Оглавление

Вступительное слово от сообщества разработчиков России и Беларуси

Перед вами необычная, почти уникальная книга по C++17 с элементами лишь частично стандартизированного C++20. Любая система образования инерционна, в этой особенности заложена одновременно сила и слабость подхода – диалектика Гегеля в действии. С одной стороны, так называемый тезис: благодаря инерционности мы получаем проверенные временем, тщательно апробированные знания и способы их донесения до читателя. С другой стороны, так называемый антитезис: мы порой не можем найти ответы на актуальные вопросы здесь и сейчас, особенно в рамках академического образования.

Автор книги попытался добиться синтеза, и, как нам кажется, это ему удалось – написать современный академический учебник по C++17 с практическими задачами от математических, вычислительных до архитектурных, построенных на базе шаблонов проектирования.

Язык C++ переживает взрывное развитие, ниша применения систематически расширяется, примерно раз в 2–3 года выходит новый стандарт, что доказывает сверхактуальность подобных материалов по C++17/20. Книга будет очень полезна студентам профильных вузов, всем тем, кто решил освоить язык C++17 самостоятельно, и, конечно, IT-профессионалам, предпочитающим изучение нового стандарта через практическую, а не теоретическую призму.

Сообщество C++-разработчиков CoreHard,
Антон Наумович и Антон Семенченко.

cpp-russia.ru

stdcpp.ru

corehard.by

Об авторе

Маурис Бансила (Marius Bancila) – инженер-программист с 15-летним опытом разработки промышленных и финансовых решений. Автор книги «Modern C++ Programming Cookbook». Отдает предпочтение технологиям Microsoft и занимается в основном разработкой настольных приложений на C++ и C#.

С удовольствием делится своим опытом с другими, и поэтому вот уже больше десяти лет ему присваивается статус Microsoft MVP. Связаться с ним можно в Twitter: `@mariusbancila`.

Я хочу выразить благодарность Никхил Боркар (Nikhil Borkar), Джиджо Малийекал (Jijo Maliyekal), Чайтанья Наир (Chaitanya Nair), Нитин Дасан (Nitin Dasan) и всем другим сотрудникам издательства Packt, принявшим участие в создании этой книги. Также я хочу сказать спасибо рецензентам, чьи бесценные отзывы помогли мне улучшить книгу. Наконец, особое спасибо моей супруге и всей моей семье, поддерживавшим меня в процессе работы над этим проектом.

О рецензентах

Айварс Кальванс (Aivars Kalvāns) – ведущий архитектор программного обеспечения в Tieto Latvia. Уже больше 16 лет он работает над платежной системой Card Suite и сопровождает большое количество программ и библиотек на C++. Также он пишет руководства по программированию на C++ и ведет курсы по безопасному программированию. Организует внутренние встречи разработчиков на C++ и выступает на них.

Я хочу сказать спасибо моей любимой супруге Анете (Anete) и сыновьям, Карлису (Kārlis), Густавсу (Gustavs) и Лео (Leo), за то, что сделали мою жизнь краше.

Арун Муралидхаран(Arun Muralidharan) – программист с более чем 8-летним опытом разработки систем и комплексных приложений. Архитектуры распределенных систем, системы событий, масштабируемость, производительность и языки программирования – вот лишь некоторые из его интересов.

Истовый сторонник языка C++ и метапрограммирования с шаблонами; ему нравится, как язык сохраняет свою индивидуальность, продолжая динамично развиваться. Поэтому чаще всего его можно найти программирующим на C++.

Хочу поблагодарить сообщество C++, которое многому меня научило за прошедшие годы.

Нибедит Дей (Nibedit Dey) – технический предприниматель с богатым техническим опытом во многих сферах. Имеет степень бакалавра в области биоинженерии и магистра в области цифрового проектирования и разработки встраиваемых систем. Прежде чем начать карьеру технического предпринимателя, несколько лет работал в L&T и Tektronix, где занимался научно-исследовательскими и опытно-конструкторскими разработками. Последние 8 лет активно использует C++ для создания сложных программных систем.

О чем рассказывается в книге

Глава 1 «*Математические задачи*» содержит ряд математических упражнений, которые помогут вам разобраться в более сложных задачах в последующих главах.

Глава 2 «*Особенности языка*» предлагает задачи, решая которые, вы попрактикуетесь в перегрузке операторов, семантике перемещения, определении пользовательских литералов и аспектах метапрограммирования шаблонов, таких как функции с переменным количеством аргументов, выражения свертки и свойства типов (type traits).

Глава 3 «*Строки и регулярные выражения*» включает несколько задач по работе со строками, такие как преобразование между строками и другими типами данных, разбиение и объединение строк, а также задачи с регулярными выражениями.

Глава 4 «*Потоки данных и файловые системы*» охватывает управление потоком вывода, а также операции с файлами и каталогами с применением библиотеки `filesystem` в C++17.

Глава 5 «*Дата и время*» знакомит с грядущими расширениями к библиотеке `chrono` в C++20, содержит несколько задач по работе с датами и часовыми поясами, которые можно решать с помощью библиотеки `date`, являющейся основой новых дополнений к стандарту.

Глава 6 «*Алгоритмы и структуры данных*» – одна из самых длинных и содержит широкий спектр задач. Для решения одних вы должны будете использовать существующие стандартные алгоритмы; для других – написать свои обобщенные алгоритмы или структуры данных, такие как циклический буфер и приоритетная очередь. В конце главы приводятся две довольно забавные задачи, программа «Weasel» Докинза (Dawkins) и программа «Game of Life» Конвея (Conway), которые познакомят вас с эволюционными алгоритмами и клеточными автоматами.

Глава 7 «*Конкуренция*» потребует от вас использовать потоки выполнения (threads) и асинхронные функции для реализации обобщенных параллельных алгоритмов, а также решить некоторые проблемы конкурентного выполнения.

Глава 8 «*Шаблоны проектирования*» предлагает ряд задач, которые можно решить с применением шаблонов проектирования, таких как «Декоратор», «Компоновщик», «Цепочка обязанностей», «Шаблонный метод» и др.

Глава 9 «*Сериализация данных*» охватывает наиболее распространенные форматы сериализованных данных, JSON и XML; но также включает задачу по созданию файлов PDF с применением любых сторонних, открытых и кроссплатформенных библиотек.

Глава 10 «*Архивы, изображения и базы данных*» научит вас работать с архивами ZIP, создавать файлы PNG, например для Captcha-подобных систем, а также пользоваться встраиваемыми базами данных SQLite.

Глава 11 «*Криптография*» в основном охватывает приемы использования библиотеки Crypto++ для шифрования и подписывания данных. Также предложит вам реализовать свои утилиты кодирования и декодирования в/из формата Base64.

Глава 12 «*Сети и службы*» предложит вам реализовать свое клиент/серверное приложение, взаимодействующее по протоколу TCP/IP, а также продемонстрировать свою способность пользоваться разнообразными REST-службами, возвращающими информацию о курсе биткойна или выполняющими преобразование текста.

Вступление

C++ – универсальный язык программирования, сочетающий разные парадигмы, такие как объектно-ориентированное, императивное, обобщенное и функциональное программирование. C++ обладает высочайшей эффективностью и предназначен в первую очередь для разработки программ, где производительность имеет ключевое значение. В течение последних нескольких десятилетий C++ остается одним из наиболее широко используемых языков в индустрии, академических кругах и вообще везде. Стандарт языка разрабатывается международной организацией по стандартизации International Organization for Standardization (ISO), которая в настоящее время работает над следующей версией стандарта C++20, выход которого ожидается в 2020 г.

Описание стандарта занимает почти 1500 страниц, соответственно, C++ – не самый простой язык для изучения. Приобрести навыки невозможно, просто читая описание стандарта или наблюдая, как программируют другие, обязательно нужна повседневная практика. Разработчики не изучают новые языки или технологии, просто читая книги, статьи или просматривая видеоуроки. Чтобы овладеть чем-то новым, они должны постоянно практиковаться, пробовать и разрабатывать новые приемы. Однако поиск хороших упражнений для развития и закрепления знаний – сложная задача. В Интернете имеется много веб-сайтов, где можно найти упражнения для разных языков программирования, но задачи, предлагаемые на них, в основном имеют отношение к математике и алгоритмам и предназначены для студентов. Такие упражнения не помогают в освоении широкого спектра возможностей языка программирования. И в этом отношении данная книга опережает их.

В этой книге собраны 100 практических задач, которые помогут вам применить на практике разнообразные возможности языка C++ и его стандартной библиотеки, а также опробовать множество сторонних, кроссплатформенных библиотек. Подавляющее большинство задач не связано непосредственно с C++, и их можно решить на многих других языках. Но цель книги состоит в том, чтобы помочь вам освоить именно C++, поэтому вы должны решать задачи на C++. Все решения в книге написаны на C++. Однако вы можете использовать книгу как справочник по приемам решения представленных задач, хотя в этом случае вы не получите главной выгоды – практическое освоения языка.

Задачи в книге сгруппированы в 12 глав. Каждая глава содержит задачи, связанные с одной или несколькими близкими темами. Задачи имеют разный уровень сложности; некоторые – простые, некоторые имеют умеренную сложность, а некоторые очень сложные. Для разных уровней сложности предлагается примерно одинаковое количество задач. Каждая глава начинается с описания предлагаемых задач. Решения представлены в виде рекомендаций, пояснений и исходного кода. В книге приводятся решения всех задач, но,

прежде чем вы перейдете к их изучению, постарайтесь сначала реализовать свое решение и только потом (или если у вас возникнут сложности) переходите к предлагаемым вариантам. В примерах исходного кода отсутствует только одна деталь – заголовки, которые вы должны подключить. Это было сделано намеренно, чтобы заставить вас заняться самостоятельными исследованиями. С другой стороны, к книге предлагаются законченные примеры решений, где вы сможете увидеть все необходимые заголовки.

На момент написания этой книги стандарт C++20 еще продолжал разрабатываться, и ему предстояло разрабатываться еще несколько лет. Однако некоторые особенности уже были утверждены, и одна из них – библиотека-расширение `chrono` с функциями для работы с датами и часовыми поясами. В главе 5 вы найдете несколько задач, посвященных этой теме, и хотя пока нет компиляторов, поддерживающих упомянутую библиотеку, вы сможете решить эти задачи с использованием библиотеки `date`, на основе которой разрабатываются новые расширения. В книге также используется множество других библиотек, в том числе Asio, Crypto++, Curl, NLohmann/json, PDF-Writer, PNGWriter, pugixml, SQLite и ZipLib. В качестве альтернативы библиотекам `std::optional` и `filesystem` вы можете использовать библиотеку Boost с компиляторами, где эти библиотеки отсутствуют. Все упомянутые библиотеки распространяются с открытым исходным кодом и являются кроссплатформенными. При их выборе я руководствовался следующими причинами: высокая производительность, хорошая документация и широкое распространение в сообществе. Однако вы можете использовать для решения задач любые другие библиотеки.

Кому адресована эта книга

Изучаете C++ и ищете практические упражнения? Тогда эта книга для вас. Книга адресована изучающим C++ независимо от их опыта использования других языков программирования и содержит практические упражнения по решению повседневных задач. Здесь вы не найдете подробного описания особенностей языка или стандартной библиотеки. Вы сможете узнать о них из других источников – книг, статей, видеоуроков. Эта книга написана в помощь обучающимся и предлагает задачи разной сложности, для решения которых вы сможете использовать знания, полученные из других источников. Многие задачи в этой книге не связаны непосредственно с конкретным языком, и вы можете использовать их в процессе освоения других языков программирования; но в этом случае вы не сможете воспользоваться представленными здесь решениями.

Что необходимо, чтобы извлечь максимум пользы из книги

Как упоминалось выше, чтобы извлечь пользу из этой книги, вы должны иметь базовое знакомство с языком C++ и стандартной библиотекой или знакомить-

ся с ним в процессе чтения. В любом случае, эта книга научит вас решать задачи, но не научит вас языку и его особенностям, используемым в решениях. Вам понадобится компилятор с поддержкой C++17; полный список необходимых библиотек, а также доступных компиляторов вы найдете в списке «Software Hardware List», входящем в состав примеров решений. В следующих разделах вы найдете подробные инструкции, описывающие, как загружать и собирать код примеров для этой книги.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф в разделе Читателям – Файлы к книгам.

После загрузки файла архива распакуйте его, использовав последнюю версию:

- WinRAR/7-Zip в Windows;
- Zipeg/iZip/UnRarX в Mac;
- 7-Zip/PeaZip в Linux.

Пакет с исходным кодом примеров доступен также в репозитории GitHub по адресу: https://github.com/PacktPublishing/The-Modern-Cpp-Challenge.

Сборка примеров

Несмотря на использование в книге большого количества сторонних библиотек, все эти библиотеки, а также решения, представленные в книге, являются кроссплатформенными и работают на всех основных платформах. Однако сам код примеров разрабатывался и тестировался в Visual Studio 2017 v15.6/7 в Windows 10 и в Xcode 9.3 в Mac OS 10.13.x.

Для тех, кто пользуется Xcode в Mac, отмечу, что набор инструментов LLVM в Xcode не поддерживает две особенности: библиотеки `filesystem` и `std::optional`. Однако обе они спроектированы на основе библиотек `Boost.Filesystem` и `Boost.Optional`, и ими легко можно заменить стандартные библиотеки, используемые в решениях. На самом деле код загружаемых примеров написан так, что может работать с любым из этих двух вариантов; выбор осуществляется с помощью нескольких макросов. Инструкции по сборке с той или иной библиотекой приводятся ниже, однако эта же информация доступна в архиве с исходным кодом.

Для поддержки как можно более широкого круга окружений разработки и систем сборки примеры сопровождаются сценариями CMake. Они применяются для создания проектов и сценариев сборки для комплекта инструментов по вашему выбору. Если у вас не установлен инструмент CMake, вы можете получить его на сайте https://cmake.org/. Далее следуют инструкции, описывающие использование сценариев CMake для создания проектов Visual Studio и

Xcode. Если вам понадобится сгенерировать проект для другой среды разработки, обращайтесь к документации CMake.

Как сгенерировать проекты для Visual Studio 2017

Выполните следующие шаги, чтобы сгенерировать проекты для Visual Studio 2017 на платформе x86:

1. Откройте командную строку, перейдите в каталог `build` в корневой папке с примерами исходного кода.
2. Запустите следующую команду:

```
cmake -G "Visual Studio 15 2017" .. -DCMAKE_USE_WINSSL=ON
-DCURL_WINDOWS_SSPI=ON -DCURL_LIBRARY=libcurl
-DCURL_INCLUDE_DIR=..\libs\curl\include -DBUILD_TESTING=OFF
-DBUILD_CURL_EXE=OFF -DUSE_MANUAL=OFF
```

3. После этого вы найдете решение для Visual Studio в `build/cppchallenger.sln`.

Чтобы использовать платформу x64, используйте генератор с именем "`Visual Studio 15 2017 Win64`". Версия Visual Studio 2017 15.4 поддерживает обе библиотеки, `filesystem` (как экспериментальную) и `std::optional`. Если у вас Visual Studio более ранней версии или просто желаете использовать библиотеки Boost, вы сможете сгенерировать проекты следующей командой, после установки Boost:

```
cmake -G "Visual Studio 15 2017" .. -DCMAKE_USE_WINSSL=ON
-DCURL_WINDOWS_SSPI=ON -DCURL_LIBRARY=libcurl
-DCURL_INCLUDE_DIR=..\libs\curl\include -DBUILD_TESTING=OFF
-DBUILD_CURL_EXE=OFF -DUSE_MANUAL=OFF -DBOOST_FILESYSTEM=ON
-DBOOST_OPTIONAL=ON -DBOOST_INCLUDE_DIR=<путь_к_заголовкам>
-DBOOST_LIB_DIR=<путь_к_библиотекам>
```

Обратите внимание: пути к заголовкам и файлам статических библиотек *не* должны завершаться обратным слешем (\).

Как сгенерировать проекты для Xcode

Некоторые решения в последней главе используют библиотеку `libcurl`. Для поддержки SSL эта библиотека должна быть скомпонована с библиотекой OpenSSL. Выполните следующие шаги, чтобы установить OpenSSL:

1. Загрузите библиотеку с сайта https://www.openssl.org/.
2. Распакуйте загруженный архив в терминале, перейдите в корневой каталог распакованного архива.
3. Соберите и установите библиотеку следующими командами (выполните их в указанном порядке):

```
./Configure darwin64-x86_64-cc shared enable-ec_nistp_64_gcc_128 no-ssl2
no-ssl3 no-comp --openssldir=/usr/local/ssl/macos-x86_64

make depend

sudo make install
```

Несмотря на то что библиотеки `std::optional` и `filesystem` доступны в Xcode Clang, вы должны использовать Boost. Выполните следующие шаги, чтобы установить и собрать библиотеки Boost:

1. Установите диспетчер пакетов Homebrew с сайта https://brew.sh/.
2. Выполните следующую команду, чтобы автоматически загрузить и установить Boost.

   ```
   brew install boost
   ```

3. После установки библиотека Boost будет доступна в каталоге `/usr/local/Cellar/boost/1.65.0`.

Чтобы сгенерировать проекты для Xcode, выполните следующие шаги:

1. Откройте терминал и перейдите в каталог `build build` в корневой папке с примерами исходного кода.
2. Выполните следующую команду:

   ```
   cmake -G Xcode .. -DOPENSSL_ROOT_DIR=/usr/local/bin
   -DOPENSSL_INCLUDE_DIR=/usr/local/include/ -DBUILD_TESTING=OFF
   -DBUILD_CURL_EXE=OFF -DUSE_MANUAL=OFF -DBOOST_FILESYSTEM=ON
   -DBOOST_OPTIONAL=ON -DBOOST_INCLUDE_DIR=/usr/local/Cellar/boost/1.65.0
   -DBOOST_LIB_DIR=/usr/local/Cellar/boost/1.65.0/lib
   ```

3. После этого вы найдете проект для Xcode в каталоге `build/cppchallenger.xcodeproj`.

Соглашения

В этой книге используется несколько разных стилей оформления текста с целью обеспечить визуальное отличие информации разных типов.

Программный код в тексте, имена таблиц баз данных, имена папок, имена файлов, расширения файлов, пути в файловой системе, ввод пользователя и ссылки в Twitter оформляются, как показано в следующем предложении: «Смонтируйте загруженный файл образа диска `WebStorm-10*.dmg`.».

Блоки программного кода оформляются так:

```
int main()
{
    std::cout << "Hello, World!\n";
}
```

Когда нам потребуется привлечь ваше внимание к определенному фрагменту в блоке программного кода, мы будем выделять его жирным шрифтом:

```
template<typename C, typename... Args>
void push_back(C& c, Args&&... args)
{
    (c.push_back(args), ...);
}
```

Ввод и вывод в командной строке будут оформляться так:

```
$ mkdir build
$ cd build
```

Новые термины и **важные определения**, а также **надписи на экране** будут выделяться в обычном тексте жирным. Например: «Выберите в панели **Administration** (Администрирование) ярлык **System info** (Информация о системе)».

Таким значком будут оформляться предупреждения и важные примечания.

Таким значком будут оформляться советы и рекомендации .

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки всё равно случаются. Если вы найдёте ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги.

Если вы найдёте какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в Интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в Интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли принять меры.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Глава 1

Математические задачи

Задачи

1. Сумма натуральных чисел, кратных 3 и 5

Напишите программу, которая вычислит и выведет сумму всех натуральных чисел, кратных 3 или 5, вплоть до числа, введенного пользователем.

2. Наибольший общий делитель

Напишите программу, принимающую два положительных целых числа, которая вычислит и выведет их наибольший общий делитель.

3. Наименьшее общее кратное

Напишите программу, принимающую два положительных целых числа, которая вычислит и выведет их наименьшее общее кратное.

4. Наибольшее простое число меньше заданного

Напишите программу, которая вычислит и выведет наибольшее простое число, меньше указанного пользователем (это должно быть положительное целое число).

5. Простые числа, отличающиеся на шесть[1]

Напишите программу, которая выведет все пары простых чисел, отличающихся на шесть, вплоть до числа, введенного пользователем.

6. Избыточные числа

Напишите программу, которая выведет все избыточные числа и величину их избыточности, вплоть до числа, введенного пользователем.

---

[1] В английском языке такие пары чисел называют «sexy primes» (от латинского названия «числа шесть» – «sex»), что добавляет термину забавную двусмысленность ввиду возможной трактовки англ. «sexy primes» как «сексуальные (возбуждающие, привлекательные) простые числа». – *Прим. перев.*

7. Дружественные числа

Напишите программу, которая выведет все пары дружественных чисел меньше 1 000 000.

8. Числа Армстронга

Напишите программу, которая выведет все числа Армстронга с тремя цифрами.

9. Простые множители числа

Напишите программу, которая выведет простые множители числа, введенного пользователем.

10. Код Грея

Напишите программу, которая для всех 5-битных чисел выведет их обычное двоичное представление, представление в виде кода Грея и декодированный код Грея.

11. Преобразование десятичных чисел в римские

Напишите программу, которая для заданного десятичного значения выведет его эквивалент в виде римского числа.

12. Наибольшая последовательность Коллатца

Напишите программу, которая определит и выведет число, меньшее 1 000 000, порождающее наибольшую последовательность Коллатца, и длину этой последовательности.

13. Вычисление значения числа π

Напишите программу, вычисляющую значение числа π с точностью до двух знаков после запятой.

14. Проверка действительности номеров ISBN

Напишите программу, проверяющую действительность 10-значных номеров ISBN-10, введенных пользователем в виде строки.

Решения

1. Сумма натуральных чисел, кратных 3 и 5

Эта задача решается перебором всех чисел от 3 (1 и 2 не делятся на 3, поэтому проверять их не имеет смысла) вплоть до числа, введенного пользователем.

Подходящие числа делятся на 3 и 5 без остатка, и для этой проверки можно использовать оператор деления по модулю (%). Однако есть одна хитрость: переменная для накопления суммы должна быть объявлена с типом `long long`, а не `int` или `long`, иначе произойдет *переполнение*, если пользователь введет число больше 100 000:

```
int main()
{
  unsigned int limit = 0;
  std::cout << "Upper limit:";
  std::cin >> limit;

  unsigned long long sum = 0;
  for (unsigned int i = 3; i < limit; ++i)
  {
    if (i % 3 == 0 || i % 5 == 0)
      sum += i;
  }

  std::cout << "sum=" << sum << std::endl;
}
```

2. Наибольший общий делитель

Наибольший общий делитель (НОД) двух и более ненулевых целых чисел – это наибольшее положительное целое число, на которое все эти числа делятся без остатка. Есть несколько способов вычисления НОД, но наиболее эффективным считается алгоритм Эвклида:

```
НОД(a,0) = a
НОД(a,b) = НОД (b, a mod b)
```

Он легко реализуется на C++ в виде рекурсивной функции:

```
unsigned int gcd(unsigned int const a, unsigned int const b)
{
  return b == 0 ? a : gcd(b, a % b);
}
```

А вот так выглядит нерекурсивная реализация алгоритма Эвклида:

```
unsigned int gcd(unsigned int a, unsigned int b)
{
  while (b != 0) {
    unsigned int r = a % b;
    a = b;
    b = r;
  }
```

```
  return a;
}
```

В C++17 имеется функция `constexpr gcd()`, объявленная в заголовке `<numeric>`, которая вычисляет наибольший общий делитель двух чисел.

3. Наименьшее общее кратное

Наименьшее общее кратное (НОК) двух и более ненулевых целых чисел – это наименьшее положительное целое число, кратное всем этим числам. Одно из решений задачи поиска наименьшего общего кратного сводится к задаче поиска наибольшего общего делителя, согласно следующей формуле:

```
НОК(a, b) = abs(a, b) / НОД(a, b)
```

Вот как выглядит функция вычисления наименьшего общего кратного:

```
int lcm(int const a, int const b)
{
  int h = gcd(a, b);
  return h ? (a * (b / h)) : 0;
}
```

Для вычисления НОК для трех и более чисел можно использовать алгоритм `std::accumulate` из заголовка `<numeric>`:

```
template<class InputIt>
int lcmr(InputIt first, InputIt last)
{
  return std::accumulate(first, last, 1, lcm);
}
```

В C++17 имеется функция `constexpr lcm()`, объявленная в заголовке `<numeric>`, которая вычисляет наименьшее общее кратное двух чисел.

4. Наибольшее простое число меньше заданного

Простыми называют числа, которые делятся без остатка только на самих себя и на 1. Чтобы найти наибольшее простое число меньше заданного, вы должны написать функцию, определяющую – является ли число простым. А затем

вызывать ее для каждого числа, начиная с заданного и заканчивая 1, пока не встретится первое простое число. Существуют разные алгоритмы определения простого числа. Вот один из них:

```
bool is_prime(int const num)
{
  if (num <= 3) { return num > 1; }
  else if (num % 2 == 0 || num % 3 == 0)
  {
    return false;
  }
  else
  {
    for (int i = 5; i * i <= num; i += 6)
    {
      if (num % i == 0 || num % (i + 2) == 0)
      {
        return false;
      }
    }
    return true;
  }
}
```

А вот как можно использовать эту функцию для решения задачи:

```
int main()
{
  int limit = 0;
  std::cout << "Upper limit:";
  std::cin >> limit;

  for (int i = limit; i > 1; i--)
  {
    if (is_prime(i))
    {
      std::cout << "Largest prime:" << i << std::endl;
      return 0;
    }
  }
}
```

5. Простые числа, отличающиеся на шесть

Простые числа, отличающиеся на шесть, – это пары целых чисел, разность которых равна шести (например, 5 и 11 или 13 и 19). Существуют также *простые близнецы*, отличающиеся на два, *простые двоюродные числа*, отличающиеся на четыре.

В решении предыдущей задачи мы реализовали функцию, которая выясняет, является ли целое число простым. Ее можно использовать в решении данной задачи. Для этого нужно просто проверить, являются ли простыми числа *n* и *n*+6, и вывести их:

```
int main()
{
  int limit = 0;
  std::cout << "Upper limit:";
  std::cin >> limit;

  for (int n = 2; n <= limit; n++)
  {
    if (is_prime(n) && is_prime(n+6))
    {
      std::cout << n << "," << n+6 << std::endl;
    }
  }
}
```

В качестве дополнительного упражнения можете попробовать найти и вывести тройки простых чисел, отличающихся на шесть, четверки и пятерки.

6. Избыточные числа

Избыточным называют число, сумма положительных собственных делителей которого превышает само число. Собственные делители числа – это положительные простые делители, отличные от самого числа. Величина, на которую сумма собственных делителей превышает само число, называется его избыточностью. Например, число 12 имеет собственные делители 1, 2, 3, 4 и 6. Их сумма равна 16, то есть число 12 является избыточным. Его избыточность равна 4 (то есть 16 – 12).

Чтобы найти сумму всех собственных делителей, нужно опробовать все числа от 2 до квадратного корня числа (все простые делители меньше или равны этому значению). Если текущее число, назовем его *i*, делит число *num* без остатка, тогда оба числа, *i* и *num*/*i*, являются делителями. Но если они равны (например, если *i* = 3 и *num* = 9, тогда *i* делит 9, но *n*/i = 3), в сумму добавляется только *i*, потому что каждый делитель должен быть добавлен только один раз. Иначе в сумму добавляются оба числа, *i* и *num*/*i*:

```
int sum_proper_divisors(int const number)
{
  int result = 1;
  for (int i = 2; i <= std::sqrt(number); i++)
  {
    if (number%i == 0)
    {
```

```
        result += (i == (number / i)) ? i : (i + number / i);
      }
    }
    return result;
  }
```

Вывод избыточных чисел реализуется как простой цикл до указанного предела, вычисляющий сумму собственных делителей и сравнивающий ее с самим числом:

```
void print_abundant(int const limit)
{
  for (int number = 10; number <= limit; ++number)
  {
    auto sum = sum_proper_divisors(number);
    if (sum > number)
    {
      std::cout << number << ", abundance="
      << sum - number << std::endl;
    }
  }
}

int main()
{
  int limit = 0;
  std::cout << "Upper limit:";
  std::cin >> limit;

  print_abundant(limit);
}
```

7. Дружественные числа

Два числа называют дружественными, если сумма собственных делителей одного из них равна другому числу. Собственные делители числа – это положительные простые делители, отличные от самого числа. Не путайте *дружественные числа* (amicable numbers) с *компанейскими числами* (friendly numbers). Например, число 220 имеет собственные делители 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 и 110, сумма которых равна 284. Число 284 имеет собственные делители 1, 2, 4, 71 и 142; их сумма равна 220. То есть числа 220 и 284 являются дружественными.

Задача решается перебором всех чисел до заданного предела. Для каждого числа определяется сумма его собственных делителей. Назовем ее `sum1`. Затем процесс повторяется, и определяется сумма собственных делителей числа `sum1`. Если результат получился равным первому числу, значит, это число и `sum1` – дружественные числа:

```
void print_amicables(int const limit)
{
  for (int number = 4; number < limit; ++number)
  {
    auto sum1 = sum_proper_divisors(number);
    if (sum1 < limit)
    {
      auto sum2 = sum_proper_divisors(sum1);
      if (sum2 == number && number != sum1)
      {
        std::cout << number << "," << sum1 << std::endl;
      }
    }
  }
}
```

Функция `sum_proper_divisors()`, используемая в этом решении, взята из решения задачи об избыточных числах.

Функция выше выведет пары чисел дважды, например «220,284» и «284,220». Измените реализацию, чтобы каждая пара выводилась только один раз.

8. Числа Армстронга

Числа Армстронга (названные в честь Майкла Ф. Армстронга (Michael F. Armstrong)), их также называют *самовлюбленными числами* и *совершенными цифровыми инвариантами*, – это числа, равные сумме своих цифр, возведенных в степень, равную количеству цифр. Например, наименьшее число Армстронга – 153, которое равно $1^3 + 5^3 + 3^3$.

Чтобы определить, является ли трехзначное число самовлюбленным, нужно сначала разбить его на цифры, чтобы потом подсчитать сумму их степеней. Однако для этого требуется использовать довольно дорогостоящие операции деления. Намного проще положиться на тот факт, что число – это сумма цифр, умноженных на 10 в степени, соответствующей позиции цифры. Иными словами, для чисел до 1000 мы имеем: `a*10^2 + b*10^1 + c`. Поскольку по условиям задачи требуется найти только числа Армстронга с тремя цифрами, можно утверждать, что `a` начинается с 1. Это решение будет работать быстрее других, потому что умножения выполняются компьютерами быстрее, чем деление. Вот как выглядит реализация нужной нам функции:

```
void print_narcissistics()
{
```

```
  for (int a = 1; a <= 9; a++)
  {
    for (int b = 0; b <= 9; b++)
    {
      for (int c = 0; c <= 9; c++)
      {
        auto abc = a * 100 + b * 10 + c;
        auto arm = a * a * a + b * b * b + c * c * c;
        if (abc == arm)
        {
          std::cout << arm << std::endl;
        }
      }
    }
  }
}
```

В качестве дополнительного упражнения можете написать функцию, которая будет определять, является ли число самовлюбленным, независимо от количества цифр в нем. Такая функция может получиться довольно медленной, потому что сначала ей придется определить последовательность цифр числа, сохранить их в контейнере, а затем сложить их степени (по числу цифр).

9. Простые множители числа

Простые множители положительного целого числа – это простые числа, которые делят данное число без остатка. Например, простые множители числа 8 – это 2 × 2 × 2, а простые множители числа 42 – это 2 × 3 × 7. Для определения простых множителей можно использовать следующий алгоритм:

1. Если n делится на 2, 2 – простой множитель и его следует добавить в список, после этого, пока возможно, n продолжает делиться на 2. По завершении этого шага у нас остается нечетное число n.
2. Выполнить перебор чисел от 3 до квадратного корня из n. Если текущее число, назовем его i, делит n без остатка и является простым, его следует добавить в список, и у нас остается результат деления n/i. После этого, пока возможно, n продолжает делиться на i, и затем i увеличивается на 2 (чтобы получить следующее нечетное число).
3. Если n – простое число, большее 2, шаги выше не приведут к превращению n в 1. Следовательно, если в конце шага 2 число n остается больше 2, значит, оно является простым множителем.

```
std::vector<unsigned long long> prime_factors(unsigned long long n)
{
  std::vector<unsigned long long> factors;
  while (n % 2 == 0) {
```

```
    factors.push_back(2);
    n = n / 2;
  }
  for (unsigned long long i = 3; i <= std::sqrt(n); i += 2)
  {
    while (n%i == 0) {
      factors.push_back(i);
      n = n / i;
    }
  }

  if (n > 2)
    factors.push_back(n);
  return factors;
}

int main()
{
  unsigned long long number = 0;
  std::cout << "number:";
  std::cin >> number;

  auto factors = prime_factors(number);
  std::copy(std::begin(factors), std::end(factors),
    std::ostream_iterator<unsigned long long>(std::cout, " "));
}
```

В качестве дополнительного упражнения определите наибольший простой множитель числа 600 851 475 143.

10. Код Грея

Код Грея, также известный как *рефлексивный двоичный код*, – это форма двоичного кодирования, в которой две соседние кодовые комбинации отличаются только одним битом. Кодирование в рефлексивный двоичный код выполняется по следующей формуле:

```
if b[i-1] = 1 then g[i] = not b[i]
else g[i] = b[i]
```

Она эквивалентна такому выражению:

```
g = b xor (b логический сдвиг вправо на 1)
```

Для декодирования рефлексивного двоичного кода используется следующая формула:

```
b[0] = g[0]
b[i] = g[i] xor b[i-1]
```

На языке C++ это можно записать (с использованием 32-битных беззнаковых целых чисел) так:

```
unsigned int gray_encode(unsigned int const num)
{
  return num ^ (num >> 1);
}

unsigned int gray_decode(unsigned int gray)
{
  for (unsigned int bit = 1U << 31; bit > 1; bit >>= 1)
  {
    if (gray & bit) gray ^= bit >> 1;
  }
  return gray;
}
```

Вывести все 5-битные целые числа с их двоичными представлениями, кодами Грея и декодированными значениями можно с помощью следующего кода:

```
std::string to_binary(unsigned int value, int const digits)
{
  return std::bitset<32>(value).to_string().substr(32-digits, digits);
}

int main()
{
  std::cout << "Number\tBinary\tGray\tDecoded\n";
  std::cout << "------\t------\t----\t-------\n";

  for (unsigned int n = 0; n < 32; ++n)
  {
    auto encg = gray_encode(n);
    auto decg = gray_decode(encg);

    std::cout
      << n << "\t" << to_binary(n, 5) << "\t"
      << to_binary(encg, 5) << "\t" << decg << "\n";
  }
}
```

11. Преобразование десятичных чисел в римские

Римские числа, известные в наши дни, записываются с помощью семи символов: I = 1, V = 5, X = 10, L = 50, C = 100, D = 500 и M = 1000. Система использует операции сложения и вычитания при составлении числовых символов. Символы от 1 до 10: I, II, III, IV, V, VI, VII, VIII, IX и X. У римлян не было символа, пред-

ставляющего нуль, и для его обозначения использовалось слово *nulla*. В этой системе символы с наибольшим значением находятся слева, а с наименьшим – справа. Например, число 1994 в римской системе записи имеет вид: MCMXCIV. Если вы незнакомы с правилами записи римских чисел, поищите их в Интернете.

Для представления чисел в римской системе записи используется следующий алгоритм:

1. Проверить уместность каждого основного римского символа, начиная с наибольшего (M).
2. Если текущее число больше значения символа, римский символ добавляется в запись, и его значение вычитается из текущего числа.
3. Процесс повторяется, пока не будет достигнут нуль.

Возьмем для примера число 42: первый основной римский символ меньше 42 – это XL, представляющий число 40. Добавляем его в запись, получая XL, и вычитаем его значение из текущего числа, получая в результате 2. Первый основной римский символ меньше 2 – это I, представляющий число 1. Добавляем его в запись, получая XLI, и вычитаем 1 из текущего числа, получая в результате 1. Добавляем один символ I в запись, получаем XLII, снова вычитаем 1 и достигаем 0:

```
std::string to_roman(unsigned int value)
{
  std::vector<std::pair<unsigned int, char const*>> roman {
    { 1000, "M" },{ 900, "CM" }, { 500, "D" },{ 400, "CD" },
    {  100, "C" },{  90, "XC" }, {  50, "L" },{  40, "XL" },
    {   10, "X" },{   9, "IX" }, {   5, "V" },{   4, "IV" },{ 1, "I" }};

  std::string result;
  for (auto const & kvp : roman) {
    while (value >= kvp.first) {
      result += kvp.second;
      value -= kvp.first;
    }
  }
  return result;
}
```

Вот как можно использовать эту функцию:

```
int main()
{
  for(int i = 1; i <= 100; ++i)
  {
    std::cout << i << "\t" << to_roman(i) << std::endl;
```

```
  }

  int number = 0;
  std::cout << "number:";
  std::cin >> number;
  std::cout << to_roman(number) << std::endl;
}
```

12. Наибольшая последовательность Коллатца

Гипотеза Коллатца, также известная как гипотеза Улама, проблема Какутани, гипотеза Туэйтса, алгоритм Хассе или сиракузская проблема, – одна из нерешенных проблем математики. Она утверждает, что последовательность, которая определяется, как описано далее, всегда достигает 1. Берем любое положительное целое число *n* и из него получаем следующий элемент последовательности: если число чётное, делим его на 2, если нечётное – умножаем на 3 и прибавляем 1.

Ваша задача: сгенерировать последовательность Коллатца для всех положительных целых чисел вплоть до 1 000 000, выбрать наибольшую и вывести ее длину и начальное число, из которого она была получена. Можно было бы просто перебрать все числа, сгенерировать для каждого последовательность Коллатца и подсчитать количество элементов. Однако есть более быстрое решение: сохранять длины всех уже сгенерированных последовательностей. Если текущий элемент последовательности с начальным числом *n* оказывается меньше *n*, значит, для него последовательность уже была определена, поэтому можно просто получить ее длину из кеша и прибавить к текущей длине накопленной последовательности, чтобы определить полную длину последовательности для данного числа *n*. Однако это решение имеет ограничение, поскольку в некоторый момент может исчерпаться память для кеша:

```
std::pair<unsigned long long, long> longest_collatz(
    unsigned long long const limit)
{
  long length = 0;
  unsigned long long number = 0;
  std::vector<int> cache(limit + 1, 0);

  for (unsigned long long i = 2; i <= limit; i++)
  {
    auto n = i;
    long steps = 0;
    while (n != 1 && n >= i)
    {
      if ((n % 2) == 0) n = n / 2;
      else n = n * 3 + 1;
      steps++;
```

```
    }
    cache[i] = steps + cache[n];

    if (cache[i] > length)
    {
      length = cache[i];
      number = i;
    }
  }

  return std::make_pair(number, length);
}
```

13. Вычисление значения числа π

Для вычисления значения числа π можно использовать метод Монте-Карло. Он основан на использовании случайных выборок для исследования поведения сложных процессов или систем. Данный метод широко применяется в самых разных сферах: в физике, технике, информатике, финансовой области, бизнесе и др.

С этой целью возьмем за основу следующую идею: площадь круга с диаметром d равна $\pi * d^2 / 4$. Площадь квадрата с длиной стороны d равна d^2. Если разделить их, мы получим $\pi/4$. Если поместить круг внутрь квадрата и генерировать случайные точки, равномерно распределенные внутри квадрата, число точек, попавших внутрь круга, должно быть прямо пропорционально площади круга, а число точек, попавших внутрь квадрата, должно быть прямо пропорционально площади квадрата. То есть, разделив количество точек, попавших внутрь квадрата, на количество точек, попавших внутрь круга, мы получим приближенное значение $\pi/4$. Чем больше точек будет сгенерировано, тем более точным получится результат.

Для генерации случайных чисел используем алгоритм Mersenne Twister и равномерное статистическое распределение:

```
template <typename E = std::mt19937,
          typename D = std::uniform_real_distribution<>>
double compute_pi(E& engine, D& dist, int const samples = 1000000)
{
  auto hit = 0;
  for (auto i = 0; i < samples; i++)
  {
    auto x = dist(engine);
    auto y = dist(engine);
    if (y <= std::sqrt(1 - std::pow(x, 2))) hit += 1;
  }
  return 4.0 * hit / samples;
```

```
}

int main()
{
  std::random_device rd;
  auto seed_data = std::array<int, std::mt19937::state_size> {};
  std::generate(std::begin(seed_data), std::end(seed_data),
                std::ref(rd));
  std::seed_seq seq(std::begin(seed_data), std::end(seed_data));
  auto eng = std::mt19937{ seq };
  auto dist = std::uniform_real_distribution<>{ 0, 1 };

  for (auto j = 0; j < 10; j++)
    std::cout << compute_pi(eng, dist) << std::endl;
}
```

14. Проверка действительности номеров ISBN

Международный стандартный книжный номер (International Standard Book Number, ISBN) – это уникальный числовой идентификатор, присваиваемый книгам. В настоящее время используется 13-значный формат. Однако в этой задаче предлагается реализовать проверку номеров ISBN в прежнем, 10-значном формате. Последняя из 10 цифр – это контрольная сумма. Она выбирается так, чтобы сумма всех десяти цифр, умноженных на их веса, уменьшающиеся от 10 до 1, была кратна 11.

Функция `validate_isbn_10`, показанная ниже, принимает номер ISBN в виде строки и возвращает `true`, если длина строки равна 10, все десять элементов являются цифрами и сумма всех цифр, умноженных на их веса (номера позиций), кратна 11:

```
bool validate_isbn_10(std::string_view isbn)
{
  auto valid = false;
  if (isbn.size() == 10 &&
      std::count_if(std::begin(isbn), std::end(isbn), isdigit) == 10)
  {
    auto w = 10;
    auto sum = std::accumulate(
      std::begin(isbn), std::end(isbn), 0,
      [&w](int const total, char const c) {
        return total + w-- * (c - '0'); });

    valid = !(sum % 11);
  }
  return valid;
}
```

В качестве дополнительного упражнения можете улучшить эту функцию, чтобы она могла проверять номера ISBN-10, включающие дефисы, такие как 3-16-148410-0. Также можете попробовать написать функцию для проверки номеров ISBN-13.