost.org>.

2. Стандартизация

Процесс стандартизации C++ довольно демократичен. Комитет по стандартизации языка называется WG21 (Working Group 21) и был сформирован в 1990–1991 годах. Подкомитетами являются:

- организатор (Convener): возглавляет WG21, назначает график встреч и группы изучения;
- редактор проекта: применяет изменения к черновику стандарта C++;
- секретарь: назначает встречи WG21.



Группы изучения в процессе стандартизации C++

Комитет организован в конвейер из трех стадий, состоящий из нескольких подгрупп.

2.1 Стадия 3

Стадия 3 предназначена для уточнения формулировок и обеспечения непротиворечивости предлагаемых добавлений и состоит из двух групп: формулировок по самому языку (core language wording, CWG) и формулировок по библиотекам (library wording).

2.2 Стадия 2

Стадия 2 состоит из двух групп: развитие самого языка (Core Language Evolution, EWG) и развитие библиотек (Library Evolution, LWEG). EWG и LWEG отвечают за новые возможности, включающие в себя расширения языка и стандартных библиотек соответственно.

2.3 Стадия 1

Стадия 1 предназначена для изучения направлений развития языка. Члены групп встречаются лицом к лицу, между встречами по телефону и видеоконференциями. Центральные группы могут изучать результаты целевых групп для обеспечения согласованности и последовательности изменений языка.

Ниже приводится список целевых групп (Study Groups):

- **SG1, Concurrency** – параллельные вычисления, включая модель памяти;
- **SG2, Modules** – темы, связанные с модулями;
- **SG3, File system** – файловая система;
- **SG4, Networking** – развитие сетевой библиотеки;
- **SG5, Transactional memory** – транзакционная память для включения в будущие релизы;
- **SG6, Numerics** – численные расчеты, числа с фиксированной точкой, числа с плавающей точкой и дроби;
- **SG7, Compile time programming** – программирование в контексте компиляции программы;
- **SG8, Concepts** – концепты;
- **SG9, Ranges** – диапазоны;
- **SG10, Feature test** – переносимые тесты для проверки того, поддерживает ли конкретная реализация конкретную возможность (feature);
- **SG11, Databases** – интерфейсы для взаимодействия с базами данных;
- **SG12, UB & Vulnerabilities** – улучшения, направленные борьбу с уязвимостями и неопределенным/незаданным поведением текущей версии стандарта языка;
- **SG13 HMI & I/O (Human/Machine Interface)** – поддержка устройств ввода/вывода;
- **SG14, Game development & low latency** – поддержка требования работы с небольшой задержкой;
- **SG15, Tooling** – инструменты для разработчиков, включая модули и пакеты;
- **SG16, Unicode** – обработка на C++ текста на юникоде;
- **SG17, EWG Incubator** – ранние обсуждения развития языка;

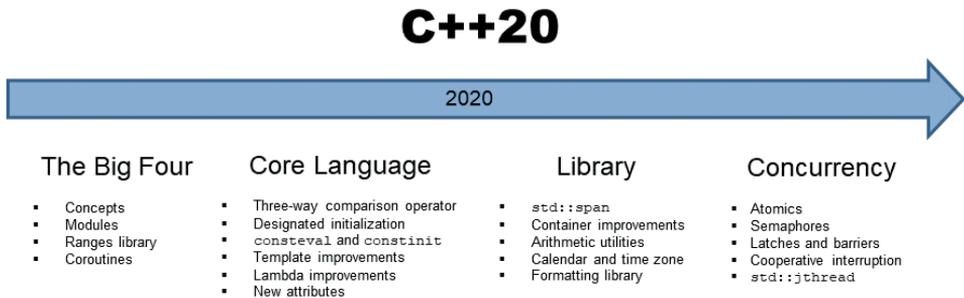
- **SG18, LWEG Incubator** – ранние обсуждения развития библиотек;
- **SG19, Machine Learning** – темы, связанные с искусственным интеллектом, и линейная алгебра;
- **SG20, Education** – материалы для обучающих курсов по C++;
- **SG21, Contracts** – поддержка языком обусловленного проектирования (Design by contract);
- **SG22 C/C++ Liason** – обсуждения взаимодействия C и C++.

Этот раздел дал вам краткий обзор стандартизации C++ и структуры комитета по стандартизации C++. Вы можете получить дополнительную информацию на сайте: <https://isocpp.org/std>.

КРАТКИЙ ОБЗОР C++20

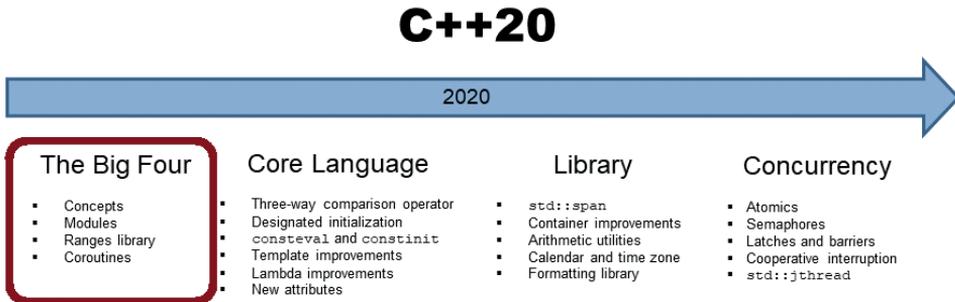
3. C++20

Прежде чем погружаться в детали C++20, я хотел бы дать краткий обзор возможностей C++20. Этот обзор служит сразу двум целям: дать первое впечатление и предоставить ссылки в соответствующие разделы, которые вы можете использовать для погружения в детали. Также в этой главе будут только фрагменты кода, а не завершенные программы.



В C++20 вошло несколько выдающихся возможностей: концепты, диапазоны, корутины и модули. Каждая заслуживает своего раздела.

3.1 Большая четверка



Каждая из новых возможностей языка меняет то, как мы программируем на современном C++. Давайте начнем с концептов.

3.1.1 Концепты

Обобщенное программирование с шаблонами позволяет определять функции и классы, которые могут быть использованы с различными типами данных. В результате бывает, что мы инстанцируем шаблон с неверным типом. В итоге мы можем получить много страниц непонятных ошибок. Эта проблема решается при помощи концептов. Концепты дают вам возможность описать требования для параметров шаблонов, которые будут проверяться компилятором, и тем самым улучшают то, как мы думаем и пишем обобщенный код. Это достигается благодаря тому, что:

- требования для параметров шаблона становятся частью публичного интерфейса шаблона;
- перегрузка функций или специализация классов может быть основана на концептах;
- мы получаем улучшенные сообщения об ошибках, поскольку компилятор проверяет требования к параметрам шаблона для переданных параметров.

Кроме того, у концептов есть еще преимущества:

- вы можете использовать уже готовые концепты для написания собственных;
- унифицируется использование ключевого слова `auto` и концептов. Вместо `auto` вы теперь можете использовать концепт;
- если объявление функции использует концепт, то эта функция автоматически становится шаблонной функцией. Написание шаблонных функций становится таким же легким, как и написание обычных функций.

Следующий фрагмент кода показывает определение и использование концепта `Integral`.

Определение и использование концепта `Integral`

```
template <typename T>
concept Integral = std::is_integral<T>::value;

Integral auto gcd(Integral auto a, Integral auto b) {
    if( b == 0 ) return a;
    else return gcd(b, a % b);
}
```

Концепт `Integral` требует от своего параметра-типа `T`, чтобы `std::is_integral<T>::value` было истинным. Оно, в свою очередь, является частью библиотеки трейтов (type traits), проверяющей во время компиляции, что данный тип является целым. Если `std::is_integral<T>::value` истинно, то все в порядке, иначе вы получите сообщение об ошибке.

Алгоритм `gcd` находит наибольший общий делитель, используя алгоритм Евклида¹. Этот фрагмент кода использует так называемый сокращенный синтаксис для функции-шаблона для определения `gcd`. Функция `gcd` требует, чтобы оба ее аргумента и выходной тип поддерживали концепт `Integral`. Другими словами, `gcd` – это что-то вроде шаблонной функции, задающей требования на входные аргументы и возвращаемый тип. Когда я уберу весь «синтаксический сахар», вы увидите природу этой функции.

Использование концепта `Integral` в опции `requires`

```
template<typename T>
requires Integral<T>
T gcd(T a, T b) {
    if( b == 0 ) return a;
    else return gcd(b, a % b);
}
```

Описатель `requires` задает требования на тип параметров `gcd`.

3.1.2 Модули

Модули обеспечивают:

- более быструю компиляцию;
- уменьшение необходимости использования макросов;
- более выраженную логическую структуру кода;
- делают устаревшими заголовочные файлы;
- помощь в избавлении от уродливых макрорешений.

Вот пример простого модуля `math`:

¹ <https://en.wikipedia.org/wiki/Euclid>.

Модуль `math`

```
1 export module math;
2
3 export int add(int fir, int sec) {
4     return fir + sec;
5 }
```

Выражение `export module math` (строка 1) является объявлением модуля. Поместив `export` перед описанием функции `add` (строка 3), мы экспортируем эту функцию. Теперь она может быть использована потребителем данного модуля.

Использование модуля `math`

```
import math;

int main() {
    add(2000, 20);
}
```

Выражение `import math` импортирует модуль `math` и делает экспортируемые имена видимыми в текущей области видимости.

3.1.3 Библиотека диапазонов (ranges)

Библиотека диапазонов (`range`) поддерживает алгоритмы, которые могут:

- работать напрямую с контейнерами, не требуя итераторов для задания диапазона;
- выполняться отложено (`lazy`);
- совмещаться/комбинироваться при помощи символа `|`.

Говоря проще: эта библиотека поддерживает функциональные шаблоны. Следующий пример демонстрирует композицию функций через символ `|`.

Композиция функций при помощи символа `|`

```
1 int main() {
2     std::vector<int> ints{0, 1, 2, 3, 4, 5};
3     auto even = [](int i){ return i % 2 == 0; };
4     auto square = [](int i) { return i * i; };
5
6     for (int i : ints | std::views::filter(even) |
7             std::views::transform(square)) {
8         std::cout << i << ' ';           // 0 4 16
9     }
10 }
```

Лямбда-выражение для `even` (строка 3) – это лямбда, возвращающая `true`, если ее аргумент четный. Лямбда `square` (строка 4) переводит свой аргумент в его квадрат. Строки 6 и 7 показывают композицию функций, которую нужно читать слева направо: `for (int i : ints | std::views::filter(even) | std::views::transform (square))`. К каждому элементу `ints` применяет фильтр `even`, и к оставшимся элементам применяется преобразование `square`. Если вы знакомы с функциональным программированием, то это совсем не ново.

3.1.4 Корутины

Корутины – это обобщенные функции, выполнение которых можно приостанавливать и возобновлять потом, сохраняя их состояние. Корутины очень удобны для написания событийно-ориентированных (event-driven) приложений. Такими приложениями могут быть симуляции, игры, серверы, пользовательские интерфейсы и даже алгоритмы. Корутины обычно используются для кооперативной многозадачности.

C++20 не предоставляет конкретных корутинов, вместо этого C++20 предоставляет фреймворк для написания корутинов. Этот фреймворк состоит из более чем 20 функций, некоторые из которых вы должны реализовать, некоторые можете переопределить. Таким образом можно приспособить корутины для своих целей.

Следующий фрагмент кода использует генератор для создания потенциально бесконечного потока данных. В главе «Корутины» содержится реализация `Generator`.

Генератор для получения бесконечного потока данных

```

1  Generator<int> getNext(int start = 0, int step = 1){
2      auto value = start;
3      while (true) {
4          co_yield value;
5          value += step;
6      }
7  }
8
9  int main() {
10
11     std::cout << '\n';
12
13     std::cout << "getNext():";
14     auto gen1 = getNext();
15     for (int i = 0; i <= 10; ++i) {
16         gen1.next();
17         std::cout << " " << gen1.getValue();
18     }
19

```

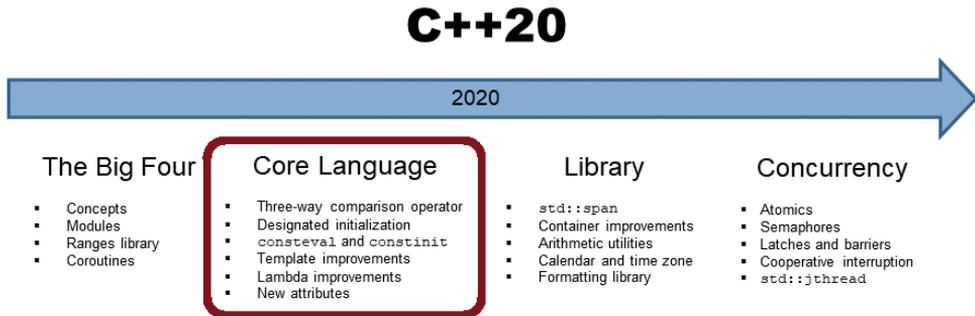
```
20     std::cout << "\n\n";
21
22     std::cout << "getNext(100, -10):";
23     auto gen2 = getNext(100, -10);
24     for (int i = 0; i <= 20; ++i) {
25         gen2.next();
26         std::cout << " " << gen2.getValue();
27     }
28
29     std::cout << "\n";
30
31 }
```

Функция `getNext` является корутином, поскольку она использует ключевое слово `co_yield`. Есть бесконечный цикл, который возвращает значение через `co_yield` (строка 4). После возвращения из вызова `getNext` корутин приостанавливается до следующего вызова. Есть один тонкий момент: возвращаемое значение – это `Generator<int>`. Здесь начинаются хитрости, описанные в разделе по корутинам.

```
Start
getNext(): 0 1 2 3 4 5 6 7 8 9 10
getNext(100, -10): 100 90 80 70 60 50 40 30 20 10 0 -10 -20 -30 -40 -50 -60 -70 -80 -90 -100
0
Finish
```

Бесконечный генератор данных

3.2 Ядро языка



3.2.1 Оператор трехстороннего сравнения

Оператор трехстороннего сравнения (spaceship operator) `<=>` сравнивает два значения – `A` и `B` – и определяет, что имеет место `A < B`, `A == B` или `A > B`.

Если вы определили оператор трехстороннего сравнения как `default`, то компилятор попытается сгенерировать соответствующий оператор для вашего класса. В этом случае вы также получите все шесть операторов сравнения: `==`, `!=`, `<`, `<=`, `>` и `>=`.

Автоматическое создание трехстороннего оператора сравнения

```
struct MyInt {
    int value;
    MyInt(int value): value{value} { }
    auto operator<=>(const MyInt&) const = default;
};
```

Созданный компилятором оператор `<=>` выполняет лексикографическое сравнение, начиная с базовых классов и используя все нестатические члены класса в порядке их описания. Чтобы более подробно разобраться с данным оператором, рекомендую довольно хитрый пример из блога Microsoft: [Simplify Your Code with Rocket Science: C++20's Spaceship operator](https://devblogs.microsoft.com/cppblog/simplify-your-code-with-rocket-science-c20s-spaceship-operator/)¹.

¹ <https://devblogs.microsoft.com/cppblog/simplify-your-code-with-rocket-science-c20s-spaceship-operator/>.

3.2.2 Назначенная инициализация

Оператор трехстороннего сравнения для производных классов

```

struct Basics {
    int i;
    char c;
    float f;
    double d;
    auto operator<=>(const Basics&) const = default;
};

struct Arrays {
    int ai[1];
    char ac[2];
    float af[3];
    double ad[2][2];
    auto operator<=>(const Arrays&) const = default;
};

struct Bases : Basics, Arrays {
    auto operator<=>(const Bases&) const = default;
};

int main() {
    constexpr Basics a = { { 0, 'c', 1.f, 1. },
        { { 1 }, { 'a', 'b' }, { 1.f, 2.f, 3.f }, { { 1., 2. }, { 3., 4. } } } };
    constexpr Basics b = { { 0, 'c', 1.f, 1. },
        { { 1 }, { 'a', 'b' }, { 1.f, 2.f, 3.f }, { { 1., 2. }, { 3., 4. } } } };
    static_assert(a == b);
    static_assert(!(a != b));
    static_assert(!(a < b));
    static_assert(a <= b);
    static_assert(!(a > b));
    static_assert(a >= b);
}

```

Я полагаю, что наиболее сложным в этом фрагменте кода является не оператор трехстороннего сравнения, а инициализация класса Base. Агрегированная инициализация (aggregate initialization) означает, что вы можете непосредственно проинициализировать члены класса (class, struct или union), если все они public. В этом случае вы можете использовать список инициализации в фигурных скобках, как в приведенном примере.

Прежде чем я приступлю к назначенной инициализации (designated initialization), давайте еще немного рассмотрим агрегатную инициализацию. Вот простой пример.

Агрегатная инициализация

```
struct Point2D{
    int x;
    int y;
};

class Point3D{
public:
    int x;
    int y;
    int z;
};

int main(){

    std::cout << "\n";

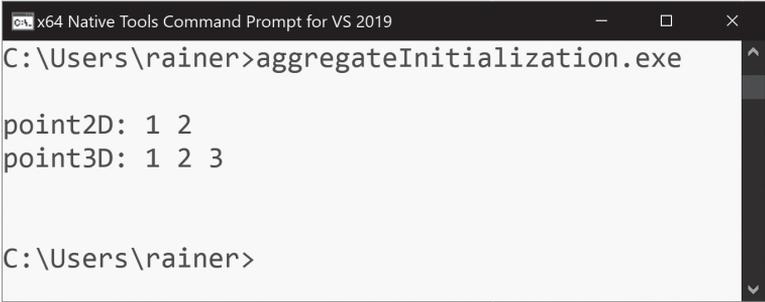
    Point2D point2D {1, 2};
    Point3D point3D {1, 2, 3};

    std::cout << "point2D: " << point2D.x << " " << point2D.y << "\n";
    std::cout << "point3D: " << point3D.x << " "
                << point3D.y << " " << point3D.z << "\n";

    std::cout << '\n';

}
```

Ниже приводится вывод данного примера.



```
x64 Native Tools Command Prompt for VS 2019
C:\Users\rainer>aggregateInitialization.exe

point2D: 1 2
point3D: 1 2 3

C:\Users\rainer>
```

Агрегатная инициализация очень чувствительная к ошибкам, поскольку вы можете поменять местами аргументы конструктора и никогда этого не заметить. Явная инициализация лучше неявной. Давайте посмотрим, что это значит. Взглянем на назначенную инициализацию из C99, которая стала частью стандарта C++.

Назначенная инициализация

```

1  struct Point2D{
2      int x;
3      int y;
4  };
5
6  class Point3D{
7  public:
8      int x;
9      int y;
10     int z;
11 };
12
13 int main(){
14
15     Point2D point2D { .x = 1, .y = 2 };
16     // Point2D point2d { .y = 2, .x = 1 };           // error
17     Point3D point3D { .x = 1, .y = 2, .z = 2 };
18     // Point3D point3D { .x = 1, .z = 2 }         // {1, 0, 2}
19
20
21     std::cout << "point2D: " << point2D.x << " " << point2D.y << "\n";
22     std::cout << "point3D: " << point3D.x << " " << point3D.y << " " << point3D.z
23         << "\n";
24
25 }
```

Аргументы для экземпляров Point2D и Point3D явно именованы. Вывод этой программы идентичен выводу предыдущей. Закомментированные строки 16 и 18 довольно интересны. Строка 16 дает ошибку, потому что порядок инициализаторов не соответствует порядку, в котором описаны члены класса. Что касается строки 18, то здесь пропущен инициализатор для y. В этом случае y будет проинициализирован нулем, так же как и при списке инициализации { 1, 0, 3}.

3.2.3 consteval и constexpr

Новый спецификатор `constexpr`, добавленный в C++20, создает непосредственную функцию (immediate function). Для такой функции каждый ее вызов должен происходить за константное время компиляции. Непосредственная функция – это просто `constexpr`-функция, но обратное не всегда верно.

Непосредственная функция

```
constexpr int sqr(int n) {
    return n*n;
}
constexpr int r = sqr(100); // OK

int x = 100;
int r2 = sqr(x);           // Error
```

Последнее присваивание приводит к ошибке, поскольку x – это не константное выражение, и поэтому вычисление $\text{sqr}(x)$ не может быть выполнено на этапе компиляции.

Опция `constexpr` гарантирует, что статические переменные или TLS-переменные будут проинициализированы во время компиляции. Статические переменные – это переменные, память для которых выделяется при запуске программы. У TLS-переменных время жизни привязано ко времени жизни потока.

`constexpr` гарантирует, что эти переменные будут проинициализированы во время компиляции. Но это не означает их константности.

3.2.4 Улучшения работы с шаблонами

C++20 предоставляет многочисленные улучшения для программирования шаблонов. Общий конструктор (generic constructor) – это универсальный конструктор, поскольку вы можете вызвать его с любым типом.

Неявный и явный общие конструкторы

```
struct Implicit {
    template <typename T>
    Implicit(T t) {
        std::cout << t << '\n';
    }
};

struct Explicit {
    template <typename T>
    explicit Explicit(T t) {
        std::cout << t << '\n';
    }
};

Explicit exp1 = "implicit"; // Error
Explicit exp2{"explicit"};
```

Общий конструктор класса `Implicit` слишком общий. Помещая ключевое слово `explicit` перед конструктором, как у класса `Explicit`, конструктор становится явным. Это значит, что неявные преобразования больше не валидны.

3.2.5 Улучшения лямбд

В C++20 лямбды получили много новых улучшений. У них могут быть шаблонные параметры, невычисляемый контекст и лямбды без собственного состояния могут быть созданы конструктором по умолчанию (`default constructor`) и конструктором копирования (`copy constructor`). Более того, компилятор может определять, когда вы неявно копируете указатель `this`, – это значит, заметная часть **неопределенного поведения** (`undefined behavior`) у лямбд ушла в прошлое.

Если вы хотите определить лямбду, которая принимает только `std::vector`, то шаблонные параметры для лямбд позволяют это сделать.

Шаблонные параметры для лямбды

```
auto foo = [] <typename T>(std::vector<T> const& vec) {  
    // do vector-specific stuff  
};
```

3.2.6 Новые атрибуты

В C++20 появились новые атрибуты, включая `[[likely]]` и `[[unlikely]]`. Оба этих атрибута позволяют дать подсказку оптимизатору, задавая, какой путь выполнения будет чаще или реже встречаться.

Атрибут `[[likely]]`

```
for(size_t i=0; i < v.size(); ++i){  
    if (v[i] < 0) [[likely]] sum -= sqrt(-v[i]);  
    else sum += sqrt(v[i]);  
}
```
