

**C++** — один из самых популярных языков программирования и используется в разных областях, таких как программирование игр, разработка графических интерфейсов и операционных систем. На протяжении многих лет C++ входил и продолжает входить в число самых востребованных языков.

Эта книга познакомит вас с наиболее примечательными особенностями C++ и покажет, как ими пользоваться в своих приложениях. Каждая задача уникальна и не просто проверяет ваше знание языка - она проверяет вашу способность думать и находить более удачные решения. И если вы окажетесь в тупике, вам не придется волноваться, потому что книга предложит вам одно из лучших решений.

Итак, вы готовы принять вызов?

В этой книге вы узнаете, как:

- осуществлять сериализацию и десериализацию данных в форматах JSON и XML;
- шифровать и подписывать данные для безопасного обмена информацией между сторонами;
- внедрять и использовать в приложениях базы данных SQLite;
- использовать потоки и асинхронные функции для реализации параллельных алгоритмов;
- упаковывать данные в ZIP-архивы и распаковывать их;
- реализовать такие структуры данных, как кольцевой буфер и приоритетная очередь;
- реализовать универсальные алгоритмы, а также алгоритмы для решения конкретных задач;
- создавать клиент/серверные приложения, взаимодействующие по протоколу TCP/IP;
- использовать HTTP-службы REST;
- применять шаблоны проектирования для решения практических задач.

Решение задач на современном C++

Мариус Бансила



Решение задач  
на современном  
C++

Интернет -магазин:  
www.dmkpress.com  
Книга - почтой:  
e-mail: orders@aliants-kniga.ru  
Оптовая продажа:  
«Альянс-книга»  
Тел/факс (499)782-3889  
e-mail: books@aliants-kniga.ru

Ракт>



ISBN 978-5-97060-666-7



Станьте опытным программистом, решая практические задачи

---

Мариус Бансила



# Решение задач на современном C++

---

Станьте опытным программистом,  
решая практические задачи





# The Modern C++ Challenge

**Become an expert programmer by solving  
real-world problems**



**Marius Bancila**

**Packt**>

BIRMINGHAM – MUMBAI

---

---

# Решение задач на современном C++

Станьте опытным программистом, решая  
практические задачи

Мариус Бансила



Москва, 2019

УДК 004.4  
ББК 32.973.202-018.2  
Б23

Б23 Мариус Бансила

Решение задач на современном C++ / пер. с англ. А. Н. Киселева – М.: ДМК Пресс, 2019. – 302 с.: ил.



**ISBN 978-5-97060-666-7**

Эта книга – сборник практических задач по языку C++17: от математических и вычислительных до архитектурных, построенных на базе шаблонов проектирования. Здесь собрано 100 задач, которые помогут вам применить на практике разнообразные возможности C++ и его стандартной библиотеки, а также опробовать множество сторонних, кроссплатформенных библиотек. Решения представлены в виде исходного кода, пояснений и рекомендаций к нему.

Книга рекомендована сообществом разработчиков C++ России и Беларуси.

Издание будет полезно студентам технических вузов, а также начинающим и опытным разработчикам.



УДК 004.4  
ББК 32.973.202-018.2

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но, поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-1-78899-386-9 (англ.)  
ISBN 978-5-97060-666-7 (рус.)

Copyright © 2018 Packt Publishing.  
© Оформление, перевод на русский язык, издание,  
ДМК Пресс, 2019

# Оглавление



<b>Вступительное слово от сообщества разработчиков России и Беларуси ....</b>	<b>12</b>
<b>Об авторе .....</b>	<b>13</b>
<b>О рецензентах .....</b>	<b>14</b>
<b>О чем рассказывается в книге .....</b>	<b>15</b>
<b>Вступление .....</b>	<b>17</b>
Кому адресована эта книга.....	18
Что необходимо, чтобы извлечь максимум пользы из книги.....	18
Скачивание исходного кода примеров .....	19
Сборка примеров.....	19
Как сгенерировать проекты для Visual Studio 2017 .....	20
Как сгенерировать проекты для Xcode .....	20
Соглашения.....	21
Отзывы и пожелания.....	22
Список опечаток.....	23
Нарушение авторских прав.....	23
<b>Глава 1. Математические задачи.....</b>	<b>24</b>
Задачи .....	24
1. Сумма натуральных чисел, кратных 3 и 5.....	24
2. Наибольший общий делитель.....	24
3. Наименьшее общее кратное.....	24
4. Наибольшее простое число меньше заданного.....	24
5. Простые числа, отличающиеся на шесть.....	24
6. Избыточные числа.....	24
7. Дружественные числа.....	25
8. Числа Армстронга .....	25
9. Простые множители числа .....	25
10. Код Грея .....	25
11. Преобразование десятичных чисел в римские .....	25
12. Наибольшая последовательность Коллатца.....	25
13. Вычисление значения числа $\pi$ .....	25
14. Проверка действительности номеров ISBN .....	25
Решения .....	25
1. Сумма натуральных чисел, кратных 3 и 5.....	25
2. Наибольший общий делитель.....	26
3. Наименьшее общее кратное.....	27

4. Наибольшее простое число меньше заданного.....	27
5. Простые числа, отличающиеся на шесть.....	28
6. Избыточные числа.....	29
7. Дружественные числа.....	30
8. Числа Армстронга.....	31
9. Простые множители числа.....	32
10. Код Грея.....	33
11. Преобразование десятичных чисел в римские.....	34
12. Наибольшая последовательность Коллатца.....	36
13. Вычисление значения числа $\pi$ .....	37
14. Проверка действительности номеров ISBN.....	38
<b>Глава 2. Особенности языка.....</b>	<b>40</b>
Задачи.....	40
15. Тип данных IPv4.....	40
16. Перечисление адресов IPv4 в заданном диапазоне.....	40
17. 2-мерный массив с поддержкой базовых операций.....	40
18. Функция выбора минимального значения с переменным числом аргументов.....	40
19. Добавление диапазона значений в контейнер.....	40
20. Проверка наличия в контейнере любого, всех и ни одного из указанных значений.....	41
21. Обертка для системных дескрипторов.....	41
22. Литералы разных температурных шкал.....	41
Решения.....	41
15. Тип данных IPv4.....	41
16. Перечисление адресов IPv4 в заданном диапазоне.....	43
17. 2-мерный массив с поддержкой базовых операций.....	45
18. Функция выбора минимального значения с переменным числом аргументов.....	47
19. Добавление диапазона значений в контейнер.....	48
20. Проверка наличия в контейнере любого, всех и ни одного из указанных значений.....	49
21. Обертка для системных дескрипторов.....	50
22. Литералы разных температурных шкал.....	54
<b>Глава 3. Строки и регулярные выражения.....</b>	<b>59</b>
Задачи.....	59
23. Преобразование чисел в строки.....	59
24. Преобразование строк в числа.....	59
25. Преобразование в верхний регистр первых букв слов.....	59
26. Объединение строк через разделитель.....	59
27. Разбиение строк на лексемы по разделителям из списка.....	60
28. Наибольшая подстрока-палиндром.....	60
29. Проверка номерного знака.....	60
30. Извлечение частей URL.....	60

31. Преобразование дат в строках.....	60
Решения.....	60
23. Преобразование чисел в строки.....	60
24. Преобразование строк в числа.....	61
25. Преобразование в верхний регистр первых букв слов.....	63
26. Объединение строк через разделитель.....	64
27. Разбиение строк на лексемы по разделителям из списка.....	65
28. Наибольшая подстрока-палиндром.....	66
29. Проверка номерного знака.....	68
30. Извлечение частей URL.....	69
31. Преобразование дат в строках.....	71



## **Глава 4. Потоки данных и файловые системы..... 72**

Задачи.....	72
32. Треугольник Паскаля.....	72
33. Табличный вывод списка процессов.....	72
34. Удаление пустых строк из текстового файла.....	72
35. Определение размера каталога.....	72
36. Удаление файлов старше заданной даты.....	73
37. Поиск файлов в каталоге, соответствующих регулярному выражению.....	73
38. Временные файлы журналов.....	73
Решения.....	73
32. Треугольник Паскаля.....	73
33. Табличный вывод списка процессов.....	74
34. Удаление пустых строк из текстового файла.....	76
35. Определение размера каталога.....	77
36. Удаление файлов старше заданной даты.....	78
37. Поиск файлов в каталоге, соответствующих регулярному выражению.....	80
38. Временные файлы журналов.....	81



## **Глава 5. Дата и время..... 83**

Задачи.....	83
39. Измерение времени выполнения функции.....	83
40. Число дней между двумя датами.....	83
41. День недели.....	83
42. День и неделя года.....	83
43. Время встречи для нескольких часовых поясов.....	83
44. Календарь на месяц.....	83
Решения.....	84
39. Измерение времени выполнения функции.....	84
40. Число дней между двумя датами.....	85
41. День недели.....	86
42. День и неделя года.....	87
43. Время встречи для нескольких часовых поясов.....	88
44. Календарь на месяц.....	90



<b>Глава 6. Алгоритмы и структуры данных .....</b>	<b>92</b>
Задачи .....	92
45. Приоритетная очередь .....	92
46. Циклический буфер .....	92
47. Двойной буфер .....	93
48. Самый часто встречающийся элемент в диапазоне .....	93
49. Текстовая гистограмма .....	93
50. Фильтрация списка телефонных номеров .....	93
51. Преобразование списка телефонных номеров .....	93
52. Генерация всех перестановок символов в строке .....	94
53. Средний рейтинг фильмов .....	94
54. Алгоритм объединения в пары .....	94
55. Алгоритм «сшивания» .....	94
56. Алгоритм выбора .....	94
57. Алгоритм сортировки .....	95
58. Кратчайший путь между узлами .....	95
59. Программа Weasel .....	96
60. Игра «Жизнь» .....	96
Решения .....	97
45. Приоритетная очередь .....	97
46. Циклический буфер .....	99
47. Двойной буфер .....	103
48. Самый часто встречающийся элемент в диапазоне .....	105
49. Текстовая гистограмма .....	107
50. Фильтрация списка телефонных номеров .....	108
51. Преобразование списка телефонных номеров .....	109
52. Генерация всех перестановок символов в строке .....	111
53. Средний рейтинг фильмов .....	113
54. Алгоритм объединения в пары .....	114
55. Алгоритм «сшивания» .....	115
56. Алгоритм выбора .....	117
57. Алгоритм сортировки .....	117
58. Кратчайший путь между узлами .....	121
59. Программа Weasel .....	125
60. Игра «Жизнь» .....	128
<b>Глава 7. Конкуренция .....</b>	<b>133</b>
Задачи .....	133
61. Алгоритм параллельного преобразования .....	133
62. Параллельные алгоритмы поиска максимального и минимального значений с использованием потоков .....	133
63. Параллельные алгоритмы поиска максимального и минимального значений с использованием асинхронных функций .....	133
64. Параллельный алгоритм сортировки .....	133
65. Потокбезопасное журналирование в консоль .....	134
66. Система обслуживания клиентов .....	134

Решения.....	134
61. Алгоритм параллельного преобразования.....	134
62. Параллельные алгоритмы поиска максимального и минимального значений с использованием потоков.....	136
63. Параллельные алгоритмы поиска максимального и минимального значений с использованием асинхронных функций.....	138
64. Параллельный алгоритм сортировки.....	140
65. Потокбезопасное журналирование в консоль.....	142
66. Система обслуживания клиентов.....	143
<b>Глава 8. Шаблоны проектирования.....</b>	<b>148</b>
67. Проверка пароля.....	148
68. Генерация случайных паролей.....	148
69. Генерация номеров социального страхования.....	148
70. Система одобрений.....	149
71. Контейнер с наблюдателями.....	149
72. Вычисление стоимости заказа с учетом скидок.....	149
Решения.....	150
67. Проверка пароля.....	150
68. Генерация случайных паролей.....	154
69. Генерация номеров социального страхования.....	158
70. Система одобрений.....	162
71. Контейнер с наблюдателями.....	166
72. Вычисление стоимости заказа с учетом скидок.....	171
<b>Глава 9. Сериализация данных.....</b>	<b>177</b>
Задачи.....	177
73. Сериализация и десериализация данных в формате XML.....	177
74. Выборка данных из XML с помощью XPath.....	177
75. Сериализация данных в формат JSON.....	178
76. Десериализация данных из формата JSON.....	178
77. Вывод списка фильмов в файл PDF.....	178
78. Создание документа PDF из коллекции изображений.....	179
Решения.....	179
73. Сериализация и десериализация данных в формате XML.....	179
74. Выборка данных из XML с помощью XPath.....	183
75. Сериализация данных в формат JSON.....	185
76. Десериализация данных из формата JSON.....	187
77. Вывод списка фильмов в файл PDF.....	189
78. Создание документа PDF из коллекции изображений.....	193
<b>Глава 10. Архивы, изображения и базы данных.....</b>	<b>196</b>
Задачи.....	196
79. Поиск файлов в архиве ZIP.....	196
80. Упаковка и извлечение файлов из архива ZIP.....	196
81. Упаковка и извлечение файлов из архива ZIP с защитой паролем.....	196



82. Создание файла PNG с изображением национального флага .....	196
83. Создание изображения PNG с контрольным текстом .....	197
84. Генератор штрихкодов EAN-13 .....	197
85. Чтение информации о фильмах из базы данных SQLite.....	198
86. Добавление информации о фильмах в базу данных SQLite .....	198
87. Обработка изображений для фильмов в базе данных SQLite.....	198
<b>Решения.....</b>	<b>199</b>
79. Поиск файлов в архиве ZIP.....	199
80. Упаковка и извлечение файлов из архива ZIP.....	201
81. Упаковка и извлечение файлов из архива ZIP с защитой паролем.....	204
82. Создание файла PNG с изображением национального флага .....	207
83. Создание изображения PNG с контрольным текстом .....	208
84. Генератор штрихкодов EAN-13 .....	211
85. Чтение информации о фильмах из базы данных SQLite.....	217
86. Добавление информации о фильмах в базу данных SQLite.....	223
87. Обработка изображений для фильмов в базе данных SQLite.....	227
<b>Глава 11. Криптография.....</b>	<b>236</b>
<b>Задачи .....</b>	<b>236</b>
88. Шифр Цезаря .....	236
89. Шифр Виженера.....	236
90. Кодирование и декодирование Base64.....	236
91. Проверка учетных данных пользователя.....	236
92. Вычисление хеш-суммы файла .....	236
93. Шифрование и расшифровывание файлов .....	237
94. Подписывание файлов.....	237
<b>Решения.....</b>	<b>237</b>
88. Шифр Цезаря .....	237
89. Шифр Виженера.....	239
90. Кодирование и декодирование Base64.....	241
91. Проверка учетных данных пользователя.....	246
92. Вычисление хеш-суммы файла .....	250
93. Шифрование и расшифровывание файлов .....	252
94. Подписывание файлов.....	254
<b>Глава 12. Сети и службы.....</b>	<b>258</b>
<b>Задачи .....</b>	<b>258</b>
95. Поиск IP-адреса хоста.....	258
96. Клиент-серверная игра Fizz-Buzz .....	258
97. Обменный курс биткойна.....	258
98. Получение почты по протоколу IMAP .....	258
99. Перевод текста на любой язык.....	259
100. Определение лиц на изображениях .....	259
<b>Решения.....</b>	<b>259</b>
95. Поиск IP-адреса хоста.....	259
96. Клиент-серверная игра Fizz-Buzz .....	261

97. Обменный курс биткойна.....	265
98. Получение почты по протоколу IMAP.....	270
99. Перевод текста на любой язык.....	274
100. Определение лиц на изображениях.....	279
<b>Библиография .....</b>	<b>291</b>
Статьи.....	291
Документация к библиотекам .....	293
<b>Предметный указатель .....</b>	<b>295</b>



---

# 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](http://cpp-russia.ru)

[stdcpp.ru](http://stdcpp.ru)

[corehard.by](http://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](https://twitter.com/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 видеуроки. 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 – практического 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](http://www.dmkpress.com) или [www.dmk.pф](http://www.dmk.pф) в разделе Читателям – Файлы к книгам.

После загрузки файла архива распакуйте его, используя 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-ecdsa_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` 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](http://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](mailto: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/](http://dmkpress.com/authors/publish_book/) или напишите в издательство по адресу [dmkpress@gmail.com](mailto: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](mailto: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](mailto: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<sup>1</sup>

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.

<sup>1</sup> В английском языке такие пары чисел называют «sexu primes» (от латинского названия «числа шесть» – «sex»), что добавляет термину забавную двусмысленность ввиду возможной трактовки англ. «sexu 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 $\pi$

Напишите программу, вычисляющую значение числа  $\pi$  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:

$$\begin{aligned} \text{НОД}(a, 0) &= a \\ \text{НОД}(a, b) &= \text{НОД}(b, a \bmod b) \end{aligned}$$

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:

$$\text{НОК}(a, b) = \text{abs}(a, b) / \text{НОД}(a, b)$$

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 двояродные 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:  $a \cdot 10^2 + b \cdot 10^1 + c$ . 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  $2 \times 2 \times 2$ , а простые множители числа 42 – это  $2 \times 3 \times 7$ . 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) {

```



```

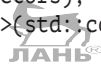
    factors.push_back(2);
    n = n / 2;
}
for (unsigned long long i = 3; i <= std::sqrt(n); i += 2)
{
    while (n%i == 0) {
        factors.push_back(i);
        n = n / i;
    }
}

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:

$$g = b \text{ xor } (b \text{ логический сдвиг вправо на } 1)$$

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 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 $\pi$

Для вычисления значения числа  $\pi$  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.





---

# Глава 2

## Особенности языка



### Задачи

#### 15. Тип данных IPv4

Напишите класс, представляющий адрес IPv4. Реализуйте функции для ввода/вывода таких адресов в консоли. Пользователь должен иметь возможность вводить значения в формате с точками, например: 127.0.0.1 или 168.192.0.100. В таком же формате адреса IPv4 должны выводиться в выходной поток.

#### 16. Перечисление адресов IPv4 в заданном диапазоне

Напишите программу, которой пользователь мог бы передать два адреса IPv4, представляющих диапазон, и получить список всех адресов в этом диапазоне. Добавьте эту функциональность в класс из предыдущего задания.

#### 17. 2-мерный массив с поддержкой базовых операций

Напишите шаблонный класс, представляющий двумерный массив, с методами доступа к элементам (`at()` и `data()`), определения емкости, заполнения, перестановки и итераторами. Класс должен поддерживать перемещение объектов данного типа.

#### 18. Функция выбора минимального значения с переменным числом аргументов

Напишите шаблонную функцию с переменным числом аргументов, возвращающую минимальное значение, которая выполняет сравнение с помощью оператора `<`. Напишите перегруженную версию этой функции, которую можно было бы параметризовать функцией сравнения с двумя аргументами, чтобы ее можно было использовать вместо оператора `<`.

#### 19. Добавление диапазона значений в контейнер

Напишите универсальную функцию для добавления произвольного количества элементов в конец контейнера, имеющего метод `push_back(T&& value)`.

## 20. Проверка наличия в контейнере любого, всех и ни одного из указанных значений

Напишите набор универсальных функций, проверяющих наличие в заданном контейнере любого, всех и ни одного из указанных аргументов. Эти функции должны позволять писать такой код:

```
std::vector<int> v{ 1, 2, 3, 4, 5, 6 };
assert(contains_any(v, 0, 3, 30));
```

```
std::array<int, 6> a{ { 1, 2, 3, 4, 5, 6 } };
assert(contains_all(a, 1, 3, 5, 6));
```



```
std::list<int> l{ 1, 2, 3, 4, 5, 6 };
assert(!contains_none(l, 0, 6));
```

## 21. Обертка для системных дескрипторов

Напишите обертку для каких-нибудь системных дескрипторов, например дескрипторов файлов, которая приобретала бы дескриптор и освобождала его, а также выполняла другие операции, такие как проверка действительности дескриптора и передача владения им из одного объекта в другой.

## 22. Литералы разных температурных шкал

Напишите небольшую библиотеку, с помощью которой можно было бы выражать значения температуры в трех наиболее распространенных шкалах – Цельсия, Фаренгейта и Кельвина – и выполнять преобразования между ними. Библиотека должна позволять записывать литералы температуры во всех трех шкалах, например `36.5_deg` для шкалы Цельсия, `97.7_f` для шкалы Фаренгейта и `309.65_k` для шкалы Кельвина; выполнять операции с этими значениями и производить преобразования с ними.

# Решения

## 15. Тип данных IPv4

По условию задачи мы должны написать класс, представляющий адрес IPv4. Это 32-битное значение, обычно записываемое в формате с точками, например: `168.192.0.100`. Каждый элемент этой формы является 8-битным значением в диапазоне от 0 до 255. Чтобы упростить представление и обработку, будем хранить адрес в виде четырех значений `unsigned char`. Значение адреса можно сконструировать или из четырех значений `unsigned char`, или из одного значения `unsigned long`. Чтобы дать возможность читать адрес из консоли (или из любого другого потока ввода) и выводить его в консоль (или в любой другой поток вывода), мы должны реализовать перегруженные версии `operator>>` и `op-`

erator<<. В листинге ниже приводится минимальная реализация, соответствующая требованиям, перечисленным в задании:

```
class ipv4
{
    std::array<unsigned char, 4> data;
public:
    constexpr ipv4() : data{ {0} } {}
    constexpr ipv4(unsigned char const a, unsigned char const b,
        unsigned char const c, unsigned char const d):
        data{{a,b,c,d}} {}
    explicit constexpr ipv4(unsigned long a) :
        data{ { static_cast<unsigned char>((a >> 24) & 0xFF),
            static_cast<unsigned char>((a >> 16) & 0xFF),
            static_cast<unsigned char>((a >> 8) & 0xFF),
            static_cast<unsigned char>(a & 0xFF) } } {}
    ipv4(ipv4 const & other) noexcept : data(other.data) {}
    ipv4& operator=(ipv4 const & other) noexcept
    {
        data = other.data;
        return *this;
    }

    std::string to_string() const
    {
        std::stringstream sstr;
        sstr << *this;
        return sstr.str();
    }

    constexpr unsigned long to_ulong() const noexcept
    {
        return (static_cast<unsigned long>(data[0]) << 24) |
            (static_cast<unsigned long>(data[1]) << 16) |
            (static_cast<unsigned long>(data[2]) << 8) |
            static_cast<unsigned long>(data[3]));
    }

    friend std::ostream& operator<<(std::ostream& os, const ipv4& a)
    {
        os << static_cast<int>(a.data[0]) << '.'
            << static_cast<int>(a.data[1]) << '.'
            << static_cast<int>(a.data[2]) << '.'
            << static_cast<int>(a.data[3]);
        return os;
    }

    friend std::istream& operator>>(std::istream& is, ipv4& a)
```



```
{
    char d1, d2, d3;
    int b1, b2, b3, b4;
    is >> b1 >> d1 >> b2 >> d2 >> b3 >> d3 >> b4;
    if (d1 == '.' && d2 == '.' && d3 == '.')
        a = ipv4(b1, b2, b3, b4);
    else
        is.setstate(std::ios_base::failbit);
    return is;
}
};
```

Далее приводится пример использования класса `ipv4`:

```
int main()
{
    ipv4 address(168, 192, 0, 1);
    std::cout << address << std::endl;

    ipv4 ip;
    std::cout << ip << std::endl;
    std::cin >> ip;
    if(!std::cin.fail())
        std::cout << ip << std::endl;
}
```



## 16. Перечисление адресов IPv4 в заданном диапазоне

Для перечисления адресов IPv4 в заданном диапазоне прежде всего необходима возможность сравнивать их. То есть, как минимум, мы должны реализовать `operator<`, но в следующем листинге содержатся реализации всех операторов сравнения: `==`, `!=`, `<`, `>`, `<=` и `>=`. Кроме того, для получения адреса IPv4, следующего за текущим, реализованы обе версии – префиксная и постфиксная – оператора `operator++`. Следующий код расширяет класс `ipv4` из предыдущей задачи:

```
ipv4& operator++()
{
    *this = ipv4(1 + to_ulong());
    return *this;
}

ipv4& operator++(int)
{
    ipv4 result(*this);
    ++(*this);
    return *this;
}
```

```

}

friend bool operator==(ipv4 const & a1, ipv4 const & a2) noexcept
{
    return a1.data == a2.data;
}

friend bool operator!=(ipv4 const & a1, ipv4 const & a2) noexcept
{
    return !(a1 == a2);
}

friend bool operator<(ipv4 const & a1, ipv4 const & a2) noexcept
{
    return a1.to_ulong() < a2.to_ulong();
}

friend bool operator>(ipv4 const & a1, ipv4 const & a2) noexcept
{
    return a2 < a1;
}

friend bool operator<=(ipv4 const & a1, ipv4 const & a2) noexcept
{
    return !(a1 > a2);
}

friend bool operator>=(ipv4 const & a1, ipv4 const & a2) noexcept
{
    return !(a1 < a2);
}

```

После добавления этих функций в класс `ipv4` из предыдущей задачи можно написать такую программу:

```

int main()
{
    std::cout << "input range: ";
    ipv4 a1, a2;
    std::cin >> a1 >> a2;
    if (a2 > a1)
    {
        for (ipv4 a = a1; a <= a2; a++)
        {
            std::cout << a << std::endl;
        }
    }
    else

```

```

    {
        std::cerr << "invalid range!" << std::endl;
    }
}

```

## 17.2-мерный массив с поддержкой базовых операций

Прежде чем заняться определением такого класса, создадим несколько тестов для его проверки. Следующий листинг демонстрирует, какие возможности мы должны реализовать:

```

int main()
{
    // доступ к элементам
    array2d<int, 2, 3> a {1, 2, 3, 4, 5, 6};
    for (size_t i = 0; i < a.size(1); ++i)
        for (size_t j = 0; j < a.size(2); ++j)
            a(i, j) *= 2;

    // итераторы
    std::copy(std::begin(a), std::end(a),
              std::ostream_iterator<int>(std::cout, " "));

    // заполнение
    array2d<int, 2, 3> b;
    b.fill(1);

    // перестановка
    a.swap(b);

    // перемещение
    array2d<int, 2, 3> c(std::move(b));
}

```



Обратите внимание, что для доступа к элементам используется `operator()` (выражение `a(i, j)` в примере выше), а не `operator[]`, например `a[i][j]`, потому что только первый из них может принимать несколько аргументов (по одному индексу для каждого измерения). Последний принимает только один аргумент, и для реализации выражений, таких как `a[i][j]`, он должен возвращать промежуточный тип (по сути, представляющий строку в двумерном массиве), который, в свою очередь, реализует перегруженный `operator[]`, возвращающий единственный элемент.

В стандартной библиотеке уже имеются контейнеры, хранящие последовательности элементов фиксированной или переменной длины. Поэтому в нашем классе двумерного массива мы можем использовать один из таких контейнеров. Наиболее предпочтительными вариантами являются `std::array` и

`std::vector`. Чтобы сделать выбор между ними, мы должны учесть, что класс `array2d` должен:

- поддерживать семантику перемещения;
- позволять инициализировать объекты этого типа списком инициализаторов.



Контейнер `std::array` поддерживает перемещение, только если хранящиеся в нем элементы поддерживают конструирование и присваивание при перемещении. С другой стороны, он не может инициализироваться из `std::initializer_list`. Поэтому вариант `std::vector` выглядит предпочтительнее.

Внутренне наш класс 2-мерного массива может хранить свои данные в векторе векторов (каждая строка имеет тип `vector<T>` с `C` элементами, а 2-мерный массив имеет `R` таких элементов, хранящихся в `vector<vector<T>>`) или в едином векторе с `R * C` элементами типа `T`. В последнем случае элемент в строке `i` и столбце `j` хранится с индексом `i * C + j`. При таком подходе все данные хранятся в одной непрерывной области памяти, и это решение реализуется достаточно просто. Поэтому возьмем его за основу.

Вот одна из возможных реализаций класса двумерного массива:

```
template <class T, size_t R, size_t C>
class array2d
{
    typedef T          value_type;
    typedef value_type* iterator;
    typedef value_type const* const_iterator;
    std::vector<T>    arr;
public:
    array2d() : arr(R*C) {}
    explicit array2d(std::initializer_list<T> l):arr(l) {}
    constexpr T* data() noexcept { return arr.data(); }
    constexpr T const * data() const noexcept { return arr.data(); }

    constexpr T& at(size_t const r, size_t const c)
    {
        return arr.at(r*C + c);
    }

    constexpr T const & at(size_t const r, size_t const c) const
    {
        return arr.at(r*C + c);
    }

    constexpr T& operator() (size_t const r, size_t const c)
    {
        return arr[r*C + c];
    }
};
```

```

}

constexpr T const & operator() (size_t const r, size_t const c) const
{
    return arr[r*C + c];
}

constexpr bool empty() const noexcept { return R == 0 || C == 0; }

constexpr size_t size(int const rank) const
{
    if (rank == 1) return R;
    else if (rank == 2) return C;
    throw std::out_of_range("Rank is out of range!");
}

void fill(T const & value)
{
    std::fill(std::begin(arr), std::end(arr), value);
}

void swap(array2d & other) noexcept { arr.swap(other.arr); }

constexpr const_iterator begin() const { return arr.data(); }
constexpr const_iterator end() const { return arr.data() + arr.size(); }
constexpr iterator begin() { return arr.data(); }
constexpr iterator end() { return arr.data() + arr.size(); }
};

```

## 18. Функция выбора минимального значения с переменным числом аргументов

Мы можем написать шаблонную функцию с переменным числом аргументов, используя рекурсию времени компиляции (которая фактически сводится к вызову множества перегруженных функций). Ниже показано, как можно реализовать такую функцию:

```

template <typename T>
T minimum(T const a, T const b) { return a < b ? a : b; }

template <typename T1, typename... T>
T1 minimum(T1 a, T... args)
{
    return minimum(a, minimum(args...));
}

int main()

```



```
{
    auto x = minimum(5, 4, 2, 3);
}
```



Чтобы добавить поддержку пользовательской функции сравнения с двумя аргументами, нужно написать другую шаблонную функцию. Функция сравнения должна передаваться в первом аргументе, потому что она не может следовать за списком аргументов переменной длины. С другой стороны, она не может быть перегруженной версией предыдущей функции `minimum` и должна иметь другое имя. Причина в том, что компилятор не сможет отличить списки параметров шаблона `<typename T1, typename... T>` и `<class Compare, typename T1, typename... T>`. Изменения минимальны и легко заметны в следующем коде:

```
template <class Compare, typename T>
T minimumc(Compare comp, T const a, T const b)
{ return comp(a, b) ? a : b; }

template <class Compare, typename T1, typename... T>
T1 minimumc(Compare comp, T1 a, T... args)
{
    return minimumc(comp, a, minimumc(comp, args...));
}

int main()
{
    auto y = minimumc(std::less<>(), 3, 2, 1, 0);
}
```



## 19. Добавление диапазона значений в контейнер

Мы можем написать функцию с переменным числом аргументов, используя тот же прием, что и в предыдущей задаче. На этот раз наша функция должна принимать контейнер в первом параметре и произвольное количество параметров, представляющих значения для добавления в конец контейнера. Реализацию такой функции можно значительно упростить с помощью выражения свертки:

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

Далее показаны примеры использования этой функции с контейнерами разных типов:

```
int main()
{
```

```

std::vector<int> v;
push_back(v, 1, 2, 3, 4);
std::copy(std::begin(v), std::end(v),
          std::ostream_iterator<int>(std::cout, " "));

std::list<int> l;
push_back(l, 1, 2, 3, 4);
std::copy(std::begin(l), std::end(l),
          std::ostream_iterator<int>(std::cout, " "));
}

```



## 20. Проверка наличия в контейнере любого, всех и ни одного из указанных значений

Требование возможности проверки наличия или отсутствия произвольного числа аргументов предполагает, что мы должны написать шаблонную функцию с переменным числом аргументов. Нам также понадобится универсальная вспомогательная функция, проверяющая присутствие элемента в контейнере и возвращающая результат типа `bool` как признак успеха или неудачи. Поскольку все функции, которые мы назовем `contains_all`, `contains_any` и `contains_none`, применяют логические операторы к результатам, возвращаемым вспомогательной функцией, мы можем использовать выражения свертки, чтобы упростить код. Оценка после развертывания выражения свертки производится по короткой схеме, то есть проверка элементов происходит только до момента, когда окончательный результат станет очевидным. Например, если мы проверяем присутствие всех значений из числа 1, 2 и 3 и значение 2 отсутствует, функция завершит выполнение, обнаружив его отсутствие, без проверки значения 3:

```

template<class C, class T>
bool contains(C const & c, T const & value)
{
    return std::end(c) != std::find(std::begin(c), std::end(c), value);
}

template<class C, class... T>
bool contains_any(C const & c, T &&... value)
{
    return (... || contains(c, value));
}

template<class C, class... T>
bool contains_all(C const & c, T &&... value)
{
    return (... && contains(c, value));
}

```

```

}

template<class C, class... T>
bool contains_none(C const & c, T &&... value)
{
    return !contains_any(c, std::forward<T>(value)...);
}

```

## 21. Обертка для системных дескрипторов

Системные дескрипторы являются ссылками на системные ресурсы. Так как все операционные системы, по крайней мере начальные их версии, написаны на C, создание и освобождение дескрипторов производится посредством специальных системных функций. Такая организация увеличивает риск утечки ресурсов в случае ошибок, таких как исключения, возникающих в процессе работы с ними. В следующем фрагменте, характерном для Windows, можно видеть функцию, которая открывает файл, читает его и в конце закрывает. Но в ней есть пара проблем: в одном случае разработчик мог бы забыть закрыть дескриптор перед выходом из функции; в другом – вызвать другую функцию, способную возбудить исключение, перед закрытием дескриптора и не предусмотреть обработку исключения, из-за чего заключительный код может не вызываться:

```

void bad_handle_example()
{
    bool condition1 = false;
    bool condition2 = true;
    HANDLE handle = CreateFile(L"sample.txt",
                              GENERIC_READ,
                              FILE_SHARE_READ,
                              nullptr,
                              OPEN_EXISTING,
                              FILE_ATTRIBUTE_NORMAL,
                              nullptr);

    if (handle == INVALID_HANDLE_VALUE)
        return;

    if (condition1)
    {
        CloseHandle(handle);
        return;
    }

    std::vector<char> buffer(1024);
    unsigned long bytesRead = 0;

```

```

ReadFile(handle,
         buffer.data(),
         buffer.size(),
         &bytesRead,
         nullptr);

if (condition2)
{
    // ой, забыли закрыть дескриптор
    return;
}

// если функция возбудит исключение,
// следующая за ней строка не выполнится
function_that_throws();

CloseHandle(handle);
}

```



Класс-обертка на C++ может гарантировать корректное освобождение дескриптора, когда объект обертки выходит из области видимости и уничтожается (в ходе нормального выполнения или в результате исключения). Правильная реализация должна учитывать разные типы дескрипторов, с диапазоном значений, определяющим недействительные дескрипторы (например, 0/null или -1). Реализация ниже поддерживает:

- явное приобретение и автоматическое освобождение дескриптора при уничтожении объекта;
- семантику перемещения для передачи права владения дескриптором;
- операторы сравнения для проверки, что два объекта ссылаются на один и тот же дескриптор;
- дополнительные операции, такие как перестановка и сброс.



Реализация, представленная ниже, – это измененная версия класса дескриптора, реализованного Кенни Керром (Kenny Kerr) и описанного в статье «Windows и C++ – C++ и Windows API», которая была опубликована в электронном журнале MSDN Magazine в июле 2011 г., <https://msdn.microsoft.com/ru-ru/magazine/hh288076.aspx>. Несмотря на то что класс, описывающий свойства дескриптора, показанный здесь, ссылается на дескрипторы Windows, для вас не составит труда написать аналогичные классы для других платформ.



```
template <typename Traits>
class unique_handle
{
    using pointer = typename Traits::pointer;
    pointer m_value;
public:
    unique_handle(unique_handle const &) = delete;
    unique_handle& operator=(unique_handle const &) = delete;

    explicit unique_handle(pointer value = Traits::invalid()) noexcept
        : m_value{ value }
    {}

    unique_handle(unique_handle && other) noexcept
        : m_value{ other.release() }
    {}

    unique_handle& operator=(unique_handle && other) noexcept
    {
        if (this != &other)
            reset(other.release());
        return *this;
    }

    ~unique_handle() noexcept
    {
        Traits::close(m_value);
    }

    explicit operator bool() const noexcept
    {
        return m_value != Traits::invalid();
    }

    pointer get() const noexcept { return m_value; }

    pointer release() noexcept
    {
        auto value = m_value;
        m_value = Traits::invalid();
        return value;
    }

    bool reset(pointer value = Traits::invalid()) noexcept
    {
        if (m_value != value)
        {
            Traits::close(m_value);
```





```
        m_value = value;
    }
    return static_cast<bool>(*this);
}

void swap(unique_handle<Traits> & other) noexcept
{
    std::swap(m_value, other.m_value);
}
};

template <typename Traits>
void swap(unique_handle<Traits> & left, unique_handle<Traits> & right)
noexcept
{
    left.swap(right);
}

template <typename Traits>
bool operator==(unique_handle<Traits> const & left,
                unique_handle<Traits> const & right) noexcept
{
    return left.get() == right.get();
}

template <typename Traits>
bool operator!=(unique_handle<Traits> const & left,
                unique_handle<Traits> const & right) noexcept
{
    return left.get() != right.get();
}

struct null_handle_traits
{
    using pointer = HANDLE;
    static pointer invalid() noexcept { return nullptr; }
    static void close(pointer value) noexcept
    {
        CloseHandle(value);
    }
};

struct invalid_handle_traits
{
    using pointer = HANDLE;
    static pointer invalid() noexcept { return INVALID_HANDLE_VALUE; }
    static void close(pointer value) noexcept
    {
```



```

        CloseHandle(value);
    }
};

```



```

using null_handle = unique_handle<null_handle_traits>;
using invalid_handle = unique_handle<invalid_handle_traits>;

```

Имея это определение типа дескриптора, можно переписать предыдущий пример функции, устранив проблему утечки ресурсов из-за отсутствия правильной обработки исключений или из-за забывчивости разработчика. Новая реализация получилась проще и надежнее:

```

void good_handle_example()
{
    bool condition1 = false;
    bool condition2 = true;

    invalid_handle handle{
        CreateFile(L"sample.txt",
            GENERIC_READ,
            FILE_SHARE_READ,
            nullptr,
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL,
            nullptr) };

    if (!handle) return;

    if (condition1) return;

    std::vector<char> buffer(1024);
    unsigned long bytesRead = 0;
    ReadFile(handle.get(),
        buffer.data(),
        buffer.size(),
        &bytesRead,
        nullptr);

    if (condition2) return;

    function_that_throws();
}

```



## 22. Литералы разных температурных шкал

Чтобы выполнить поставленные требования, необходимо реализовать несколько типов, операторов и функций:

- перечисление поддерживаемых шкал с именем `scale`;
- шаблонный класс `quantity` для представления значения температуры, параметризованный шкалой;
- операторы сравнения `==`, `!=`, `<`, `>`, `<=` и `>=` для сравнения двух значений температуры;
- арифметические операторы `+` и `-` для сложения и вычитания значений температуры; дополнительно можно реализовать операторы `+=` и `-=`;
- шаблонную функцию `temperature_cast` для преобразования температур из одной шкалы в другую; для преобразования функция использует сами свойства типов;
- литеральные операторы `""_deg`, `""_f` и `""_k` для создания пользовательских литералов температуры.



Для краткости следующий листинг содержит только код реализации шкал Цельсия и Фаренгейта. Вы должны самостоятельно добавить код поддержки шкалы Кельвина. Код в загружаемых примерах содержит полную реализацию для всех трех шкал.



Функция `are_equal()` – это вспомогательная функция для сравнения вещественных значений:

```
bool are_equal(double const d1, double const d2,
              double const epsilon = 0.001)
{
    return std::fabs(d1 - d2) < epsilon;
}
```

Далее определяются перечисление с поддерживаемыми температурными шкалами и класс, представляющий значение температуры:

```
namespace temperature
{
    enum class scale { celsius, fahrenheit, kelvin };

    template <scale S>
    class quantity
    {
        const double amount;
    public:
        constexpr explicit quantity(double const a) : amount(a) {}
        explicit operator double() const { return amount; }
    };
}
```



### Операторы сравнения для класса `quantity<S>`:

```

namespace temperature
{
    template <scale S>
    inline bool operator==(quantity<S> const & lhs, quantity<S> const & rhs)
    {
        return are_equal(static_cast<double>(lhs), static_cast<double>(rhs));
    }

    template <scale S>
    inline bool operator!=(quantity<S> const & lhs, quantity<S> const & rhs)
    {
        return !(lhs == rhs);
    }

    template <scale S>
    inline bool operator< (quantity<S> const & lhs, quantity<S> const & rhs)
    {
        return static_cast<double>(lhs) < static_cast<double>(rhs);
    }

    template <scale S>
    inline bool operator> (quantity<S> const & lhs, quantity<S> const & rhs)
    {
        return rhs < lhs;
    }

    template <scale S>
    inline bool operator<=(quantity<S> const & lhs, quantity<S> const & rhs)
    {
        return !(lhs > rhs);
    }

    template <scale S>
    inline bool operator>=(quantity<S> const & lhs, quantity<S> const & rhs)
    {
        return !(lhs < rhs);
    }

    template <scale S>
    constexpr quantity<S> operator+(quantity<S> const &q1,
                                     quantity<S> const &q2)
    {
        return quantity<S>(static_cast<double>(q1) +
                           static_cast<double>(q2));
    }

    template <scale S>

```



```
constexpr quantity<S> operator-(quantity<S> const &q1,
                                quantity<S> const &q2)
{
    return quantity<S>(static_cast<double>(q1) -
                       static_cast<double>(q2));
}
}
```

Для преобразования значений температур между шкалами мы определим шаблонные функции `temperature_cast()`, использующие определенные выше классы. Здесь они показаны не все, так же как классы; другие реализации вы найдете в загружаемых примерах кода:

```
namespace temperature
{
    template <scale S, scale R>
    struct conversion_traits
    {
        static double convert(double const value) = delete;
    };

    template <>
    struct conversion_traits<scale::celsius, scale::fahrenheit>
    {
        static double convert(double const value)
        {
            return (value * 9) / 5 + 32;
        }
    };

    template <>
    struct conversion_traits<scale::fahrenheit, scale::celsius>
    {
        static double convert(double const value)
        {
            return (value - 32) * 5 / 9;
        }
    };

    template <scale R, scale S>
    constexpr quantity<R> temperature_cast(quantity<S> const q)
    {
        return quantity<R>(conversion_traits<S, R>::convert(
            static_cast<double>(q)));
    }
}
```

В следующем листинге показаны операторы литералов для создания значений температуры. Эти операторы объявлены в отдельном пространстве имен

`temperature_scale_literals`, что считается хорошей практикой, способствующей снижению риска конфликта имен с другими операторами литералов:

```
namespace temperature
{
    namespace temperature_scale_literals
    {
        constexpr quantity<scale::celsius> operator "" _deg(
            long double const amount)
        {
            return quantity<scale::celsius> {static_cast<double>(amount)};
        }

        constexpr quantity<scale::fahrenheit> operator "" _f(
            long double const amount)
        {
            return quantity<scale::fahrenheit> {static_cast<double>(amount)};
        }
    }
}
```

Далее приводится пример, демонстрирующий, как определить два значения температур, одно в шкале Цельсия и другое в шкале Фаренгейта, и выполнять преобразования между ними:

```
int main()
{
    using namespace temperature;
    using namespace temperature_scale_literals;

    auto t1{ 36.5_deg };
    auto t2{ 79.0_f };

    auto tf = temperature_cast<scale::fahrenheit>(t1);
    auto tc = temperature_cast<scale::celsius>(tf);
    assert(t1 == tc);
}
```

---

# Глава 3

## Строки и регулярные выражения



### Задачи

#### 23. Преобразование чисел в строки

Напишите функцию, которая преобразует коллекцию 8-битных целых чисел (массив или вектор) в строку с шестнадцатеричными их представлениями. Функция должна поддерживать вывод символов в верхнем и в нижнем регистрах. Вот несколько примеров входных данных и соответствующих им строк на выходе:

Вход: { 0xBA, 0xAD, 0xF0, 0x0D }, выход: "BAADF00D" или "baadf00d"  
Вход: { 1, 2, 3, 4, 5, 6 }, выход: "010203040506"

#### 24. Преобразование строк в числа

Напишите функцию, которая преобразует строку с шестнадцатеричными цифрами в коллекцию 8-битных целых чисел (массив или вектор) в строку с шестнадцатеричными их представлениями. Вот несколько примеров:

Вход: "BAADF00D" или "baadf00d", выход: {0xBA, 0xAD, 0xF0, 0x0D}  
Вход: "010203040506", выход: {1, 2, 3, 4, 5, 6}

#### 25. Преобразование в верхний регистр первых букв слов

Напишите функцию, которая преобразует в верхний регистр первые буквы каждого слова в тексте и в нижний – все остальные. Например, текст "the c++ challenger" эта функция должна преобразовать в "The C++ Challenger".

#### 26. Объединение строк через разделитель

Напишите функцию, которая принимает список строк и разделитель и создает новую строку, объединяя входные строки через разделитель. Разделитель не должен появляться после последней строки, и для пустого списка функция должна возвращать пустую строку.

Пример:

входной список { "this", "is", "an", "example" } и разделитель ' ' (пробел),  
выход: "this is an example".



## 27. Разбиение строк на лексемы по разделителям из списка

Напишите функцию, которая принимает строку и список возможных символов-разделителей, разбивает строку на лексемы по встреченным разделителям и возвращает их в виде `std::vector`.

Пример:

входная строка "this.is.a sample!!" и разделители ",.!",  
выход: {"this", "is", "a", "sample"}.

## 28. Наибольшая подстрока-палиндром

Напишите функцию, которая принимает строку и отыскивает в ней наибольшую последовательность символов, являющуюся палиндромом. Если в строке присутствует несколько подстрок-палиндромов одинаковой наибольшей длины, функция должна вернуть первую из них.

## 29. Проверка номерного знака

Для номерных знаков вида *LLL-LL DDD* или *LLL-LL DDDD* (где *L* – буква верхнего регистра от *A* до *Z*, а *D* – цифра) напишите функции:

- одна должна проверять соответствие номера указанному формату;
- другая – извлекать из текста все найденные номерные знаки.



## 30. Извлечение частей URL

Напишите функцию, которая принимает строку с URL и извлекает из нее отдельные части (протокол, домен, порт, путь, параметры запроса и имя фрагмента).

## 31. Преобразование дат в строках

Напишите функцию, которая принимает строку с датами в формате *dd.mm.yyyy* или *dd-mm-yyyy* и преобразует эти даты в формат *yyuu-mm-dd*.

# Решения

## 23. Преобразование чисел в строки

Чтобы функция могла обрабатывать коллекции разных видов, такие как `std::array`, `std::vector`, C-подобные массивы и др., мы должны написать шаб-

лонную функцию. Далее приводятся две перегруженные версии; одна принимает контейнер и флаг, определяющий регистр символов в выходной строке, а другая принимает пару итераторов (указывающих на первый и последний элементы последовательности) и флаг регистра символов. Содержимое диапазона записывается в объект `std::ostringstream` с использованием соответствующих манипуляторов ввода/вывода, таких как ширина, заполняющий символ и флаг регистра:

```
template <typename Iter>
std::string bytes_to_hexstr(Iter begin, Iter end,
                           bool const uppercase = false)
{
    std::ostringstream oss;
    if(uppercase) oss.setf(std::ios_base::uppercase);
    for (; begin != end; ++begin)
        oss << std::hex << std::setw(2) << std::setfill('0')
            << static_cast<int>(*begin);
    return oss.str();
}

template <typename C>
std::string bytes_to_hexstr(C const & c, bool const uppercase = false)
{
    return bytes_to_hexstr(std::cbegin(c), std::cend(c), uppercase);
}
```

Далее приводится пример использования этих функций:

```
int main()
{
    std::vector<unsigned char> v{ 0xBA, 0xAD, 0xF0, 0x0D };
    std::array<unsigned char, 6> a{ {1,2,3,4,5,6} };
    unsigned char buf[5] = {0x11, 0x22, 0x33, 0x44, 0x55};

    assert(bytes_to_hexstr(v, true) == "BAADF00D");
    assert(bytes_to_hexstr(a, true) == "010203040506");
    assert(bytes_to_hexstr(buf, true) == "1122334455");

    assert(bytes_to_hexstr(v) == "baadf00d");
    assert(bytes_to_hexstr(a) == "010203040506");
    assert(bytes_to_hexstr(buf) == "1122334455");
}
```

## 24. Преобразование строк в числа

В этой задаче предлагается реализовать операцию, противоположную предыдущей задаче. Однако на этот раз мы можем написать обычную функцию, а не шаблонную. На входе она будет принимать `std::string_view` – легковесную

обертку последовательности символов, а на выходе возвращать вектор 8-битных целых чисел без знака. Следующая функция `hexstr_to_bytes` преобразует каждые два символа в значение типа `unsigned char` ("A0" превращается в `0xA0`), сохраняет результаты в `std::vector` и возвращает его:

```

unsigned char hexchar_to_int(char const ch)
{
    if (ch >= '0' && ch <= '9') return ch - '0';
    if (ch >= 'A' && ch <= 'F') return ch - 'A' + 10;
    if (ch >= 'a' && ch <= 'f') return ch - 'a' + 10;
    throw std::invalid_argument("Invalid hexadecimal character");
}

std::vector<unsigned char> hexstr_to_bytes(std::string_view str)
{
    std::vector<unsigned char> result;
    for (size_t i = 0; i < str.size(); i += 2)
    {
        result.push_back(
            (hexchar_to_int(str[i]) << 4) | hexchar_to_int(str[i+1]));
    }
    return result;
}

```



Эта функция предполагает, что входная строка содержит четное число шестнадцатеричных цифр. Если это число окажется нечетным, последняя цифра будет отброшена (то есть строка "BAD" превратится в вектор `{0xBA}`). В качестве дополнительного упражнения измените предыдущую функцию, чтобы она не отбрасывала последнюю цифру, а предполагала, что строка начинается с символа '0', то есть строка "BAD" должна превратиться в вектор `{0x0B, 0xAD}`. Как еще одно упражнение напишите версию функции, которая распознавала бы строки шестнадцатеричных чисел с разделителями, например пробелом ("BA AD F0 0D").

Следующий листинг показывает, как использовать эту функцию:

```

int main()
{
    std::vector<unsigned char> expected{ 0xBA, 0xAD, 0xF0, 0x0D, 0x42 };
    assert(hexstr_to_bytes("BAADF00D42") == expected);
    assert(hexstr_to_bytes("BaaDf00d42") == expected);
}

```

## 25. Преобразование в верхний регистр первых букв слов

Шаблонная функция `capitalize()`, реализация которой представлена ниже, работает со строками любых символов. Она не изменяет входную строку, а создает новую. Для этого используется `std::stringstream`. Функция перебирает все символы во входной строке и устанавливает признак нового слова всякий раз, когда встречается пробел или знак препинания. Входные символы преобразуются в верхний регистр, если они начинают новые слова, и в нижний во всех остальных случаях:

```
template <class Elem>
using tstring = std::basic_string<Elem, std::char_traits<Elem>,
    std::allocator<Elem>>;
template <class Elem>
using tstringstream = std::basic_stringstream<
    Elem, std::char_traits<Elem>, std::allocator<Elem>>;

template <class Elem>
tstring<Elem> capitalize(tstring<Elem> const & text)
{
    tstringstream<Elem> result;
    bool newWord = true;
    for (auto const ch : text)
    {
        newWord = newWord || std::ispunct(ch) || std::isspace(ch);
        if (std::isalpha(ch))
        {
            if (newWord)
            {
                result << static_cast<Elem>(std::toupper(ch));
                newWord = false;
            }
            else
                result << static_cast<Elem>(std::tolower(ch));
        }
        else result << ch;
    }
    return result.str();
}
```

Следующая программа демонстрирует использование этой функции:

```
int main()
{
    using namespace std::string_literals;
    assert("The C++ Challenger"s ==
        capitalize("the c++ challenger"s));
    assert("This Is An Example, Should Work!"s ==
```



```

    capitalize("THIS IS an ExAmplE, should wORk!"s));
}

```



## 26. Объединение строк через разделитель

В следующем листинге представлены две перегруженные функции с именем `join_strings()`. Одна принимает контейнер строк и указатель на последовательность символов-разделителей, а другая принимает два итератора с произвольным доступом, представляющих первый и последний элементы последовательности, и указатель на последовательность символов-разделителей. Обе возвращают новую строку, созданную путем объединения всех входных строк с использованием `std::ostringstream` и функции `std::copy`. Универсальная функция копирует все элементы из входного диапазона в выходной, представленный итератором вывода. Здесь мы использовали `std::ostream_iterator`, который применяет `operator<<` для записи указанного значения в заданный поток вывода, когда ему присваивается новое значение:

```

template <typename Iter>
std::string join_strings(Iter begin, Iter end,
                        char const * const separator)
{
    std::ostringstream os;
    std::copy(begin, end-1,
              std::ostream_iterator<std::string>(os, separator));
    os << *(end-1);
    return os.str();
}

template <typename C>
std::string join_strings(C const & c, char const * const separator)
{
    if (c.size() == 0) return std::string{};
    return join_strings(std::begin(c), std::end(c), separator);
}

int main()
{
    using namespace std::string_literals;
    std::vector<std::string> v1{ "this", "is", "an", "example" };
    std::vector<std::string> v2{ "example" };
    std::vector<std::string> v3{ };

    assert(join_strings(v1, " ") == "this is an example"s);
    assert(join_strings(v2, " ") == "example"s);
    assert(join_strings(v3, " ") == ""s);
}

```





Как дополнительное упражнение попробуйте изменить перегруженную версию, принимающую итераторы, чтобы она могла работать с другими типами итераторов, такими как двунаправленные итераторы. Это даст возможность использовать функцию со списками и другими контейнерами.



## 27. Разбиение строк на лексемы по разделителям из списка

В листинге ниже приводятся две версии функции разбиения строк на лексемы.

- Первая принимает единственный символ разделителя. Она разбивает входную строку с использованием строкового потока, инициализированного ее содержимым, применяя `std::getline()` для чтения фрагментов, пока не встретится следующий разделитель или признак конца строки.
- Вторая принимает список возможных разделителей в виде `std::string`. Она использует `std::string::find_first_of()` для поиска первого из символов-разделителей и переходит к первому символу за ним. Операция повторяется в цикле, пока не будет обработана вся входная строка. Извлеченные подстроки добавляются в вектор, играющий роль результата:

```
template <class Elem>
using tstring = std::basic_string<Elem, std::char_traits<Elem>,
                                std::allocator<Elem>>;

template <class Elem>
using tstringstream = std::basic_stringstream<
    Elem, std::char_traits<Elem>, std::allocator<Elem>>;

template<typename Elem>
inline std::vector<tstring<Elem>> split(tstring<Elem> text,
                                     Elem const delimiter)
{
    auto sstr = tstringstream<Elem>{ text };
    auto tokens = std::vector<tstring<Elem>>{};
    auto token = tstring<Elem>{};
    while (std::getline(sstr, token, delimiter))
    {
        if (!token.empty()) tokens.push_back(token);
    }
    return tokens;
}
```

```

template<typename Elem>
inline std::vector<tstring<Elem>> split(tstring<Elem> text,
                                       tstring<Elem> const & delimiters)
{
    auto tokens = std::vector<tstring<Elem>>{};
    size_t pos, prev_pos = 0;
    while ((pos = text.find_first_of(delimiters, prev_pos)) !=
           std::string::npos)
    {
        if (pos > prev_pos)
            tokens.push_back(text.substr(prev_pos, pos - prev_pos));
        prev_pos = pos + 1;
    }
    if (prev_pos < text.length())
        tokens.push_back(text.substr(prev_pos, std::string::npos));
    return tokens;
}

```

Следующий листинг демонстрирует два примера разбиения строк по одному или по нескольким разделителям:

```

int main()
{
    using namespace std::string_literals;
    std::vector<std::string> expected{"this", "is", "a", "sample"};
    assert(expected == split("this is a sample"s, ' '));
    assert(expected == split("this,is a.sample!!"s, "!.! "s));
}

```

## 28. Наибольшая подстрока-палиндром

Простейшее решение этой задачи – использовать метод прямого перебора, проверяющего каждую подстроку. Однако это означает, что нам придется проверить  $C(N, 2)$  подстрок (где  $N$  – число символов в строке), а временная сложность такого решения равна  $O(N^3)$ . Сложность можно уменьшить, сохраняя результаты подзадач. Для этого нам понадобится таблица логических значений с размерами  $N \times N$ , где элемент  $[i, j]$  сообщает, является ли палиндромом подстрока с  $i$ -го по  $j$ -й символы в исходной строке. Сначала инициализируем значением `true` все элементы  $[i, i]$  (односимвольные палиндромы) и все элементы  $[i, i+1]$  для всех пар одинаковых символов (двухсимвольные палиндромы). Затем переходим к проверке подстрок длиннее двух символов и записи значения `true` во все элементы  $[i, j]$ , если элемент  $[i+1, j-1]$  содержит значение `true` и в позициях  $i$  и  $j$  в исходной строке находятся одинаковые символы. Попутно запоминаем начальную позицию и длину наибольшей подстроки-палиндрома, чтобы получить ее после заполнения таблицы.

Вот как выглядит это решение в коде:

```

std::string longest_palindrome(std::string_view str)
{
    size_t const len = str.size();
    size_t longestBegin = 0;
    size_t maxLen = 1;

    std::vector<bool> table(len * len, false);
    for (size_t i = 0; i < len; i++)
        table[i*len + i] = true;

    for (size_t i = 0; i < len - 1; i++)
    {
        if (str[i] == str[i + 1])
        {
            table[i*len + i + 1] = true;
            if (maxLen < 2)
            {
                longestBegin = i;
                maxLen = 2;
            }
        }
    }

    for (size_t k = 3; k <= len; k++)
    {
        for (size_t i = 0; i < len - k + 1; i++)
        {
            size_t j = i + k - 1;
            if (str[i] == str[j] && table[(i + 1)*len + j - 1])
            {
                table[i*len + j] = true;
                if (maxLen < k)
                {
                    longestBegin = i;
                    maxLen = k;
                }
            }
        }
    }
    return std::string(str.substr(longestBegin, maxLen));
}

```



Следующий листинг демонстрирует тестирование функции `longest_palindrome()`:

```

int main()
{
    using namespace std::string_literals;

```

```

assert(longest_palindrome("sahararahnide") == "hararah");
assert(longest_palindrome("level") == "level");
assert(longest_palindrome("s") == "s");
}

```



## 29. Проверка номерного знака

Простейший способ решения этой задачи – использовать регулярные выражения. Описанному формату соответствует регулярное выражение "[A-Z]{3}-[A-Z]{2} \d{3,4}".

Первая функция просто проверяет входную строку на соответствие регулярному выражению. Для этого можно использовать `std::regex_match()`:

```

bool validate_license_plate_format(std::string_view str)
{
    std::regex rx(R"([A-Z]{3}-[A-Z]{2} \d{3,4})");
    return std::regex_match(str.data(), rx);
}

int main()
{
    assert(validate_license_plate_format("ABC-DE 123"));
    assert(validate_license_plate_format("ABC-DE 1234"));
    assert(!validate_license_plate_format("ABC-DE 12345"));
    assert(!validate_license_plate_format("abc-de 1234"));
}

```



Вторая функция немного отличается. Вместо сопоставления входной строки с регулярным выражением она должна выявить все совпадения с регулярным выражением внутри этой строки. Как следствие регулярное выражение изменится на "[A-Z]{3}-[A-Z]{2} \d{3,4})\*". Для обхода всех совпадений используем `std::sregex_iterator`:

```

std::vector<std::string> extract_license_plate_numbers(
    std::string const & str)
{
    std::regex rx(R"(([A-Z]{3}-[A-Z]{2} \d{3,4})*)");
    std::smatch match;
    std::vector<std::string> results;

    for(auto i = std::sregex_iterator(std::cbegin(str), std::cend(str), rx);
        i != std::sregex_iterator(); ++i)
    {
        if((*i)[1].matched)
            results.push_back(i->str());
    }

    return results;
}

```

```

}

int main()
{
    std::vector<std::string> expected {
        "AAA-AA 123", "ABC-DE 1234", "XYZ-WW 0001"};
    std::string text("AAA-AA 123qwe-ty 1234 ABC-DE 123456..XYZ-WW 0001");
    assert(expected == extract_license_plate_numbers(text));
}

```

## 30. Извлечение частей URL

Эта задача тоже легко решается с помощью регулярных выражений. Однако найти регулярное выражение, соответствующее любому URL, – очень непростая задача. Цель этого упражнения – дать вам возможность попрактиковать навыки владения библиотекой `regex`, а не тратить время на поиск регулярного выражения для данной конкретной цели. Поэтому регулярное выражение в этом примере следует считать лишь учебным примером.



У вас есть возможность опробовать свои регулярные выражения с помощью онлайн-инструментов и отладчиков, таких как <https://regex101.com/>. Это может пригодиться при создании своих регулярных выражений и даст вам возможность опробовать их на разных массивах данных.

Для целей этой задачи предположим, что URL включает следующие части: обязательные *протокол* и *домен* и необязательные *порт*, *путь*, *запрос* и *фрагмент*. Для получения результатов парсинга URL используется следующая структура (также можно было бы возвращать кортеж и использовать структурированную привязку для присваивания переменным разных элементов кортежа):

```

struct uri_parts
{
    std::string          protocol;
    std::string          domain;
    std::optional<int>   port;
    std::optional<std::string> path;
    std::optional<std::string> query;
    std::optional<std::string> fragment;
};

```

Далее показана одна из возможных реализаций функции, анализирующей адрес URL, извлекающей и возвращающей его элементы. Обратите внимание: возвращаемое значение имеет тип `std::optional<uri_parts>`, потому что функ-

ция может потерпеть неудачу при попытке сопоставить строку с регулярным выражением; в этом случае функция вернет значение `std::nullopt`:

```
std::optional<uri_parts> parse_uri(std::string uri)
{
    std::regex rx(R"^(^(\w+):\\\/(\\w.-]+)(:(\d+))?(\\w\\\/.]+)?(\?([\w=&]*)|#?(\\w+))?)?$)");
    auto matches = std::smatch{};
    if (std::regex_match(uri, matches, rx))
    {
        if (matches[1].matched && matches[2].matched)
        {
            uri_parts parts;
            parts.protocol = matches[1].str();
            parts.domain = matches[2].str();
            if (matches[4].matched)
                parts.port = std::stoi(matches[4]);
            if (matches[5].matched)
                parts.path = matches[5];
            if (matches[7].matched)
                parts.query = matches[7];
            if (matches[9].matched)
                parts.fragment = matches[9];
            return parts;
        }
    }
    return {};
}
```



Следующая программа тестирует функцию `parse_uri()` с двумя адресами URL, состоящими из разных элементов:

```
int main()
{
    auto p1 = parse_uri("https://packt.com");
    assert(p1.has_value());
    assert(p1->protocol == "https");
    assert(p1->domain == "packt.com");
    assert(!p1->port.has_value());
    assert(!p1->path.has_value());
    assert(!p1->query.has_value());
    assert(!p1->fragment.has_value());

    auto p2 = parse_uri("https://bbc.com:80/en/index.html?lite=true#ui");
    assert(p2.has_value());
    assert(p2->protocol == "https");
    assert(p2->domain == "bbc.com");
    assert(p2->port == 80);
}
```

```

assert(p2->path.value() == "/en/index.html");
assert(p2->query.value() == "lite=true");
assert(p2->fragment.value() == "ui");
}

```

## 31. Преобразование дат в строках



Преобразование текста тоже можно выполнять с помощью регулярных выражений, используя `std::regex_replace()`. Датам в указанных в условиях форматах соответствует регулярное выражение `(\d{1,2})(\.|-|/)(\d{1,2})(\.|-|/)(\d{4})`. Это выражение состоит из пяти сохраняющих групп; группа 1 сохраняет день, группа 2 – разделитель (. или -), группа 3 – месяц, группа 4 – снова разделитель (. или -), и группа 5 – год.

Поскольку нам требуется преобразовать даты из формата `dd.mm.yyyy` или `dd-mm-yyyy` в формат `yyyy-mm-dd`, регулярное выражение замены для `std::regex_replace()` должно иметь вид: `"($5-$3-$1)"`:



```

std::string transform_date(std::string_view text)
{
    auto rx = std::regex{ R"((\d{1,2})(\.|-|/)(\d{1,2})(\.|-|/)(\d{4}))" };
    return std::regex_replace(text.data(), rx, R"($5-$3-$1)");
}

int main()
{
    using namespace std::string_literals;
    assert(transform_date("today is 01.12.2017!"s) ==
           "today is 2017-12-01!"s);
}

```



---

# Глава 4

## Потоки данных и файловые системы

### Задачи

#### 32. Треугольник Паскаля

Напишите функцию, выводящую 10 строк треугольника Паскаля.

#### 33. Табличный вывод списка процессов

Представьте, что у вас есть список всех процессов в системе. Для каждого процесса известны имя, идентификатор, состояние (*работает* или *приостановлен*), имя пользователя (от имени которого запущен процесс), объем памяти в байтах и платформа (32 или 64 бита). Напишите функцию, которая принимает такой список процессов и выводит его в консоль в алфавитном порядке и в табличной форме. Данные во всех столбцах должны выводиться с выравниванием по левому краю, кроме столбца с объемом памяти, в котором числа должны выводиться с выравниванием по правому краю. Объем памяти должен выводиться в килобайтах. Ниже приводится пример вывода этой функции:

chrome.exe	1044	Running	marius.bancila	25180	32-bit
chrome.exe	10100	Running	marius.bancila	227756	32-bit
cmd.exe	512	Running	SYSTEM	48	64-bit
explorer.exe	7108	Running	marius.bancila	29529	64-bit
skype.exe	22456	Suspended	marius.bancila	656	64-bit

#### 34. Удаление пустых строк из текстового файла

Напишите программу, которая принимает путь к текстовому файлу и удаляет из этого файла все пустые строки. Строки, содержащие только пробельные символы, тоже считаются пустыми.

#### 35. Определение размера каталога

Напишите функцию, рекурсивно определяющую суммарный размер каталога в байтах. Она должна принимать флаг, определяющий необходимость следования по символическим ссылкам.

### 36. Удаление файлов старше заданной даты

Напишите функцию, которая принимает путь к каталогу и возраст и рекурсивно удаляет все записи (файлы и подкаталоги) с возрастом больше указанного. Допускается представлять возраст в любых единицах – днях, часах, минутах, секундах – или их комбинациях, таких как «один час и двадцать минут». Если указанный каталог сам старше заданного возраста, он должен быть удален целиком.

### 37. Поиск файлов в каталоге, соответствующих регулярному выражению

Напишите функцию, которая принимает путь к каталогу и регулярное выражение и возвращает список всех элементов каталога, имена которых соответствуют регулярному выражению.

### 38. Временные файлы журналов



Создайте класс журналов, который записывает текстовые сообщения во временный текстовый файл. Файл должен иметь уникальное имя и находиться во временном каталоге. Если не указано иное, файл должен удаляться при уничтожении экземпляра класса. При этом должна быть предусмотрена возможность сохранения файла путем перемещения его в другой каталог.

## Решения

### 32. Треугольник Паскаля

Треугольник Паскаля – это конструкция с биномиальными коэффициентами. Первая строка треугольника содержит единственное значение 1. Элементы каждой следующей строки образуются суммированием чисел в строке выше, слева направо, при этом отсутствующие значения принимаются равными 0. Вот пример треугольника с пятью строками:

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1

```

Чтобы вывести треугольник, мы должны:

- сдвинуть позицию вывода вправо на соответствующее количество пробелов, чтобы вершина треугольника оказалась точно над серединой его основания;
- вычислить каждое значение как сумму чисел слева и справа в строке выше: самая простая формула: значение  $x$  в строке  $i$  и столбце  $j$  равно

предыдущему значению  $x$ , умноженному на  $(i - j) / (j + 1)$ , где  $x$  начинается с 1.

Вот одна из возможных реализаций функции вывода треугольника Паскаля:

```
unsigned int number_of_digits(unsigned int const i)
{
    return i > 0 ? (int)log10((double)i) + 1 : 1;
}

void print_pascal_triangle(int const n)
{
    for (int i = 0; i < n; i++)
    {
        auto x = 1;
        std::cout << std::string((n - i - 1)*(n / 2), ' ');
        for (int j = 0; j <= i; j++)
        {
            auto y = x;
            x = x * (i - j) / (j + 1);
            auto maxlen = number_of_digits(x) - 1;
            std::cout << y << std::string(n - 1 - maxlen - n%2, ' ');
        }
        std::cout << std::endl;
    }
}
```

Следующая программа предлагает пользователю ввести число строк и выводит треугольник в консоль:

```
int main()
{
    int n = 0;
    std::cout << "Levels (up to 10): ";
    std::cin >> n;
    if (n > 10)
        std::cout << "Value too large" << std::endl;
    else
        print_pascal_triangle(n);
}
```

### 33. Табличный вывод списка процессов

Для решения этой задачи используем следующий класс, представляющий информацию о процессе:

```
enum class procstatus {suspended, running};
enum class platforms {p32bit, p64bit};

struct procinfo
```

```

{
    int          id;
    std::string name;
    procstatus  status;
    std::string account;
    size_t      memory;
    platforms   platform;
};

```



Чтобы вывести состояние и платформу в текстовом виде, нам понадобятся функции преобразования элементов перечислений в строки `std::string`:

```

std::string status_to_string(procstatus const status)
{
    if (status == procstatus::suspended) return "suspended";
    else return "running";
}

std::string platform_to_string(platforms const platform)
{
    if (platform == platforms::p32bit) return "32-bit";
    else return "64-bit";
}

```



Процессы в списке должны быть отсортированы по их именам. То есть первым шагом мы отсортируем диапазон процессов. А для вывода информации используем манипуляторы:

```

void print_processes(std::vector<procinfo> processes)
{
    std::sort(
        std::begin(processes), std::end(processes),
        [](procinfo const & p1, procinfo const & p2) {
            return p1.name < p2.name; });

    for (auto const & pi : processes)
    {
        std::cout << std::left << std::setw(25) << std::setfill(' ')
            << pi.name;
        std::cout << std::left << std::setw(8) << std::setfill(' ')
            << pi.id;
        std::cout << std::left << std::setw(12) << std::setfill(' ')
            << status_to_string(pi.status);
        std::cout << std::left << std::setw(15) << std::setfill(' ')
            << pi.account;
        std::cout << std::right << std::setw(10) << std::setfill(' ')
            << (int)(pi.memory/1024);
        std::cout << std::left << ' ' << platform_to_string(pi.platform);
    }
}

```

```

    std::cout << std::endl;
}
}

```

Следующая программа определяет список процессов (вы можете использовать для этого специализированный системный API) и выводит его с учетом требований:

```

int main()
{
    using namespace std::string_literals;

    std::vector<procinfo> processes
    {
        {512, "cmd.exe"s, procstatus::running, "SYSTEM"s,
         148293, platforms::p64bit },
        {1044, "chrome.exe"s, procstatus::running, "marius.bancila"s,
         25180454, platforms::p32bit},
        {7108, "explorer.exe"s, procstatus::running, "marius.bancila"s,
         2952943, platforms::p64bit },
        {10100, "chrome.exe"s, procstatus::running, "marius.bancila"s,
         227756123, platforms::p32bit},
        {22456, "skype.exe"s, procstatus::suspended, "marius.bancila"s,
         16870123, platforms::p64bit },
    };
    print_processes(processes);
}

```

## 34. Удаление пустых строк из текстового файла

Вот один из возможных алгоритмов решения этой задачи:

1. Создать пустой текстовый файл для сохранения только непустых строк из исходного файла.
2. Читать содержимое исходного файла строку за строкой и записывать во временный файл непустые строки.
3. Удалить исходный файл после его обработки.
4. Переименовать временный файл, присвоив ему имя исходного файла.

Как вариант можно переместить содержимое из временного файла в исходный, с затиранием имевшегося в нем текста. Реализация ниже следует шагам, описанным выше. Временный файл создается во временном каталоге, который возвращает `filesystem::temp_directory_path()`:

```

namespace fs = std::experimental::filesystem;

void remove_empty_lines(fs::path filepath)

```

```

{
    std::ifstream filein(filepath.native(), std::ios::in);
    if (!filein.is_open())
        throw std::runtime_error("cannot open input file");

    auto temp_path = fs::temp_directory_path() / "temp.txt";
    std::ofstream fileout(temp_path.native(),
        std::ios::out | std::ios::trunc);
    if (!fileout.is_open())
        throw std::runtime_error("cannot create temporary file");

    std::string line;
    while (std::getline(filein, line))
    {
        if (line.length() > 0 &&
            line.find_first_not_of(' ') != line.npos)
        {
            fileout << line << '\n';
        }
    }
    filein.close();
    fileout.close();

    fs::remove(filepath);
    fs::rename(temp_path, filepath);
}

```

## 35. Определение размера каталога

Чтобы определить размер каталога, нужно выполнить обход всех файлов в нем и подсчитать сумму их размеров.

Выполнить рекурсивный обход всех элементов каталога можно с помощью итератора `filesystem::recursive_directory_iterator` из библиотеки `filesystem`. Он имеет несколько конструкторов, часть из которых принимает значение типа `filesystem::directory_options`, определяющее необходимость следования по символическим ссылкам. Для подсчета суммы можно использовать универсальный алгоритм `std::accumulate()`. Поскольку общий размер каталога может превысить 2 Гбайт, для хранения суммы следует использовать значение типа `unsigned long long`, а не `int` или `long`. Следующая функция демонстрирует одну из возможных реализаций задачи:

```

namespace fs = std::experimental::filesystem;

std::uintmax_t get_directory_size(fs::path const & dir,
    bool const follow_symlinks = false)
{
    auto iterator = fs::recursive_directory_iterator(

```

```

dir,
follow_symlinks ? fs::directory_options::follow_directory_symlink :
                fs::directory_options::none);

return std::accumulate(
    fs::begin(iterator), fs::end(iterator),
    0ull,
    [](std::uintmax_t const total,
       fs::directory_entry const & entry) {
        return total + (fs::is_regular_file(entry) ?
                        fs::file_size(entry.path()) : 0);
    });
}

int main()
{
    std::string path;
    std::cout << "Path: ";
    std::cin >> path;
    std::cout << "Size: " << get_directory_size(path) << std::endl;
}

```



## 36. Удаление файлов старше заданной даты

Для выполнения операций с файловой системой следует использовать библиотеку `filesystem`. Для работы со временем и временными интервалами нужно применять библиотеку `chrono`. Вот что должна делать требуемая функция:

1. Проверить существование указанного каталога и, если он старше заданного возраста, удалить его.
2. Если каталог существует и моложе заданного возраста, выполнить обход всех его элементов, вызывая себя рекурсивно:

```

namespace fs = std::experimental::filesystem;
namespace ch = std::chrono;

template <typename Duration>
bool is_older_than(fs::path const & path, Duration const duration)
{
    auto ftimeduration = fs::last_write_time(path).time_since_epoch();
    auto nowduration = (ch::system_clock::now() - duration)
        .time_since_epoch();
    return ch::duration_cast<Duration>(nowduration - ftimeduration)
        .count() > 0;
}

template <typename Duration>

```

```

void remove_files_older_than(fs::path const & path,
                             Duration const duration)
{
    try
    {
        if (fs::exists(path))
        {
            if (is_older_than(path, duration))
            {
                fs::remove(path);
            }
            else if (fs::is_directory(path))
            {
                for (auto const & entry : fs::directory_iterator(path))
                {
                    remove_files_older_than(entry.path(), duration);
                }
            }
        }
    }
    catch (std::exception const & ex)
    {
        std::cerr << ex.what() << std::endl;
    }
}

```



Вместо `directory_iterator` и рекурсивного вызова `remove_files_older_than()` можно использовать `recursive_directory_iterator` и просто удалять элементы, если они старше заданного возраста. Однако этот подход может привести к непредсказуемым результатам, потому что если после создания рекурсивного итератора будет добавлен или удален какой-то файл или каталог, стандарт не гарантирует, что это изменение будет замечено итератором. Поэтому данный способ лучше не использовать.

Шаблонная функция `is_older_than()` определяет время, прошедшее от начала эпохи системных часов до текущего момента и до момента последнего изменения файла, вычисляет их разность и сравнивает результат с заданным возрастом.

Вот как можно использовать функцию `remove_files_older_than()`:

```

int main()
{
    using namespace std::chrono_literals;

#ifdef _WIN32
    auto path = R"(..\Test\)";
#else

```



```

    auto path = R"(..Test/)";
#endif
    remove_files_older_than(path, 1h + 20min);
}

```

## 37. Поиск файлов в каталоге, соответствующих регулярному выражению

Эта функция реализуется просто: она должна выполнить рекурсивный обход всех элементов заданного каталога и добавить в список только имена, принадлежащие обычным файлам и соответствующие регулярному выражению. С этой целью используем следующие инструменты:

- `filesystem::recursive_directory_iterator` для обхода элементов каталога;
- `regex` и `regex_match()` для проверки соответствия имен регулярному выражению;
- `copy_if()` и `back_inserter` для копирования в конец вектора элементов, соответствующих заданным критериям.

Вот одна из возможных реализаций такой функции:

```

namespace fs = std::experimental::filesystem;

std::vector<fs::directory_entry> find_files(
    fs::path const & path,
    std::string_view regex)
{
    std::vector<fs::directory_entry> result;
    std::regex rx(regex.data());

    std::copy_if(
        fs::recursive_directory_iterator(path),
        fs::recursive_directory_iterator(),
        std::back_inserter(result),
        [&rx](fs::directory_entry const & entry) {
            return fs::is_regular_file(entry.path()) &&
                std::regex_match(entry.path().filename().string(), rx);
        });
    return result;
}

```

В следующем листинге приводится программа, использующая эту функцию:

```

int main()
{
    auto dir = fs::temp_directory_path();
    auto pattern = R"(wct[0-9a-zA-Z]{3}\.tmp)";
}

```

```

auto result = find_files(dir, pattern);

for (auto const & entry : result)
{
    std::cout << entry.path().string() << std::endl;
}
}

```



## 38. Временные файлы журналов

Класс журналирования, который необходимо реализовать для решения этой задачи, должен:

- иметь конструктор, создающий текстовый файл во временном каталоге и открывающий его для записи;
- иметь деструктор, проверяющий наличие файла, закрывающий и удаляющий его;
- иметь метод, закрывающий файл и перемещающий его в указанный каталог;
- перегружать `operator<<` для записи текстовых сообщений в файл.

Чтобы получить уникальное имя файла, используем UUID (также известный как GUID). Стандарт C++ не поддерживает никакой функциональности, связанной с этим, зато имеются сторонние библиотеки, такие как `boost::uuid`, `Cross-Guid` или `stduuid` (последняя из них написана мной). В данной реализации я использую последнюю библиотеку из перечисленных. Вы найдете ее по адресу: <https://github.com/mariusbancila/stduuid>:

```

namespace fs = std::experimental::filesystem;

class logger
{
    fs::path logpath;
    std::ofstream logfile;
public:
    logger()
    {
        auto name = uuids::to_string(uuids::uuid_random_generator{}());
        logpath = fs::temp_directory_path() / (name + ".tmp");
        logfile.open(logpath.c_str(), std::ios::out|std::ios::trunc);
    }

    ~logger() noexcept
    {
        try {
            if(logfile.is_open()) logfile.close();
            if (!logpath.empty()) fs::remove(logpath);
        }
    }
}

```



```
    }
    catch (...) {}
}

void persist(fs::path const & path)
{
    logfile.close();
    fs::rename(logpath, path);
    logpath.clear();
}

logger& operator<<(std::string_view message)
{
    logfile << message.data() << '\n';
    return *this;
}
};
```

А вот пример использования этого класса:

```
int main()
{
    logger log;
    try
    {
        log << "this is a line" << "and this is another one";
        throw std::runtime_error("error");
    }
    catch (...)
    {
        log.persist(R"(lastlog.txt)");
    }
}
```



---

# Глава 5



## Дата и время

---

### Задачи

#### 39. Измерение времени выполнения функции



Напишите функцию, измеряющую время выполнения другой функции (с произвольным числом аргументов) в секундах, миллисекундах, микросекундах.

#### 40. Число дней между двумя датами

Напишите функцию, принимающую две даты и возвращающую число дней между ними. Функция не должна зависеть от порядка, в каком указаны даты.

#### 41. День недели

Напишите функцию, принимающую дату и возвращающую день недели. Функция должна возвращать числовое значение между 1 (понедельник) и 7 (воскресенье).

#### 42. День и неделя года

Напишите функцию, принимающую дату и возвращающую номер дня в году (от 1 до 365 или 366 для високосного года), и вторую функцию, также принимающую дату и возвращающую номер недели в году, в которую попадает эта дата.

#### 43. Время встречи для нескольких часовых поясов

Напишите функцию, принимающую список участников встречи и их часовые пояса и выводящую местное время встречи для каждого участника.

#### 44. Календарь на месяц

Напишите функцию, принимающую год и месяц и выводящую в консоль календарь на этот месяц. Ниже показан пример вывода (для декабря 2017 г.):

Mon	Tue	Wed	Thu	Fri	Sat	Sun
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31



## Решения

### 39. Измерение времени выполнения функции

Чтобы измерить время выполнения функции, нужно получить текущее время, вызвать функцию, затем снова получить текущее время и определить разность между двумя значениями времени. Для удобства напомним шаблонную функцию с переменным числом аргументов, которая принимает функцию для выполнения и ее аргументы и:

- использует `std::high_resolution_clock` для определения текущего времени;
- использует `std::invoke()` для вызова оцениваемой функции с ее аргументами;
- возвращает продолжительность выполнения в микросекундах, а не в тактах системы; это важно, чтобы не проиграть в точности; такой подход позволяет складывать продолжительность, измеренную с разной точностью, что было бы невозможно, если бы измерения производились в тактах системных часов:

```
template <typename Time = std::chrono::microseconds,
         typename Clock = std::chrono::high_resolution_clock>
struct perf_timer
{
    template <typename F, typename... Args>
    static Time duration(F&& f, Args... args)
    {
        auto start = Clock::now();
        std::invoke(std::forward<F>(f), std::forward<Args>(args)...);
        auto end = Clock::now();
        return std::chrono::duration_cast<Time>(end - start);
    }
};
```

Вот как можно использовать эту функцию:

```
void f()
{
    // имитировать работу
```



```
std::this_thread::sleep_for(2s);
}

void g(int const a, int const b)
{
    // имитировать работу
    std::this_thread::sleep_for(1s);
}

int main()
{
    auto t1 = perf_timer<std::chrono::microseconds>::duration(f);
    auto t2 = perf_timer<std::chrono::milliseconds>::duration(g, 1, 2);

    auto total = std::chrono::duration<double, std::nano>(t1 + t2).count();
}
```

## 40. Число дней между двумя датами

Стандартная библиотека `chrono` из стандарта C++17 пока не поддерживает работу с датами, неделями, календарями, часовыми поясами и другими интересными величинами. Но предполагается, что ситуация изменится с выходом стандарта C++20, так как поддержка часовых поясов и календарей уже была добавлена в стандарт на встрече в Джексонвилле в марте 2018 г. Новые дополнения основаны на открытой библиотеке `date`, созданной поверх `chrono` Ховардом Хиннантом (Howard Hinnant) и доступной на GitHub: <https://github.com/HowardHinnant/date>. Мы используем эту библиотеку в решениях некоторых задач из данной главы. В этой реализации используется пространство имен `date`, но в C++20 оно станет частью `std::chrono`. Если вы читаете эту книгу, когда стандарт C++20 уже вышел, вам достаточно будет изменить только название пространства имен, а остальной код останется неизменным.

Для решения этой задачи используем класс `date::sys_days`, объявленный в заголовке `date.h`. Он представляет количество дней, прошедших с начала эпохи `std::system_clock`. Время измеряется этим классом с точностью до одного дня и неявно преобразуется в `std::system_clock::time_point`. Фактически мы должны сконструировать два объекта этого типа и потом найти разность между ними. Результатом будет число дней между двумя датами. Вот простейшая реализация такой функции:

```
inline int number_of_days(
    int const y1, unsigned int const m1, unsigned int const d1,
    int const y2, unsigned int const m2, unsigned int const d2)
{
    using namespace date;

    return (sys_days{ year{ y1 } / month{ m1 } / day{ d1 } } -
```

```

        sys_days{ year{ y2 } / month{ m2 } / day{ d2 } }).count();
    }

    inline int number_of_days(date::sys_days const & first,
                              date::sys_days const & last)
    {
        return (last - first).count();
    }

```



Далее приводится пример использования этих перегруженных функций:

```

int main()
{
    auto diff1 = number_of_days(2016, 9, 23, 2017, 5, 15);

    using namespace date::literals;
    auto diff2 = number_of_days(2016_y/sep/23, 15_d/may/2017);
}

```

## 41. День недели

Эта задача тоже решается относительно просто, если использовать ту же библиотеку `date`. Но на сей раз мы должны использовать следующие типы:

- `date::year_month_day`, структура, представляющая день, имеющая поля, хранящие год, месяц (от 1 до 12) и день месяца (от 1 до 31);
- `date::iso_week::year_weeknum_weekday`, из заголовка `iso_week.h`, структура с полями для хранения года, номера недели в году и номера дня в неделе (от 1 до 7). Экземпляры этого класса неявно преобразуются в/из `date::sys_days`, что позволяет явно преобразовывать в любые другие календарные системы, неявно преобразуемые в/из `date::sys_days`, такие как `date::year_month_day`.

Итак, чтобы решить задачу, нужно создать объект `year_month_day` для представления желаемой даты и затем из него получить объект `year_weeknum_weekday`, из которого извлечь день недели с помощью `weekday()`:

```

unsigned int week_day(int const y, unsigned int const m,
                      unsigned int const d)
{
    using namespace date;

    if(m < 1 || m > 12 || d < 1 || d > 31) return 0;

    auto const dt = date::year_month_day{year{ y }, month{ m }, day{ d }};
    auto const tiso = iso_week::year_weeknum_weekday{ dt };

    return (unsigned int)tiso.weekday();
}

```

```

}

int main()
{
    auto wday = week_day(2018, 5, 9);
}

```



## 42. День и неделя года

Эта задача легко решается после решения двух предыдущих задач:

- чтобы определить день года, нужно найти разность двух объектов `date::sys_days`, один из которых представляет заданную дату, а другой 0 января того же года; как вариант можно определить второй объект как 1 января и прибавить к разности 1;
- чтобы определить номер недели в году, нужно создать объект `year_weeknum_weekday`, как в решении предыдущей задачи, и получить значение `weeknum()`:

```

int day_of_year(int const y, unsigned int const m,
               unsigned int const d)
{
    using namespace date;
    if(m < 1 || m > 12 || d < 1 || d > 31) return 0;

    return (sys_days{ year{ y } / month{ m } / day{ d } } -
            sys_days{ year{ y } / jan / 0 }).count();
}

```



```

unsigned int calendar_week(int const y, unsigned int const m,
                          unsigned int const d)
{
    using namespace date;
    if(m < 1 || m > 12 || d < 1 || d > 31) return 0;

    auto const dt = date::year_month_day{year{ y }, month{ m }, day{ d }};
    auto const tiso = iso_week::year_weeknum_weekday{ dt };

    return (unsigned int)tiso.weeknum();
}

```

Вот как можно использовать эти функции:

```

int main()
{
    int y = 0;
}

```



```

unsigned int m = 0, d = 0;
std::cout << "Year:"; std::cin >> y;
std::cout << "Month:"; std::cin >> m;
std::cout << "Day:"; std::cin >> d;

std::cout << "Calendar week:" << calendar_week(y, m, d) << std::endl;
std::cout << "Day of year:" << day_of_year(y, m, d) << std::endl;
}

```



### 43. Время встречи для нескольких часовых поясов

Для работы с часовыми поясами нужно подключить заголовок `tz.h` из библиотеки `date`. Но при этом на компьютер должна быть загружена и установлена база данных часовых поясов IANA Time Zone Database.

Вот как подготовить базу данных часовых поясов для использования библиотекой `date`:

- загрузите последнюю версию базы данных с сайта <https://www.iana.org/time-zones>. В настоящее время она доступна в виде архива с именем `tzdata2017c.tar.gz`;
- распакуйте архив в любой каталог на своем компьютере, в подкаталог с именем `tzdata`; допустим, что вы решили установить базу данных в каталог `c:\work\challenges\libs\date` (на компьютере с Windows); тогда в нем нужно создать подкаталог `tzdata`;
- для Windows загрузите файл `windowsZones.xml`, содержащий соответствия между часовыми поясами Windows и IANA; он доступен по адресу: <https://unicode.org/repos/clldr/trunk/common/supplemental/windowsZones.xml>; файл следует сохранить в тот же подкаталог `tzdata`, созданный выше;
- в настройках проекта определите макрос препроцессора с именем `INSTALL`, определяющий родительский каталог для подкаталога `tzdata`; для данного примера этот макрос должен определяться так: `INSTALL=c:\work\challenges\libs\date` (обратите внимание на повторяющиеся обратные слэши, они необходимы, потому что макрос используется для определения пути).

Для решения задачи определим следующую структуру с минимально необходимой информацией, такой как имя и часовой пояс. Значение часового пояса будет создаваться вызовом функции `date::locate_zone()`:

```

struct user
{
    std::string Name;
    date::time_zone const * Zone;

    explicit user(std::string_view name, std::string_view zone)

```

```

        : Name{name.data()}, Zone(date::locate_zone(zone.data()))
    {}
};

```

Функция, которая выводит список пользователей и местное время начала встречи для каждого, должна преобразовать время из базового часового пояса в их часовые пояса. Для этого можно использовать конструктор преобразования класса `date::zoned_time`:

```

template <class Duration, class TimeZonePtr>
void print_meeting_times(
    date::zoned_time<Duration, TimeZonePtr> const & time,
    std::vector<user> const & users)
{
    std::cout
        << std::left << std::setw(15) << std::setfill(' ')
        << "Local time: "
        << time << std::endl;

    for (auto const & user : users)
    {
        std::cout
            << std::left << std::setw(15) << std::setfill(' ')
            << user.Name
            << date::zoned_time<Duration, TimeZonePtr>(user.Zone, time)
            << std::endl;
    }
}

```

Вот как можно использовать эту функцию, где время (часы и минуты) представлено в текущем часовом поясе:

```

int main()
{
    std::vector<user> users{
        user{ "Ildiko", "Europe/Budapest" },
        user{ "Jens", "Europe/Berlin" },
        user{ "Jane", "America/New_York" }
    };

    unsigned int h, m;
    std::cout << "Hour:"; std::cin >> h;
    std::cout << "Minutes:"; std::cin >> m;

    date::year_month_day today =
        date::floor<date::days>(ch::system_clock::now());

    auto localtime = date::zoned_time<std::chrono::minutes>(

```

```

    date::current_zone(),
    static_cast<date::local_days>(today)+ch::hours{h}+ch::minutes{m});

    print_meeting_times(localtime, users);
}

```

## 44. Календарь на месяц



Решение этой задачи отчасти основано на решениях предыдущих задач. Чтобы вывести календарь на месяц, как определено в условиях задачи, вы должны узнать:

- каким днем недели является первое число месяца; ответить на этот вопрос можно с помощью функции `week_day()`, написанной нами в решении предыдущей задачи;
- сколько дней в месяце; это число можно определить с помощью структуры `date::year_month_day_last` и метода `day()`.

Получив ответы на эти вопросы, далее необходимо:

- вывести пробелы в первой неделе, предшествующие первому числу месяца;
- вывести числа, с применением правильного форматирования, от 1 до последнего дня месяца;
- вставить перенос строки после каждого седьмого дня, начиная с первого дня первой недели, даже если он принадлежит другому месяцу.

Далее показана реализация всех этих пунктов:

```

unsigned int week_day(int const y, unsigned int const m,
                    unsigned int const d)
{
    using namespace date;

    if(m < 1 || m > 12 || d < 1 || d > 31) return 0;

    auto const dt = date::year_month_day{year{ y }, month{ m }, day{ d }};
    auto const tiso = iso_week::year_weeknum_weekday{ dt };

    return (unsigned int)tiso.weekday();
}

void print_month_calendar(int const y, unsigned int m)
{
    using namespace date;
    std::cout << "Mon Tue Wed Thu Fri Sat Sun" << std::endl;

    auto first_day_weekday = week_day(y, m, 1);

```

```
auto last_day = (unsigned int)year_month_day_last(
    year{ y }, month_day_last{ month{ m } }).day();

unsigned int index = 1;
for (unsigned int day = 1; day < first_day_weekday; ++day, ++index)
{
    std::cout << " ";
}

for (unsigned int day = 1; day <= last_day; ++day)
{
    std::cout << std::right << std::setfill(' ') << std::setw(3)
        << day << ' ';
    if (index++ % 7 == 0) std::cout << std::endl;
}

std::cout << std::endl;
}

int main()
{
    print_month_calendar(2017, 12);
}
```



---

# Глава 6

## Алгоритмы и структуры данных



### Задачи

#### 45. Приоритетная очередь

Напишите структуру данных, представляющую приоритетную очередь, которая гарантирует постоянное время поиска наибольшего элемента, но логарифмическое время добавления и удаления элементов. Очередь должна вставлять элементы в конец и удалять из начала. По умолчанию очередь должна использовать `operator<` для сравнения элементов, но также давать пользователю возможность указать свою функцию сравнения объектов, возвращающую `true`, если первый аргумент меньше второго. Реализация должна поддерживать, по меньшей мере, следующие операции:

- `push()` для добавления нового элемента;
- `pop()` для удаления элемента из начала очереди;
- `top()` для доступа к элементу в начале очереди;
- `size()` для получения количества элементов в очереди;
- `empty()` для получения признака пустой очереди.

#### 46. Циклический буфер

Напишите структуру данных, представляющую циклический буфер фиксированного размера. Циклический буфер затирает существующие элементы, когда он заполнен до предела и производится попытка добавить новый элемент. Структура должна:

- запрещать создание с помощью конструктора по умолчанию;
- поддерживать создание объектов заданного размера;
- позволять проверять емкость буфера и его состояние (`empty()`, `full()`, `size()`, `capacity()`);

- давать возможность добавлять новые элементы с возможным затиранием наиболее старых элементов;
- удалять самые старые элементы;
- поддерживать обход элементов.

## 47. Двойной буфер

Напишите класс, представляющий буфер, поддерживающий возможность одновременного чтения и записи без конфликтов операций. Операция чтения должна давать доступ к старым элементам буфера, пока выполняется операция записи нового элемента. Вновь записанные данные должны быть доступны для чтения сразу после завершения операции записи.

## 48. Самый часто встречающийся элемент в диапазоне

Напишите функцию, принимающую диапазон и возвращающую самый часто встречающийся элемент в этом диапазоне, а также сколько раз встретился этот элемент. Если в диапазоне имеется несколько самых часто встречающихся элементов с одинаковой частотой, тогда функция должна вернуть все элементы. Например, для диапазона  $\{1, 1, 3, 5, 8, 13, 3, 5, 8, 8, 5\}$  она должна вернуть  $\{5, 3\}$  и  $\{8, 3\}$ .

## 49. Текстовая гистограмма

Напишите программу, которая принимает входной текст, и определяет частоту появления всех алфавитных символов и выводит гистограмму. Под частотой в данном случае понимается процент появлений каждого символа в тексте. Программа должна подсчитывать вхождения только алфавитных символов и игнорировать цифры, знаки препинания и другие возможные символы. Частота должна определяться на основе числа алфавитных символов, а также общего размера текста.

## 50. Фильтрация списка телефонных номеров

Напишите функцию, которая принимает список телефонных номеров и возвращает только те из них, которые принадлежат указанной стране. Страна определяется телефонным кодом страны, например 44 – код Великобритании. Телефонные номера, включающие код страны, могут начинаться со знака «плюс» (+), за которым следует сам код, но могут не иметь кода страны. Такие номера без кода страны должны игнорироваться.

## 51. Преобразование списка телефонных номеров

Напишите функцию, которая принимает список телефонных номеров, преобразует их так, чтобы они начинались с кода страны, которому предшествует знак «плюс» (+). Также из номеров должны удаляться любые пробельные символы. Вот пример списка на входе и на выходе:

```

07555 123456 => +447555123456
07555123456 => +447555123456
+44 7555 123456 => +447555123456
44 7555 123456 => +447555123456
7555 123456 => +447555123456

```

## 52. Генерация всех перестановок символов в строке

Напишите функцию, которая выведет в консоль все возможные перестановки символов в строке. Реализуйте две версии функции: рекурсивную и ~~не~~рекурсивную.



## 53. Средний рейтинг фильмов

Напишите программу, которая вычислит и выведет средний рейтинг по списку фильмов. Каждый фильм в списке имеет рейтинг от 1 до 10 (где 1 – низший рейтинг, а 10 – высший). Вычисляя средний рейтинг, отбросьте 5% фильмов с высшим и низшим рейтингами и по остатку вычислите среднее. Результат должен выводиться с десятичной точкой.

## 54. Алгоритм объединения в пары

Напишите универсальную функцию, которая принимает диапазон элементов и возвращает новый диапазон, состоящий из пар соседних элементов. Если число элементов во входном диапазоне нечетное, последний элемент должен игнорироваться. Например, для диапазона {1, 1, 3, 5, 8, 13, 21} функция должна вернуть { {1, 1}, {3, 5}, {8, 13} }.

## 55. Алгоритм «сшивания»

Напишите функцию, принимающую два диапазона и возвращающую новый диапазон с парами элементов из обоих диапазонов. Если два диапазона имеют разную длину, результат должен иметь длину по наименьшему из них. Например, для входных диапазонов { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 } и { 1, 1, 3, 5, 8, 13, 21 } должен получиться результат { {1,1}, {2,1}, {3,3}, {4,5}, {5,8}, {6,13}, {7,21} }.

## 56. Алгоритм выбора

Напишите функцию, которая принимает диапазон значений и функцию проекции, преобразует каждое значение в новое и возвращает новый диапазон с выбранными значениями. Например, если предположить, что у нас имеется тип book с полями id, title и author и диапазон значений типа book, функция должна позволить выбрать только названия книг. Вот пример использования функции:

```

struct book
{

```

```

int      id;
std::string title;
std::string author;
};

std::vector<book> books{
    {101, "The C++ Programming Language", "Bjarne Stroustrup"},
    {203, "Effective Modern C++", "Scott Meyers"},
    {404, "The Modern C++ Programming Cookbook", "Marius Bancila"}};

auto titles = select(books, [](book const & b) {return b.title; });

```

## 57. Алгоритм сортировки

Напишите функцию, которая принимает пару итераторов с произвольным доступом, определяющих границы диапазона, и сортирует элементы в этом диапазоне с использованием алгоритма быстрой сортировки. Вы должны определить две перегруженные версии функции: одну, использующую для сравнения элементов `operator<`, и другую, принимающую пользовательскую функцию сравнения.

## 58. Кратчайший путь между узлами

Напишите программу, принимающую сеть узлов и расстояний между ними, вычисляющую и отображающую кратчайшие расстояния из указанного узла во все остальные, а также путь между начальным и конечным узлами. Например, для неориентированного графа на рис. 6.1 и узла A.

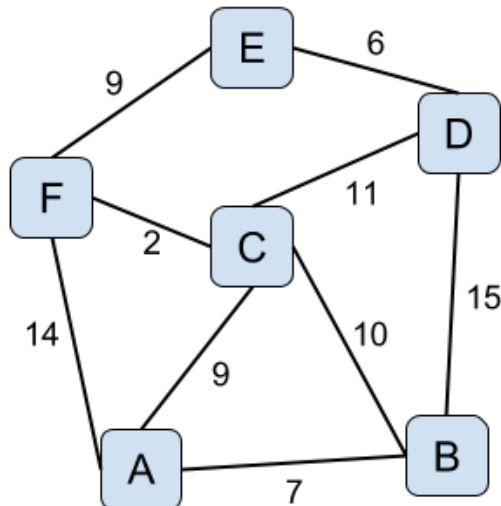


Рис. 6.1. Пример графа для задачи



Программа должна вывести следующее:

```
A -> A : 0      A
A -> B : 7      A -> B
A -> C : 9      A -> C
A -> D : 20     A -> C -> D
A -> E : 20     A -> C -> F -> E
A -> F : 11     A -> C -> F
```



## 59. Программа Weasel

Напишите программу, реализующую алгоритм эволюции Ричарда Докинза (Richard Dawkins), описанный Докинзом в книге «The Blind Watchmaker»<sup>1</sup> (глава 3):

*Мы снова используем нашу компьютерную обезьяну, но внесем критическое изменение в программу. Она, как и раньше, снова начинает, выбирая случайную последовательность из 28 букв... реплицирует ее неоднократно, но с некоторой вероятностью случайной ошибки – «мутации» – при копировании. Компьютер исследует мутировавшую бессмысленную фразу, «потомка» первоначальной, и выбирает ту, которая более всего (хотя бы чуть-чуть) походит на нашу искомую фразу, «METHINKS IT IS LIKE A WEASEL»<sup>2</sup>.*

## 60. Игра «Жизнь»

Напишите программу, реализующую клеточный автомат «Игра "Жизнь"», предложенный Джоном Хортоном Конвеем (John Horton Conway). Вселенной для этой игры является сетка из квадратных ячеек. Каждая клетка может находиться в одном из двух состояний: живая или мертвая. Каждая клетка взаимодействует с соседними клетками, подчиняясь следующим правилам на каждом шаге игры:

- любая живая клетка погибает от «одиночества», если по соседству оказывается меньше двух живых клеток;
- любая живая клетка с двумя или тремя живыми клетками по соседству выживает и переходит в следующее поколение;
- любая живая клетка погибает от «перенаселенности», если по соседству оказывается больше трех живых соседей;
- в мертвой клетке зарождается жизнь, если рядом с ней оказывается точно три живые клетки.

<sup>1</sup> Докинз Ричард. Слепой часовщик. Как эволюция доказывает отсутствие замысла во Вселенной. 1987, Corpus (ACT). ISBN: 978-5-17-086374-7. – Прим. перев.

<sup>2</sup> «Мне кажется, оно похоже на горностаю» – цитата из произведения «Гамлет», У. Шекспир. – Прим. перев.

Состояние игрового поля должно отображаться в консоли после каждой итерации. Для удобства выберите не очень большой размер поля, например 20 строк × 50 столбцов.

## Решения



### 45. Приоритетная очередь

Приоритетная очередь – это абстрактный тип данных, элементы которого имеют приоритеты. Очередь этого типа работает не как контейнер «первым пришел, первым ушел», а выдает элементы в порядке их приоритетов. Эта структура данных используется, например, в алгоритме Дейкстры (Dijkstra) поиска кратчайшего пути, в алгоритме Прима (Prim) сортировки кучи, в алгоритме поиска  $A^*$ , в кодах Хаффмана (Huffman) для сжатия данных и др.

Самый простой подход к реализации приоритетной очереди – использовать в качестве внутреннего контейнера `std::vector` и всегда поддерживать его в отсортированном состоянии, когда минимальный и максимальный элементы находятся на противоположных концах. Однако этот подход не поддерживает некоторых эффективных операций.

Наиболее подходящей структурой данных в подобном случае является куча. Это древовидная структура данных, обладающая следующими свойствами: если  $P$  – родительский узел для  $C$ , тогда ключ (значение) узла  $P$  больше или равно (в невозрастающей куче) или меньше или равно (в неубывающей куче) ключу узла  $C$ .

В стандартной библиотеке имеется несколько операций для работы с кучами:

- `std::make_heap()`: создает невозрастающую кучу для заданного диапазона с использованием `operator<` или пользовательской функции сравнения для упорядочения элементов;
- `std::push_heap()`: вставляет новый элемент в конец невозрастающей кучи;
- `std::pop_heap()`: удаляет первый элемент из кучи (меняя местами значения в первой и последней позициях и усекая диапазон `[first, last-1)` невозрастающей кучи).

Ниже показана реализация приоритетной очереди, использующая `std::vector` для хранения данных и стандартные функции для управления кучей:

```
template <class T,
class Compare = std::less<typename std::vector<T>::value_type>>
class priority_queue
{
    typedef typename std::vector<T>::value_type value_type;
```

```

typedef typename std::vector<T>::size_type size_type;
typedef typename std::vector<T>::reference reference;
typedef typename std::vector<T>::const_reference const_reference;
public:
    bool empty() const noexcept { return data.empty(); }
    size_type size() const noexcept { return data.size(); }

    void push(value_type const & value)
    {
        data.push_back(value);
        std::push_heap(std::begin(data), std::end(data), comparer);
    }

    void pop()
    {
        std::pop_heap(std::begin(data), std::end(data), comparer);
        data.pop_back();
    }

    const_reference top() const { return data.front(); }

    void swap(priority_queue& other) noexcept
    {
        swap(data, other.data);
        swap(comparer, other.comparer);
    }
private:
    std::vector<T> data;
    Compare comparer;
};

template<class T, class Compare>
void swap(priority_queue<T, Compare>& lhs,
          priority_queue<T, Compare>& rhs)
noexcept(noexcept(lhs.swap(rhs)))
{
    lhs.swap(rhs);
}

```



В следующем листинге демонстрируется порядок использования этого класса:

```

int main()
{
    priority_queue<int> q;
    for (int i : {1, 5, 3, 1, 13, 21, 8})
    {
        q.push(i);
    }
}

```



```
}  
  
assert(!q.empty());  
assert(q.size() == 7);  
  
while (!q.empty())  
{  
    std::cout << q.top() << ' ';  
    q.pop();  
}  
}
```

## 46. Циклический буфер

Циклический буфер – это контейнер фиксированного размера, действующий так, будто своими концами он связан в кольцо. Основное его преимущество в том, что не требует большого объема памяти для хранения данных, потому что старые записи затираются новыми. Циклические буферы используются в буферизации ввода/вывода, ограниченном журналировании (когда требуется хранить только самые последние сообщения), для буферов в асинхронной обработке и в других сценариях.

Мы можем выделить две разные ситуации:

- 1) число элементов в буфере не достигло его емкости (фиксированного размера, определяемого пользователем). В этом случае циклический буфер действует как обычный линейный контейнер, такой как вектор;
- 2) число элементов в буфере достигло его емкости. В этом случае память буфера используется повторно и новые элементы затирают старые.

Реализовать такую структуру можно с использованием:

- обычного контейнера с предварительно выделенной памятью для заданного числа элементов;
- указателя на начало, отмечающего позицию последнего вставленного элемента;
- счетчика элементов, хранящего количество элементов в контейнере, которое не может превысить емкость буфера (с этого момента новые элементы начнут затирать старые).

Циклический буфер должен поддерживать две основные операции:

- добавление нового элемента в буфер; элемент всегда должен добавляться в ячейку, следующую за указателем на начало; это действие выполняет метод `push()` в листинге ниже;
- удаление существующего элемента из буфера; удаляться всегда должен самый старый элемент, находящийся в позиции, на которую ссылается указатель на начало, минус количество элементов в буфере (при этом

необходимо учитывать циклическую природу буфера); это действие выполняет метод `pop()` в листинге ниже.

Реализация циклического буфера представлена ниже:

```
template <class T>
class circular_buffer
{
    typedef circular_buffer_iterator<T> const_iterator;

    circular_buffer() = delete;
public:
    explicit circular_buffer(size_t const size) :data_(size)
    {}

    bool clear() noexcept { head_ = -1; size_ = 0; }
    bool empty() const noexcept { return size_ == 0; }
    bool full() const noexcept { return size_ == data_.size(); }
    size_t capacity() const noexcept { return data_.size(); }
    size_t size() const noexcept { return size_; }

    void push(T const item)
    {
        head_ = next_pos();
        data_[head_] = item;
        if (size_ < data_.size()) size_++;
    }

    T pop()
    {
        if (empty()) throw std::runtime_error("empty buffer");
        auto pos = first_pos();
        size_--;
        return data_[pos];
    }

    const_iterator begin() const
    {
        return const_iterator(*this, first_pos(), empty());
    }

    const_iterator end() const
    {
        return const_iterator(*this, next_pos(), true);
    }
private:
    std::vector<T> data_;
    size_t head_ = -1;

```

```

size_t size_ = 0;

size_t next_pos() const noexcept
{ return size_ == 0 ? 0 : (head_ + 1) % data_.size(); }
size_t first_pos() const noexcept
{ return size_ == 0 ? 0 : (head_ + data_.size() - size_ + 1) %
  data_.size(); }

friend class circular_buffer_iterator<T>;
};

```



Поскольку циклический буфер хранится в памяти с линейной организацией, итератор для этого класса не может быть указателем. Итераторы должны иметь возможность ссылаться на элементы, с применением к индексу операции деления по модулю. Вот возможная реализация такого итератора:

```

template <class T>
class circular_buffer_iterator
{
    typedef circular_buffer_iterator    self_type;
    typedef T                          value_type;
    typedef T&                         reference;
    typedef T const&                   const_reference;
    typedef T*                          pointer;
    typedef std::random_access_iterator_tag iterator_category;
    typedef ptrdiff_t                  difference_type;
public:
    circular_buffer_iterator(circular_buffer<T> const & buf,
                           size_t const pos, bool const last) :
        buffer_(buf), index_(pos), last_(last)
    {}

    self_type & operator++ ()
    {
        if (last_)
            throw std::out_of_range("Iterator cannot be incremented past the end
of range.");
        index_ = (index_ + 1) % buffer_.data_.size();
        last_ = index_ == buffer_.next_pos();
        return *this;
    }

    self_type operator++ (int)
    {
        self_type tmp = *this;
        ++*this;
        return tmp;
    }
};

```





```

}

bool operator== (self_type const & other) const
{
    assert(compatible(other));
    return index_ == other.index_ && last_ == other.last_;
}

bool operator!= (self_type const & other) const
{
    return !(*this == other);
}

const_reference operator* () const
{
    return buffer_.data_[index_];
}

const_reference operator-> () const
{
    return buffer_.data_[index_];
}
private:
bool compatible(self_type const & other) const
{
    return &buffer_ == &other.buffer_;
}

circular_buffer<T> const & buffer_;
size_t index_;
bool last_;
};

```



Закончив реализацию, можно написать код, как показано ниже. Обратите внимание, что в первом диапазоне, в комментариях, показано фактическое содержимое внутреннего вектора, а во втором – логическое содержимое, доступное посредством итераторов:

```

int main()
{
    circular_buffer<int> cbuf(5); // {0, 0, 0, 0, 0} -> {}

    cbuf.push(1);                // {1, 0, 0, 0, 0} -> {1}
    cbuf.push(2);                // {1, 2, 0, 0, 0} -> {1, 2}
    cbuf.push(3);                // {1, 2, 3, 0, 0} -> {1, 2, 3}

    auto item = cbuf.pop();      // {1, 2, 3, 0, 0} -> {2, 3}
    cbuf.push(4);                // {1, 2, 3, 4, 0} -> {2, 3, 4}
}

```



```

cbuf.push(5);           // {1, 2, 3, 4, 5} -> {2, 3, 4, 5}
cbuf.push(6);           // {6, 2, 3, 4, 5} -> {2, 3, 4, 5, 6}

cbuf.push(7);           // {6, 7, 3, 4, 5} -> {3, 4, 5, 6, 7}
cbuf.push(8);           // {6, 7, 8, 4, 5} -> {4, 5, 6, 7, 8}

item = cbuf.pop();      // {6, 7, 8, 4, 5} -> {5, 6, 7, 8}
item = cbuf.pop();      // {6, 7, 8, 4, 5} -> {6, 7, 8}
item = cbuf.pop();      // {6, 7, 8, 4, 5} -> {7, 8}

item = cbuf.pop();      // {6, 7, 8, 4, 5} -> {8}
item = cbuf.pop();      // {6, 7, 8, 4, 5} -> {}

cbuf.push(9);           // {6, 7, 8, 9, 5} -> {9}
}

```

## 47. Двойной буфер

Здесь описывается типичная ситуация двойной буферизации. Двойная буферизация является наиболее распространенным случаем множественной буферизации – приема, позволяющего читателям видеть полные версии данных и не замечать версии, запись которых еще не завершена. Этот прием широко используется в компьютерной графике для предотвращения эффекта мерцания.

Для поддержки требуемой функциональности класс буфера, который мы должны написать, должен иметь два внутренних буфера: один для временных, записываемых данных, а второй для полных (или подтвержденных) данных. В момент завершения операции записи содержимое временного буфера переписывается в основной буфер. В качестве внутренних буферов представленная ниже реализация использует экземпляры `std::vector`. Когда операция записи завершается, вместо копирования данных из одного буфера в другой мы просто меняем их местами – эта операция выполняется намного быстрее. Доступ к подтвержденным данным осуществляется или посредством функции `read()`, которая копирует содержимое буфера чтения в указанный контейнер, или посредством прямого доступа к элементам (перегруженный `operator[]`). Доступ к буферу чтения синхронизируется с использованием `std::mutex`, чтобы предотвратить конфликты между читающими и пишущими потоками выполнения:

```

template <typename T>
class double_buffer
{
    typedef T          value_type;
    typedef T&         reference;
    typedef T const & const_reference;
    typedef T*         pointer;

```



```

public:
    explicit double_buffer(size_t const size) :
        rdbuf(size), wrbuf(size)
    {}

    size_t size() const noexcept { return rdbuf.size(); }

    void write(T const * const ptr, size_t const size)
    {
        std::unique_lock<std::mutex> lock(mt);
        auto length = std::min(size, wrbuf.size());
        std::copy(ptr, ptr + length, std::begin(wrbuf));
        wrbuf.swap(rdbuf);
    }

    template <class Output>
    void read(Output it) const
    {
        std::unique_lock<std::mutex> lock(mt);
        std::copy(std::cbegin(rdbuf), std::cend(rdbuf), it);
    }

    pointer data() const
    {
        std::unique_lock<std::mutex> lock(mt);
        return rdbuf.data();
    }

    reference operator[](size_t const pos)
    {
        std::unique_lock<std::mutex> lock(mt);
        return rdbuf[pos];
    }

    const_reference operator[](size_t const pos) const
    {
        std::unique_lock<std::mutex> lock(mt);
        return rdbuf[pos];
    }

    void swap(double_buffer other)
    {
        std::swap(rdbuf, other.rdbuf);
        std::swap(wrbuf, other.wrbuf);
    }

private:
    std::vector<T> rdbuf;
    std::vector<T> wrbuf;

```



```
mutable std::mutex mt;
};
```

Далее приводится пример использования класса двойного буфера для чтения и записи из двух конкурирующих потоков выполнения:



```
template <typename T>
void print_buffer(double_buffer<T> const & buf)
{
    buf.read(std::ostream_iterator<T>(std::cout, " "));
    std::cout << std::endl;
}

int main()
{
    double_buffer<int> buf(10);

    std::thread t([&buf]() {
        for (int i = 1; i < 1000; i += 10)
        {
            int data[] = { i, i + 1, i + 2, i + 3, i + 4,
                          i + 5, i + 6, i + 7, i + 8, i + 9 };
            buf.write(data, 10);

            using namespace std::chrono_literals;
            std::this_thread::sleep_for(100ms);
        }
    });

    auto start = std::chrono::system_clock::now();
    do
    {
        print_buffer(buf);

        using namespace std::chrono_literals;
        std::this_thread::sleep_for(150ms);
    } while (std::chrono::duration_cast<std::chrono::seconds>(
        std::chrono::system_clock::now() - start).count() < 12);
    t.join();
}
```



## 48. Самый часто встречающийся элемент в диапазоне

Чтобы найти и вернуть самый часто встречающийся элемент в диапазоне, нужно выполнить следующее:

- подсчитать количество вхождений каждого элемента в `std::map`, роль ключа в котором должен играть сам элемент, а значения – число вхождений;

- определить элемент словаря `std::map` с наибольшим значением вызовом функции `std::max_element()`, которая вернет элемент, то есть пару ключ/значение;
- скопировать все элементы с этим же максимальным значением (счетчиком вхождений) и вернуть окончательный результат.

Ниже показана реализация описанных выше шагов:

```
template <typename T>
std::vector<std::pair<T, size_t>> find_most_frequent(
    std::vector<T> const & range)
{
    std::map<T, size_t> counts;
    for (auto const & e : range) counts[e]++;

    auto maxelem = std::max_element(
        std::cbegin(counts), std::cend(counts),
        [](auto const & e1, auto const & e2) {

            return e1.second < e2.second;
        });

    std::vector<std::pair<T, size_t>> result;

    std::copy_if(
        std::begin(counts), std::end(counts),
        std::back_inserter(result),
        [maxelem](auto const & kvp) {
            return kvp.second == maxelem->second;
        });

    return result;
}
```



Далее приводится пример использования функции `find_most_frequent()`:

```
int main()
{
    auto range = std::vector<int>{1,1,3,5,8,13,3,5,8,8,5};
    auto result = find_most_frequent(range);

    for (auto const & e : result)
    {
        std::cout << e.first << " : " << e.second << std::endl;
    }
}
```

## 49. Текстовая гистограмма

Гистограмма – это графическое представление распределения числовых данных. В фотографии и обработке изображений широко применяются цветковые гистограммы. Текстовая гистограмма, о которой здесь идет речь, представляет частоту встречаемости букв в заданном тексте. Отчасти эта задача напоминает предыдущую, за исключением того, что диапазон элементов представляет буквы и нам требуется определить частоту встречаемости каждой из них. Чтобы решить эту задачу, нужно:

- подсчитать вхождение каждой буквы с помощью словаря, роль ключей в котором будут играть сами буквы, а роль значений – счетчик вхождений;
- при подсчете мы должны игнорировать все символы, не являющиеся буквами, буквы в верхнем и нижнем регистрах должны интерпретироваться как одна и та же буква;
- использовать `std::accumulate()` для подсчета общего количества вхождений всех букв в тексте;
- использовать `std::for_each()` или диапазонный цикл `for` для обхода всех элементов словаря и преобразовать количества вхождений в частоты.

Вот одна из возможных реализаций этой задачи:

```
std::map<char, double> analyze_text(std::string_view text)
{
    std::map<char, double> frequencies;
    for (char ch = 'a'; ch <= 'z'; ch++)
        frequencies[ch] = 0;

    for (auto ch : text)
    {
        if (isalpha(ch))
            frequencies[tolower(ch)]++;
    }

    auto total = std::accumulate(
        std::cbegin(frequencies), std::cend(frequencies),
        0ull,
        [](auto sum, auto const & kvp) {
            return sum + static_cast<unsigned long long>(kvp.second);
        });

    std::for_each(
        std::begin(frequencies), std::end(frequencies),
        [total](auto & kvp) {
            kvp.second = (100.0 * kvp.second) / total;
        });
}
```

```

    });

    return frequencies;
}

```



Следующая программа выводит частоты букв в консоль:

```

int main()
{
    auto result = analyze_text(R"(Lorem ipsum dolor sit amet, consectetur
        adipiscing elit, sed do eiusmod tempor incididunt ut labore et
        dolore magna aliqua.)");

    for (auto const & kvp : result)
    {
        std::cout << kvp.first << " : "
            << std::fixed
            << std::setw(5) << std::setfill(' ')
            << std::setprecision(2) << kvp.second << std::endl;
    }
}

```



## 50. Фильтрация списка телефонных номеров

Эта задача решается относительно просто: мы должны выполнить обход всех телефонных номеров и скопировать в отдельный контейнер (например, `std::vector`) номера, начинающиеся с кода страны. Если, к примеру, задан код страны 44, значит, мы должны скопировать номера, начинающиеся с 44 и +44. Выполнить подобную фильтрацию диапазона можно с помощью функции `std::copy_if()`. Вот как выглядит решение задачи в программном коде:

```

bool starts_with(std::string_view str, std::string_view prefix)
{
    return str.find(prefix) == 0;
}

template <typename InputIt>
std::vector<std::string> filter_numbers(InputIt begin, InputIt end,
                                       std::string const & countryCode)
{
    std::vector<std::string> result;
    std::copy_if(
        begin, end,
        std::back_inserter(result),
        [countryCode](auto const & number) {
            return starts_with(number, countryCode) ||
                starts_with(number, "+" + countryCode);
        });
}

```

```

    });

    return result;
}

std::vector<std::string> filter_numbers(
    std::vector<std::string> const & numbers,
    std::string const & countryCode)
{
    return filter_numbers(std::cbegin(numbers), std::cend(numbers),
                          countryCode);
}

```



А это пример использования функции:

```

int main()
{
    std::vector<std::string> numbers{
        "+40744909080",
        "44 7520 112233",
        "+44 7555 123456",
        "40 7200 123456",
        "7555 123456"
    };

    auto result = filter_numbers(numbers, "44");

    for (auto const & number : result)
    {
        std::cout << number << std::endl;
    }
}

```



## 51. Преобразование списка телефонных номеров

Эта задача имеет определенное сходство с предыдущей. Только вместо выбора телефонных номеров, начинающихся с указанного кода страны, мы должны преобразовать номера так, чтобы они начинались с заданного кода и символа + перед ним. При этом мы должны учесть следующие аспекты:

- если номер начинается с 0, значит, он не содержит кода страны; для преобразования такого номера нужно заменить 0 кодом страны с предшествующим ему знаком +;
- если номер начинается с кода страны, мы должны добавить начальный знак +;
- если номер начинается со знака +, за которым следует код страны, значит, он уже имеет необходимый нам формат и не должен изменяться;

- во всех остальных случаях достаточно просто добавить код страны в начало номера с предшествующим ему знаком +.



Для простоты игнорируем вероятность встречи номера с кодом другой страны. Вы можете реализовать учет этого обстоятельства в качестве самостоятельного упражнения и изменить решение так, чтобы оно обрабатывало телефонные номера с кодами разных стран. Эти номера должны удаляться из списка.

Во всех предыдущих случаях телефонные номера могут содержать пробелы. Согласно требованиям задачи, они должны удаляться. Для этой цели используются функции `std::remove_if()` и `isspace()`.

Далее приводится реализация описанного решения:

```
bool starts_with(std::string_view str, std::string_view prefix)
{
    return str.find(prefix) == 0;
}

void normalize_phone_numbers(std::vector<std::string>& numbers,
                             std::string const & countryCode)
{
    std::transform(
        std::cbegin(numbers), std::cend(numbers),
        std::begin(numbers),
        [countryCode](std::string const & number) {
            std::string result;
            if (number.size() > 0)
            {
                if (number[0] == '0')
                    result = "+" + countryCode +
                        number.substr(1);
                else if (starts_with(number, countryCode))
                    result = "+" + number;
                else if (starts_with(number, "+" + countryCode))
                    result = number;
                else
                    result = "+" + countryCode + number;
            }

            result.erase(
                std::remove_if(std::begin(result), std::end(result),
                    [](const char ch) {return isspace(ch); }),
                std::end(result));

            return result;
        });
}
```

```
});
}
```

Программа ниже нормализует список телефонных номеров согласно требованиям и выводит их в консоль:

```
int main()
{
    std::vector<std::string> numbers{
        "07555 123456",
        "07555123456",
        "+44 7555 123456",
        "44 7555 123456",
        "7555 123456"
    };

    normalize_phone_numbers(numbers, "44");

    for (auto const & number : numbers)
    {
        std::cout << number << std::endl;
    }
}
```



## 52. Генерация всех перестановок символов в строке

В решении этой задачи нам могут помочь некоторые универсальные алгоритмы из стандартной библиотеки. Простейшей из двух требуемых версий является нерекурсивная, по крайней мере при использовании `std::next_permutation()`. Эта функция преобразует входной диапазон (то есть подлежащий сортировке) в следующую перестановку из множества возможных, упорядоченных лексикографически с помощью `operator<` или заданной функции сравнения. Если такая перестановка уже существует, возвращается `true`, иначе диапазон преобразуется в первую перестановку и возвращается `false`. Итак, нерекурсивная реализация на основе `std::next_permutation()` выглядит так:

```
void print_permutations(std::string str)
{
    std::sort(std::begin(str), std::end(str));
    do
    {
        std::cout << str << std::endl;
    } while (std::next_permutation(std::begin(str), std::end(str)));
}
```

Рекурсивная альтернатива немного сложнее. Один из способов реализовать ее – использовать входную и выходную строки; первоначально входная



строка – это исходная строка, для которой нужно найти все перестановки, а выходная строка – пустая. Мы берем по одному символу из входной строки и переносим в выходную. Когда входная строка опустеет, выходная строка будет содержать следующую перестановку. Рекурсивный алгоритм, осуществляющий это, можно описать так:

- если входная строка пустая, вывести выходную строку и выйти;
- иначе выполнить итерации по всем символам во входной строке и для каждого:
  - вызвать метод рекурсивно, удалив первый символ из входной строки и добавив его в конец выходной;
  - прокрутить входную строку так, чтобы первый символ стал последним, второй – первым и т. д.

На рис. 6.2 приводится диаграмма с визуальным описанием алгоритма.

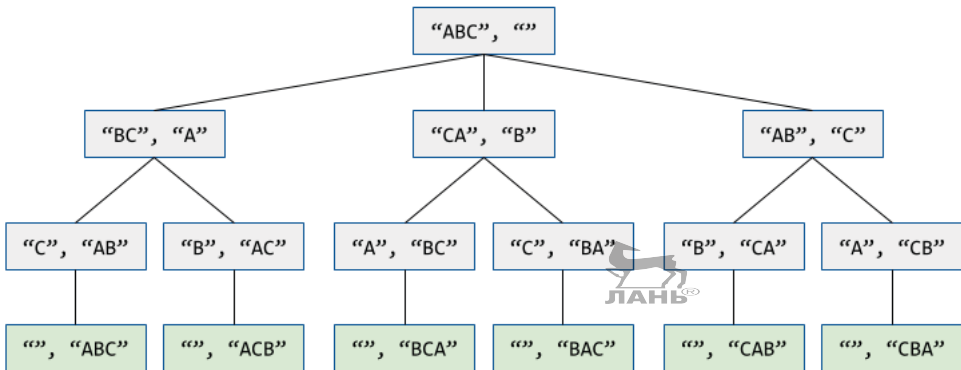


Рис. 6.2. Рекурсивный алгоритм поиска перестановок

Для прокрутки входной строки можно использовать стандартную функцию `std::rotate()`, которая выполняет прокрутку диапазона влево. Вот как выглядит реализация рекурсивного алгоритма:

```

void next_permutation(std::string str, std::string perm)
{
    if (str.empty()) std::cout << perm << std::endl;
    else
    {
        for (size_t i = 0; i < str.size(); ++i)
        {
            next_permutation(str.substr(1), perm + str[0]);

            std::rotate(std::begin(str), std::begin(str) + 1, std::end(str));
        }
    }
}
  
```

```

    }
}

void print_permutations_recursive(std::string str)
{
    next_permutation(str, "");
}

```

Далее демонстрируется пример использования обеих версий:

```

int main()
{
    std::cout << "non-recursive version" << std::endl;
    print_permutations("main");

    std::cout << "recursive version" << std::endl;
    print_permutations_recursive("main");
}

```

## 53. Средний рейтинг фильмов

По условиям задачи мы должны найти усеченное среднее рейтингов фильмов. Эта статистическая оценка среднего значения вычисляется после выбора из выборки наибольших и наименьших значений. Обычно это осуществляется удалением определенного процента измерений с обоих концов. В данной задаче мы должны удалить по 5% измерений с обоих концов.

Функция, вычисляющая усеченное среднее для заданного диапазона, должна:

- отсортировать диапазон, расположив элементы по порядку (по возрастанию или по убыванию);
- удалить заданный процент элементов с обоих концов;
- вычислить сумму оставшихся элементов;
- найти среднее, разделив сумму на количество оставшихся элементов.

Функция `truncated_mean()`, показанная ниже, реализует описанный алгоритм:

```

double truncated_mean(std::vector<int> values, double const percentage)
{
    std::sort(std::begin(values), std::end(values));
    auto remove_count = static_cast<size_t>(
        values.size() * percentage + 0.5);

    values.erase(std::begin(values), std::begin(values) + remove_count);
    values.erase(std::end(values) - remove_count, std::end(values));

    auto total = std::accumulate(

```

```

    std::cbegin(values), std::cend(values),
    0ull,
    [](auto const sum, auto const e) {
        return sum + e; });
return static_cast<double>(total) / values.size();
}

```

Следующая программа использует эту функцию для вычисления и вывода среднего рейтинга:

```

struct movie
{
    int          id;
    std::string  title;
    std::vector<int> ratings;
};

void print_movie_ratings(std::vector<movie> const & movies)
{
    for (auto const & m : movies)
    {
        std::cout << m.title << " : "
                    << std::fixed << std::setprecision(1)
                    << truncated_mean(m.ratings, 0.05) << std::endl;
    }
}

int main()
{
    std::vector<movie> movies
    {
        { 101, "The Matrix", {10, 9, 10, 9, 9, 8, 7, 10, 5, 9, 9, 8} },
        { 102, "Gladiator", {10, 5, 7, 8, 9, 8, 9, 10, 10, 5, 9, 8, 10} },
        { 103, "Interstellar", {10, 10, 10, 9, 3, 8, 8, 9, 6, 4, 7, 10} }
    };

    print_movie_ratings(movies);
}

```

## 54. Алгоритм объединения в пары

Функция `pairwise`, решающая эту задачу, должна составить пары из смежных элементов входного диапазона и вернуть диапазон элементов типа `std::pair`. В следующем листинге представлены две реализации:

- первая шаблонная функция принимает итераторы, `begin` и `end`, определяющие входной диапазон, а возвращаемый итератор `result` определяет позицию в выходном диапазоне, куда должен быть вставлен результат;

- перегруженная версия принимает `std::vector<T>` и возвращает `std::vector<std::pair<T, T>>`; она просто вызывает первую версию функции:

```

template <typename Input, typename Output>
void pairwise(Input begin, Input end, Output result)
{
    auto it = begin;
    while (it != end)
    {
        auto v1 = *it++; if (it == end) break;
        auto v2 = *it++;
        result++ = std::make_pair(v1, v2);
    }
}

template <typename T>
std::vector<std::pair<T, T>> pairwise(std::vector<T> const & range)
{
    std::vector<std::pair<T, T>> result;
    pairwise(std::begin(range), std::end(range),
             std::back_inserter(result));
    return result;
}

```

Следующая программа объединяет в пары элементы вектора целых чисел и выводит пары в консоль:

```

int main()
{
    std::vector<int> v{ 1, 1, 3, 5, 8, 13, 21 };
    auto result = pairwise(v);

    for (auto const & p : result)
    {
        std::cout << '{' << p.first << ', ' << p.second << '}' << std::endl;
    }
}

```

## 55. Алгоритм «сшивания»

Эта задача очень похожа на предыдущую, только вместо одного диапазона на вход подаются два. Результатом также является диапазон пар `std::pair`. Однако входные диапазоны могут содержать элементы разных типов. И снова представленная здесь реализация содержит две перегруженные версии:

- основная функция принимает итераторы, определяющие начало и конец двух диапазонов, а выходной итератор – позицию в выходном диапазоне, куда должен быть вставлен результат;

- вторая функция принимает два вектора `std::vector`, один из которых хранит элементы типа `T`, а второй – элементы типа `U` и возвращает `std::vector<std::pair<T, U>>`; она просто вызывает первую версию функции:

```
template <typename Input1, typename Input2, typename Output>
void zip(Input1 begin1, Input1 end1,
        Input2 begin2, Input1 end2,
        Output result)
{
    auto it1 = begin1;
    auto it2 = begin2;
    while (it1 != end1 && it2 != end2)
    {
        result++ = std::make_pair(*it1++, *it2++);
    }
}
```



```
template <typename T, typename U>
std::vector<std::pair<T, U>> zip(
    std::vector<T> const & range1,
    std::vector<U> const & range2)
{
    std::vector<std::pair<T, U>> result;

    zip(std::begin(range1), std::end(range1),
        std::begin(range2), std::end(range2),
        std::back_inserter(result));

    return result;
}
```



Программа в следующем листинге демонстрирует «сшивание» двух векторов целых чисел и выводит результат в консоль:

```
int main()
{
    std::vector<int> v1{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    std::vector<int> v2{ 1, 1, 3, 5, 8, 13, 21 };

    auto result = zip(v1, v2);
    for (auto const & p : result)
    {
        std::cout << '{' << p.first << ', ' << p.second << '}' << std::endl;
    }
}
```

## 56. Алгоритм выбора



Функция `select()`, которую нам предстоит реализовать, должна принимать вектор `std::vector<T>` и функцию типа `F` и возвращать `std::vector<R>`, где `R` – это результат применения `F` к `T`. Для вывода типа значения, возвращаемого выражением вызова на этапе компиляции, можно использовать `std::result_of()`. Внутренне функция `select()` должна использовать `std::transform()` для обхода элементов входного вектора, применять функцию к каждому элементу и вставлять результат в выходной вектор.

Реализация функции показана в следующем листинге:

```
template <
    typename T, typename A, typename F,
    typename R = typename std::decay<typename std::result_of<
        typename std::decay<F>::type&(
            typename std::vector<T, A>::const_reference)>::type>::type>
std::vector<R> select(std::vector<T, A> const & c, F&& f)
{
    std::vector<R> v;
    std::transform(std::cbegin(c), std::cend(c),
        std::back_inserter(v),
        std::forward<F>(f));

    return v;
}
```

Следующая программа демонстрирует порядок использования этой функции:



```
int main()
{
    std::vector<book> books{
        {101, "The C++ Programming Language", "Bjarne Stroustrup"},
        {203, "Effective Modern C++", "Scott Meyers"},
        {404, "The Modern C++ Programming Cookbook", "Marius Bancila"}};

    auto titles = select(books, [](book const & b) {return b.title; });

    for (auto const & title : titles)
    {
        std::cout << title << std::endl;
    }
}
```

## 57. Алгоритм сортировки

**Алгоритм быстрой сортировки** предназначен для сортировки элементов массива, для которых можно определить порядковое значение. При надлежа-

шей реализации он действует существенно быстрее, чем сортировка слиянием или пирамидальная сортировка.

В худшем случае (когда диапазон уже отсортирован) этот алгоритм выполняет  $O(n^2)$  сравнений, средняя сложность составляет всего  $O(n \times \log(n))$ . Алгоритм быстрой сортировки действует по принципу «разделяй и властвуй»; он рекурсивно делит большой диапазон на меньшие и сортирует их. Существует несколько схем деления. В реализации, представленной ниже, мы используем оригинальную схему, разработанную Тони Хоаром (Tony Hoare). Вот как выглядит этот алгоритм на псевдокоде:

```
algorithm quicksort(A, lo, hi) is
  if lo < hi then
    p := partition(A, lo, hi)
    quicksort(A, lo, p)
    quicksort(A, p + 1, hi)
```

```
algorithm partition(A, lo, hi) is
  pivot := A[lo]
  i := lo - 1
  j := hi + 1
  loop forever
    do
      i := i + 1
      while A[i] < pivot

    do
      j := j - 1
      while A[j] > pivot

  if i >= j then
    return j

  swap A[i] with A[j]
```



Универсальная реализация алгоритма должна использовать итераторы вместо массивов и индексов. Главное требование реализации ниже состоит в том, что итераторы должны поддерживать произвольный доступ (то есть перемещаться к любому элементу за постоянное время):

```
template <class RandomIt>
RandomIt partition(RandomIt first, RandomIt last)
{
  auto pivot = *first;
  auto i = first + 1;
  auto j = last - 1;
  while (i <= j)
  {
```

```

    while (i <= j && *i <= pivot) i++;
    while (i <= j && *j > pivot) j--;
    if (i < j) std::iter_swap(i, j);
}

std::iter_swap(i - 1, first);

return i - 1;
}

template <class RandomIt>
void quicksort(RandomIt first, RandomIt last)
{
    if (first < last)
    {
        auto p = partition(first, last);
        quicksort(first, p);
        quicksort(p + 1, last);
    }
}

```



Функцию `quicksort()`, как показано ниже, можно использовать для сортировки контейнеров разных типов:



```

int main()
{
    std::vector<int> v{ 1,5,3,8,6,2,9,7,4 };
    quicksort(std::begin(v), std::end(v));

    std::array<int, 9> a{ 1,2,3,4,5,6,7,8,9 };
    quicksort(std::begin(a), std::end(a));

    int a[]{ 9,8,7,6,5,4,3,2,1 };
    quicksort(std::begin(a), std::end(a));
}

```

По условиям задачи алгоритм сортировки должен позволять передавать ему пользовательскую функцию сравнения. Для этого достаточно изменить только функцию деления диапазона и для сравнения текущего элемента с опорной точкой вместо операторов `<` и `>` использовать указанную функцию сравнения:

```

template <class RandomIt, class Compare>
RandomIt partitionc(RandomIt first, RandomIt last, Compare comp)
{
    auto pivot = *first;
    auto i = first + 1;
    auto j = last - 1;
    while (i <= j)

```



```

    {
        while (i <= j && comp(*i, pivot)) i++;
        while (i <= j && !comp(*j, pivot)) j--;
        if (i < j) std::iter_swap(i, j);
    }

    std::iter_swap(i - 1, first);

    return i - 1;
}

template <class RandomIt, class Compare>
void quicksort(RandomIt first, RandomIt last, Compare comp)
{
    if (first < last)
    {
        auto p = partitionc(first, last, comp);
        quicksort(first, p, comp);
        quicksort(p + 1, last, comp);
    }
}

```



С помощью этой перегруженной версии можно отсортировать диапазон в порядке убывания, как показано в следующем примере:

```

int main()
{
    std::vector<int> v{ 1,5,3,8,6,2,9,7,4 };
    quicksort(std::begin(v), std::end(v), std::greater<>());
}

```

Также можно реализовать итеративную версию алгоритма быстрой сортировки. Итеративная версия будет иметь ту же производительность, что и рекурсивная ( $O(n \times \log(n))$  в большинстве случаев и падать до  $O(n^2)$  в худшем случае, когда диапазон уже отсортирован). Преобразование рекурсивной версии в итеративную осуществляется просто: достаточно использовать структуру стека для эмуляции рекурсивных вызовов и сохранять границы разделов. Ниже показана итеративная реализация, использующая `operator<` для сравнения элементов:

```

template <class RandomIt>
void quicksorti(RandomIt first, RandomIt last)
{
    std::stack<std::pair<RandomIt, RandomIt>> st;
    st.push(std::make_pair(first, last));
    while (!st.empty())
    {

```

```

auto iters = st.top();
st.pop();

if (iters.second - iters.first < 2) continue;

auto p = partition(iters.first, iters.second);

st.push(std::make_pair(iters.first, p));
st.push(std::make_pair(p+1, iters.second));
}
}

```



Итеративная версия используется точно так же, как рекурсивная:

```

int main()
{
    std::vector<int> v{ 1,5,3,8,6,2,9,7,4 };
    quicksorti(std::begin(v), std::end(v));
}

```

## 58. Кратчайший путь между узлами

Для решения этой задачи используем алгоритм Дейкстры (Dijkstra) поиска кратчайшего пути в графе. Оригинальный алгоритм отыскивает кратчайший путь между двумя заданными узлами, но по условию задачи нам требуется найти кратчайшие пути между заданным узлом и всеми остальными узлами в графе, а это уже другая версия алгоритма.

Для эффективной реализации алгоритма можно использовать приоритетную очередь. Вот как выглядит реализация алгоритма (см. [https://ru.wikipedia.org/wiki/Алгоритм\\_Дейкстры](https://ru.wikipedia.org/wiki/Алгоритм_Дейкстры)) на псевдокоде:

```

function Dijkstra(Graph, source):
    dist[source] ← 0 // Инициализация

    create vertex set Q
    for each vertex v in Graph:
        if v ≠ source
            dist[v] ← INFINITY // Неизвестное расстояние от source до v
            prev[v] ← UNDEFINED // Предшествует узлу v

    Q.add_with_priority(v, dist[v])

    while Q is not empty: // Главный цикл
        u ← Q.extract_min() // Извлечь и вернуть лучшую вершину
        for each neighbor v of u: // Пока в Q не останется только v
            alt ← dist[u] + length(u, v)
            if alt < dist[v]

```

```

    dist[v] ← alt
    prev[v] ← u
    Q.decrease_priority(v, alt)

return dist[], prev[]

```

Для представления ориентированных и неориентированных графов можно использовать следующую структуру данных. Класс поддерживает добавление новых вершин и ребер и может вернуть список вершин и список соседей заданной вершины (то есть узлов и расстояний до них):

```

template <typename Vertex = int, typename Weight = double>
class graph
{
public:
    typedef Vertex          vertex_type;
    typedef Weight         weight_type;
    typedef std::pair<Vertex, Weight> neighbor_type;
    typedef std::vector<neighbor_type> neighbor_list_type;
public:
    void add_edge(Vertex const source, Vertex const target,
                  Weight const weight, bool const bidirectional = true)
    {
        adjacency_list[source].push_back(std::make_pair(target, weight));
        adjacency_list[target].push_back(std::make_pair(source, weight));
    }

    size_t vertex_count() const { return adjacency_list.size(); }

    std::vector<Vertex> verteces() const
    {
        std::vector<Vertex> keys;
        for (auto const & kvp : adjacency_list)
            keys.push_back(kvp.first);
        return keys;
    }

    neighbor_list_type const & neighbors(Vertex const & v) const
    {
        auto pos = adjacency_list.find(v);
        if (pos == adjacency_list.end())
            throw std::runtime_error("vertex not found");
        return pos->second;
    }

    constexpr static Weight Infinity =
        std::numeric_limits<Weight>::infinity();

```

```
private:
    std::map<vertex_type, neighbor_list_type> adjacency_list;
};
```

Далее показана одна из возможных реализаций алгоритма поиска кратчайшего пути, описанного выше в псевдокоде. Здесь вместо приоритетной очереди используется `std::set` (то есть самобалансирующееся дерево двоичного поиска). `std::set` имеет ту же сложность  $O(\log(n))$  для добавления и удаления верхнего элемента, что и двоичная куча (используемая в реализации приоритетной очереди). С другой стороны, `std::set` позволяет находить и удалять другие элементы за время  $O(\log(n))$ , что очень хорошо для шага уменьшения ключа путем удаления и повторной вставки:

```
template <typename Vertex, typename Weight>
void shortest_path(
    graph<Vertex, Weight> const & g,
    Vertex const source,
    std::map<Vertex, Weight>& min_distance,
    std::map<Vertex, Vertex>& previous)
{
    auto const n = g.vertex_count();
    auto const verteces = g.verteces();

    min_distance.clear();
    for (auto const & v : verteces)
        min_distance[v] = graph<Vertex, Weight>::Infinity;
    min_distance[source] = 0;

    previous.clear();

    std::set<std::pair<Weight, Vertex> > vertex_queue;
    vertex_queue.insert(std::make_pair(min_distance[source], source));

    while (!vertex_queue.empty())
    {
        auto dist = vertex_queue.begin()->first;
        auto u = vertex_queue.begin()->second;

        vertex_queue.erase(std::begin(vertex_queue));

        auto const & neighbors = g.neighbors(u);
        for (auto const & neighbor : neighbors)
        {
            auto v = neighbor.first;
            auto w = neighbor.second;
            auto dist_via_u = dist + w;
            if (dist_via_u < min_distance[v])
```



```

    {
        vertex_queue.erase(std::make_pair(min_distance[v], v));
        min_distance[v] = dist_via_u;
        previous[v] = u;
        vertex_queue.insert(std::make_pair(min_distance[v], v));
    }
}
}
}

```



Следующая вспомогательная функция выводит результаты в требуемом формате:

```

template <typename Vertex>
void build_path(
    std::map<Vertex, Vertex> const & prev, Vertex const v,
    std::vector<Vertex> & result)
{
    result.push_back(v);

    auto pos = prev.find(v);
    if (pos == std::end(prev)) return;

    build_path(prev, pos->second, result);
}

template <typename Vertex>
std::vector<Vertex> build_path(std::map<Vertex, Vertex> const & prev,
                             Vertex const v)
{
    std::vector<Vertex> result;
    build_path(prev, v, result);
    std::reverse(std::begin(result), std::end(result));
    return result;
}

template <typename Vertex>
void print_path(std::vector<Vertex> const & path)
{
    for (size_t i = 0; i < path.size(); ++i)
    {
        std::cout << path[i];
        if (i < path.size() - 1) std::cout << " -> ";
    }
}

```



А вот как выглядит программа, решающая поставленную задачу:

```

int main()
{
    graph<char, double> g;
    g.add_edge('A', 'B', 7);
    g.add_edge('A', 'C', 9);
    g.add_edge('A', 'F', 14);
    g.add_edge('B', 'C', 10);
    g.add_edge('B', 'D', 15);
    g.add_edge('C', 'D', 11);
    g.add_edge('C', 'F', 2);
    g.add_edge('D', 'E', 6);
    g.add_edge('E', 'F', 9);

    char source = 'A';
    std::map<char, double> min_distance;
    std::map<char, char> previous;
    shortest_path(g, source, min_distance, previous);

    for (auto const & kvp : min_distance)
    {
        std::cout << source << " -> " << kvp.first << " : "
            << kvp.second << '\t';

        print_path(build_path(previous, kvp.first));

        std::cout << std::endl;
    }
}

```



## 59. Программа Weasel

Программа Weasel – это мысленный эксперимент, проведенный Ричардом Докинзом (Richard Dawkins), цель которого состояла в том, чтобы продемонстрировать, как накопление небольших улучшений (мутаций в ходе естественного отбора, улучшающих индивидуальные качества) может давать быстрые результаты, в противоположность бытующему мнению, что эволюция происходит большими скачками. Алгоритм программы Weasel описан в Википедии ([https://en.wikipedia.org/wiki/Weasel\\_program](https://en.wikipedia.org/wiki/Weasel_program)):

1. Дано: случайная строка с 28 символами.
2. Создать 100 копий этой строки, заменяя каждый символ другим случайным символом с 5%-ной вероятностью.
3. Сравнить каждую новую версию строки с целевой строкой **METHINKS IT IS LIKE A WEASEL**<sup>3</sup> и определить оценку (количество букв в строке, стоящих на своем месте).

<sup>3</sup> «Мне кажется, оно похоже на горностаю» – цитата из произведения «Гамлет», У. Шекспир. – *Прим. перев.*

4. Если любая из новых строк получила наивысшую оценку (28), остановить работу.
5. Иначе взять строку с наивысшей оценкой и вернуться к шагу 2.

Одна из возможных реализаций показана ниже. Функция `make_random()` создает случайную строку той же длины, что и целевая; функция `fitness()` вычисляет оценку каждой новой строки (то есть степень сходства с целевой строкой); функция `mutate()` производит новую строку из заданной и с заданной вероятностью изменения каждого символа:

```
class weasel
{
    std::string target;
    std::uniform_int_distribution<> chardist;
    std::uniform_real_distribution<> ratedist;
    std::mt19937 mt;
    std::string const allowed_chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ ";
public:
    weasel(std::string_view t) :
        target(t), chardist(0, 26), ratedist(0, 100)
    {
        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));
        mt.seed(seq);
    }

    void run(int const copies)
    {
        auto parent = make_random();
        int step = 1;
        std::cout << std::left << std::setw(5) << std::setfill(' ')
            << step << parent << std::endl;

        do
        {
            std::vector<std::string> children;
            std::generate_n(std::back_inserter(children), copies,
                [parent, this]() {return mutate(parent, 5); });

            parent = *std::max_element(
                std::begin(children), std::end(children),
                [this](std::string_view c1, std::string_view c2) {
                    return fitness(c1) < fitness(c2); });

            std::cout << std::setw(5) << std::setfill(' ') << step
```



```

        << parent << std::endl;
        step++;
    } while (parent != target);
}
private:
    weasel() = delete;

    double fitness(std::string_view candidate) const
    {
        int score = 0;
        for (size_t i = 0; i < candidate.size(); ++i)
        {
            if (candidate[i] == target[i])
                score++;
        }
        return score;
    }

    std::string mutate(std::string_view parent, double const rate)
    {
        std::stringstream sstr;
        for (auto const c : parent)
        {
            auto nc = ratedist(mt) > rate ? c : allowed_chars[chardist(mt)];
            sstr << nc;
        }
        return sstr.str();
    }

    std::string make_random()
    {
        std::stringstream sstr;
        for (size_t i = 0; i < target.size(); ++i)
        {
            sstr << allowed_chars[chardist(mt)];
        }
        return sstr.str();
    }
};

```

Далее приводится пример использования этого класса:

```

int main()
{
    weasel w("METHINKS IT IS LIKE A WEASEL");
    w.run(100);
}

```



## 60. Игра «Жизнь»

Класс `universe`, представленный ниже, реализует описанную игру. Вот наиболее интересные функции в нем:

- `initialize()` генерирует начальное состояние игрового поля; код в примерах, сопровождающих книгу, содержит больше вариантов, но здесь показаны только два: `random`, генерирует случайное игровое поле, и `ten_cell_row`, создает линию из 10 ячеек в середине сетки;
- `reset()` очищает все ячейки (делает их мертвыми);
- `count_neighbors()` возвращает количество живых соседей; использует вспомогательную шаблонную функцию `count_alive()` с переменным числом аргументов, вместо которой можно было бы применять выражение свертки, но такие выражения пока не поддерживаются в Visual C++, поэтому я не использовал их здесь;
- `next_generation()` вычисляет состояние игры на следующем шаге, согласно правилам;
- `display()` выводит игровое поле в консоль; для очистки консоли использует вызов `system`, но вы можете попробовать применять другие средства, например программный интерфейс конкретной операционной системы;
- `run()` инициализирует игровое поле и создает новые поколения с интервалом, указанным пользователем, выполняя указанное пользователем число итераций или до бесконечности (если число итераций задано равным 0).

```
class universe
{
private:
    universe() = delete;
public:
    enum class seed
    {
        random, ten_cell_row
    };
public:
    universe(size_t const width, size_t const height):
        rows(height), columns(width), grid(width * height), dist(0, 4)
    {
        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));
        mt.seed(seq);
    }
};
```



```

}

void run(seed const s, int const generations,
         std::chrono::milliseconds const ms =
         std::chrono::milliseconds(100))
{
    reset();
    initialize(s);
    display();

    int i = 0;
    do
    {
        next_generation();
        display();

        using namespace std::chrono_literals;
        std::this_thread::sleep_for(ms);
    } while (i++ < generations || generations == 0);
}
private:
void next_generation()
{
    std::vector<unsigned char> newgrid(grid.size());

    for (size_t r = 0; r < rows; ++r)
    {
        for (size_t c = 0; c < columns; ++c)
        {
            auto count = count_neighbors(r, c);

            if (cell(c, r) == alive)
            {
                newgrid[r * columns + c] =
                    (count == 2 || count == 3) ? alive : dead;
            }
            else
            {
                newgrid[r * columns + c] = (count == 3) ? alive : dead;
            }
        }
    }

    grid.swap(newgrid);
}

void reset_display()
{
#ifdef WIN32

```



```

    system("cls");
#endif
}

void display()
{
    reset_display();

    for (size_t r = 0; r < rows; ++r)
    {
        for (size_t c = 0; c < columns; ++c)
        {
            std::cout << (cell(c, r) ? '*' : ' ');
        }
        std::cout << std::endl;
    }
}

void initialize(seed const s)
{
    if (s == seed::ten_cell_row)
    {
        for (size_t c = columns / 2 - 5; c < columns / 2 + 5; c++)
            cell(c, rows / 2) = alive;
    }
    else
    {
        for (size_t r = 0; r < rows; ++r)
        {
            for (size_t c = 0; c < columns; ++c)
            {
                cell(c, r) = dist(mt) == 0 ? alive : dead;
            }
        }
    }
}

void reset()
{
    for (size_t r = 0; r < rows; ++r)
    {
        for (size_t c = 0; c < columns; ++c)
        {
            cell(c, r) = dead;
        }
    }
}

int count_alive() { return 0; }

template<typename T1, typename... T>

```



```

auto count_alive(T1 s, T... ts) { return s + count_alive(ts...); }

int count_neighbors(size_t const row, size_t const col)
{
    if (row == 0 && col == 0)
        return count_alive(cell(1, 0), cell(1,1), cell(0, 1));
    if (row == 0 && col == columns - 1)
        return count_alive(cell(columns - 2, 0), cell(columns - 2, 1),
            cell(columns - 1, 1));

    if (row == rows - 1 && col == 0)
        return count_alive(cell(0, rows - 2), cell(1, rows - 2),
            cell(1, rows - 1));

    if (row == rows - 1 && col == columns - 1)
        return count_alive(cell(columns - 1, rows - 2),
            cell(columns - 2, rows - 2),
            cell(columns - 2, rows - 1));

    if (row == 0 && col > 0 && col < columns - 1)
        return count_alive(cell(col - 1, 0), cell(col - 1, 1),
            cell(col, 1), cell(col + 1, 1),
            cell(col + 1, 0));

    if (row == rows - 1 && col > 0 && col < columns - 1)
        return count_alive(cell(col - 1, row), cell(col - 1, row - 1),
            cell(col, row - 1), cell(col + 1, row - 1),
            cell(col + 1, row));

    if (col == 0 && row > 0 && row < rows - 1)
        return count_alive(cell(0, row - 1), cell(1, row - 1),
            cell(1, row), cell(1, row + 1),
            cell(0, row + 1));

    if (col == columns - 1 && row > 0 && row < rows - 1)
        return count_alive(cell(col, row - 1), cell(col - 1, row - 1),
            cell(col - 1, row), cell(col - 1, row + 1),
            cell(col, row + 1));

    return count_alive(cell(col - 1, row - 1), cell(col, row - 1),
        cell(col + 1, row - 1), cell(col + 1, row),
        cell(col + 1, row + 1), cell(col, row + 1),
        cell(col - 1, row + 1), cell(col - 1, row));
}

unsigned char& cell(size_t const col, size_t const row)
{
    return grid[row * columns + col];
}

```

```
private:
    size_t rows;
    size_t columns;

    std::vector<unsigned char> grid;
    const unsigned char alive = 1;
    const unsigned char dead = 0;

    std::uniform_int_distribution<> dist;
    std::mt19937 mt;
};
```

Вот как можно запустить игру, чтобы выполнить 100, начиная со случайного состояния:

```
int main()
{
    using namespace std::chrono_literals;
    universe u(50, 20);
    u.run(universe::seed::random, 100, 100ms);
}
```

На рис. 6.3 изображен пример вывода программы (скриншот представляет единственную итерацию в игре «Жизнь»).

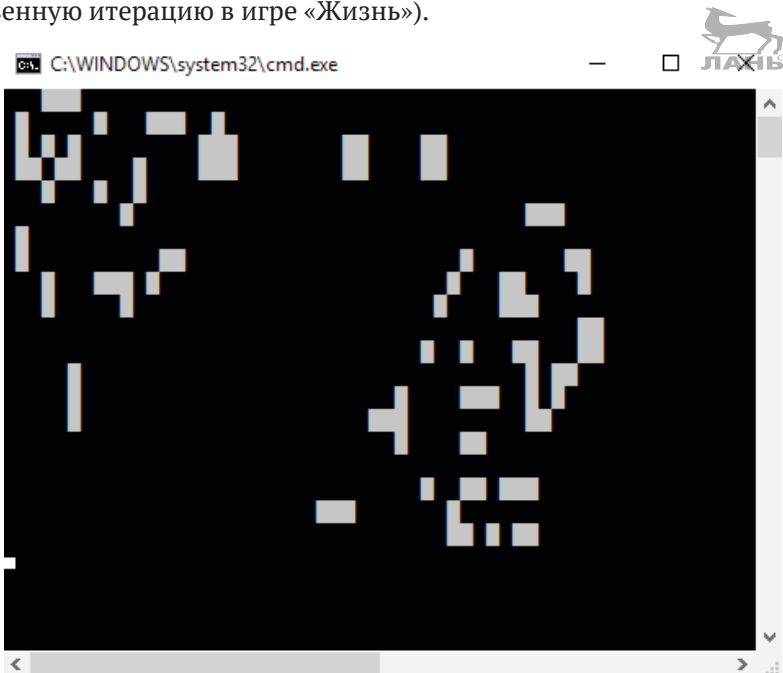


Рис. 6.3. Игра «Жизнь»

# Глава 7

## Конкуренция

### Задачи

#### 61. Алгоритм параллельного преобразования

Напишите универсальный алгоритм, параллельно применяющий заданную унарную функцию к элементам в диапазоне. Унарная функция не должна нарушать действительность итераторов преобразуемого диапазона или изменять его элементы. Уровень параллелизма, то есть количество потоков выполнения и способ их запуска, считать деталью реализации.

#### 62. Параллельные алгоритмы поиска максимального и минимального значений с использованием потоков

Реализуйте обобщенные параллельные алгоритмы поиска максимального и минимального значений в заданном диапазоне. Параллельный поиск должен осуществляться с помощью потоков выполнения, но их количество можно считать деталью реализации.

#### 63. Параллельные алгоритмы поиска максимального и минимального значений с использованием асинхронных функций

Реализуйте обобщенные параллельные алгоритмы поиска максимального и минимального значений в заданном диапазоне. Параллельный поиск должен осуществляться с помощью асинхронных функций, при этом количество одновременно действующих асинхронных вызовов можно считать деталью реализации.

#### 64. Параллельный алгоритм сортировки

Напишите параллельную версию алгоритма сортировки, как определено в задаче 57 «Алгоритм сортировки» в главе 6 «Алгоритмы и структуры данных», который принимает пару итераторов с произвольным доступом, определяющих нижнюю и верхнюю границы диапазоны, и сортирует элементы в этом

диапазоне с помощью алгоритма быстрой сортировки. Для сравнения элементов в диапазоне алгоритм должен использовать операторы сравнения. Уровень параллелизма, то есть количество потоков выполнения и способ их запуска, считать деталью реализации.

## 65. Потокобезопасное журналирование в консоль

Напишите класс, позволяющий компонентам, действующим в разных потоках выполнения, безопасно выводить сообщения в консоль, синхронизируя доступ к потоку стандартного вывода, чтобы гарантировать целостность отображаемых сообщений. Этот класс должен иметь метод `log()` со строковым аргументом, в котором передается сообщение для вывода.

## 66. Система обслуживания клиентов

Напишите программу, имитирующую обслуживание клиентов в офисе. В офисе имеется три стойки для обслуживания сразу трех клиентов. Клиенты могут входить в офис в любой момент. Они получают талон с номером очереди из автомата электронной очереди и ожидают вызова к одной из стоек для обслуживания. Клиенты должны обслуживаться в порядке их входа в офис, точнее, в порядке получения талонов. Каждый раз, когда завершается обслуживание очередного клиента, к освободившейся стойке вызывается следующий клиент из очереди. Программа должна прекращать выдавать талоны после определенного количества и завершать работу после обслуживания всех клиентов.

# Решения

## 61. Алгоритм параллельного преобразования

В стандартной библиотеке имеется обобщенная функция `std::transform()`, которая применяет заданную функцию к диапазону элементов и сохраняет результаты в другом (или в том же) диапазоне. По условиям задачи мы должны реализовать параллельную версию функции `std::transform()`. Она должна принимать пару итераторов, определяющих начало и конец диапазона. Поскольку унарная функция одинаково применяется ко всем элементам диапазона, мы легко сможем распараллелить эту операцию. Используем для этого потоки выполнения. Поскольку в условиях не указано, сколько должно быть одновременно выполняющихся потоков, мы можем воспользоваться `std::thread::hardware_concurrency()`. Эта функция возвращает рекомендованное реализацией число потоков.

Параллельная версия алгоритма даст выигрыш в производительности в сравнении с последовательной версией, только если размер диапазона превышает определенный порог, который может меняться в зависимости от флагов компиляции, платформы или оборудования. В следующей реализации исполь-

зается искусственный порог в 10 000 элементов. Вы можете поэкспериментировать с разными значениями порога и размерами диапазона и посмотреть, как это влияет на время выполнения.

Следующая функция, `pttransform()`, реализует требуемый параллельный алгоритм. Она просто вызывает `std::transform()`, если размер диапазона не превышает установленного порога. Иначе разбивает диапазон на несколько равных частей, по одной для каждого потока, и вызывает `std::transform()` в каждом потоке, передавая соответствующий поддиапазон. В этом случае функция блокирует вызывающий поток до завершения всех рабочих потоков:

```
template <typename RandomAccessIterator, typename F>
void pttransform(RandomAccessIterator begin, RandomAccessIterator end,
                F&& f)
{
    auto size = std::distance(begin, end);
    if (size <= 10000)
    {
        std::transform(begin, end, begin, std::forward<F>(f));
    }
    else
    {
        std::vector<std::thread> threads;
        int thread_count = std::thread::hardware_concurrency();
        auto first = begin;
        auto last = first;
        size /= thread_count;
        for (int i = 0; i < thread_count; ++i)
        {
            first = last;
            if (i == thread_count - 1) last = end;
            else std::advance(last, size);

            threads.emplace_back([first, last, &f]() {
                std::transform(first, last, first, std::forward<F>(f));
            });
        }

        for (auto & t : threads) t.join();
    }
}
```



Функция `palter()`, показанная ниже, – это вспомогательная функция, которая применяет `pttransform()` к `std::vector` и возвращает другой `std::vector` с результатом:

```
template <typename T, typename F>
std::vector<T> palter(std::vector<T> data, F&& f)
```



```
{
    ptransform(std::begin(data), std::end(data),
              std::forward<F>(f));
    return data;
}
```

Эту функцию можно использовать, как показано ниже (законченный пример можно найти в загружаемом коде):



```
int main()
{
    std::vector<int> data(1000000);
    // инициализация данных
    auto result = palter(data, [](int const e) {return e * e; });
}
```



В C++17 имеется группа стандартных обобщенных алгоритмов, включая `std::transform()`, для которых реализованы перегруженные параллельные версии, действующие согласно установленным правилам выполнения.



## 62. Параллельные алгоритмы поиска максимального и минимального значений с использованием потоков

Эта задача и ее решение имеет много общего с предыдущей. Отличие состоит лишь в том, что конкурентно выполняемая функция должна возвращать минимальное или максимальное значение в поддиапазоне. Шаблонная функция `pprocess()`, показанная ниже, – это высокоуровневая функция, реализующая обобщенное решение:

- она принимает итераторы начала и конца диапазона и объект функции для обработки диапазона, которую мы назовем `f`;
- если размер диапазона меньше заданного порога, по умолчанию равного 10 000 элементов, она просто выполняет объект функции `f`;
- иначе разбивает диапазон на несколько поддиапазонов равного размера, по одному для каждого потока выполнения; каждый поток выполняет функцию `f` для указанного поддиапазона;
- результаты параллельных вызовов `f` собираются в вектор `std::vector`, и когда все потоки завершатся, вновь вызывается функция `f` для выбора окончательного значения из промежуточных результатов:

```
template <typename Iterator, typename F>
auto pprocess(Iterator begin, Iterator end, F&& f)
```

```

{
    auto size = std::distance(begin, end);
    if (size <= 10000)
    {
        return std::forward<F>(f)(begin, end);
    }
    else
    {
        int thread_count = std::thread::hardware_concurrency();
        std::vector<std::thread> threads;
        std::vector<typename std::
            iterator_traits<Iterator>::value_type>
            mins(thread_count);

        auto first = begin;
        auto last = first;
        size /= thread_count;
        for (int i = 0; i < thread_count; ++i)
        {
            first = last;
            if (i == thread_count - 1) last = end;
            else std::advance(last, size);

            threads.emplace_back([first, last, &f, &r=mins[i]]() {
                r = std::forward<F>(f)(first, last);
            });
        }

        for (auto & t : threads) t.join();

        return std::forward<F>(f)(std::begin(mins), std::end(mins));
    }
}

```



Следующие две функции, `pmin()` и `pmax()`, реализуют обобщенные алгоритмы поиска максимального и минимального значений соответственно. Они вызывают `pprocess()`, передавая ей в третьем аргументе лямбда-выражение, использующее стандартный алгоритм `std::min_element()` или `std::max_element()`:

```

template <typename Iterator>
auto pmin(Iterator begin, Iterator end)
{
    return pprocess(begin, end,
        [](auto b, auto e){return *std::min_element(b, e);});
}

template <typename Iterator>
auto pmax(Iterator begin, Iterator end)

```



```
{
    return pprocess(begin, end,
                   [](auto b, auto e){return *std::max_element(b, e);});
}
```

Вот как можно использовать эти функции:

```
int main()
{
    std::vector<int> data(count);
    // инициализация данных
    auto rmin = pmin(std::begin(data), std::end(data));
    auto rmax = pmin(std::begin(data), std::end(data));
}
```



В качестве самостоятельного упражнения попробуйте реализовать обобщенный параллельный алгоритм, вычисляющий сумму элементов диапазона в нескольких потоках.

## 63. Параллельные алгоритмы поиска максимального и минимального значений с использованием асинхронных функций

Единственное отличие этой задачи от предыдущей – метод реализации конкурентного выполнения. В предыдущей задаче требовалось использовать потоки выполнения. В этой мы должны использовать асинхронные функции. Функцию можно вызвать асинхронно с помощью `std::async()`. Эта функция создает объект *promise*, который асинхронно возвращает результат заданной функции. Объект *promise* может хранить общедоступное состояние (значение, возвращенное функцией, или исключение, возникшее в ходе ее выполнения) и ассоциированный объект *future*, обеспечивающий доступ к состоянию из другого потока. Пара *promise/future* определяет канал для обмена значениями между потоками. `std::async()` возвращает объект *future*, связанный с созданным ею объектом *promise*.

Следующая реализация `pprocess()` вместо потоков выполнения использует функцию `std::async()`. Обратите внимание, что в первом аргументе функции `std::async()` нужно передать флаг `std::launch::async`, чтобы гарантировать асинхронное, а не отложенное выполнение. Изменений, по сравнению с предыдущей версией, очень немного, и вам не составит труда разобраться в коде, следуя за описанием алгоритма в предыдущем решении:

```
template <typename Iterator, typename F>
```

```
auto pprocess(Iterator begin, Iterator end, F&& f)
{
    auto size = std::distance(begin, end);
    if (size <= 10000)
    {
        return std::forward<F>(f)(begin, end);
    }
    else
    {
        int task_count = std::thread::hardware_concurrency();
        std::vector<std::future<
            typename std::iterator_traits<Iterator>::value_type>> tasks;

        auto first = begin;
        auto last = first;
        size /= task_count;
        for (int i = 0; i < task_count; ++i)
        {
            first = last;
            if (i == task_count - 1) last = end;
            else std::advance(last, size);

            tasks.emplace_back(std::async(
                std::launch::async,
                [first, last, &f]() {
                    return std::forward<F>(f)(first, last);
                }));
        }

        std::vector<typename std::iterator_traits<Iterator>::value_type>
            mins;

        for (auto & t : tasks)
            mins.push_back(t.get());

        return std::forward<F>(f)(std::begin(mins), std::end(mins));
    }
}

template <typename Iterator>
auto pmin(Iterator begin, Iterator end)
{
    return pprocess(begin, end,
        [](auto b, auto e){return *std::min_element(b, e);});
}

template <typename Iterator>
auto pmax(Iterator begin, Iterator end)
```



```
{
    return pprocess(begin, end,
                   [](auto b, auto e){return *std::max_element(b, e);});
}
```



В следующем листинге показано, как используются эти функции:

```
int main()
{
    std::vector<int> data(count);
    // инициализация данных
    auto rmin = pmin(std::begin(data), std::end(data));
    auto rmax = pmax(std::begin(data), std::end(data));
}
```



В качестве самостоятельного упражнения попробуйте реализовать обобщенный параллельный алгоритм, вычисляющий сумму элементов диапазона в нескольких потоках.



## 64. Параллельный алгоритм сортировки

В предыдущей главе мы видели последовательную реализацию алгоритма быстрой сортировки. Алгоритм быстрой сортировки действует по принципу «разделяй и властвуй». Он разбивает диапазон на две части, одна из которых содержит только элементы меньше выбранного элемента, который называют также опорным, а другая – больше. Затем он рекурсивно применяет тот же алгоритм к каждой из двух частей, пока не останется один элемент или ни одного. Благодаря его природе этот алгоритм легко распараллелить, рекурсивно применяя алгоритм к двум частям конкурентно.

Следующая функция `quick_sort()` использует для этой цели асинхронные функции. Однако распараллеливание дает положительный эффект только на больших массивах данных. Существует порог, ниже которого накладные расходы на переключение контекста выполнения оказываются слишком большими и время выполнения параллельного алгоритма оказывается больше времени выполнения последовательного алгоритма. В следующей реализации установлен порог 100 000 элементов, но вы можете поэкспериментировать и опробовать разные значения порога, сравнивая время выполнения параллельной и последовательной версий:

```
template <class RandomIt>
RandomIt partition(RandomIt first, RandomIt last)
{
    auto pivot = *first;
```

```

auto i = first + 1;
auto j = last - 1;
while (i <= j)
{
    while (i <= j && *i <= pivot) i++;
    while (i <= j && *j > pivot) j--;
    if (i < j) std::iter_swap(i, j);
}
std::iter_swap(i - 1, first);

return i - 1;
}

template <class RandomIt>
void pquicksort(RandomIt first, RandomIt last)
{
    if (first < last)
    {
        auto p = partition(first, last);

        if (last - first <= 100000)
        {
            pquicksort(first, p);
            pquicksort(p + 1, last);
        }
        else
        {
            auto f1 = std::async(std::launch::async,
                [first, p]() { pquicksort(first, p); });
            auto f2 = std::async(std::launch::async,
                [last, p]() { pquicksort(p+1, last); });
            f1.wait();
            f2.wait();
        }
    }
}
}

```



В следующем листинге показано, как отсортировать большой вектор случайных целых чисел (от 1 до 1000) с помощью функции `pquicksort()`:

```

int main()
{
    std::random_device rd;
    std::mt19937 mt;
    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));
}

```

```

mt.seed(seq);
std::uniform_int_distribution<> ud(1, 1000);

const size_t count = 1000000;
std::vector<int> data(count);
std::generate_n(std::begin(data), count,
               [&mt, &ud]() {return ud(mt); });

pquicksort(std::begin(data), std::end(data));
}

```



## 65. Потокобезопасное журналирование в консоль

В C++ отсутствует понятие консоли и для операций ввода/вывода с последовательными носителями используется идея потоков данных, но в нем имеются глобальные объекты `std::cout` и `std::wcout`, управляющие выводом в буфер, связанный с потоком вывода `stdout`. Эти глобальные объекты не поддерживают безопасное использование из разных потоков выполнения, а это означает, что вы должны синхронизировать обращения к ним. Именно такая синхронизация является конечной целью компонента, который предлагается написать для решения данной задачи.

Класс `logger`, показанный ниже, использует `std::mutex` в методе `log()` для синхронизации доступа к объекту `std::cout`. Класс реализован как потокобезопасный синглтон (одиночка). Статический метод `instance()` возвращает ссылку на локальный статический объект. В C++11 статические объекты создаются только один раз, даже если одновременно выполняется несколько попыток инициализации. В данном случае конкурирующие потоки блокируются, пока не выполнится инициализация, запущенная первым потоком. По этой причине нет необходимости предусматривать дополнительные механизмы синхронизации:

```

class logger
{
protected:
    logger() {}
public:
    static logger& instance()
    {
        static logger lg;
        return lg;
    }

    logger(logger const &) = delete;

    logger& operator=(logger const &) = delete;

    void log(std::string_view message)

```

```

    {
        std::lock_guard<std::mutex> lock(mt);
        std::cout << "LOG: " << message << std::endl;
    }

private:
    std::mutex mt;
};

```

Вот как можно использовать класс `logger` для вывода в консоль сообщений из нескольких потоков выполнения:

```

int main()
{
    std::vector<std::thread> modules;

    for(int id = 1; id <= 5; ++id)
    {
        modules.emplace_back([id]() {
            std::random_device rd;
            std::mt19937 mt(rd());
            std::uniform_int_distribution<> ud(100, 1000);

            logger::instance().log(
                "module " + std::to_string(id) + " started");

            std::this_thread::sleep_for(std::chrono::milliseconds(ud(mt)));

            logger::instance().log(
                "module " + std::to_string(id) + " finished");
        });
    }

    for(auto & m : modules) m.join();
}

```

## 66. Система обслуживания клиентов

Для реализации электронной очереди обслуживания клиентов в офисе, как определено условиями задачи, нам понадобится несколько классов. Класс `ticketing_machine` реализует очень простой автомат выдачи талонов с постоянно увеличивающимися номерами, начиная с некоторого начального значения, заданного пользователем. Класс `customer` представляет клиента, который входит в офис и получает талон из автомата. Этот класс перегружает `operator<`, для упорядочения клиентов в приоритетной очереди, из которой они вызываются в порядке номеров талонов. Кроме того, для вывода сообщений в консоль используется класс `logger` из предыдущей задачи:



```

class ticketing_machine
{
public:
    ticketing_machine(int const start) :
        last_ticket(start), first_ticket(start)
    {}

    int next() { return last_ticket++; }
    int last() const { return last_ticket - 1; }
    void reset() { last_ticket = first_ticket; }
private:
    int first_ticket;
    int last_ticket;
};

class customer
{
public:
    customer(int const no) : number(no) {}

    int ticket_number() const noexcept { return number; }
private:
    int number;
    friend bool operator<(customer const & l, customer const & r);
};

bool operator<(customer const & l, customer const & r)
{
    return l.number > r.number;
}

```



Стойки обслуживания в офисе моделируются потоками выполнения. Очередь клиентов, вошедших в офис и вставших в очередь к автомату для получения талона, моделируется с помощью отдельного потока выполнения. В следующем примере каждые 200–500 миллисекунд в офис входит новый клиент, получает талон, и информация о нем помещается в приоритетную очередь. Работа офиса заканчивается после обслуживания 25 клиентов. Для взаимодействия потоков используется `std::condition_variable`, посредством которой передается уведомление, что новый клиент встал в очередь или один из ожидающих клиентов был обслужен. Потоки, представляющие стойки обслуживания, выполняются, пока установлен признак, что офис открыт, но прекращают работу только после обслуживания всех клиентов в очереди. В данной имитации на обслуживание одного клиента тратится от 2000 до 3000 миллисекунд:

```

int main()
{
    std::priority_queue<customer> customers;

```

```

bool office_open = true;
std::mutex mt;
std::condition_variable cv;

std::vector<std::thread> desks;
for (int i = 1; i <= 3; ++i)
{
    desks.emplace_back([i, &office_open, &mt, &cv, &customers]() {
        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));
        std::mt19937 eng(seq);
        std::uniform_int_distribution<> ud(2000, 3000);

        logger::instance().log("desk " + std::to_string(i) + " open");

        while (office_open || !customers.empty())
        {
            std::unique_lock<std::mutex> locker(mt);

            cv.wait_for(locker, std::chrono::seconds(1),
                       [&customers]() {return !customers.empty(); });

            if (!customers.empty())
            {
                auto const c = customers.top();
                customers.pop();

                logger::instance().log(
                    "[ - ] desk " + std::to_string(i) + " handling customer "
                    + std::to_string(c.ticket_number()));

                logger::instance().log(
                    "[ = ] queue size: " + std::to_string(customers.size()));

                locker.unlock();
                cv.notify_one();

                std::this_thread::sleep_for(
                    std::chrono::milliseconds(ud(eng)));

                logger::instance().log(
                    "[ ] desk " + std::to_string(i) + " done with customer "
                    + std::to_string(c.ticket_number()));
            }
        }
    });
}

```



```

    logger::instance().log("desk " + std::to_string(i) + " closed");
  });
}

std::thread store([&office_open, &customers, &mt, &cv](){
    ticketing_machine tm(100);

    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));
    std::mt19937 eng(seq);
    std::uniform_int_distribution<> ud(200, 500);

    for (int i = 1; i <= 25; ++i)
    {
        customer c(tm.next());
        customers.push(c);

        logger::instance().log("[+] new customer with ticket " +
                                std::to_string(c.ticket_number()));
        logger::instance().log("[=] queue size: " +
                                std::to_string(customers.size()));

        cv.notify_one();

        std::this_thread::sleep_for(std::chrono::milliseconds(ud(eng)));
    }

    office_open = false;
});

store.join();
for (auto & desk : desks) desk.join();
}

```

Вот пример вывода программы, имитирующей работу офиса:

```

LOG: desk 1 open
LOG: desk 2 open
LOG: desk 3 open
LOG: [+] new customer with ticket 100
LOG: [-] desk 2 handling customer 100
LOG: [=] queue size: 0
LOG: [=] queue size: 0
LOG: [+] new customer with ticket 101

```

```
LOG: [=] queue size: 1
LOG: [-] desk 3 handling customer 101
LOG: [=] queue size: 0
LOG: [+] new customer with ticket 102
LOG: [=] queue size: 1
LOG: [-] desk 1 handling customer 102
LOG: [=] queue size: 0
LOG: [+] new customer with ticket 103
LOG: [=] queue size: 1
...
LOG: [+] new customer with ticket 112
LOG: [=] queue size: 7
LOG: [+] new customer with ticket 113
LOG: [=] queue size: 8
LOG: [ ] desk 2 done with customer 103
LOG: [-] desk 2 handling customer 106
LOG: [=] queue size: 7
...
LOG: [ ] desk 1 done with customer 120
LOG: [-] desk 1 handling customer 123
LOG: [=] queue size: 1
LOG: [ ] desk 2 done with customer 121
LOG: [-] desk 2 handling customer 124
LOG: [=] queue size: 0
LOG: [ ] desk 3 done with customer 122
LOG: desk 3 closed
LOG: [ ] desk 1 done with customer 123
LOG: desk 1 closed
LOG: [ ] desk 2 done with customer 124
LOG: desk 2 closed
```



В качестве самостоятельного упражнения попробуйте изменить интервал появления новых клиентов, количество обслуживаемых клиентов до закрытия офиса, продолжительность их обслуживания и количество работающих стоек.

---

# Глава 8

## Шаблоны проектирования

### Задачи

#### 67. Проверка пароля

Напишите программу для проверки мощности пароля на основе predetermined правил, которые могут выбираться в разных комбинациях. Каждый пароль, как минимум, должен соответствовать требованию длины. Дополнительно могут вводиться в действие другие правила, такие как наличие хотя бы одного неалфавитного символа, цифры, букв нижнего и верхнего регистра и т. д.

#### 68. Генерация случайных паролей

Напишите программу, генерирующую случайные пароли, отвечающие некоторым predetermined требованиям. Каждый пароль должен иметь длину не меньше установленной. Также могут применяться дополнительные правила, такие как наличие хотя бы одного неалфавитного символа, цифры, букв нижнего и верхнего регистров и т. д. Должна иметься возможность определять такие дополнительные правила и вводить их в действие.

#### 69. Генерация номеров социального страхования

Напишите программу, генерирующую номера социального страхования для двух стран, Северландии и Югостана, имеющие разные, но схожие форматы:

- в Северландии номера социального страхования имеют формат  $SYYYMMDDNNNNNS$ , где  $s$  – цифра, определяющая пол, 9 для женщин и 7 для мужчин,  $YYYYMMDD$  – дата рождения,  $NNNNN$  – пять случайных цифр, уникальных для даты (то есть для разных дат может использоваться одно и то же число), и  $s$  – цифра, выбираемая так, чтобы контрольная сумма номера, алгоритм вычисления которой описывается ниже, была кратна 11;
- в Югостане номера социального страхования имеют формат  $SYYYMMDDNNNNNS$ , где  $s$  – цифра, определяющая пол, 1 для женщин и 2 для мужчин,  $YYYYMMDD$  – дата рождения,  $NNNN$  – четырехзначное случайное

число, уникальное для даты рождения, и  $s$  – цифра, выбираемая так, чтобы контрольная сумма номера, алгоритм вычисления которой описывается ниже, была кратна 10.

Контрольная сумма в обоих случаях – это сумма всех цифр, умноженных на их весовые коэффициенты (позиции в номере). Например, контрольная сумма для номера 12017120134895 социального страхования в Югостане вычисляется так:

$$\begin{aligned} \text{срс} &= 14*1 + 13*2 + 12*0 + 11*1 + 10*7 + 9*1 + 8*2 + 7*0 + 6*1 + 5*3 \\ &\quad + 4*4 + 3*8 + 2*9 + 1*5 \\ &= 230 = 23 * 10 \end{aligned}$$

## 70. Система одобрений

Напишите программу для отдела закупок компании, которая позволит сотрудникам утверждать новые покупки (или расходы). Однако, учитывая положение в иерархии компании, каждый сотрудник может высказать свое одобрение только для затрат до определенного размера. Например, рядовые сотрудники могут одобрить расходы величиной до 1000 валютных единиц, руководители отделов – до 10 000, а руководители департаментов – до 100 000. Любые расходы выше этих сумм должны получить одобрение со стороны президента компании.

## 71. Контейнер с наблюдателями

Напишите шаблонный класс, который действует подобно вектору, но может уведомлять зарегистрированных подписчиков об изменении внутреннего состояния. Как минимум класс должен поддерживать:

- разнообразные конструкторы для создания новых экземпляров класса;
- `operator=` для присваивания значений контейнеру;
- `push_back()` для добавления новых элементов в конец контейнера;
- `pop_back()` для удаления последнего элемента из контейнера;
- `clear()` для удаления всех элементов из контейнера;
- `size()` для получения количества элементов в контейнере;
- `empty()` для получения признака отсутствия элементов в контейнере.

`operator=`, `push_back()`, `pop_back()` и `clear()` должны уведомлять подписчиков об изменении состояния контейнера. Уведомления должны включать тип изменения и, если это имеет смысл, индекс изменившегося элемента (например, добавленного или удаленного).

## 72. Вычисление стоимости заказа с учетом скидок

Розничный магазин продает разные товары и может предлагать различные виды скидок некоторым клиентам на определенные виды товаров или в зависимости от стоимости заказа, например:

- фиксированная скидка 5% на все товары независимо от суммы заказа;
- оптовая скидка 10% при приобретении товаров больше определенного количества;
- скидка 15% при приобретении одного товара на общую сумму больше установленной величины, например \$100; если покупатель приобретает 30 единиц товара стоимостью \$5, общая сумма за товар составит \$150, соответственно, к этому товару применяется скидка 15%;
- скидка с общей стоимости заказа (независимо от товаров, вошедших в заказ).

Напишите программу, вычисляющую окончательную стоимость заказа. Стоимость можно вычислять разными способами; например, все скидки могут накапливаться или, если к какому-то товару применена скидка, он исключается из учета при начислении других скидок.

## Решения

### 67. Проверка пароля

Данная задача является типичным кандидатом на применение шаблона проектирования «Декоратор». Этот шаблон позволяет расширять поведение объекта, не влияя на поведение других объектов этого же типа, что достигается путем обертывания одного объекта другим. Допускается обертывать одни декораторы другими, каждый из которых добавляет свои функциональные возможности. В данном случае под возможностями подразумеваются разные правила, которым должен соответствовать пароль.

Диаграмма классов на рис. 8.1 описывает шаблон проверки паролей.

Далее приводится реализация шаблона, изображенного на этой диаграмме:

```
class password_validator
{
public:
    virtual bool validate(std::string_view password) = 0;
    virtual ~password_validator() {}
};

class length_validator final : public password_validator
{
public:
    length_validator(unsigned int min_length): length(min_length)
    {}

    virtual bool validate(std::string_view password) override
    {
        return password.length() >= length;
    }
};
```

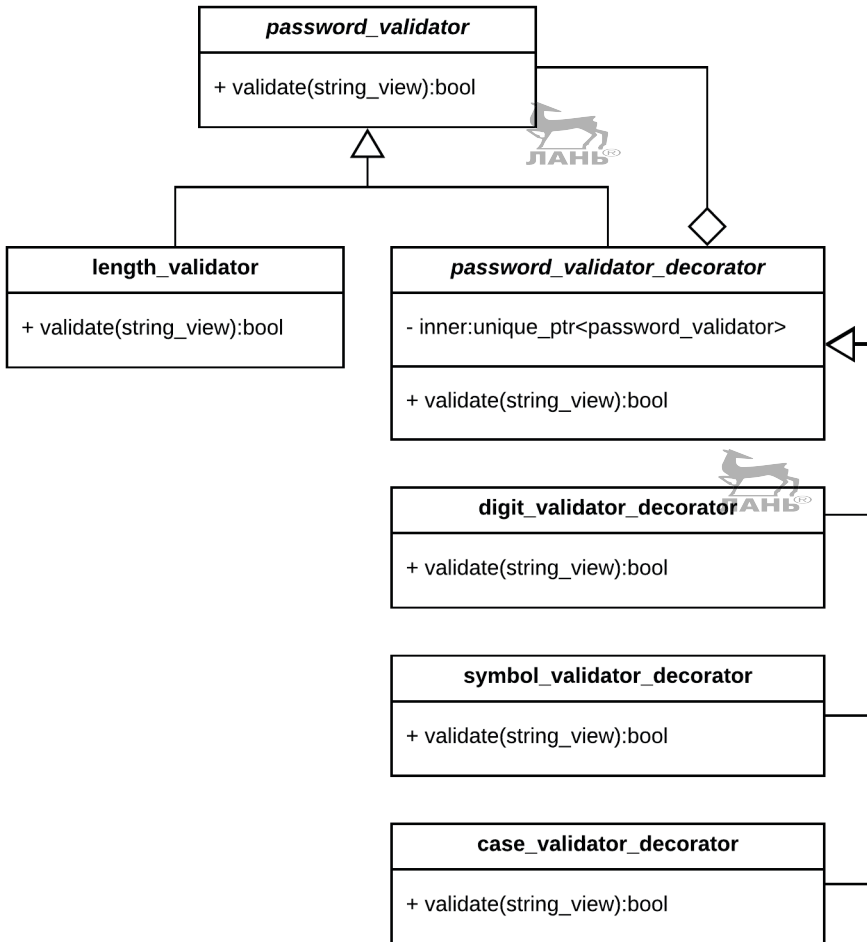


Рис. 8.1. Диаграмма классов шаблона проверки паролей

```

}

private:
    unsigned int length;
};

class password_validator_decorator : public password_validator
{
public:
    explicit password_validator_decorator(
        std::unique_ptr<password_validator> validator):
        inner(std::move(validator))
  
```



```

    {
    }

    virtual bool validate(std::string_view password) override
    {
        return inner->validate(password);
    }

private:
    std::unique_ptr<password_validator> inner;
};

class digit_password_validator final : public password_validator_decorator
{
public:
    explicit digit_password_validator(
        std::unique_ptr<password_validator> validator):
        password_validator_decorator(std::move(validator))
    {
    }

    virtual bool validate(std::string_view password) override
    {
        if(!password_validator_decorator::validate(password))
            return false;

        return password.find_first_of("0123456789") != std::string::npos;
    }
};

class case_password_validator final : public password_validator_decorator
{
public:
    explicit case_password_validator(
        std::unique_ptr<password_validator> validator):
        password_validator_decorator(std::move(validator))
    {
    }

    virtual bool validate(std::string_view password) override
    {
        if(!password_validator_decorator::validate(password))
            return false;

        bool haslower = false;
        bool hasupper = false;

        for(size_t i = 0; i < password.length() && !(hasupper && haslower); ++i)

```



```

    {
        if(islower(password[i])) haslower = true;
        else if(isupper(password[i])) hasupper = true;
    }

    return haslower && hasupper;
}
};

class symbol_password_validator final : public password_validator_decorator
{
public:
    explicit symbol_password_validator(
        std::unique_ptr<password_validator> validator):
        password_validator_decorator(std::move(validator))
    {
    }

    virtual bool validate(std::string_view password) override
    {
        if(!password_validator_decorator::validate(password))
            return false;

        return password.find_first_of("!@#%$%^&*(){}[]?<>") !=
            std::string::npos;
    }
};

```



`password_validator` – базовый класс с единственным виртуальным методом `validate()`, который принимает строковый аргумент с паролем.

`length_validator` – наследует этот класс и реализует обязательную проверку минимальной длины.

`password_validator_decorator` – тоже наследует класс `password_validator` и содержит внутренний компонент `password_validator`. Его метод `validate()` просто вызывает `inner->validate()`.

Другие классы – `digit_password_validator`, `symbol_password_validator` и `case_password_validator` – наследуют `password_validator_decorator` и реализуют свои требования к паролю.

Вот пример комбинирования этих классов для создания разных компонентов проверки паролей:

```

int main()
{
    auto validator1 = std::make_unique<digit_password_validator>(
        std::make_unique<length_validator>(8));

    assert(validator1->validate("abc123!@#"));
}

```



```

assert(!validator1->validate("abcde!@#"));

auto validator2 =
    std::make_unique<symbol_password_validator>(
        std::make_unique<case_password_validator>(
            std::make_unique<digit_password_validator>(
                std::make_unique<length_validator>(8))));

assert(validator2->validate("Abc123!@#"));
assert(!validator2->validate("Abc123567"));
}

```

## 68. Генерация случайных паролей

Эту задачу можно решить применением шаблона проектирования «Компоновщик». Согласно этому шаблону, объекты объединяются в древовидную иерархию и эти группы (или деревья) интерпретируются как отдельные объекты того же типа. Диаграмма на рис. 8.2 перечисляет классы, которые можно использовать для генерации паролей.

`password_generator` – базовый класс, имеющий несколько виртуальных методов:

`generate()` возвращает новую случайную строку, `length()` возвращает минимальную длину генерируемой строки, `allowed_chars()` возвращает строку со всеми допустимыми символами, которые могут использоваться при генерации пароля, и `add()` добавляет новый дочерний компонент, являющийся составной частью генератора.

Класс `basic_password_generator` наследует базовый класс и определяет генератор паролей с заданной минимальной длиной.

Классы `digit_generator`, `symbol_generator`, `upper_letter_generator` и `lower_letter_generator` наследуют `basic_password_generator` и переопределяют метод `allowed_chars()`, задавая подмножества допустимых символов.

Класс `composite_password_generator` тоже наследует `password_generator` и имеет коллекцию объектов `password_generator`, которые использует для конструирования случайного текста в своей версии метода `generate()`, который объединяет все строки, сгенерированные дочерними компонентами, и затем случайно перемешивает их, чтобы получить окончательный пароль:

```

class password_generator
{
public:
    virtual std::string generate() = 0;

    virtual std::string allowed_chars() const = 0;
    virtual size_t length() const = 0;

    virtual void add(std::unique_ptr<password_generator> generator) = 0;

```

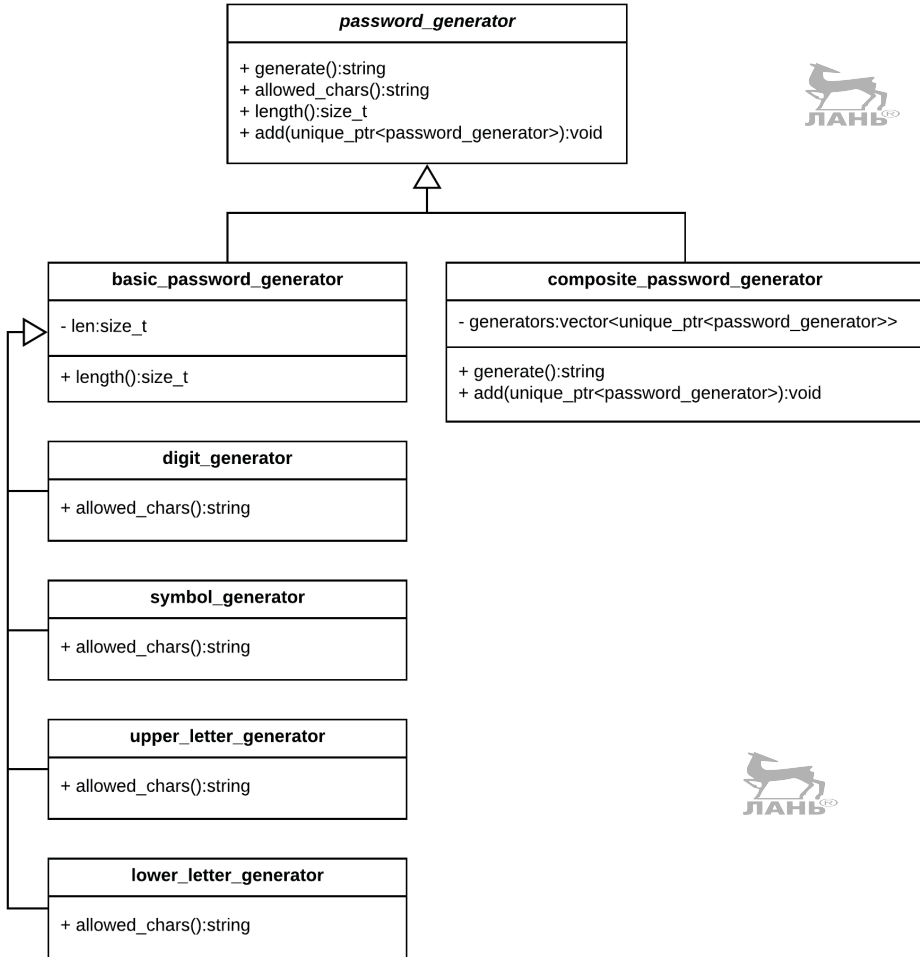


Рис. 8.2. Классы для генерации паролей

```

virtual ~password_generator(){}
};

class basic_password_generator : public password_generator
{
    size_t len;
public:
    explicit basic_password_generator(size_t const len) noexcept : len(len)
    {}

    virtual std::string generate() override
  
```

```

    { throw std::runtime_error("not implemented"); }

    virtual void add(std::unique_ptr<password_generator>) override
    { throw std::runtime_error("not implemented"); }

    virtual size_t length() const override final
    {return len;}
};

class digit_generator : public basic_password_generator
{
public:
    explicit digit_generator(size_t const len) noexcept
        : basic_password_generator(len) {}

    virtual std::string allowed_chars() const override
    {return "0123456789";}
};

class symbol_generator : public basic_password_generator
{
public:
    explicit symbol_generator(size_t const len) noexcept
        : basic_password_generator(len) {}

    virtual std::string allowed_chars() const override
    {return "!@#$%^&*(){}[]?<>";}
};

class upper_letter_generator : public basic_password_generator
{
public:
    explicit upper_letter_generator(size_t const len) noexcept
        : basic_password_generator(len) {}

    virtual std::string allowed_chars() const override
    {return "ABCDEFGHIJKLMNOPQRSTUVWXYZ";}
};

class lower_letter_generator : public basic_password_generator
{
public:
    explicit lower_letter_generator(size_t const len) noexcept
        : basic_password_generator(len) {}

    virtual std::string allowed_chars() const override
    {return "abcdefghijklmnopqrstuvwxyz";}
};

```



```

};

class composite_password_generator : public password_generator
{
    virtual std::string allowed_chars() const override
    { throw std::runtime_error("not implemented"); };

    virtual size_t length() const override
    { throw std::runtime_error("not implemented"); };
public:
    composite_password_generator()
    {
        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));
        eng.seed(seq);
    }

    virtual std::string generate() override
    {
        std::string password;
        for(auto & generator : generators)
        {
            std::string chars = generator->allowed_chars();
            std::uniform_int_distribution<> ud(
                0, static_cast<int>(chars.length() - 1));

            for(size_t i = 0; i < generator->length(); ++i)
                password += chars[ud(eng)];
        }

        std::shuffle(std::begin(password), std::end(password), eng);

        return password;
    }

    virtual void add(std::unique_ptr<password_generator> generator) override
    {
        generators.push_back(std::move(generator));
    }

private:
    std::random_device rd;
    std::mt19937 eng;
    std::vector<std::unique_ptr<password_generator>> generators;
};

```



Вот как можно использовать эти классы для генерации паролей:

```
int main()
{
    composite_password_generator generator;
    generator.add(std::make_unique<symbol_generator>(2));
    generator.add(std::make_unique<digit_generator>(2));
    generator.add(std::make_unique<upper_letter_generator>(2));
    generator.add(std::make_unique<lower_letter_generator>(4));
    auto password = generator.generate();
}
```



Желающие могут попробовать использовать программу из предыдущего решения для проверки соответствия генерируемых паролей установленным требованиям.

## 69. Генерация номеров социального страхования

Номера социального страхования в обеих странах имеют схожий формат, отличаясь лишь мелкими деталями:

- значением цифры, определяющей пол;
- длиной случайной части номера и, соответственно, общей длиной номера;
- числом, которому должна быть кратна контрольная сумма.

Эта задача может быть решена с применением шаблона проектирования «Шаблонный метод», который определяет скелет алгоритма и позволяет переопределять конкретные операции в подклассах, как показано на рис. 8.3.

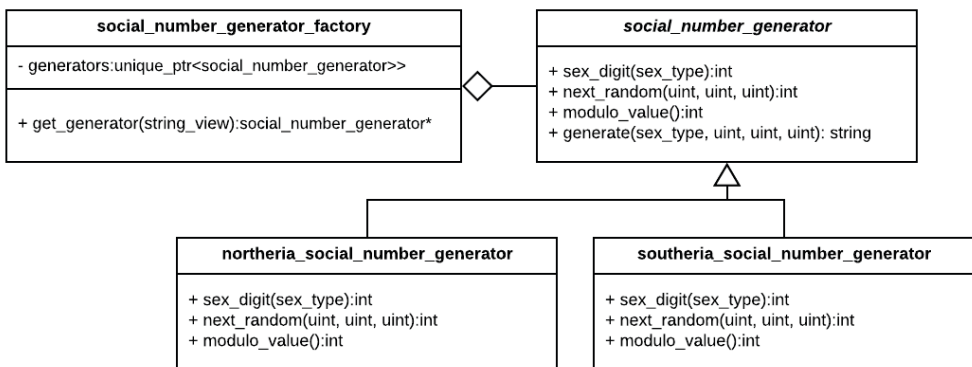


Рис. 8.3. Диаграмма классов шаблона проектирования «Шаблонный метод»

`social_number_generator` – базовый класс с общедоступным методом `generate()`, который возвращает новый номер социального страхования для заданного пола и даты рождения. Этот метод внутренне вызывает несколько за-

щищенных виртуальных методов: `sex_digit()`, `next_random()` и `modulo_value()`. Эти виртуальные методы переопределяются в двух дочерних классах, `north-eria_social_number_generator` и `southeria_social_number_generator`. Кроме того, имеется фабричный класс, который хранит экземпляры этих двух классов и возвращает их по запросу клиентов:

```
enum class sex_type {female, male};
class social_number_generator
{
protected:
    virtual int sex_digit(sex_type const sex) const noexcept = 0;
    virtual int next_random(unsigned const year, unsigned const month,
                            unsigned const day) = 0;
    virtual int modulo_value() const noexcept = 0;

    social_number_generator(int const min, int const max):ud(min, max)
    {
        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));
        eng.seed(seq);
    }

public:
    std::string generate(
        sex_type const sex,
        unsigned const year, unsigned const month, unsigned const day)
    {
        std::stringstream snumber;

        snumber << sex_digit(sex);

        snumber << year << month << day;

        snumber << next_random(year, month, day);

        auto number = snumber.str();

        auto index = number.length();
        auto sum = std::accumulate(std::begin(number), std::end(number), 0,
            [&index](int const s, char const c) {
                return s + index-- * (c-'0');});

        auto rest = sum % modulo_value();
        snumber << modulo_value() - rest;

        return snumber.str();
    }
};
```



```

    }

    virtual ~social_number_generator() {}

protected:
    std::map<unsigned, int> cache;
    std::mt19937 eng;
    std::uniform_int_distribution<> ud;
};

class southeria_social_number_generator final :
public social_number_generator
{
public:
    southeria_social_number_generator():
        social_number_generator(1000, 9999)
    {
    }

protected:
    virtual int sex_digit(sex_type const sex) const noexcept override
    {
        if(sex == sex_type::female) return 1;
        else return 2;
    }

    virtual int next_random(unsigned const year, unsigned const month,
                            unsigned const day) override
    {
        auto key = year * 10000 + month * 100 + day;
        while(true)
        {
            auto number = ud(eng);
            auto pos = cache.find(number);
            if(pos == std::end(cache))
            {
                cache[key] = number;
                return number;
            }
        }
    }

    virtual int modulo_value() const noexcept override
    {
        return 11;
    }
};

class northeria_social_number_generator final :

```

```

public social_number_generator
{
public:
    northeria_social_number_generator():
        social_number_generator(10000, 99999)
    {
    }

protected:
    virtual int sex_digit(sex_type const sex) const noexcept override
    {
        if(sex == sex_type::female) return 9;
        else return 7;
    }

    virtual int next_random(unsigned const year, unsigned const month,
                            unsigned const day) override
    {
        auto key = year * 10000 + month * 100 + day;
        while(true)
        {
            auto number = ud(eng);
            auto pos = cache.find(number);
            if(pos == std::end(cache))
            {
                cache[key] = number;
                return number;
            }
        }
    }

    virtual int modulo_value() const noexcept override
    {
        return 11;
    }
};

class social_number_generator_factory
{
public:
    social_number_generator_factory()
    {
        generators["northeria"] =
            std::make_unique<northeria_social_number_generator>();
        generators["southeria"] =
            std::make_unique<southeria_social_number_generator>();
    }

    social_number_generator* get_generator(std::string_view country) const

```



```

{
    auto it = generators.find(country.data());
    if(it != std::end(generators))
        return it->second.get();

    throw std::runtime_error("invalid country");
}

private:
    std::map<std::string,
    std::unique_ptr<social_number_generator>> generators;
};

```



Вот как с помощью этих классов можно генерировать номера социального страхования:

```

int main()
{
    social_number_generator_factory factory;

    auto sn1 = factory.get_generator("northeria")->generate(
        sex_type::female, 2017, 12, 25);
    auto sn2 = factory.get_generator("northeria")->generate(
        sex_type::female, 2017, 12, 25);
    auto sn3 = factory.get_generator("northeria")->generate(
        sex_type::male, 2017, 12, 25);

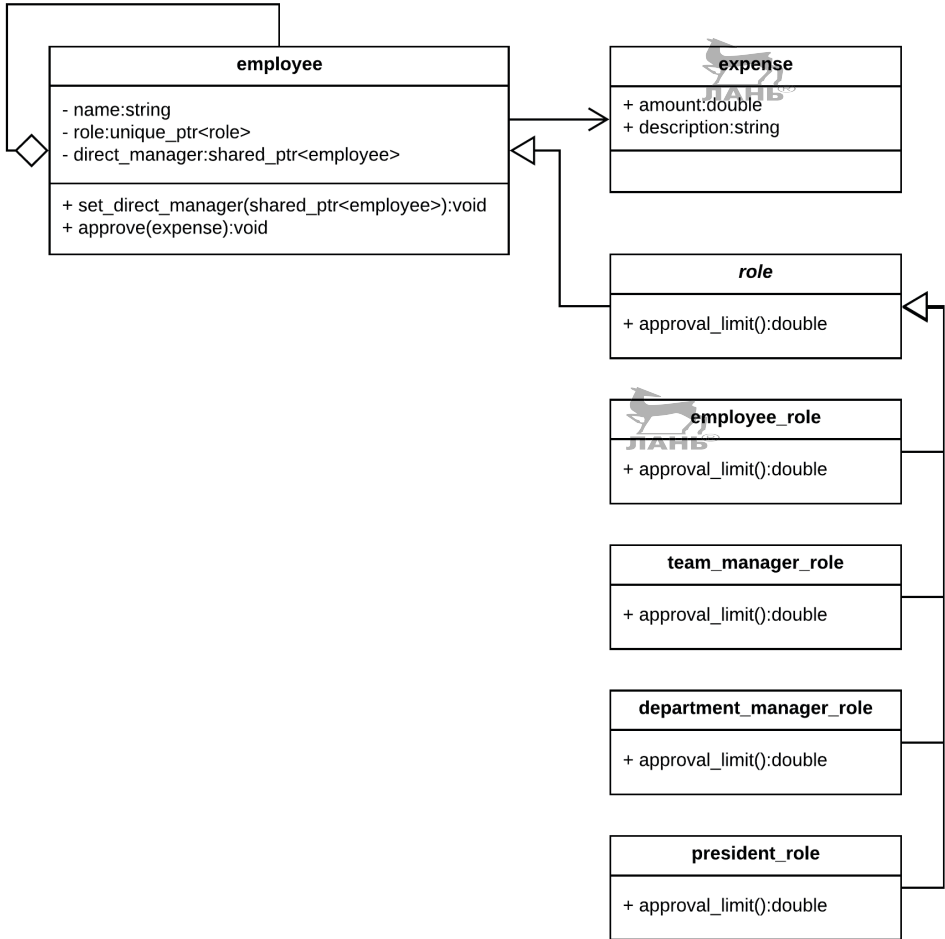
    auto sss1 = factory.get_generator("southeria")->generate(
        sex_type::female, 2017, 12, 25);
    auto ss2 = factory.get_generator("southeria")->generate(
        sex_type::female, 2017, 12, 25);
    auto ss3 = factory.get_generator("southeria")->generate(
        sex_type::male, 2017, 12, 25);
}

```

## 70. Система одобрений

Описанную задачу можно решить с помощью последовательности инструкций `if ... else if ... else ... endif`. В объектно-ориентированной версии эту идиому выражает шаблон проектирования «Цепочка обязанностей». Этот шаблон определяет цепочку объектов, которые обрабатывают запрос или передают его следующему объекту в цепочке, если имеется. Диаграмма классов на рис. 8.4 иллюстрирует возможную реализацию этого шаблона.

`employee` – класс, представляющий сотрудника компании. Сотрудник может иметь прямого руководителя, который назначается вызовом метода `set_direct_manager()`. Каждый сотрудник имеет имя и роль, определяющую уровень его ответственности и полномочий.



**Рис. 8.4.** Диаграмма классов шаблона проектирования «Цепочка обязанностей»

`role` – абстрактный базовый класс для всех возможных ролей. Имеет чистый виртуальный метод `approval_limit()`, который наследуется дочерними классами, такими как `employee_role`, `team_manager_role`, `department_manager_role` и `president_role`, и переопределяющими его, задавая ограничение на сумму расходов, которую сотрудник с данной ролью может одобрить.

Метод `approve()` класса `employee` используется классом `employee` для одобрения расходов. Если роль сотрудника позволяет одобрить заданную сумму, он выдает такое одобрение; иначе запрос передается прямому руководителю, если он определен:

```

class role
{

```

```
public:
    virtual double approval_limit() const noexcept = 0;
    virtual ~role() {}
};

class employee_role : public role
{
public:
    virtual double approval_limit() const noexcept override
    {
        return 1000;
    }
};

class team_manager_role : public role
{
public:
    virtual double approval_limit() const noexcept override
    {
        return 10000;
    }
};

class department_manager_role : public role
{
public:
    virtual double approval_limit() const noexcept override
    {
        return 100000;
    }
};

class president_role : public role
{
public:
    virtual double approval_limit() const noexcept override
    {
        return std::numeric_limits<double>::max();
    }
};

struct expense
{
    double amount;
    std::string description;

    expense(double const amount, std::string_view desc):
```



```

    amount(amount), description(desc)
    {
    }
};

class employee
{
public:
    explicit employee(std::string_view name, std::unique_ptr<role> ownrole)
        : name(name), own_role(std::move(ownrole))
    {
    }

    void set_direct_manager(std::shared_ptr<employee> manager)
    {
        direct_manager = manager;
    }

    void approve(expense const & e)
    {
        if(e.amount <= own_role->approval_limit())
            std::cout << name << " approved expense '" << e.description
                << "', cost=" << e.amount << std::endl;
        else if(direct_manager != nullptr)
            direct_manager->approve(e);
    }

private:
    std::string          name;
    std::unique_ptr<role> own_role;
    std::shared_ptr<employee> direct_manager;
};

```

Следующий код демонстрирует, как осуществляется одобрение расходов с использованием этих классов:

```

int main()
{
    auto john = std::make_shared<employee>("john smith",
        std::make_unique<employee_role>());

    auto robert = std::make_shared<employee>("robert booth",
        std::make_unique<team_manager_role>());

    auto david = std::make_shared<employee>("david jones",
        std::make_unique<department_manager_role>());

    auto cecil = std::make_shared<employee>("cecil williamson",

```

```

std::make_unique<president_role>());

john->set_direct_manager(robert);
robert->set_direct_manager(david);
david->set_direct_manager(cecil);

john->approve(expense{500, "magazins"});
john->approve(expense{5000, "hotel accomodation"});
john->approve(expense{50000, "conference costs"});
john->approve(expense{200000, "new lorry"});
}

```



## 71. Контейнер с наблюдателями

Контейнер с наблюдателями, который нужно реализовать в этой задаче, – типичный случай для применения шаблона проектирования «Наблюдатель». Этот шаблон описывает объект, который называется **субъектом**, и поддерживает список зависимых объектов, которые называются **наблюдателями**. Субъект извещает наблюдателей обо всех изменениях в своем состоянии, вызывая их методы. Диаграмма классов на рис. 8.5 иллюстрирует одно из возможных решений данной задачи.

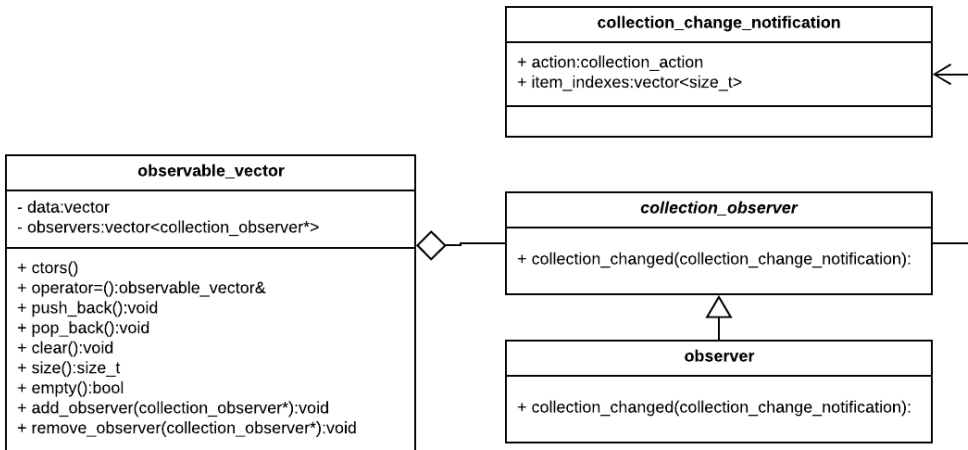



Рис. 8.5. Диаграмма классов шаблона проектирования «Наблюдатель»

`observable_vector` – это класс, обертывающий `std::vector` и реализующий необходимые операции. Он также поддерживает список указателей на объекты `collection_observer` – базового класса, экземпляры которого должны уведомляться о любых изменениях в состоянии `observable_vector`. Он имеет виртуальный метод `collection_changed()` с аргументом типа `collection_change_notification`, посредством которого передается информация об изменении.

Когда происходит любое изменение состояния объекта `observable_vector`, он вызывает этот метод всех зарегистрированных наблюдателей.

Наблюдатели могут регистрироваться вызовом метода `add_observer()` **ВЕКТО-**ра и удаляться из списка наблюдателей вызовом `remove_observer()`: 

```
enum class collection_action
{
    add,
    remove,
    clear,
    assign
};

std::string to_string(collection_action const action)
{
    switch(action)
    {
        case collection_action::add:    return "add";
        case collection_action::remove: return "remove";
        case collection_action::clear:   return "clear";
        case collection_action::assign:  return "assign";
    }
}

struct collection_change_notification
{
    collection_action action;
    std::vector<size_t> item_indexes;
};

class collection_observer
{
public:
    virtual void collection_changed(
        collection_change_notification notification) = 0;
    virtual ~collection_observer() {}
};

template <typename T, class Allocator = std::allocator<T>>
class observable_vector final
{
    typedef typename std::vector<T, Allocator>::size_type size_type;
public:
    observable_vector() noexcept(noexcept(Allocator()))
        : observable_vector( Allocator() ) {}
    explicit observable_vector( const Allocator& alloc ) noexcept
        : data(alloc){}
};
```





```

observable_vector( size_type count, const T& value,
                  const Allocator& alloc = Allocator())
    : data(count, value, alloc){}
explicit observable_vector( size_type count,
                          const Allocator& alloc = Allocator() )
    :data(count, alloc){}
observable_vector(observable_vector&& other) noexcept
    :data(other.data){}
observable_vector(observable_vector&& other,
                const Allocator& alloc)
    :data(other.data, alloc){}
observable_vector(std::initializer_list<T> init,
                const Allocator& alloc = Allocator())
    :data(init, alloc){}
template<class InputIt>
observable_vector(InputIt first, InputIt last, const
                Allocator& alloc = Allocator())
    :data(first, last, alloc){}

```



```

observable_vector& operator=(observable_vector const & other)
{
    if(this != &other)
    {
        data = other.data;

        for(auto o : observers)
        {
            if(o != nullptr)
            {
                o->collection_changed({
                    collection_action::assign,
                    std::vector<size_t> {}
                });
            }
        }
    }

    return *this;
}

```



```

observable_vector& operator=(observable_vector&& other)
{
    if(this != &other)
    {
        data = std::move(other.data);

        for(auto o : observers)

```

```
{
    if(o != nullptr)
    {
        o->collection_changed({
            collection_action::assign,
            std::vector<size_t> {}
        });
    }
}

return *this;
}

void push_back(T&& value)
{
    data.push_back(value);

    for(auto o : observers)
    {
        if(o != nullptr)
        {
            o->collection_changed({
                collection_action::add,
                std::vector<size_t> {data.size()-1}
            });
        }
    }
}

void pop_back()
{
    data.pop_back();

    for(auto o : observers)
    {
        if(o != nullptr)
        {
            o->collection_changed({
                collection_action::remove,
                std::vector<size_t> {data.size()+1}
            });
        }
    }
}

void clear() noexcept
{
```



```

    data.clear();

    for(auto o : observers)
    {
        if(o != nullptr)
        {
            o->collection_changed({
                collection_action::clear,
                std::vector<size_t> {}
            });
        }
    }
}

size_type size() const noexcept
{
    return data.size();
}

[[nodiscard]] bool empty() const noexcept
{
    return data.empty();
}

void add_observer(collection_observer * const o)
{
    observers.push_back(o);
}

void remove_observer(collection_observer const * const o)
{
    observers.erase(std::remove(std::begin(observers),
                                std::end(observers), o),
                    std::end(observers));
}

private:
    std::vector<T, Allocator> data;
    std::vector<collection_observer*> observers;
};

class observer : public collection_observer
{
public:
    virtual void collection_changed(
        collection_change_notification notification) override
    {
        std::cout << "action: " << to_string(notification.action);
    }
};

```



```

if(!notification.item_indexes.empty())
{
    std::cout << ", indexes: ";
    for(auto i : notification.item_indexes)
        std::cout << i << ' ';
}
std::cout << std::endl;
}
};

```



В следующем листинге приводится пример использования класса `observable_vector` и получения уведомлений при изменении его внутреннего состояния:

```

int main()
{
    observable_vector<int> v;
    observer o;

    v.add_observer(&o);
    v.push_back(1);
    v.push_back(2);
    v.pop_back();
    v.clear();

    v.remove_observer(&o);

    v.push_back(3);
    v.push_back(4);

    v.add_observer(&o);

    observable_vector<int> v2 {1,2,3};
    v = v2;
    v = observable_vector<int> {7,8,9};
}

```



Желающие могут продолжить развитие этого решения и добавить в `observable_vector` возможность доступа к элементам вектора посредством итераторов.

## 72. Вычисление стоимости заказа с учетом скидок

Предложенную задачу можно решить с помощью шаблона проектирования «Стратегия». Он определяет семейство алгоритмов и обеспечивает их взаи-

мозаменяемость в пределах семейства. В данной конкретной задаче шаблон «Стратегия» можно применить для реализации калькуляторов скидков и окончательной стоимости заказа. Диаграмма на рис. 8.6 иллюстрирует иерархию типов скидков и их взаимозаменяемость в других классах: `customer`, `article`, `order_line` и `order`.

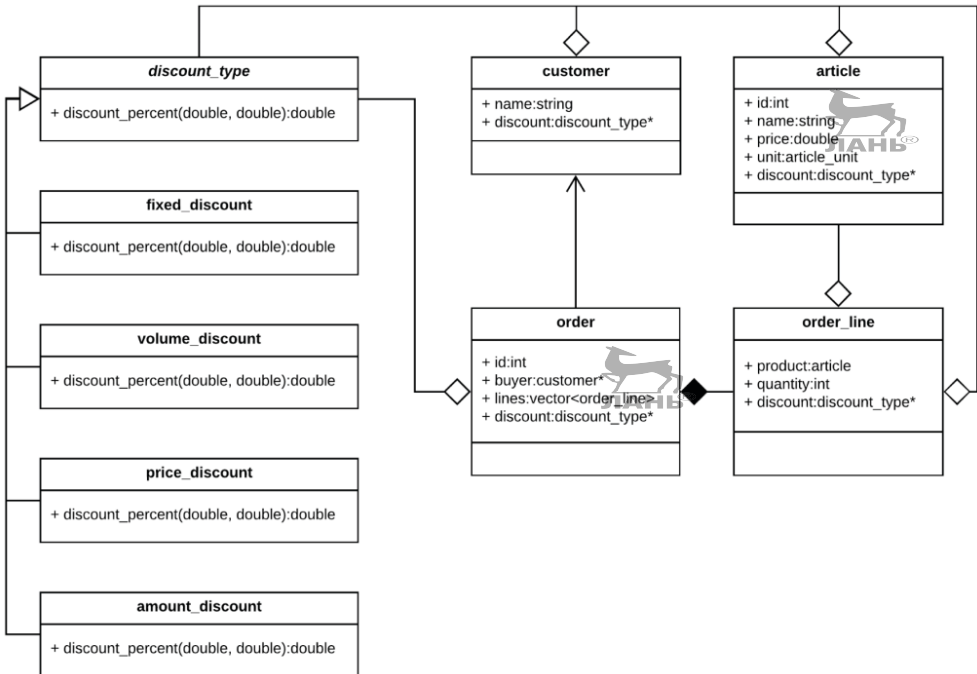


Рис. 8.6. Диаграмма классов шаблона проектирования «Стратегия»

Вот одна из возможных реализаций типов скидков:

```

struct discount_type
{
    virtual double discount_percent(
        double const price, double const quantity) const noexcept = 0;
    virtual ~discount_type() {}
};

```

```

struct fixed_discount final : public discount_type
{
    explicit fixed_discount(double const discount) noexcept
        : discount(discount) {}
    virtual double discount_percent(
        double const, double const) const noexcept

```

```
{return discount;}

private:
    double discount;
};

struct volume_discount final : public discount_type
{
    explicit volume_discount(double const quantity,
                             double const discount) noexcept
        : discount(discount), min_quantity(quantity) {}
    virtual double discount_percent(
        double const, double const quantity) const noexcept
        {return quantity >= min_quantity ? discount : 0;}

private:
    double discount;
    double min_quantity;
};

struct price_discount : public discount_type
{
    explicit price_discount(double const price,
                            double const discount) noexcept
        : discount(discount), min_total_price(price) {}
    virtual double discount_percent(
        double const price, double const quantity) const noexcept
        {return price*quantity >= min_total_price ? discount : 0;}

private:
    double discount;
    double min_total_price;
};

struct amount_discount : public discount_type
{
    explicit amount_discount(double const price,
                             double const discount) noexcept
        : discount(discount), min_total_price(price) {}
    virtual double discount_percent(
        double const price, double const) const noexcept
        {return price >= min_total_price ? discount : 0;}

private:
    double discount;
    double min_total_price;
};
```



Для простоты примера классы, моделирующие клиента (customer), товары (article) и заказы (order) имеют минимальную структуру:

```
struct customer
{
    std::string    name;
    discount_type* discount;
};

enum class article_unit
{
    piece, kg, meter, sqmeter, cmeter, liter
};

struct article
{
    int            id;
    std::string    name;
    double         price;
    article_unit   unit;
    discount_type* discount;
};

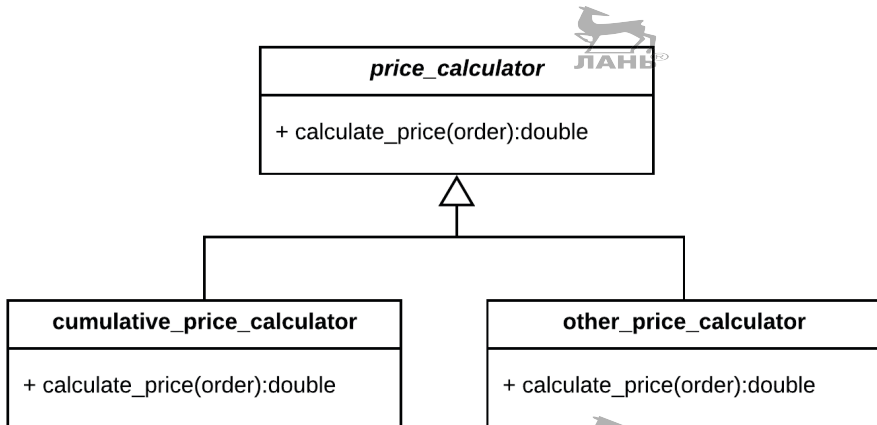
struct order_line
{
    article        product;
    int            quantity;
    discount_type* discount;
};

struct order
{
    int            id;
    customer*      buyer;
    std::vector<order_line> lines;
    discount_type* discount;
};
```



Для вычисления окончательной стоимости заказа можно использовать разные типы калькуляторов. На рис. 8.7 показан еще один пример реализации шаблона «Стратегия».

price\_calculator – абстрактный базовый класс, имеющий чистый виртуальный метод calculate\_price(). Дочерние классы, наследующие price\_calculator, такие как cumulative\_price\_calculator, реализуют фактические алгоритмы вычислений, переопределяя calculate\_price(). Для простоты в этой реализации показана только одна конкретная стратегия вычисления стоимости. Реализацию остальных я оставляю читателям как самостоятельное упражнение:



**Рис. 8.7.** Еще одна диаграмма классов шаблона проектирования «Стратегия»

```

struct price_calculator
{
    virtual double calculate_price(order const & o) = 0;
};

struct cumulative_price_calculator : public price_calculator
{
    virtual double calculate_price(order const & o) override
    {
        double price = 0;

        for(auto ol : o.lines)
        {
            double line_price = ol.product.price * ol.quantity;

            if(ol.product.discount != nullptr)
                line_price *= (1.0 - ol.product.discount->discount_percent(
                    ol.product.price, ol.quantity));

            if(ol.discount != nullptr)
                line_price *= (1.0 - ol.discount->discount_percent(
                    ol.product.price, ol.quantity));

            if(o.buyer != nullptr && o.buyer->discount != nullptr)
                line_price *= (1.0 - o.buyer->discount->discount_percent(
                    ol.product.price, ol.quantity));

            price += line_price;
        }

        if(o.discount != nullptr)
  
```



```

        price *= (1.0 - o.discount->discount_percent(price, 0));

    return price;
}
};

```

В следующем листинге показан пример вычисления окончательной стоимости с помощью `cumulative_price_calculator`:

```

inline bool are_equal(double const d1, double const d2,
                     double const diff = 0.001)
{
    return std::abs(d1 - d2) <= diff;
}

int main()
{
    fixed_discount d1(0.1);
    volume_discount d2(10, 0.15);
    price_discount d3(100, 0.05);
    amount_discount d4(100, 0.05);

    customer c1 {"default", nullptr};
    customer c2 {"john", &d1};
    customer c3 {"joane", &d3};

    article a1 {1, "pen", 5, article_unit::piece, nullptr};
    article a2 {2, "expensive pen", 15, article_unit::piece, &d1};
    article a3 {3, "scissors", 10, article_unit::piece, &d2};

    cumulative_price_calculator calc;

    order o1 {101, &c1, {{a1, 1, nullptr}}, nullptr};
    assert(are_equal(calc.calculate_price(o1), 5));

    order o3 {103, &c1, {{a2, 1, nullptr}}, nullptr};
    assert(are_equal(calc.calculate_price(o3), 13.5));

    order o6 {106, &c1, {{a3, 15, nullptr}}, nullptr};
    assert(are_equal(calc.calculate_price(o6), 127.5));

    order o9 {109, &c3, {{a2, 20, &d1}}, &d4};
    assert(are_equal(calc.calculate_price(o9), 219.3075));
}

```



---

# Глава 9

## Сериализация данных



### Задачи

#### 73. Сериализация и десериализация данных в формате XML

Напишите программу, сохраняющую список фильмов в файл XML и читающую этот список обратно. Для каждого фильма определены: числовой идентификатор, название, год выхода на экраны, продолжительность в минутах, список режиссеров, список сценаристов и список главных ролей с именами актеров. Вот как мог бы выглядеть такой файл XML:

```
<?xml version="1.0"?>
<movies>
  <movie id="9871" title="Forrest Gump" year="1994" length="202">
    <cast>
      <role star="Tom Hanks" name="Forrest Gump" />
      <role star="Sally Field" name="Mrs. Gump" />
      <role star="Robin Wright" name="Jenny Curran" />
      <role star="Mykelti Williamson" name="Bubba Blue" />
    </cast>
    <directors>
      <director name="Robert Zemeckis" />
    </directors>
    <writers>
      <writer name="Winston Groom" />
      <writer name="Eric Roth" />
    </writers>
  </movie>
  <!-- другие элементы movie -->
</movies>
```

#### 74. Выборка данных из XML с помощью XPath

Напишите программу, которая извлекает из файла XML с фильмами, созданного в предыдущей задаче, и выводит в консоль:

- названия всех фильмов, вышедших после указанного года;
- имя последнего актера в списке для каждого фильма.



## 75. Сериализация данных в формат JSON

Напишите программу, сохраняющую список фильмов в файл JSON, как определено в задаче 73. Для каждого фильма определены: числовой идентификатор, название, год выхода на экраны, продолжительность в минутах, список режиссеров, список сценаристов и список главных ролей с именами актеров. Вот как должен выглядеть такой файл JSON:

```
{
  "movies": [{
    "id": 9871,
    "title": "Forrest Gump",
    "year": 1994,
    "length": 202,
    "cast": [{
      "star": "Tom Hanks",
      "name": "Forrest Gump"
    },
    {
      "star": "Sally Field",
      "name": "Mrs. Gump"
    },
    {
      "star": "Robin Wright",
      "name": "Jenny Curran"
    },
    {
      "star": "Mykelti Williamson",
      "name": "Bubba Blue"
    }
  ],
  "directors": ["Robert Zemeckis"],
  "writers": ["Winston Groom", "Eric Roth"]
}]
}
```



## 76. Десериализация данных из формата JSON

Напишите программу, которая извлекает данные из файла JSON с фильмами, созданного в предыдущей задаче.

## 77. Вывод списка фильмов в файл PDF

Напишите программу, которая выводит список фильмов в файл PDF в табличном виде, согласно следующим требованиям:



- список должен быть озаглавлен как «List of movies»; заголовок должен присутствовать только на первой странице документа;
- для каждого фильма должны выводиться: название, год выхода и его продолжительность;
- год выхода должен заключаться в круглые скобки и выравниваться по левому краю;
- продолжительность должна выводиться в часах и минутах (например, 2:12) и выравниваться по правому краю;
- перед и после списка на каждой странице должна выводиться пустая строка.

На рис. 9.1 показан пример вывода списка в формате PDF.

List of movies	
The Matrix (1999)	2:16
Forrest Gump (1994)	2:22
The Truman Show (1998)	1:43
The Pursuit of Happyness (2006)	1:57
Fight Club (1999)	2:19

Рис. 9.1. Пример списка в формате PDF

## 78. Создание документа PDF из коллекции изображений



Напишите программу, создающую документ PDF с изображениями из каталога, указанного пользователем. Изображения должны отображаться друг за другом. Если какое-то изображение не умещается в оставшейся части страницы, программа должна поместить его на следующую страницу.

На рис. 9.2 показан пример такого документа PDF с несколькими фотографиями Альберта Эйнштейна (они входят в состав загружаемого пакета с исходным кодом примеров для книги).

## Решения

### 73. Сериализация и десериализация данных в формате XML

В стандартной библиотеке C++ отсутствует какая-либо поддержка формата XML, зато существует богатый выбор открытых, кроссплатформенных библиотек. Некоторые библиотеки очень простые и поддерживают только базовые особенности XML, другие более сложные и обладают богатой функциональностью. Вам будет из чего выбирать для каждого конкретного проекта.

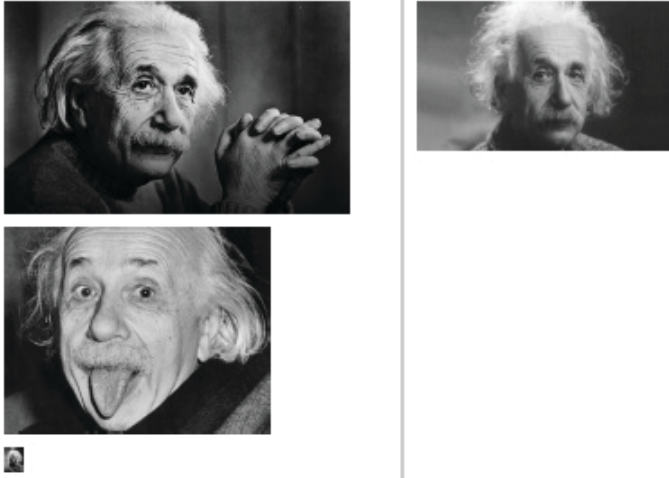


Рис. 9.2. Пример документа PDF с изображениями



В числе библиотек, с которыми вам определенно стоит познакомиться, можно назвать: *Xerces-C++*, *libxml++*, *tinycl* или *tinycl2*, *pugixml*, *gSOAP* и *RapidXml*. Для решения этой задачи я выбрал *pugixml*. Это кроссплатформенная, легковесная библиотека с быстрым парсером XML. Она предлагает DOM-подобный интерфейс с богатыми возможностями навигации по элементам и их изменения, с поддержкой Юникода и XPath 1.0. Из ограничений библиотеки следует упомянуть отсутствие поддержки проверки схем. Получить библиотеку *pugixml* можно по адресу: <https://pugixml.org/>.

Для представления фильмов используем следующие структуры:

```
struct casting_role
{
    std::string actor;
    std::string role;
};

struct movie
{
    unsigned int          id;
    std::string           title;
    unsigned int          year;
    unsigned int          length;
    std::vector<casting_role> cast;
    std::vector<std::string> directors;
    std::vector<std::string> writers;
};

using movie_list = std::vector<movie>;
```

Создание документов XML осуществляется с помощью класса `pugi::xml_document`. После конструирования дерева DOM его можно сохранить в файл вызовом `save_file()`. Узлы добавляются вызовом `append_child()`, а атрибуты – вызовом `append_attribute()`. Следующий метод сериализует список фильмов в требуемый формат:

```
void serialize(movie_list const & movies, std::string_view filepath)
{
    pugi::xml_document doc;
    auto root = doc.append_child("movies");

    for (auto const & m : movies)
    {
        auto movie_node = root.append_child("movie");

        movie_node.append_attribute("id").set_value(m.id);
        movie_node.append_attribute("title").set_value(m.title.c_str());
        movie_node.append_attribute("year").set_value(m.year);
        movie_node.append_attribute("length").set_value(m.length);

        auto cast_node = movie_node.append_child("cast");
        for (auto const & c : m.cast)
        {
            auto node = cast_node.append_child("role");
            node.append_attribute("star").set_value(c.actor.c_str());
            node.append_attribute("name").set_value(c.role.c_str());
        }

        auto directors_node = movie_node.append_child("directors");
        for (auto const & director : m.directors)
        {
            directors_node.append_child("director")
                .append_attribute("name")
                .set_value(director.c_str());
        }

        auto writers_node = movie_node.append_child("writers");
        for (auto const & writer : m.writers)
        {
            writers_node.append_child("writer")
                .append_attribute("name")
                .set_value(writer.c_str());
        }
    }

    doc.save_file(filepath.data());
}
```

Выполнить противоположную операцию – загрузить содержимое файла XML – можно с помощью класса `pugi::xml_document`, вызовом его метода `load_file()`. Доступ к узлам в этом классе осуществляется с помощью методов `child()` и `next_sibling()`, а атрибуты доступны посредством метода `attribute()`. Метод `deserialize()`, показанный ниже, читает дерево DOM и конструирует список фильмов:

```

movie_list deserialize(std::string_view filepath)
{
    pugi::xml_document doc;
    movie_list movies;

    auto result = doc.load_file(filepath.data());
    if (result)
    {
        auto root = doc.child("movies");
        for (auto movie_node = root.child("movie");
            movie_node;
            movie_node = movie_node.next_sibling("movie"))
        {
            movie m;
            m.id = movie_node.attribute("id").as_uint();
            m.title = movie_node.attribute("title").as_string();
            m.year = movie_node.attribute("year").as_uint();
            m.length = movie_node.attribute("length").as_uint();

            for (auto role_node :
                movie_node.child("cast").children("role"))
            {
                m.cast.push_back(casting_role{
                    role_node.attribute("star").as_string(),
                    role_node.attribute("name").as_string() });
            }

            for (auto director_node :
                movie_node.child("directors").children("director"))
            {
                m.directors.push_back(
                    director_node.attribute("name").as_string());
            }

            for (auto writer_node :
                movie_node.child("writers").children("writer"))
            {
                m.writers.push_back(
                    writer_node.attribute("name").as_string());
            }

            movies.push_back(m);
        }
    }
}

```



```

    }
}

return movies;
}

```



Пример использования этих функций показан в следующем листинге:

```

int main()
{
    movie_list movies
    {
        {
            11001, "The Matrix", 1999, 196,
            { {"Keanu Reeves", "Neo"},
              {"Laurence Fishburne", "Morpheus"},
              {"Carrie-Anne Moss", "Trinity"},
              {"Hugo Weaving", "Agent Smith"} },
            {"Lana Wachowski", "Lilly Wachowski"},
            {"Lana Wachowski", "Lilly Wachowski"},
        },
        {
            9871, "Forrest Gump", 1994, 202,
            { {"Tom Hanks", "Forrest Gump"},
              {"Sally Field", "Mrs. Gump"},
              {"Robin Wright", "Jenny Curran"},
              {"Mykelti Williamson", "Bubba Blue"} },
            {"Robert Zemeckis"},
            {"Winston Groom", "Eric Roth"},
        }
    };

    serialize(movies, "movies.xml");
    auto result = deserialize("movies.xml");

    assert(result.size() == 2);
    assert(result[0].title == "The Matrix");
    assert(result[1].title == "Forrest Gump");
}

```



## 74. Выборка данных из XML с помощью XPath

Навигацию по элементам и атрибутам файла XML можно осуществлять с помощью выражений *XPath*, предназначенных именно для этой цели. Библиотека *pugixml* поддерживает выражения XPath, для выполнения которых можно использовать метод `select_nodes()` класса `xml_document`. Обратите внимание, что если в процессе выполнения выражения XPath произойдет ошибка, библиотека возбудит исключение `xpath_exception`.



Для выбора узлов, в соответствии с требованиями задачи, можно использовать следующие выражения XPath:

- для выбора фильмов, вышедших после указанного года (в этом примере выбран 1995 год):  
/movies/movie[@year>1995];
- для выбора последнего актера в списке для каждого фильма:  
/movies/movie/cast/role[last()]

Вот как определен документ XML:

```
std::string text = R"(
<?xml version="1.0"?>
<movies>
  <movie id="11001" title="The Matrix" year="1999" length="196">
    <cast>
      <role star="Keanu Reeves" name="Neo" />
      <role star="Laurence Fishburne" name="Morpheus" />
      <role star="Carrie-Anne Moss" name="Trinity" />
      <role star="Hugo Weaving" name="Agent Smith" />
    </cast>
    <directors>
      <director name="Lana Wachowski" />
      <director name="Lilly Wachowski" />
    </directors>
    <writers>
      <writer name="Lana Wachowski" />
      <writer name="Lilly Wachowski" />
    </writers>
  </movie>
  <movie id="9871" title="Forrest Gump" year="1994" length="202">
    <cast>
      <role star="Tom Hanks" name="Forrest Gump" />
      <role star="Sally Field" name="Mrs. Gump" />
      <role star="Robin Wright" name="Jenny Curran" />
      <role star="Mykelti Williamson" name="Bubba Blue" />
    </cast>
    <directors>
      <director name="Robert Zemeckis" />
    </directors>
    <writers>
      <writer name="Winston Groom" />
      <writer name="Eric Roth" />
    </writers>
  </movie>
</movies>
)";
```

Код в следующем листинге загружает документ XML из строкового буфера и затем выбирает узлы, используя выражения XPath, перечисленные выше.

```

pugi::xml_document doc;
if (doc.load_string(text.c_str()))
{
    try
    {
        auto titles = doc.select_nodes("/movies/movie[@year>1995]");

        for (auto it : titles)
        {
            std::cout << it.node().attribute("title").as_string()
                << std::endl;
        }
    }
    catch (pugi::xpath_exception const & e)
    {
        std::cout << e.result().description() << std::endl;
    }

    try
    {
        auto titles = doc.select_nodes("/movies/movie/cast/role[last()]");

        for (auto it : titles)
        {
            std::cout << it.node().attribute("star").as_string()
                << std::endl;
        }
    }
    catch (pugi::xpath_exception const & e)
    {
        std::cout << e.result().description() << std::endl;
    }
}

```

## 75. Сериализация данных в формат JSON

Так же как в случае с XML, в стандартной библиотеке C++ отсутствует поддержка формата JSON. Однако имеется огромное количество кроссплатформенных библиотек. На момент написания этих строк в проекте *nativejson-benchmark* (<https://github.com/miloyip/nativejson-benchmark>) было перечислено более 40 таких библиотек.

Цель этого проекта – оценка удобства и производительности (скорость работы, потребление памяти и объем кода) открытых библиотек на C/C++, реализующих средства создания/чтения данных в формате JSON. Такое раз-



нообразии немного затрудняет выбор, тем не менее в числе главных претендентов можно назвать: *RapidJSON*, *Nlohmann*, *taocpp/json*, *Configuru*, *json\_spirit*, *jsoncpp*. Для решения этой задачи я выбрал библиотеку *nlohmann/json*. Эта кроссплатформенная библиотека для C++11 состоит только из заголовков, имеет простой и понятный синтаксис и хорошую документацию. Получить библиотеку можно по адресу: <https://github.com/nlohmann/json>.

Для представления фильмов в этом решении мы будем использовать те же структуры данных, которые применялись в задаче «Сериализация и десериализация данных в формате XML». Библиотека *nlohmann* предоставляет `nlohmann::json` как главный тип данных для представления объектов JSON. Значения JSON можно создавать явно, но есть также возможность использовать неявные преобразования скалярных типов и типов контейнеров. Кроме того, есть возможность реализовать поддержку неявных преобразований для пользовательских типов, добавив методы `to_json()` и `from_json()` в пространство имен преобразуемого типа. К этим функциям предъявляются определенные требования, с которыми вы можете познакомиться в документации.

Следующий код демонстрирует именно такой подход. Поскольку типы `movie` и `casting_role` определены в глобальном пространстве имен, перегруженные версии `to_json()` для их сериализации также определяются в глобальном пространстве имен. С другой стороны, тип `movie_list` фактически является псевдонимом типа `std::vector<movie>` и может сериализоваться и десериализоваться непосредственно, потому что, как отмечалось выше, библиотека поддерживает неявные преобразования для стандартных контейнеров:

```
using json = nlohmann::json;

void to_json(json& j, casting_role const & c)
{
    j = json{ {"star", c.actor}, {"name", c.role} };
}

void to_json(json& j, movie const & m)
{
    j = json::object({
        {"id", m.id},
        {"title", m.title},
        {"year", m.year},
        {"length", m.length},
        {"cast", m.cast },
        {"directors", m.directors},
        {"writers", m.writers}
    });
}

void serialize(movie_list const & movies, std::string_view filepath)
```

```

{
    json jdata{ { "movies", movies } };

    std::ofstream ofile(filepath.data());
    if (ofile.is_open())
    {
        ofile << std::setw(2) << jdata << std::endl;
    }
}

```

Функцию `serialize()` можно использовать, как показано в следующем примере:

```

int main()
{
    movie_list movies
    {
        {
            11001, "The Matrix", 1999, 196,
            { {"Keanu Reeves", "Neo"},
              {"Laurence Fishburne", "Morpheus"},
              {"Carrie-Anne Moss", "Trinity"},
              {"Hugo Weaving", "Agent Smith"} },
            {"Lana Wachowski", "Lilly Wachowski"},
            {"Lana Wachowski", "Lilly Wachowski"},
        },
        {
            9871, "Forrest Gump", 1994, 202,
            { {"Tom Hanks", "Forrest Gump"},
              {"Sally Field", "Mrs. Gump"},
              {"Robin Wright", "Jenny Curran"},
              {"Mykelti Williamson", "Bubba Blue"} },
            {"Robert Zemeckis"},
            {"Winston Groom", "Eric Roth"},
        }
    };

    serialize(movies, "movies.json");
}

```



## 76. Десериализация данных из формата JSON

Для решения этой задачи мы снова воспользуемся библиотекой *nlohmann/json*. Но вместо создания функций `from_json()`, как в предыдущем решении, мы предпримем более явный подход. Содержимое файла в формате JSON можно загрузить в объект `nlohmann::json` с помощью перегруженного оператора `>>`. Для доступа к значениям в объект используем метод `at()` вместо `operator[]`, по-

тому что первый возбуждает исключение (которое можно обработать), если указанный ключ не существует, тогда как поведение второго в этом случае не определено. Чтобы получить значение определенного типа `T`, используем метод `get<T>()`. Однако для этого необходимо, чтобы тип `T` имел конструктор по умолчанию.

Функция `deserialize()`, показанная ниже, возвращает вектор `std::vector<movie>`, сконструированный из содержимого заданного файла JSON:

```
using json = nlohmann::json;

movie_list deserialize(std::string_view filepath)
{
    movie_list movies;

    std::ifstream ifile(filepath.data());
    if (ifile.is_open())
    {
        json jdata;

        try
        {
            ifile >> jdata;

            if (jdata.is_object())
            {
                for (auto & element : jdata.at("movies"))
                {
                    movie m;

                    m.id = element.at("id").get<unsigned int>();
                    m.title = element.at("title").get<std::string>();
                    m.year = element.at("year").get<unsigned int>();
                    m.length = element.at("length").get<unsigned int>();

                    for (auto & role : element.at("cast"))
                    {
                        m.cast.push_back(casting_role{
                            role.at("star").get<std::string>(),
                            role.at("name").get<std::string>() });
                    }

                    for (auto & director : element.at("directors"))
                    {
                        m.directors.push_back(director);
                    }

                    for (auto & writer : element.at("writers"))
```



```

        {
            m.writers.push_back(writer);
        }

        movies.push_back(m);
    }
}
}
catch (std::exception const & ex)
{
    std::cout << ex.what() << std::endl;
}
}

return movies;
}

```



Следующий листинг демонстрирует применение функции десериализации:

```

int main()
{
    auto movies = deserialize("movies.json");

    assert(movies.size() == 2);
    assert(movies[0].title == "The Matrix");
    assert(movies[1].title == "Forrest Gump");
}

```



## 77. Вывод списка фильмов в файл PDF

Для C++ есть несколько библиотек для работы с файлами PDF. В числе примеров открытых и кроссплатформенных библиотек можно назвать: *HaHu*, *PoDoFo*, *JagPDF* и *PDF-Writer* (также известная под названием *Hummus*). В этой книге я буду использовать библиотеку *PDF-Writer*, которая доступна по адресу: <https://github.com/galkahana/PDF-Writer>. Это бесплатная, быстрая и расширяемая библиотека с базовым набором возможностей, включающих поддержку текста, изображений и геометрических фигур в виде операторов PDF и высокоуровневых функций (которые я и буду использовать в решении этой задачи).

Функция `print_pdf()`, показанная в листинге ниже, реализует следующий алгоритм:

- создается новый документ PDF вызовом `PDFWriter::StartPDF()`;
- на каждую страницу выводится до 25 названий фильмов; каждая страница представлена объектом `PDFPage` и имеет объект `PageContentContext`, который создается вызовом `PDFPage::StartPageContentContext()` и используется для рисования элементов на странице;

- на первую страницу выводится заголовок «List of movies»; вывод текста на страницу осуществляется вызовом `PageContentContext::WriteText()`;
- информация о фильмах выводится другим шрифтом;
- на каждой странице перед списком и после него рисуются линии с помощью `PageContentContext::DrawPath()`;
- по завершении вывода на страницу вызываются функции `PDFWriter::EndPageContentContext()` и `PDFWriter::WritePageAndRelease()`;
- по завершении вывода в документ PDF вызывается `PDFWriter::EndPDF()`.



За информацией о типах и методах, использованных в решении, а также за дополнительными сведениями о создании документов PDF и работе с текстом, фигурами и изображениями обращайтесь к документации с описанием проекта по адресу: <https://github.com/galkahana/PDF-Writer/wiki>.

```

#ifdef _WIN32
static const std::string fonts_dir = R"(c:\windows\fonts\)";
#elif defined (__APPLE__)
static const std::string fonts_dir = R"(/Library/Fonts/)";
#else
static const std::string fonts_dir = R"(/usr/share/fonts)";
#endif

void print_pdf(movie_list const & movies,
              std::string_view path)
{
    const int height = 842;
    const int width = 595;
    const int left = 60;
    const int top = 770;
    const int right = 535;
    const int bottom = 60;
    const int line_height = 28;

    PDFWriter pdf;
    pdf.StartPDF(path.data(), ePDFVersion13);
    auto font = pdf.GetFontForFile(fonts_dir + "arial.ttf");

    AbstractContentContext::GraphicOptions pathStrokeOptions(
        AbstractContentContext::eStroke,
        AbstractContentContext::eRGB,
        0xff000000,
        1);

    PDFPage* page = nullptr;

```

```

PageContentContext* context = nullptr;
int index = 0;
for (size_t i = 0; i < movies.size(); ++i)
{
    index = i % 25;
    if (index == 0)
    {
        if (page != nullptr)
        {
            DoubleAndDoublePairList pathPoints;
            pathPoints.push_back(DoubleAndDoublePair(left, bottom));
            pathPoints.push_back(DoubleAndDoublePair(right, bottom));
            context->DrawPath(pathPoints, pathStrokeOptions);

            pdf.EndPageContentContext(context);
            pdf.WritePageAndRelease(page);
        }

        page = new PDFPage();
        page->SetMediaBox(PDFRectangle(0, 0, width, height));
        context = pdf.StartPageContentContext(page);

        {
            DoubleAndDoublePairList pathPoints;
            pathPoints.push_back(DoubleAndDoublePair(left, top));
            pathPoints.push_back(DoubleAndDoublePair(right, top));
            context->DrawPath(pathPoints, pathStrokeOptions);
        }
    }

    if (i == 0)
    {
        AbstractContentContext::TextOptions const textOptions(
            font, 26, AbstractContentContext::eGray, 0);
        context->WriteText(left, top + 15,
            "List of movies", textOptions);
    }

    auto textw = 0;
    {
        AbstractContentContext::TextOptions const textOptions(
            font, 20, AbstractContentContext::eGray, 0);

        context->WriteText(left, top - 20 - line_height * index,
            movies[i].title, textOptions);
        auto textDimensions = font->CalculateTextDimensions(
            movies[i].title, 20);
        textw = textDimensions.width;
    }
}

```







```

    }

    {
        AbstractContentContext::TextOptions const textOptions(
            font, 16, AbstractContentContext::eGray, 0);

        context->WriteText(left + textw + 5,
                          top - 20 - line_height * index,
                          " (" + std::to_string(movies[i].year) + ")",
                          textOptions);

        std::stringstream s;
        s << movies[i].length / 60 << ':' << std::setw(2)
          << std::setfill('0') << movies[i].length % 60;

        context->WriteText(right - 30, top - 20 - line_height * index,
                          s.str(),
                          textOptions);
    }
}

DoubleAndDoublePairList pathPoints;
pathPoints.push_back(
    DoubleAndDoublePair(left, top - line_height * (index + 1)));
pathPoints.push_back(
    DoubleAndDoublePair(right, top - line_height * (index + 1)));
context->DrawPath(pathPoints, pathStrokeOptions);

if (page != nullptr)
{
    pdf.EndPageContentContext(context);
    pdf.WritePageAndRelease(page);
}

pdf.EndPDF();
}

```

Далее демонстрируется пример использования функции `print_pdf()`:

```

int main()
{
    movie_list movies
    {
        { 1, "The Matrix", 1999, 136},
        { 2, "Forrest Gump", 1994, 142},
        // .. другие фильмы
        { 28, "L.A. Confidential", 1997, 138},
        { 29, "Shutter Island", 2010, 138},
    }
}

```

```
};

print_pdf(movies, "movies.pdf");
}
```



## 78. Создание документа PDF из коллекции изображений

Для решения этой задачи используем ту же библиотеку *PDF-Writer*, что применялась в предыдущем решении. Я советую внимательно изучить предыдущую реализацию, если вы этого еще не сделали, и только потом продолжить знакомство с этой реализацией.

Следующая функция `get_images()` возвращает вектор строк с именами файлов изображений в формате JPG из указанного каталога:

```
namespace fs = std::experimental::filesystem;

std::vector<std::string> get_images(fs::path const & dirpath)
{
    std::vector<std::string> paths;

    for (auto const & p : fs::directory_iterator(dirpath))
    {
        if (p.path().extension() == ".jpg")
            paths.push_back(p.path().string());
    }

    return paths;
}
```



Функция `print_pdf()` создает документ PDF и помещает в него изображения JPG из заданного каталога. Она реализует следующий алгоритм:

- создается новый документ PDF вызовом `PDFWriter::StartPDF()`;
- создается страница с содержимым, и в нее вставляется столько изображений, сколько поместится, с выравниванием по вертикали;
- когда следующее изображение не помещается на странице, текущая страница закрывается вызовом `PDFWriter::EndPageContentContext()` и `PDFWriter::SavePageAndRelease()`, после чего создается новая страница;
- добавление изображений на страницу осуществляется вызовом `PageContentContext::DrawImage()`;
- по завершении вывода в документ PDF вызывается `PDFWriter::EndPDF()`.

```
void print_pdf(fs::path const & pdfpath,
              fs::path const & dirpath)
```

```
{
    const int height = 842;
    const int width = 595;
    const int margin = 20;

    auto image_paths = get_images(dirpath);

    PDFWriter pdf;
    pdf.StartPDF(pdfpath.string(), ePDFVersion13);

    PDFPage* page = nullptr;
    PageContentContext* context = nullptr;

    auto top = height - margin;
    for (size_t i = 0; i < image_paths.size(); ++i)
    {
        auto dims = pdf.GetImageDimensions(image_paths[i]);

        if (i == 0 || top - dims.second < margin)
        {
            if (page != nullptr)
            {
                pdf.EndPageContentContext(context);
                pdf.WritePageAndRelease(page);
            }

            page = new PDFPage();
            page->SetMediaBox(PDFRectangle(0, 0, width, height));
            context = pdf.StartPageContentContext(page);
            top = height - margin;
        }

        context->DrawImage(margin, top - dims.second, image_paths[i]);

        top -= dims.second + margin;
    }

    if (page != nullptr)
    {
        pdf.EndPageContentContext(context);
        pdf.WritePageAndRelease(page);
    }

    pdf.EndPDF();
}
```



Следующий пример иллюстрирует использование функции `print_pdf()`. Здесь `sample.pdf` – это имя файла документа PDF, а `res` – имя каталога с изображениями:

```
int main()
{
    print_pdf("sample.pdf", "res");
}
```



---

# Глава 10

## Архивы, изображения и базы данных



### Задачи

#### 79. Поиск файлов в архиве ZIP

Напишите программу, которая отыщет и выведет имена всех файлов в архиве ZIP, соответствующие регулярному выражению, заданному пользователем (например, `^.*\.*jpg$` для поиска всех файлов с расширением `.jpg`).

#### 80. Упаковка и извлечение файлов из архива ZIP



Напишите программу, выполняющую следующие действия:

- рекурсивно упаковывает в архив ZIP все файлы и подкаталоги из указанного каталога;
- извлекает содержимое архива ZIP в указанный каталог.

#### 81. Упаковка и извлечение файлов из архива ZIP с защитой паролем

Напишите программу, выполняющую следующие действия:

- рекурсивно упаковывает в архив ZIP, защищенный паролем, все файлы и подкаталоги из указанного каталога;
- извлекает содержимое архива ZIP, защищенного паролем, в указанный каталог.

#### 82. Создание файла PNG с изображением национального флага

Напишите программу, генерирующую файл PNG с изображением национального флага Румынии, показанного на рис. 10.1. Размер изображения в пикселях, а также путь к создаваемому файлу должны указываться пользователем.



Рис. 10.1. Национальный флаг Румынии

### 83. Создание изображения PNG с контрольным текстом

Напишите программу, создающую Captcha-подобные изображения PNG для защиты от ботов. Изображения должны иметь:

- цветной градиентный фон;
- последовательность случайно выбранных символов, изображенных с поворотом вправо и влево на разные углы;
- несколько случайных линий разного цвета, пересекающих изображение (поверх текста).

Пример такого изображения показан на рис. 10.2.

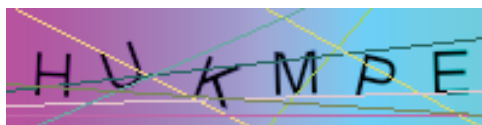


Рис. 10.2. Пример Captcha-подобного изображения

### 84. Генератор штрихкодов EAN-13

Напишите программу, генерирующую изображение PNG со штрихкодом EAN-13 для любого международного номера товара версии 13 стандарта. Для простоты изображение должно содержать только штрихкод, без номера EAN-13 под ним. На рис. 10.3 показан пример такого штрихкода для номера 5901234123457:



Рис. 10.3. Пример штрихкода для номера 5901234123457

## 85. Чтение информации о фильмах из базы данных SQLite



Напишите программу, которая читала бы информацию о фильмах из базы данных SQLite и выводила ее в консоль. Для каждого фильма должны выводиться: числовой идентификатор, название, год выхода, продолжительность в минутах, список режиссеров, список сценаристов и список ролей с именами актеров. На рис. 10.4 показана диаграмма со схемой базы данных для этой задачи.

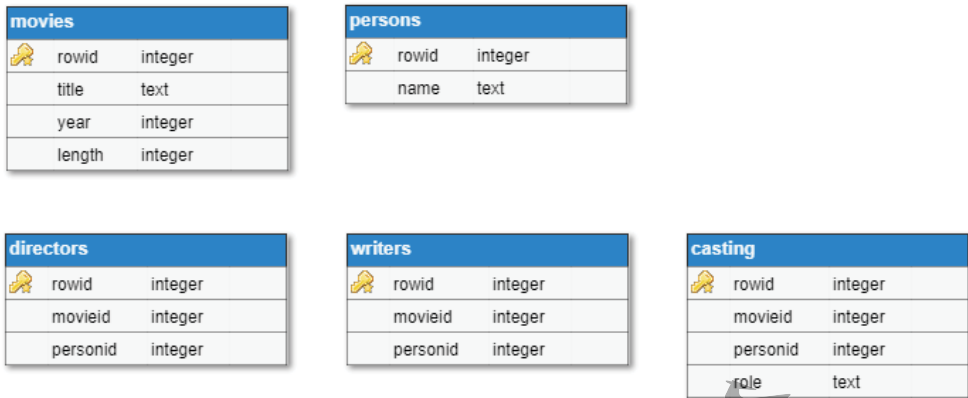


Рис. 10.4. Схема базы данных со списком фильмов

## 86. Добавление информации о фильмах в базу данных SQLite

Добавьте в программу из предыдущей задачи возможность добавления информации о фильмах. Информация должна приниматься с клавиатуры или из текстового файла. Вставка в таблицы базы данных должна выполняться в единой транзакции.

## 87. Обработка изображений для фильмов в базе данных SQLite

Измените программу, написанную для решения предыдущей задачи, и реализуйте возможность добавления медиафайлов (например, изображений или видеороликов) для фильмов. Эти файлы должны сохраняться в отдельной таблице в виде двоичных блоков blob, иметь уникальные числовые идентификаторы, имена (обычно имена файлов), ссылки на идентификаторы соответствующих фильмов и необязательные описания. На рис. 10.5 показана схема таблицы, которую нужно добавить в базу данных.


media	
 rowid	integer
movieid	integer
name	text
description	text
content	blob



Рис. 10.5. Таблица в базе данных для хранения медиафайлов

Программа с решением этой задачи должна поддерживать:

- вывод списка всех фильмов, соответствующих критерию поиска (например, по названию);
- вывод информации обо всех медиафайлах, соответствующих указанному фильму;
- добавление нового медиафайла в базу данных для заданного фильма;
- удаление существующего медиафайла из базы данных.

## Решения

### 79. Поиск файлов в архиве ZIP



Существует множество библиотек, реализующих поддержку архивов ZIP. В числе бесплатных особой популярностью пользуются следующие: *ZipLib*, *Info-Zip*, *MiniZip* и *LZMA SDK* из проекта *7z*.

Имеются также коммерческие реализации. В решениях задач, связанных с использованием архивов ZIP, я решил использовать библиотеку *ZipLib*. Эта легковесная, открытая и кроссплатформенная библиотека, соответствующая стандарту C++11, основана на использовании поддержки потоков данных в стандартной библиотеке и не имеет никаких других зависимостей. Библиотеку и документацию к ней можно найти по адресу: <https://bitbucket.org/wbenny/ziplib>.

Программа, реализующая решение этой задачи, должна:

- открыть архив ZIP с помощью `ZipFile::Open()`;
- выполнить итерации по всем элементам в архиве с помощью `ZipArchive::GetEntry()` и `ZipArchive::GetEntryCount()`;
- выполнить итерации по всем элементам в архиве и проверить совпадение их имен с заданным регулярным выражением с помощью `ZipArchiveEntry::GetName()`;



- для всех элементов, соответствующих регулярному выражению, добавить полное имя в список результатов с помощью `ZipArchiveEntry::GetFullName()`.

Описанный алгоритм реализует функция `find_in_archive()`, представленная ниже:

```
namespace fs = std::experimental::filesystem;

std::vector<std::string> find_in_archive(
    fs::path const & archivepath,
    std::string_view pattern)
{
    std::vector<std::string> results;

    if (fs::exists(archivepath))
    {
        try
        {
            auto archive = ZipFile::Open(archivepath.string());

            for (size_t i = 0; i < archive->GetEntriesCount(); ++i)
            {
                auto entry = archive->GetEntry(i);
                if (entry)
                {
                    if (!entry->IsDirectory())
                    {
                        auto name = entry->GetName();
                        if (std::regex_match(name,
                            std::regex{ pattern.data() }))
                        {
                            results.push_back(entry->GetFullName());
                        }
                    }
                }
            }
        }
        catch (std::exception const & ex)
        {
            std::cout << ex.what() << std::endl;
        }
    }

    return results;
}
```





```

}
else
{
    for (auto const & p : fs::recursive_directory_iterator(source))
    {
        if (reporter) reporter("Compressing " + p.path().string());

        if (fs::is_directory(p))
        {
            auto zipArchive = ZipFile::Open(archive.string());
            auto entry = zipArchive->CreateEntry(p.path().string());
            entry->SetAttributes(ZipArchiveEntry::Attributes::Directory);
            ZipFile::SaveAndClose(zipArchive, archive.string());
        }
        else if (fs::is_regular_file(p))
        {
            ZipFile::AddFile(archive.string(), p.path().string(),
                LzmaMethod::Create());
        }
    }
}
}
}
}

```

Для реализации противоположной операции, распаковки, нужно:

- открыть архив ZIP с помощью `ZipFile::Open()`;
- выполнить обход всех элементов в архиве вызовом `ZipArchive::GetEntriesCount()` и `ZipArchive::GetEntry()`;
- если элемент – каталог, создать его рекурсивно в целевом каталоге;
- если элемент – обычный файл, создать соответствующий файл в целевом каталоге и скопировать в него содержимое сжатого файла вызовом `ZipArchiveEntry::GetDecompressionStream()`.

Описанный алгоритм реализует функция `decompress()`, показанная в следующем листинге. Она также принимает три аргумента: первый – путь к целевому каталогу, второй – путь к архиву ZIP, который требуется распаковать, и третий – объект функции для передачи информации о процессе выполнения операции:

```

void decompress(fs::path const & destination,
               fs::path const & archive,
               std::function<void(std::string_view)> reporter)
{
    ensure_directory_exists(destination);

    auto zipArchive = ZipFile::Open(archive.string());

    for (int i = 0; i < zipArchive->GetEntriesCount(); ++i)

```

```

{
  auto entry = zipArchive->GetEntry(i);
  if (entry)
  {
    auto filepath = destination / fs::path{
      entry->GetFullName() }.relative_path();
    if (reporter) reporter("Creating " + filepath.string());

    if (entry->IsDirectory())
    {
      ensure_directory_exists(filepath);
    }
    else
    {
      ensure_directory_exists(filepath.parent_path());
      std::ofstream destFile;
      destFile.open(filepath.string().c_str(),
                    std::ios::binary | std::ios::trunc);

      if (!destFile.is_open())
      {
        if (reporter)
          reporter("Cannot create destination file!");
      }

      auto dataStream = entry->GetDecompressionStream();
      if (dataStream)
      {
        utils::stream::copy(*dataStream, destFile);
      }
    }
  }
}
}
}

```

Эта функция использует `ensure_directory_exists()`, чтобы создать дерево каталогов, если оно еще не создано. Вот как выглядит ее реализация:

```

void ensure_directory_exists(fs::path const & dir)
{
  if (!fs::exists(dir))
  {
    std::error_code err;
    fs::create_directories(dir, err);
  }
}

```

Следующая программа дает пользователю возможность ввести команду для выполнения, сжатия или распаковки, а также исходный путь и путь к каталогу

назначения. Она вызывает функции `compress()` и `decompress()`, показанные выше, и передает им лямбда-функции для вывода информации о ходе выполнения операции в консоль:

```
int main()
{
    char option = 0;
    std::cout << "Select [c]ompress/[d]ecompress?";
    std::cin >> option;

    if (option == 'c')
    {
        std::string archivepath;
        std::string inputpath;
        std::cout << "Enter file or dir to compress:";
        std::cin >> inputpath;
        std::cout << "Enter archive path:";
        std::cin >> archivepath;

        compress(inputpath, archivepath,
                [](std::string_view message) {
                    std::cout << message << std::endl; });
    }
    else if (option == 'd')
    {
        std::string archivepath;
        std::string outputpath;
        std::cout << "Enter dir to decompress:";
        std::cin >> outputpath;
        std::cout << "Enter archive path:";
        std::cin >> archivepath;

        decompress(outputpath, archivepath,
                [](std::string_view message) {
                    std::cout << message << std::endl; });
    }
    else
    {
        std::cout << "invalid option" << std::endl;
    }
}
```



## 81. Упаковка и извлечение файлов из архива ZIP с защитой паролем

Эта задача очень похожа на предыдущую, с одним дополнением: файлы должны шифроваться. Библиотека *ZipLib* поддерживает только шифрование

*PKWare*. Если вам понадобится использовать другой алгоритм шифрования, используйте другую библиотеку. Функции `compress()` и `decompress()`, показанные ниже, имеют реализации, похожие на одноименные функции в предыдущей задаче, но принимают по одному дополнительному аргументу, представляющему пароль для шифрования/расшифровывания файлов:

- добавление зашифрованных файлов в архив выполняется с помощью `ZipFile::AddEncryptedFile()` вместо `ZipFile::AddFile()`;
- при распаковывании, если элемент защищен паролем, пароль нужно установить вызовом `ZipArchiveEntry::SetPassword()`.

Вот как выглядит реализация функции `compress()` с вышеупомянутыми изменениями:

```
namespace fs = std::experimental::filesystem;

void compress(fs::path const & source,
             fs::path const & archive,
             std::string_view password,
             std::function<void(std::string_view)> reporter)
{
    if (fs::is_regular_file(source))
    {
        if (reporter) reporter("Compressing " + source.string());
        ZipFile::AddEncryptedFile(
            archive.string(),
            source.string(),
            source.filename().string(),
            password.data(),
            LzmaMethod::Create());
    }
    else
    {
        for (auto const & p : fs::recursive_directory_iterator(source))
        {
            if (reporter) reporter("Compressing " + p.path().string());

            if (fs::is_directory(p))
            {
                auto zipArchive = ZipFile::Open(archive.string());
                auto entry = zipArchive->CreateEntry(p.path().string());
                entry->SetAttributes(ZipArchiveEntry::Attributes::Directory);
                ZipFile::SaveAndClose(zipArchive, archive.string());
            }
            else if (fs::is_regular_file(p))
            {
                ZipFile::AddEncryptedFile(
                    archive.string(),
```



```

        p.path().string(),
        p.path().filename().string(),
        password.data(),
        LzmaMethod::Create());
    }
}
}
}

```



Как уже упоминалось, функция `decompress()` должна установить пароль перед копированием зашифрованного содержимого файла. Ее реализация приводится ниже:

```

void decompress(fs::path const & destination,
               fs::path const & archive,
               std::string_view password,
               std::function<void(std::string_view)> reporter)
{
    ensure_directory_exists(destination);

    auto zipArchive = ZipFile::Open(archive.string());

    for (size_t i = 0; i < zipArchive->GetEntriesCount(); ++i)
    {
        auto entry = zipArchive->GetEntry(i);
        if (entry)
        {
            auto filepath = destination / fs::path{
                entry->GetFullName() }.relative_path();
            if (reporter) reporter("Creating " + filepath.string());

            if(entry->IsPasswordProtected())
                entry->SetPassword(password.data());

            if (entry->IsDirectory())
            {
                ensure_directory_exists(filepath);
            }
            else
            {
                ensure_directory_exists(filepath.parent_path());

                std::ofstream destFile;
                destFile.open(filepath.string().c_str(),
                             std::ios::binary | std::ios::trunc);

                if (!destFile.is_open())
                {

```



```
        if (reporter)
            reporter("Cannot create destination file!");
    }

    auto dataStream = entry->GetDecompressionStream();
    if (dataStream)
    {
        utils::stream::copy(*dataStream, destFile);
    }
}
}
}
}
```

Функция `ensure_directory_exists()` идентична одноименной функции в предыдущей задаче и здесь не повторяется.

Пользоваться этими функциями можно точно так же, как показано в предыдущей задаче, с той лишь разницей, что вы должны передать пароль.

## 82. Создание файла PNG с изображением национального флага

Наиболее широкими возможностями для работы с файлами PNG обладает библиотека *libpng* – кроссплатформенная библиотека с открытым исходным кодом, написанная на языке C. Существуют также библиотеки на C++, некоторые из которых являются простыми обертками вокруг *libpng*, такие как *png++*, *lodepng* и *PNGWriter*. Для задач в этой книге я выбрал последнюю из них, *PNGWriter*. Это открытая библиотека, работающая в Linux, Unix, macOS и Windows. Она поддерживает: открытие файлов изображений в формате PNG; чтение и вывод пикселей в форматах RGB, HSV и CMYK; рисование простых геометрических фигур; масштабирование; билинейную интерполяцию; рисование текста шрифтами TrueType со сглаживанием и его поворот; а также рисование кривых Безье. Это обертка для библиотеки *libpng* и требует наличия библиотеки *FreeType2* для поддержки рисования текста.

Исходный код библиотеки и документацию к ней можно найти по адресу: <https://github.com/pngwriter/pngwriter>. Установите библиотеку, следуя инструкциям в документации.

Класс `pngwriter` представляет изображение PNG. Его конструктор позволяет указать ширину и высоту в пикселях, цвет фона и путь к файлу, куда следует сохранить изображение. Класс имеет богатое разнообразие функций-членов для рисования пикселей, фигур и текста. Для решения этой задачи нам понадобится нарисовать три прямоугольника и залить их разными цветами. Для этого можно воспользоваться функцией `filledsquare()`. Завершив формирование изображения в памяти, нужно вызвать метод `close()`, чтобы сохранить его в файл.



Следующая функция создает изображение трехцветного флага с размерами, указанными в аргументах, и сохраняет его в указанном файле:

```
void create_flag(int const width, int const height,
                std::string_view filepath)
{
    pngwriter flag { width, height, 0, filepath.data() };

    int const size = width / 3;
    // красный прямоугольник
    flag.filledsquare(0, 0, size, 2 * size, 65535, 0, 0);
    // желтый прямоугольник
    flag.filledsquare(size, 0, 2 * size, 2 * size, 65535, 65535, 0);
    // синий прямоугольник
    flag.filledsquare(2 * size, 0, 3 * size, 2 * size, 0, 0, 65535);

    flag.close();
}
```



Следующая программа позволяет пользователю указать ширину и высоту изображения, а также путь к файлу для сохранения, и вызывает `create_flag()`, чтобы сгенерировать изображение PNG:

```
int main()
{
    int width = 0, height = 0;
    std::string filepath;

    std::cout << "Width: ";
    std::cin >> width;

    std::cout << "Heigh: ";
    std::cin >> height;

    std::cout << "Output: ";
    std::cin >> filepath;

    create_flag(width, height, filepath);
}
```

### 83. Создание изображения PNG с контрольным текстом

Эта задача решается аналогично предыдущей, где создавался национальный флаг. Если вы еще не исследовали ее, я советую сделать это прямо сейчас и только потом продолжать чтение.

Согласно определению задачи, изображение должно включать три элемента:

- фон с цветной градиентной заливкой. Этого можно добиться рисованием линий (вертикальных или горизонтальных), изменяя цвет от одной стороны изображения к другой. Рисовать линии можно с помощью функции `pngwriter::line()`. Существует несколько перегруженных версий этой функции; одна, используется в следующем коде, принимает начальную и конечную позиции и три значения каналов цвета – красный, зеленый и синий – RGB;
- последовательность случайно выбранных символов, изображенных с поворотом вправо и влево на разные углы. Рисование текста можно выполнить с помощью функции `pngwriter::plot_text()`. Для ее вызова требуется наличие в системе библиотеки *FreeType2*. Перегруженная версия, используемая в этом решении, позволяет указать файл шрифта и размер, позицию вывода текста, угол поворота в радианах, сам текст и цвет;
- случайные линии разного цвета, пересекающие изображение поверх текста. И снова линии можно нарисовать с помощью `pngwriter::line()`.

Для получения случайных символов, цветов и позиций линий в данном решении используется генератор псевдослучайных чисел `std::mt19937` и несколько целочисленных равномерных распределений:

```
void create_image(int const width, int const height,
                 std::string_view font, int const font_size,
                 std::string_view filepath)
{
    pngwriter image { width, height, 0, filepath.data() };

    std::random_device rd;
    std::mt19937 mt;
    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));
    mt.seed(seq);
    std::uniform_int_distribution<> udx(0, width);
    std::uniform_int_distribution<> udy(0, height);
    std::uniform_int_distribution<> udc(0, 65535);
    std::uniform_int_distribution<> udt(0, 25);

    // фон с градиентной заливкой
    for (int iter = 0; iter < width; iter++)
    {
        image.line(
            iter, 0, iter, height,
            65535 - int(65535 * ((double)iter) / (width)),
            int(65535 * ((double)iter) / (width)),
            65535);
    }
}
```



```

}

// случайный текст
std::string font_family = font.data();
for (int i = 0; i < 6; ++i)
{
    image.plot_text(
        // шрифт
        font_family.data(), font_size,
        // позиция
        i*width / 6 + 10, height / 2 - 10,
        // угол
        (i % 2 == 0 ? -1 : 1)*(udt(mt) * 3.14) / 180,
        // текст
        std::string(1, char('A' + udt(mt))).data(),
        // цвет
        0, 0, 0);
}

// случайные линии
for (int i = 0; i < 4; ++i)
{
    image.line(udx(mt), 0, udx(mt), height,
              udc(mt), udc(mt), udc(mt));

    image.line(0, udy(mt), width, udy(mt),
              udc(mt), udc(mt), udc(mt));
}

image.close();
}

```



Следующий листинг показывает, как можно использовать эту функцию. Обратите внимание, что путь к файлу шрифта (в данном случае Arial) жестко задан для Windows и Apple, но в других платформах должен указываться пользователем:

```

int main()
{
    std::string font_path;

#ifdef WIN32
    font_path = R"(c:\windows\fonts\arial.ttf)";
#elif defined (__APPLE__)
    font_path = R"(/Library/Fonts/Arial.ttf)";
#else
    std::cout << "Font path: ";
    std::cin >> font_path;

```

```
#endif

create_image(200, 50,
             font_path, 18,
             "validation.png");
}
```



Цветовая схема для фона в функции `create_image()` всегда производит одну и ту же градиентную заливку для изображений с одинаковой шириной. Вы можете попробовать изменить функцию так, чтобы она выбирала случайный цвет для фона и текста.

## 84. Генератор штрихкодов EAN-13

*Международный номер товара* – International Article Number, или *Европейский номер товара* (European Article Number, EAN), как описывается в Википедии, – это европейский стандарт штрихкода для кодирования идентификатора товара и производителя. Наибольшее распространение получил 13-значный стандарт EAN-13. Описание стандарта, включая информацию о правилах формирования штрихкода, можно найти в Википедии: [https://ru.wikipedia.org/wiki/European\\_Article\\_Number](https://ru.wikipedia.org/wiki/European_Article_Number), поэтому я не буду приводить его в этой книге. На рис. 10.6 изображен штрихкод EAN-13 для числа 5901234123457, которое указано как пример в условиях задачи:



**Рис. 10.6.** Штрихкод EAN-13 для числа 5901234123457

Класс `ean13`, представленный в следующем листинге, создает штрихкод EAN-13 для заданного числа. Число может быть представлено в виде строки или значения типа `unsigned long long`. Класс поддерживает вычисление 13-го знака с контрольной суммой, если конструктору передать 12-значное число, или проверку 13-й цифры, если конструктору передать 13-значное число. Контрольная сумма – это 1-значное число, которое нужно добавить во взвешенную сумму первых 12 цифр, чтобы она стала кратной 10:

```
struct ean13
{
public:
```

```

ean13(std::string_view code)
{
    if (code.length() == 13)
    {
        if (code[12] != '0' + get_crc(code.substr(0,12)))
            throw std::runtime_error("Not an EAN-13 format.");

        number = code;
    }
    else if (code.length() == 12)
    {
        number = code.data() + std::string(1, '0' + get_crc(code));
    }
}

ean13(unsigned long long code) : ean13(std::to_string(code))
{ }

std::array<unsigned char, 13> to_array() const
{
    std::array<unsigned char, 13> result;
    for (int i = 0; i < 13; ++i)
        result[i] = static_cast<unsigned char>(number[i] - '0');
    return result;
}

std::string to_string() const noexcept { return number; }

private:
unsigned char get_crc(std::string_view code)
{
    unsigned char weights[12] = { 1,3,1,3,1,3,1,3,1,3,1,3 };
    size_t index = 0;
    auto sum = std::accumulate(
        std::begin(code), std::end(code), 0,
        [&weights, &index](int const total, char const c) {
            return total + weights[index++] * (c - '0'); });
    return 10 - sum % 10;
}

std::string number;
};

```

Как описывается в Википедии, штрихкод состоит из 95 равных областей, слева направо:

- три области – маркер начала;
- 42 области для группы из шести цифр слева. Эти 42 области можно разделить на шесть групп по семь областей, кодирующих цифры 2–7. Эти

коды могут иметь признак четности или нечетности, и все вместе эти признаки кодируют первую цифру EAN-13;

- 5 областей – маркер середины;
- 42 области для группы из шести цифр справа. Эти 42 области можно разделить на шесть групп по семь областей, кодирующих цифры 8–13. Все эти цифры кодируются с четным признаком четности. Цифра 13 – контрольная;
- 3 области – маркер конца.



Следующие таблицы, взятые из Википедии, демонстрируют кодирование двух групп по шесть цифр (табл. 10.1) и самих цифр, согласно значению первой цифры (табл. 10.2):

**Таблица 10.1.** Структура кода EAN-13

Первая цифра	Первая (левая) группа цифр	Вторая (правая) группа цифр
0	LLLLLL	RRRRRR
1	LLGLGG	RRRRRR
2	LLGGLG	RRRRRR
3	LLGGGL	RRRRRR
4	LGLLGG	RRRRRR
5	LGGLLG	RRRRRR
6	LGGGLL	RRRRRR
7	LGLGLG	RRRRRR
8	LGLGGL	RRRRRR
9	LGGLGL	RRRRRR



**Таблица 10.2.** Кодирование цифр

Цифра	L-код	R-код	G-код
0	0001101	1110010	0100111
1	0011001	1100110	0110011
2	0010011	1101100	0011011
3	0111101	1000010	0100001
4	0100011	1011100	0011101
5	0110001	1001110	0111001
6	0101111	1010000	0000101
7	0111011	1000100	0010001
8	0110111	1001000	0001001
9	0001011	1110100	0010111

Класс `ean13_barcode_generator` включает функции для создания PNG-изображения штрихкода EAN-13 из числового представления `ean13` и сохранения его в файл. Вот некоторые члены класса:

- `create()` – единственная общедоступная функция в классе. Она принимает число EAN-13, путь к выходному файлу, ширину в пикселях каждого фрагмента, высоту штрихкода и ширину пустой рамки вокруг штрихкода. Она последовательно рисует маркер начала, затем первые шесть цифр, маркер середины, последние шесть цифр и маркер конца, а потом сохраняет изображение в файл;
- `draw_digit()` – закрытая вспомогательная функция, которая с помощью метода `pngwriter::filledsquare()` рисует 7-битные цифры и маркеры начала, середины и конца;
- закрытые переменные-члены с таблицами кодирования, а также значениями маркеров: `eandigits`, `marker_start`, `marker_end` и `marker_center`.

Определение класса `ean13_barcode_generator` приводится в следующем листинге:

```
struct ean13_barcode_generator
{
    void create(ean13 const & code,
               std::string_view filename,
               int const digit_width = 3,
               int const height = 50,
               int const margin = 10);
private:
    int draw_digit(unsigned char code, unsigned int size,
                  pngwriter& image,
                  int const x, int const y,
                  int const digit_width, int const height)
    {
        std::bitset<7> bits(code);
        int pos = x;
        for (int i = size - 1; i >= 0; --i)
        {
            if (bits[i])
            {
                image.filledsquare(pos, y, pos + digit_width, y + height,
                                   0, 0, 0);
            }

            pos += digit_width;
        }

        return pos;
    }
};
```



```

}

unsigned char encodings[10][3] =
{
    { 0b0001101, 0b0100111, 0b1110010 },
    { 0b0011001, 0b0110011, 0b1100110 },
    { 0b0010011, 0b0011011, 0b1101100 },
    { 0b0111101, 0b0100001, 0b1000010 },
    { 0b0100011, 0b0011101, 0b1011100 },
    { 0b0110001, 0b0111001, 0b1001110 },
    { 0b0101111, 0b0000101, 0b1010000 },
    { 0b0111011, 0b0010001, 0b1000100 },
    { 0b0110111, 0b0001001, 0b1001000 },
    { 0b0001011, 0b0010111, 0b1110100 },
};

```



```

unsigned char eandigits[10][6] =
{
    { 0,0,0,0,0,0 },
    { 0,0,1,0,1,1 },
    { 0,0,1,1,0,1 },
    { 0,0,1,1,1,0 },
    { 0,1,0,0,1,1 },
    { 0,1,1,0,0,1 },
    { 0,1,1,1,0,0 },
    { 0,1,0,1,0,1 },
    { 0,1,0,1,1,0 },
    { 0,1,1,0,1,0 },
};

```



```

unsigned char marker_start = 0b101;
unsigned char marker_end   = 0b101;
unsigned char marker_center = 0b01010;
};

```

Далее следует реализация метода create():

```

void ean13_barcode_generator::create(ean13 const & code,
                                     std::string_view filename,
                                     int const digit_width = 3,
                                     int const height = 50,
                                     int const margin = 10)
{
    pngwriter image(
        margin * 2 + 95 * digit_width,
        height + margin * 2,

```





```
65535,
filename.data());

std::array<unsigned char, 13> digits = code.to_array();

int x = margin;
x = draw_digit(marker_start, 3, image, x, margin,
               digit_width, height);

for (int i = 0; i < 6; ++i)
{
    int code = encodings[digits[1 + i]][eandigits[digits[0]][i]];
    x = draw_digit(code, 7, image, x, margin, digit_width, height);
}

x = draw_digit(marker_center, 5, image, x, margin,
               digit_width, height);

for (int i = 0; i < 6; ++i)
{
    int code = encodings[digits[7 + i]][2];
    x = draw_digit(code, 7, image, x, margin, digit_width, height);
}

x = draw_digit(marker_end, 3, image, x, margin,
               digit_width, height);

image.close();
}
```



В следующем листинге показано, как можно использовать этот класс:

```
int main()
{
    ean13_barcode_generator generator;
    generator.create(ean13("5901234123457"), "5901234123457.png",
                   5, 150, 30);
}
```



В качестве самостоятельного упражнения можете попробовать реализовать вывод цифр числа EAN-13 под штрихкодом.

## 85. Чтение информации о фильмах из базы данных SQLite



SQLite – это библиотека поддержки встраиваемых баз данных, написанная на C (существуют также библиотеки-обертки для использования SQLite во многих других языках программирования). SQLite не является движком баз данных типа клиент/сервер, она встраивается в само приложение. База данных может содержать таблицы, индексы, триггеры и представления и хранится в виде единственного файла на диске. Поскольку доступ к базе данных в конечном счете заключается в обращении к локальному дисковому файлу, без использования инструментов взаимодействий процессов, SQLite обладает более высокой производительностью в сравнении с другими реляционными базами данных. SQLite, как можно заключить из названия, основывается на использовании языка запросов SQL, хотя и не реализует все его возможности (такие как RIGHT OUTER JOIN). SQLite используется не только в веб-браузерах (некоторые основные браузеры позволяют сохранять и извлекать данные из базы данных SQLite, используя технологию Web SQL Database), веб-фреймворках (таких как Bugzilla, Django, Drupal и Ruby on Rails) и операционных системах (включая Android, Windows 10, FreeBSD, OpenBSD, Symbian OS и др.), но также в мобильных приложениях и играх. SQLite имеет также свои ограничения, наиболее заметными из которых является отсутствие механизма управления пользователями. Стороннее расширение *SQLCipher* обеспечивает поддержку 256-битного шифрования AES баз данных SQLite. Получить библиотеку можно по адресу: <https://www.sqlite.org/>.

Исходный код библиотеки SQLite хранится в большом количестве исходных файлов и сценариев, однако имеется сокращенная версия, которая рекомендуется к применению во всех приложениях. Сокращенная версия состоит из двух файлов, `sqlite3.h` и `sqlite3.c`, которые можно скомпилировать с приложением. Пакет с сокращенной версией, а также другие пакеты библиотеки, включая дополнительные инструменты, можно загрузить по адресу: <https://www.sqlite.org/download.html>.

Как отмечалось выше, библиотека написана на C. Однако существуют различные обертки для нее на C++, включая *SQLiteCPP*, *CppSQLite*, *sqlite3cc* и *sqlite\_modern\_cpp*. В этой книге мы будем использовать последнюю из них, *sqlite\_modern\_cpp*, потому что это легковесная обертка, написанная на C++, с поддержкой особенностей C++17, а также *SQLCipher*. Она доступна по адресу: [https://github.com/SQLiteModernCpp/sqlite\\_modern\\_cpp](https://github.com/SQLiteModernCpp/sqlite_modern_cpp). Чтобы воспользоваться библиотекой, нужно подключить заголовок `sqlite_modern_cpp.h`.

Прежде чем начать писать код с решением этой задачи, нужно создать базу данных. Структура базы данных показана в описании задачи, на рис. 10.3. Для этого можно использовать инструмент командной строки `sqlite3`. Чтобы создать новую или открыть существующую базу данных, выполните команду:

```
sqlite3 <filename>
```

В примерах исходного кода, сопровождающих книгу, вы найдете уже созданную базу данных в файле `srcchallenger85.db`. Однако желающие могут создать свою базу командой выше и настроить ее структуру, как показано ниже:

```
create table movies(title text not null,
                   year integer not null,
                   length integer not null);

create table persons(name text not null);

create table directors(movieid integer not null,
                       personid integer not null);

create table writers(movieid integer not null,
                    personid integer not null);

create table casting(movieid integer not null,
                    personid integer not null,
                    role text not null);
```



Обратите внимание, что кроме столбцов, указанных здесь, SQLite автоматически добавит неявные столбцы `rowid` с автоматическим инкрементом для хранения 64-битных целых со знаком, уникально идентифицирующих записи в таблице. База данных `srcchallenger85.db` содержит описание нескольких фильмов, добавленных следующими командами:

```
insert into movies values ('The Matrix', 1999, 196);
insert into movies values ('Forrest Gump', 1994, 202);

insert into persons values('Keanu Reeves');
insert into persons values('Laurence Fishburne');
insert into persons values('Carrie-Anne Moss');
insert into persons values('Hugo Weaving');
insert into persons values('Lana Wachowski');
insert into persons values('Lilly Wachowski');
insert into persons values('Tom Hanks');
insert into persons values('Sally Field');
insert into persons values('Robin Wright');
insert into persons values('Mykelti Williamson');
insert into persons values('Robert Zemeckis');
insert into persons values('Winston Groom');
insert into persons values('Eric Roth');

insert into directors values(1, 5);
insert into directors values(1, 6);
insert into directors values(2, 11);

insert into writers values(1, 5);
```

```

insert into writers values(1, 6);
insert into writers values(2, 12);
insert into writers values(2, 13);

insert into casting values(1, 1, 'Neo');
insert into casting values(1, 2, 'Morpheus');
insert into casting values(1, 3, 'Trinity');
insert into casting values(1, 4, 'Agent Smith');
insert into casting values(2, 7, 'Forrest Gump');
insert into casting values(2, 8, 'Mrs. Gump');
insert into casting values(2, 9, 'Jenny Curran');
insert into casting values(2, 10, 'Bubba Blue');

```



После создания и заполнения базы данных можно переходить к следующему этапу решения задачи. Далее следуют определения классов, которые будут использоваться в этой и в следующих задачах для представления фильмов:

```

struct casting_role
{
    std::string actor;
    std::string role;
};

struct movie
{
    unsigned int          id;
    std::string           title;
    int                   year;
    unsigned int          length;
    std::vector<casting_role> cast;
    std::vector<std::string> directors;
    std::vector<std::string> writers;
};

using movie_list = std::vector<movie>;

```





`sqlite::database` – главный класс в библиотеке *sqlite\_modern\_cpp*, с которым мы будем работать. Он поддерживает такие функции, как подключение к базе данных, подготовку и выполнение инструкций SQL, привязку параметров и обратных вызовов и обработку транзакций. Чтобы открыть базу данных, достаточно передать путь к файлу в конструктор `sqlite::database`. Если во время работы SQLite возникнет какое-либо исключение, приложение получит исключение `sqlite::sqlite_exception`. В следующем листинге показана функция `main()` программы, которая подключается к базе данных `cppchallenge85.db` (хранящейся в текущем каталоге). В случае успешного соединения она извлекает информацию обо всех фильмах и выводит ее в консоль:

```

int main()
{
    try
    {
        sqlite::database db(R"(cppchallenger85.db)");

        auto movies = get_movies(db);
        for (auto const & m : movies)
            print_movie(m);
    }
    catch (sqlite::sqlite_exception const & e)
    {
        std::cerr << e.get_code() << ": " << e.what() << " during "
            << e.get_sql() << std::endl;
    }
    catch (std::exception const & e)
    {
        std::cerr << e.what() << std::endl;
    }
}

```

Вывод в консоль осуществляет функция `print_movie()`:

```

void print_movie(movie const & m)
{
    std::cout << "[" << m.id << "]" "
        << m.title << " (" << m.year << ") "
        << m.length << "min" << std::endl;
    std::cout << " directed by: ";
    for (auto const & d : m.directors) std::cout << d << ", ";
    std::cout << std::endl;
    std::cout << " written by: ";
    for (auto const & w : m.writers) std::cout << w << ", ";
    std::cout << std::endl;
    std::cout << " cast: ";
    for (auto const & r : m.cast)
        std::cout << r.actor << " (" << r.role << "), ";
    std::cout << std::endl << std::endl;
}

```

Класс `sqlite::database` перегружает операторы `<<` и `>>`, первый служит для подготовки инструкций и привязки параметров и выполняет другие операции ввода в базу данных, а последний извлекает данные из базы данных. Чтобы выполнить привязку параметра, нужно добавить символ `?` в инструкцию SQL, а затем передать фактическое значение параметра с помощью перегруженного оператора `<<`. Привязка выполняется в порядке следования символов `?`. Для каждой записи в результатах запроса вызывается функция-обработчик.

Библиотека *sqlite\_modern\_cpp* позволяет передавать лямбда-функции (соответствующего типа) с параметрами для каждого столбца в записи. Для столбцов, которые могут иметь пустое значение, следует использовать `std::unique_ptr<T>` или `std::optional<T>`, если компилятор поддерживает эти особенности стандарта C++17.

Следующая функция `get_directors()` читает информацию о режиссерах фильма из таблиц `directors` и `persons`. Обратите внимание, что в этой и в следующих инструкциях SQL используется неявный столбец `rowid`:

```
std::vector<std::string> get_directors(sqlite3_int64 const movie_id,
                                       sqlite::database & db)
{
    std::vector<std::string> result;
    db << R"(select p.name from directors as d
              join persons as p on d.personid = p.rowid
              where d.movieid = ?;)"
    << movie_id
    >> [&result](std::string const name)
    {
        result.emplace_back(name);
    };

    return result;
}
```



Очень похожая функция `get_writers()` читает информацию о сценаристах фильма из таблицы `writers`:

```
std::vector<std::string> get_writers(sqlite3_int64 const movie_id,
                                       sqlite::database & db)
{
    std::vector<std::string> result;
    db << R"(select p.name from writers as w
              join persons as p on w.personid = p.rowid
              where w.movieid = ?;)"
    << movie_id
    >> [&result](std::string const name)
    {
        result.emplace_back(name);
    };

    return result;
}
```

Список актеров извлекается из таблицы `casting` с помощью функции `get_cast()`:

```

std::vector<casting_role> get_cast(sqlite3_int64 const movie_id,
                                sqlite::database & db)
{
    std::vector<casting_role> result;
    db << R"(select p.name, c.role from casting as c
              join persons as p on c.personid = p.rowid
              where c.movieid = ?;)"
    << movie_id
    >> [&result](std::string const name, std::string role)
    {
        result.emplace_back(casting_role{ name, role });
    };

    return result;
}

```



Все эти функции вызываются из функции `get_movies()`, которая возвращает список всех фильмов из базы данных:

```

movie_list get_movies(sqlite::database & db)
{
    movie_list movies;

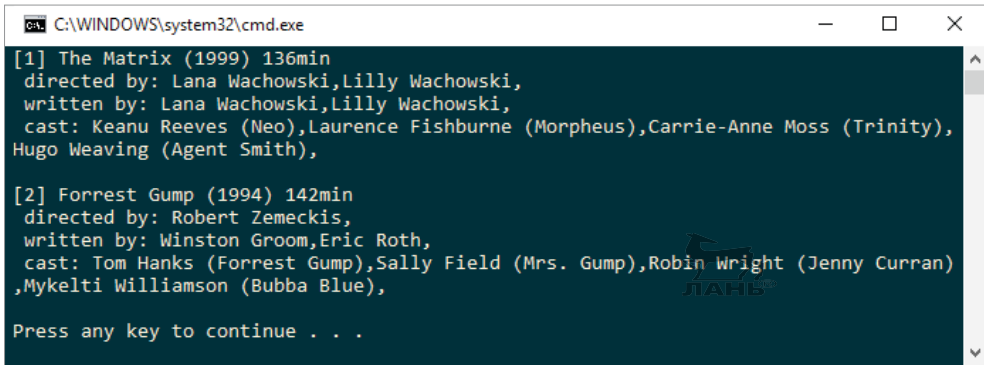
    db << R"(select rowid, * from movies;)"
    >> [&movies, &db](sqlite3_int64 const rowid,
                    std::string const & title,
                    int const year, int const length)
    {
        movies.emplace_back(movie{
            static_cast<unsigned int>(rowid),
            title,
            year,
            static_cast<unsigned int>(length),
            get_cast(rowid, db),
            get_directors(rowid, db),
            get_directors(rowid, db)
        });
    };

    return movies;
}

```



На этом реализацию решения задачи можно считать законченной. На рис. 10.7 показан скриншот с выводом данных, добавленных кодом, который мы видели выше.



```

C:\WINDOWS\system32\cmd.exe
[1] The Matrix (1999) 136min
    directed by: Lana Wachowski,Lilly Wachowski,
    written by: Lana Wachowski,Lilly Wachowski,
    cast: Keanu Reeves (Neo),Laurence Fishburne (Morpheus),Carrie-Anne Moss (Trinity),
    Hugo Weaving (Agent Smith),

[2] Forrest Gump (1994) 142min
    directed by: Robert Zemeckis,
    written by: Winston Groom,Eric Roth,
    cast: Tom Hanks (Forrest Gump),Sally Field (Mrs. Gump),Robin Wright (Jenny Curran)
    ,Mykelti Williamson (Bubba Blue),

Press any key to continue . . .

```

Рис. 10.7. Вывод ранее добавленных данных

## 86. Добавление информации о фильмах в базу данных SQLite

Это решение основано на решении предыдущей задачи. Вы должны решить ее, прежде чем приступать к этой. Кроме того, здесь используется функция `split()`, реализованная в главе 3 «Строки и регулярные выражения», в задаче 27 «Разбиение строк на лексемы по разделителям из списка». Поэтому я не буду повторять ее код здесь. В исходном коде примеров для книги вы найдете файл базы данных `cppchallenger86.db`, в котором есть несколько записей, созданных для этой задачи.

Следующая функция `read_movie()` читает информацию о фильме с клавиатуры (название, год выхода, продолжительность в минутах, режиссеры, сценаристы и актеры), создает объект `movie` и возвращает его. Информация об актерах должна передаваться в виде списка элементов через запятую в формате *имя актера=имя роли*. Например, для фильма «The Matrix» (Матрица), который мы уже видели в предыдущей задаче, список актеров должен вводиться как одна строка: Киану Ривз=Нео, Лоренс Фишборн=Морфеус, Керри-Энн Мосс=Тринити, Хьюго Уивинг=Агент Смит. Для чтения строк с пробелами следует использовать функцию `std::getline()`; если использовать объект `std::cin`, он прочитает только часть строки до первого встретившегося пробела:

```

movie read_movie()
{
    movie m;

    std::cout << "Enter movie" << std::endl;
    std::cout << "Title: ";
    std::getline(std::cin, m.title);
    std::cout << "Year: "; std::cin >> m.year;
    std::cout << "Length: "; std::cin >> m.length;
}

```



```

std::cin.ignore();
std::string directors;
std::cout << "Directors: ";
std::getline(std::cin, directors);
m.directors = split(directors, ',');
std::string writers;
std::cout << "Writers: ";
std::getline(std::cin, writers);
m.writers = split(writers, ',');
std::string cast;
std::cout << "Cast: ";
std::getline(std::cin, cast);
auto roles = split(cast, ',');
for (auto const & r : roles)
{
    auto pos = r.find_first_of('=');
    casting_role cr;
    cr.actor = r.substr(0, pos);
    cr.role = r.substr(pos + 1, r.size() - pos - 1);
    m.cast.push_back(cr);
}

return m;
}

```



Следующая функция, `get_person_id()`, возвращает числовой идентификатор персоны – значение поля `rowid`, автоматически добавляемого библиотекой SQLite в таблицу при ее создании (если иное не определено). Столбец `rowid` имеет тип `sqlite_int64` – 64-битное целое со знаком:

```

sqlite_int64 get_person_id(std::string const & name, sqlite::database & db)
{
    sqlite_int64 id = 0;
    db << "select rowid from persons where name=?;"
    << name
    >> [&id](sqlite_int64 const rowid) {id = rowid; };

    return id;
}

```

Функции `insert_person()`, `insert_directors()`, `insert_writers()` и `insert_cast()` вставляют новые записи в таблицы `persons`, `directors`, `writers` и `casting`. Для этого они используют объект `sqlite::database`, который передается им функцией `main()` как аргумент. Перед вставкой новых записей в таблицы `directors`, `writers` и `casting` проверяется наличие указанной персоны в таблице `persons`, и если она отсутствует, добавляется новая запись в эту таблицу:

```

sqlite_int64 insert_person(std::string_view name, sqlite::database & db)
{
    db << "insert into persons values(?);"
        << name.data();
    return db.last_insert_rowid();
}

void insert_directors(sqlite_int64 const movie_id,
                     std::vector<std::string> const & directors,
                     sqlite::database & db)
{
    for (auto const & director : directors)
    {
        auto id = get_person_id(director, db);

        if (id == 0)
            id = insert_person(director, db);

        db << "insert into directors values(?, ?);"
            << movie_id
            << id;
    }
}

void insert_writers(sqlite_int64 const movie_id,
                   std::vector<std::string> const & writers,
                   sqlite::database & db)
{
    for (auto const & writer : writers)
    {
        auto id = get_person_id(writer, db);

        if (id == 0)
            id = insert_person(writer, db);

        db << "insert into writers values(?, ?);"
            << movie_id
            << id;
    }
}

void insert_cast(sqlite_int64 const movie_id,
                std::vector<casting_role> const & cast,
                sqlite::database & db)
{
    for (auto const & cr : cast)
    {

```

```

auto id = get_person_id(cr.actor, db);

if (id == 0)
    id = insert_person(cr.actor, db);

db << "insert into casting values(?,?,?);"
    << movie_id
    << id
    << cr.role;
}
}

```



Функция `insert_movie()` вставляет новую запись в таблицу `movies` и затем вызывает функции, описанные выше, чтобы также добавить информацию о режиссерах, сценаристах и актерах. Все эти операции выполняются в рамках одной транзакции. Поддержка транзакций в классе `sqlite::database` реализована в виде команд `begin`; `commit`; и `rollback`; (обратите внимание на точку с запятой в конце каждой команды). Эти команды выполняются перегруженной версией `operator<<` в классе `sqlite::database`. В начале функции производится запуск транзакции, а в конце она подтверждается. Если в ходе выполнения команд SQL возникнет исключение, автоматически произойдет откат транзакции:

```

void insert_movie(movie& m, sqlite::database & db)
{
    try
    {
        db << "begin;";

        db << "insert into movies values(?,?,?);"
            << m.title
            << m.year
            << m.length;

        auto movieid = db.last_insert_rowid();

        insert_directors(movieid, m.directors, db);
        insert_writers(movieid, m.writers, db);
        insert_cast(movieid, m.cast, db);

        m.id = static_cast<unsigned int>(movieid);

        db << "commit;";
    }
    catch (std::exception const &)
    {
        db << "rollback;";
    }
}

```

```

    }
}

```

Теперь можно приступить к основной программе. Она должна открыть базу данных SQLite `cppchallenger86.db`, прочитать описание фильма с клавиатуры, добавить ее в базу данных и вывести полный список фильмов в консоль:

```

int main()
{
    try
    {
        sqlite::database db(R"(cppchallenger86.db)");

        auto movie = read_movie();
        insert_movie(movie, db);

        auto movies = get_movies(db);
        for (auto const & m : movies)
            print_movie(m);
    }
    catch (sqlite::sqlite_exception const & e)
    {
        std::cerr << e.get_code() << ": " << e.what() << " during "
                    << e.get_sql() << std::endl;
    }
    catch (std::exception const & e)
    {
        std::cerr << e.what() << std::endl;
    }
}

```



## 87. Обработка изображений для фильмов в базе данных SQLite

Прежде чем приступить к этой задаче, решите сначала предыдущие, если вы этого еще не сделали. В этой задаче мы должны расширить модель базы данных, добавив в нее дополнительную таблицу для хранения изображений и, возможно, других медиафайлов, таких как видеоролики. Содержимое медиафайлов должно храниться в виде двоичных блоков `blob`, но также должны сохраняться другие атрибуты, такие как описание и имя файла.

Чтобы создать дополнительную таблицу для медиафайлов (которую мы назовем `media`), откройте файл базы данных с помощью `sqlite3`, как было показано в задаче 85, и выполните следующую ниже команду. В исходном коде примеров для книги вы найдете файл базы данных `cppchallenger87.db`, который уже содержит расширенную версию базы данных:

```
create table media(movieid integer not null,
                  name text not null,
                  description text,
                  content blob not null);
```



Когда возникает необходимость хранить в базе данных большие объекты, у вас на выбор есть два варианта: хранить их непосредственно в базе данных, в виде двоичных блоков blob, или в виде отдельных файлов, а в базе данных сохранять только пути к этим файлам. Согласно тестам, которые проводят разработчики SQLite, объекты с размерами меньше 100 Кбайт читаются быстрее, когда они хранятся непосредственно в базе данных. Объекты больше 100 Кбайт читаются быстрее, когда они хранятся отдельно. Учитывайте этот аспект при проектировании своих баз данных. Однако в этой книге мы проигнорируем фактор производительности и будем сохранять медиа-файлы непосредственно в базе данных.

Поле `description` может содержать значение `null`. Для доступа к нему с помощью *sqlite\_modern\_cpp* вы можете использовать тип `std::optional<T>`, если ваш компилятор поддерживает эту особенность стандарта C++17. Однако для этого нужно объявить макрос `MODERN_SQLITE_STD_OPTIONAL_SUPPORT`. Иначе используйте тип `std::unique_ptr<T>`.

Для работы с объектами из таблицы `media` мы будем использовать типы, показанные ниже. Поле `rowid` имеет тип `sqlite3_int64`, но здесь мы используем тип `unsigned int`, исключительно для согласованности с типом `movie`, который мы видели в двух предыдущих решениях:

```
struct media
{
    unsigned int          id;
    unsigned int          movie_id;
    std::string           name;
    std::optional<std::string> text;
    std::vector<char>     blob;
};

using media_list = std::vector<media>;
```

Функции `add_media()`, `get_media()` и `delete_media()` добавляют, извлекают и удаляют медиафайлы, связанные с фильмом. Они должны получиться простыми, как подсказывает опыт использования *sqlite\_modern\_cpp*, который мы получили в предыдущих задачах. Важно отметить, что при извлечении полей из таблицы – в данном случае таблицы `media` – необходимо явно указывать поле

rowid, ПОТОМУ ЧТО ОНО НЕ ИЗВЛЕКАЕТСЯ ЗВЕЗДОЧКОЙ (\*), КОГДА ЗАПРОС ИЗВЛЕКАЕТ ВСЕ ПОЛЯ:

```
bool add_media(sqlite_int64 const movieid,
               std::string_view name,
               std::string_view description,
               std::vector<char> content,
               sqlite::database & db)
{
    try
    {
        db << "insert into media values(?,?,?,?)"
          << movieid
          << name.data()
          << description.data()
          << content;
        return true;
    }
    catch (...) { return false; }
}

media_list get_media(sqlite_int64 const movieid,
                    sqlite::database & db)
{
    media_list list;

    db << "select rowid, * from media where movieid = ?;"
      << movieid
      >> [&list](sqlite_int64 const rowid,
                sqlite_int64 const movieid,
                std::string const & name,
                std::optional<std::string> const text,
                std::vector<char> const & blob
        )
        {
            list.emplace_back(media{
                static_cast<unsigned int>(rowid),
                static_cast<unsigned int>(movieid),
                name,
                text,
                blob});
        };

    return list;
}

bool delete_media(sqlite_int64 const mediaid,
                 sqlite::database & db)
```





```

{
  try
  {
    db << "delete from media where rowid = ?;"
      << mediaid;
    return true;
  }
  catch (...) { return false; }
}

```

Медиафайлы связываются с фильмом посредством идентификатора фильма. Чтобы получить этот идентификатор по названию, мы используем функцию `get_movies()`, показанную далее. Она извлекает список всех фильмов, соответствующих указанному названию. Если таких фильмов оказывается больше одного, мы можем выбрать, с каким из фильмов должен быть связан данный медиафайл:

```

movie_list get_movies(std::string_view title, sqlite::database & db)
{
  movie_list movies;

  db << R"(select rowid, * from movies where title=?);"
    << title.data()
    >> [&movies, &db](sqlite3_int64 const rowid,
                      std::string const & title,
                      int const year, int const length)
    {
      movies.emplace_back(movie{
        static_cast<unsigned int>(rowid),
        title,
        year,
        static_cast<unsigned int>(length),
        {},
        {},
        {}
      });
    };

  return movies;
}

```

Главная программа реализована как маленькая утилита, которая принимает команды и выводит в консоль результат их выполнения. В число поддерживаемых команд входят: поиск, добавление, вывод списка и удаление медиафайлов. Функция `print_commands()` выводит список поддерживаемых команд:

```

void print_commands()
{

```

```

std::cout
  << "find <title>                finds a movie ID\n"
  << "list <movieid>             lists the images of a movie\n"
  << "add <movieid>,<path>,<description> adds a new image\n"
  << "del <imageid>              delete an image\n"
  << "help                        shows available commands\n"
  << "exit                        exists the application\n";
}

```



Реализация функции `main()` показана в листинге ниже. Она сначала открывает базу данных SQLite с именем `cppchallenger87.db`. Затем входит в бесконечный цикл чтения ввода пользователя и выполнения команд. Цикл и сама программа завершаются, когда пользователь введет команду `exit`:

```

int main()
{
  try
  {
    sqlite::database db(R"(cppchallenger87.db)");

    while (true)
    {
      std::string line;
      std::getline(std::cin, line);

      if (line == "help") print_commands();
      else if (line == "exit") break;
      else
      {
        if (starts_with(line, "find"))
          run_find(line, db);
        else if (starts_with(line, "list"))
          run_list(line, db);
        else if (starts_with(line, "add"))
          run_add(line, db);
        else if (starts_with(line, "del"))
          run_del(line, db);
        else
          std::cout << "unknown command" << std::endl;
      }

      std::cout << std::endl;
    }
  }
  catch (sqlite::sqlite_exception const & e)
  {
    std::cerr << e.get_code() << ": " << e.what() << " during "
              << e.get_sql() << std::endl;
  }
}

```





```

    }
    catch (std::exception const & e)
    {
        std::cerr << e.what() << std::endl;
    }
}

```



Все поддерживаемые команды реализованы в виде отдельных функций. `run_find()`, `run_list()`, `run_add()` и `run_del()` анализируют ввод пользователя, вызывают соответствующую функцию доступа к базе данных и выводят результат. Эти функции не выполняют тщательной проверки ввода пользователя. Команды чувствительны к регистру символов и должны вводиться строчными буквами.

Функция `run_find()` извлекает название фильма из ввода пользователя, вызывает `get_movie()`, чтобы получить список всех фильмов с таким названием, и выводит его в консоль:

```

void run_find(std::string_view line, sqlite::database & db)
{
    auto title = trim(line.substr(5));

    auto movies = get_movies(title, db);
    if(movies.empty())
        std::cout << "empty" << std::endl;
    else
    {
        for (auto const m : movies)
        {
            std::cout << m.id << " | "
                << m.title << " | "
                << m.year << " | "
                << m.length << "min"
                << std::endl;
        }
    }
}

```



Функция `run_list()` извлекает числовой идентификатор фильма из ввода пользователя, вызывает `get_media()`, чтобы получить список всех медиафайлов для этого фильма, и выводит его в консоль. Эта функция выводит только размер поля `blob`, но не сам объект:

```

void run_list(std::string_view line, sqlite::database & db)
{
    auto movieid = std::stoi(trim(line.substr(5)));
    if (movieid > 0)
    {
        auto list = get_media(movieid, db);
    }
}

```



```

if (list.empty())
{
    std::cout << "empty" << std::endl;
}
else
{
    for (auto const & m : list)
    {
        std::cout
            << m.id << " | "
            << m.movie_id << " | "
            << m.name << " | "
            << m.text.value_or("(null)") << " | "
            << m.blob.size() << " bytes"
            << std::endl;
    }
}
}
else
    std::cout << "input error" << std::endl;
}

```

Добавление нового медиафайла для фильма производится с помощью функции `run_add()`. Эта функция извлекает из ввода пользователя идентификатор фильма, путь к файлу и его описание и возвращает в виде списка элементов, разделенных запятыми (например: `add <movieid>, <path>, <description>`), загружает содержимое файла с диска с помощью вспомогательной функции `load_image()` и добавляет новую запись в таблицу `media`. Реализация, представленная здесь, не проверяет тип файла, поэтому в базу данных можно загрузить любой файл, не только изображение или видеоролик. Желающие могут добавить дополнительную проверку как самостоятельное упражнение:

```

std::vector<char> load_image(std::string_view filepath)
{
    std::vector<char> data;

    std::ifstream ifile(filepath.data(), std::ios::binary | std::ios::ate);
    if (ifile.is_open())
    {
        auto size = ifile.tellg();
        ifile.seekg(0, std::ios::beg);

        data.resize(static_cast<size_t>(size));
        ifile.read(reinterpret_cast<char*>(data.data()), size);
    }

    return data;
}

```

```

}

void run_add(std::string_view line, sqlite::database & db)
{
    auto parts = split(trim(line.substr(4)), '/', ',');
    if (parts.size() == 3)
    {
        auto movieid = std::stoi(parts[0]);
        auto path = std::experimental::filesystem::path{parts[1]};
        auto desc = parts[2];

        auto content = load_image(parts[1]);
        auto name = path.filename().string();

        auto success = add_media(movieid, name, desc, content, db);
        if (success)
            std::cout << "added" << std::endl;
        else
            std::cout << "failed" << std::endl;
    }
    else
        std::cout << "input error" << std::endl;
}

```

Нам осталось реализовать последнюю команду – удаляющую медиафайлы. Функция `run_del()` принимает идентификатор записи в таблице `media`, которую требуется удалить, и вызывает `delete_media()`:

```

void run_del(std::string_view line, sqlite::database & db)
{
    auto mediaid = std::stoi(trim(line.substr(4)));
    if (mediaid > 0)
    {
        auto success = delete_media(mediaid, db);
        if (success)
            std::cout << "deleted" << std::endl;
        else
            std::cout << "failed" << std::endl;
    }
    else
        std::cout << "input error" << std::endl;
}

```

В предыдущем листинге используется несколько вспомогательных функций: `split()`, которая разбивает текст на лексемы по указанным символам-разделителям; `starts_with()`, которая проверяет, что заданная строка начинается с указанной подстроки; и `trim()`, которая удаляет все пробелы в начале и в конце строки:

```

std::vector<std::string> split(std::string text, char const delimiter)
{
    auto sstr = std::stringstream{ text };
    auto tokens = std::vector<std::string>{};
    auto token = std::string{};
    while (std::getline(sstr, token, delimiter))
    {
        if (!token.empty()) tokens.push_back(token);
    }
    return tokens;
}

inline bool starts_with(std::string_view text, std::string_view part)
{
    return text.find(part) == 0;
}

inline std::string trim(std::string_view text)
{
    auto first{ text.find_first_not_of(' ') };
    auto last{ text.find_last_not_of(' ') };
    return text.substr(first, (last - first + 1)).data();
}

```



В следующем листинге приводится пример выполнения нескольких команд. Сначала мы выводим все фильмы с названием *The Matrix* (в базе данных только один такой фильм). Затем выводим все медиафайлы, ассоциированные с этим фильмом (в данном случае нет ни одного такого файла). После этого добавляем файл `the_matrix.jpg` из папки `res` и снова выводим список медиафайлов. Наконец, удаляем только что добавленный файл и опять выводим список медиафайлов, чтобы убедиться, что список пуст:

```

find The Matrix
1 | The Matrix | 1999 | 196min

list 1
empty

add 1,.\res\the_matrix.jpg,Main poster
added

list 1
1 | 1 | the_matrix.jpg | Main poster | 193906 bytes

del 1
deleted

list 1
empty

```

---

# Глава 11



## Криптография

### Задачи

#### 88. Шифр Цезаря

Напишите программу, которая шифрует и расшифровывает сообщения с использованием шифра Цезаря со сдвигом вправо на произвольное расстояние. Для простоты программа должна шифровать только символы верхнего регистра и только буквы, игнорируя цифры, знаки препинания и другие символы.

#### 89. Шифр Виженера

Напишите программу, которая шифрует и расшифровывает сообщения с использованием шифра Виженера (Vigenère). Для простоты программа должна шифровать только буквы верхнего регистра.

#### 90. Кодирование и декодирование Base64

Напишите программу, кодирующую и декодирующую двоичные данные с применением схемы кодирования base64. Вы должны написать свои функции кодирования и декодирования, без использования сторонних библиотек. Используйте для кодирования таблицу, указанную в спецификации MIME.

#### 91. Проверка учетных данных пользователя

Напишите программу, имитирующую аутентификацию пользователя в защищенной системе. Чтобы войти в систему, пользователь должен быть зарегистрирован в ней. Пользователь вводит имя и пароль, а программа должна проверить соответствие введенных данных имеющейся учетной записи и разрешить или отклонить доступ. По соображениям безопасности система не должна хранить пароли в открытом виде – только в виде хеш-суммы SHA.

#### 92. Вычисление хеш-суммы файла

Напишите программу, которая принимает путь к файлу, вычисляет для него хеш-суммы SHA1, SHA256 и MD5 и выводит их в консоль.

## 93. Шифрование и расшифровывание файлов

Напишите программу шифрования и расшифровывания файлов с использованием алгоритма блочного шифрования Advanced Encryption Standard (AES или Rijndael). Она должна принимать путь к исходному и целевому файлу, а также пароль.

## 94. Подписывание файлов

Напишите программу, которая будет подписывать файлы и проверять наличие изменений в подписанном файле с помощью алгоритма RSA. Подпись должна сохраняться в отдельном файле и использоваться позже при проверке. Программа должна содержать, по меньшей мере, две функции: одну для создания подписи (должна принимать путь к подписываемому файлу, путь к закрытому ключу RSA и путь к файлу сигнатуры), а вторую – для проверки (должна принимать путь к подписанному файлу, путь к открытому ключу RSA и путь к файлу сигнатуры).

# Решения

## 88. Шифр Цезаря

*Шифр Цезаря*, также известный как *код Цезаря* или *сдвиг Цезаря*, – один из самых простых и наиболее широко известных методов шифрования, заключающийся в замене каждой буквы в тексте некоторой другой буквой, отстоящей от данной на фиксированное число позиций в алфавите. Этот метод использовался Юлием Цезарем для защиты важных посланий военачальникам. Сам он использовал сдвиг на три буквы, то есть заменял букву А буквой D, букву В буквой Е и т. д. При таком способе кодирования строка CPPCHALLENGER превратится в FSSFKDOOHQJHU. Шифр подробно описан в Википедии: [https://ru.wikipedia.org/wiki/Шифр\\_Цезаря](https://ru.wikipedia.org/wiki/Шифр_Цезаря).



Шифр Цезаря не используется в современной криптографии, потому что легко взламывается, но его все еще часто применяют на веб-форумах и в новостных рассылках для скремблирования оскорбительных слов, решений головоломок и т. д. Единственная цель этой задачи – служить разминкой перед решением других задач в этой главе. Этот алгоритм не следует использовать для защиты информации.

Чтобы решить предложенную задачу, нужно реализовать две функции: одну для шифрования текста и вторую для расшифровывания.

- `caesar_encrypt()` – принимает объект `string_view`, представляющий простой текст, и величину сдвига в алфавите для замены букв. Эта функ-

ция замещает только буквы верхнего регистра и оставляет нетронутыми все остальные символы. Алфавит моделируется как циклическая последовательность, в результате при смещении вправо на 3 позиции буква X превратится в A, буква Y превратится в B, а буква Z превратится в C.

- `caesar_decrypt()` принимает объект `string_view`, представляющий зашифрованный текст и величину сдвига в алфавите, использовавшуюся при шифровании. По аналогии с функцией `caesar_encrypt()`, она должна преобразовывать только буквы верхнего регистра и оставлять нетронутыми все остальные символы.

```
std::string caesar_encrypt(std::string_view text, int const shift)
{
    std::string str;
    str.reserve(text.length());
    for (auto const c : text)
    {
        if (isalpha(c) && isupper(c))
            str += 'A' + (c - 'A' + shift) % 26;
        else
            str += c;
    }

    return str;
}
```

```
std::string caesar_decrypt(std::string_view text, int const shift)
{
    std::string str;
    str.reserve(text.length());
    for (auto const c : text)
    {
        if (isalpha(c) && isupper(c))
            str += 'A' + (26 + c - 'A' - shift) % 26;
        else
            str += c;
    }

    return str;
}
```

В следующем листинге приводится пример использования этих функций. Роль текста для шифрования здесь играет латинский алфавит, а функции шифрования/расшифровывания вызываются для всех возможных величин сдвига:

```
int main()
{
    auto text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    for (int i = 1; i <= 26; ++i)
```

```

{
    auto enc = caesar_encrypt(text, i);
    auto dec = caesar_decrypt(enc, i);
    assert(text == dec);
}
}
}

```



## 89. Шифр Виженера

Шифр Виженера – это алгоритм, использующий комбинацию шифров Цезаря. Впервые этот метод описал Джованни Батиста Беллазо (Giovan Battista Bellaso) в 1553 году, но по ошибке авторство было присвоено в XIX веке Блейзу де Виженеру (Blaise de Vigenère), да так и закрепилось за его именем. Этот способ шифрования подробно описан в Википедии: [https://ru.wikipedia.org/wiki/Шифр\\_Виженера](https://ru.wikipedia.org/wiki/Шифр_Виженера). Здесь я лишь кратко охарактеризую его.



Шифр Виженера не могли взломать три столетия, но в наши дни он легко вскрывается, так же как шифр Цезаря, на котором он основан. Подобно предыдущей задаче, эта служит лишь для разминки, и ее решение не имеет никакой практической ценности.

Метод основан на использовании **таблицы Виженера** (или **tabula recta**). Таблица для латинского алфавита состоит из 26 строк и 26 столбцов, в которой каждая строка представляет весь алфавит с циклическим сдвигом, как это принято в шифре Цезаря. Эта таблица показана на рис. 11.1.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Рис. 11.1. Таблица Виженера (tabula recta) для латинского алфавита



Для шифрования и расшифровывания необходим ключ. Ключ циклически записывается, пока его длина не сравняется с длиной текста, который нужно зашифровать или расшифровать. Шифрование выполняется заменой каждой буквы в тексте другой буквой, находящейся в таблице Виженера на пересечении строки, соответствующей букве из ключа, и столбца, соответствующего букве из текста. При расшифровывании нужно найти в таблице Виженера строку, соответствующую букве в ключе; в этой строке найти первый символ из зашифрованного текста и заменить зашифрованную букву буквой из заголовка этого столбца.

Ниже приводится функция `vigenere_encrypt()`, которая реализует шифрование. Она принимает текст сообщения с ключом, шифрует текст, как описано выше, и возвращает зашифрованный текст:

```
std::string vigenere_encrypt(std::string_view text, std::string_view key)
{
    std::string result;
    result.reserve(text.length());
    static auto table = build_vigenere_table();

    for (size_t i = 0; i < text.length(); ++i)
    {
        auto row = key[i%key.length()] - 'A';
        auto col = text[i] - 'A';

        result += table[row * 26 + col];
    }

    return result;
}
```



Парная ей функция – `vigenere_decrypt()` – принимает зашифрованный текст с ключом, расшифровывает его, как описано выше, и возвращает получившийся текст:

```
std::string vigenere_decrypt(std::string_view text, std::string_view key)
{
    std::string result;
    result.reserve(text.length());
    static auto table = build_vigenere_table();

    for (size_t i = 0; i < text.length(); ++i)
    {
        auto row = key[i%key.length()] - 'A';

        for (size_t col = 0; col < 26; col++)
        {
            if (table[row * 26 + col] == text[i])
```

```

        {
            result += 'A' + col;
            break;
        }
    }
}

return result;
}

```



Обе эти функции используют третью, вспомогательную функцию с именем `build_vigenere_table()`, которая создает таблицу Виженера, шифруя алфавит методом Цезаря 26 раз, каждый раз с новым смещением. Таблица хранится в программе в виде одной строки:

```

std::string build_vigenere_table()
{
    std::string table;
    table.reserve(26*26);
    for (int i = 0; i < 26; ++i)
        table += caesar_encrypt("ABCDEFGHIJKLMNOPQRSTUVWXYZ", i);

    return table;
}

```



Следующий листинг демонстрирует применение этих функций для шифрования и расшифровывания:

```

int main()
{
    auto text = "THECPPCHALLENGER";
    auto enc = vigenere_encrypt(text, "SAMPLE");
    auto dec = vigenere_decrypt(enc, "SAMPLE");
    assert(text == dec);
}

```

## 90. Кодирование и декодирование Base64

Base64 – это схема кодирования для представления двоичных данных в формате ASCII с использованием алфавита из 64 символов. Все реализации используют одни и те же первые 62 символа (A-Z, a-z и 0-9), но последние два значения могут отличаться. В спецификации MIME используются символы + и /. Каждая цифра в схеме base64 представляет 6 бит данных, то есть четыре цифры base64 кодируют точно три байта (8-битных) двоичных данных. Когда число байтов не кратно трем, перед преобразованием base64 в конец добавляются нулевые байты. В конце закодированных данных могут добавляться символы == или =, указывающие на наличие одного или двух пустых байтов в конечной тройке.

Вот пример кодирования текста `cpp`. В данном случае получается результат `Y3Bw`:

Исходный текст ASCII	<code>cpp</code>
Исходные октеты	<code>0x63 0x70 0x70</code>
Исходный текст в двоичном представлении	<code>01100011 01110000 01110000</code>
Двоичный код base64	<code>011000 110111 000001 110000</code>
Десятичный код base64	<code>24 55 1 48</code>
Код base64	<code>Y3Bw</code>



Алгоритм подробно описан в Википедии: <https://ru.wikipedia.org/wiki/Base64>. Для проверки результатов кодирования/декодирования можно использовать онлайн-кодировщики, например <https://www.base64encode.org/>.

Класс `encoded`, показанный ниже, имеет два общедоступных метода: `to_base64()` кодирует вектор байтов в base64 и возвращает результат в виде строки, а `from_base64()` декодирует строку с кодом base64 в вектор байтов и возвращает его. Для кодирования и декодирования используются две таблицы. Таблица для кодирования – это обычная строка, `table_enc`, содержащая алфавит base64. Таблица для декодирования, `table_dec`, – это массив с 256 целыми числами, представляющими индексы в таблице кодирования (`table_enc`) для всех 6-битных цифр base64:


```
class encoder
{
    std::string const table_enc =
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
    char const padding_symbol = '=';

    char const table_dec[256] =
    {
        -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,64,-1,-1,-1,-1,-1,-1,
        -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
        -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,62,-1,-1,-1,63,
        52,53,54,55,56,57,58,59,60,61,-1,-1,-1,65,-1,-1,
        -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,
        15,16,17,18,19,20,21,22,23,24,25,-1,-1,-1,-1,-1,-1,
        -1,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,
        41,42,43,44,45,46,47,48,49,50,51,-1,-1,-1,-1,-1,-1,
        -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
        -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
        -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
        -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
        -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
    }
```

```

-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
};
char const invalid_char = -1;
char const padding_char = 65;
public:
    std::string to_base64(std::vector<unsigned char> const & data);
    std::vector<unsigned char> from_base64(std::string data);
};

```



Ниже приводится реализация метода `to_base64()`. Он добавляет = или == в конец закодированной строки, чтобы подсказать длину исходных данных:

```

std::string encoder::to_base64(std::vector<unsigned char> const & data)
{
    std::string result;
    result.resize((data.size() / 3 + ((data.size() % 3 > 0) ? 1 : 0)) * 4);
    auto result_ptr = &result[0];
    size_t i = 0;
    size_t j = 0;
    while (j++ < data.size() / 3)
    {
        unsigned int value = (data[i] << 16) | (data[i+1] << 8) | data[i+2];
        i += 3;

        *result_ptr++ = table_enc[(value & 0x00fc0000) >> 18];
        *result_ptr++ = table_enc[(value & 0x0003f000) >> 12];
        *result_ptr++ = table_enc[(value & 0x00000fc0) >> 6];
        *result_ptr++ = table_enc[(value & 0x0000003f)];
    };

    auto rest = data.size() - i;
    if (rest == 1)
    {
        *result_ptr++ = table_enc[(data[i] & 0x000000fc) >> 2];
        *result_ptr++ = table_enc[(data[i] & 0x00000003) << 4];
        *result_ptr++ = padding_symbol;
        *result_ptr++ = padding_symbol;
    }
    else if (rest == 2)
    {
        unsigned int value = (data[i] << 8) | data[i + 1];

        *result_ptr++ = table_enc[(value & 0x0000fc00) >> 10];
        *result_ptr++ = table_enc[(value & 0x000003f0) >> 4];
    }
}

```

```

    *result_ptr++ = table_enc[(value & 0x0000000f) << 2];
    *result_ptr++ = padding_symbol;
}

return result;
}

```



Далее показан метод `from_base64()`. Он обеспечивает декодирование строк с заключительными символами = или без них:

```

std::vector<unsigned char> encoded::from_base64(std::string data)
{
    size_t padding = data.size() % 4;
    if (padding == 0)
    {
        if (data[data.size() - 1] == padding_symbol) padding++;
        if (data[data.size() - 2] == padding_symbol) padding++;
    }
    else
    {
        data.append(2, padding_symbol);
    }
}

```

```

std::vector<unsigned char> result;
result.resize((data.length() / 4) * 3 - padding);
auto result_ptr = &result[0];

```



```

size_t i = 0;
size_t j = 0;
while (j++ < data.size() / 4)
{
    unsigned char c1 = table_dec[static_cast<int>(data[i++])];
    unsigned char c2 = table_dec[static_cast<int>(data[i++])];
    unsigned char c3 = table_dec[static_cast<int>(data[i++])];
    unsigned char c4 = table_dec[static_cast<int>(data[i++])];

    if (c1 == invalid_char || c2 == invalid_char ||
        c3 == invalid_char || c4 == invalid_char)
        throw std::runtime_error("invalid base64 encoding");

    if (c4 == padding_char && c3 == padding_char)
    {
        unsigned int value = (c1 << 6) | c2;
        *result_ptr++ = (value >> 4) & 0x000000ff;
    }
    else if (c4 == padding_char)
    {
        unsigned int value = (c1 << 12) | (c2 << 6) | c3;

```

```

        *result_ptr++ = (value >> 10) & 0x000000ff;
        *result_ptr++ = (value >> 2) & 0x000000ff;
    }
    else
    {
        unsigned int value = (c1 << 18) | (c2 << 12) | (c3 << 6) | c4;

        *result_ptr++ = (value >> 16) & 0x000000ff;
        *result_ptr++ = (value >> 8) & 0x000000ff;
        *result_ptr++ = value & 0x000000ff;
    }
}

return result;
}

```



Поскольку класс `encoder` кодирует двоичные данные в строку `base64` и декодирует строку `base64` в двоичные данные, дополнительно реализован вспомогательный класс для преобразования строки в последовательность байтов и обратно. Класс `converter` имеет два статических метода: `from_string()` принимает объект `string_view` и возвращает вектор байтов с содержимым строки, а `from_range()` конструирует строку из вектора байтов:

```

struct converter
{
    static std::vector<unsigned char> from_string(std::string_view data)
    {
        std::vector<unsigned char> result;

        std::copy(
            std::begin(data), std::end(data),
            std::back_inserter(result));

        return result;
    }

    static std::string from_range(std::vector<unsigned char> const & data)
    {
        std::string result;

        std::copy(
            std::begin(data), std::end(data),
            std::back_inserter(result));

        return result;
    }
};

```



Следующий листинг демонстрирует применение классов `encoder` и `converter` для кодирования и декодирования данных разной длины. Правильность кодирования/декодирования проверяется сравнением с оригиналом:

```
int main()
{
    std::vector<std::vector<unsigned char>> data
    {
        { 's' },
        { 's', 'a' },
        { 's', 'a', 'm' },
        { 's', 'a', 'm', 'p' },
        { 's', 'a', 'm', 'p', 'l' },
        { 's', 'a', 'm', 'p', 'l', 'e' },
    };

    encoder enc;

    for (auto const & v : data)
    {
        auto encv = enc.to_base64(v);

        auto decv = enc.from_base64(encv);

        assert(v == decv);
    }

    auto text = "cppchallenge";
    auto textenc = enc.to_base64(converter::from_string(text));
    auto textdec = converter::from_range(enc.from_base64(textenc));
    assert(text == textdec);
}
```



Несмотря на функциональную полноту, представленная здесь реализация имеет не самую высокую скорость выполнения. Как показали результаты тестирования, она сопоставима с реализацией в *Boost.Beast*. Однако я не могу рекомендовать ее к использованию в промышленном коде. Лучше воспользуйтесь более известными реализациями, например из *Boost.Beast*, *Crypto++* или других библиотек.

## 91. Проверка учетных данных пользователя

Для нужд криптографии хорошим выбором может служить бесплатная кроссплатформенная библиотека на *Crypto++*. Эта библиотека широко ис-

пользуется в коммерческих и некоммерческих проектах, а также в академических и студенческих кругах благодаря ее проверенной и надежной реализации. Библиотека поддерживает AES и другие блочные шифры, кодирование сообщений аутентификации, хеш-функции, криптографические алгоритмы с открытым ключом и множество других возможностей, включая некриптографические функции, такие как генераторы псевдослучайных чисел, генераторы простых чисел, алгоритм сжатия DEFLATE, схемы кодирования, функции проверки контрольных сумм и многое другое. Библиотека доступна по адресу: <https://www.cryptopp.com/> и будет использоваться в этой главе для решения криптографических задач.



В загружаемом пакете вы найдете несколько проектов, соответствующих разным конфигурациям библиотеки. Используйте `cryptolib`, который создает статическую библиотеку. Версия динамической библиотеки, `cryptodll`, была проверена на соответствие требованиям NIST и CSE for FIPS 140-2 Level 1 Conformance. FIPS 140-2 – это группа стандартов компьютерной безопасности правительства США, определяющих требования к криптографической стойкости. Версия `cryptodll` не содержит ничего, что не соответствует этим требованиям, включая алгоритмы DES и MD5.

Для решения задачи смоделируем систему, поддерживающую базу данных с информацией о зарегистрированных пользователях. Каждому пользователю присваивается числовой идентификатор, имя пользователя, хеш-значение пароля, а также (необязательно) имя и фамилия. Для этого используется следующий класс `user`:

```
struct user
{
    int        id;
    std::string username;
    std::string password;
    std::string firstname;
    std::string lastname;
};
```

Вычисление хеш-значения пароля осуществляется функцией `get_hash()`. Она принимает объект `string_view`, представляющий пароль (или любой текст), и возвращает его хеш SHA512. Библиотека *Crypto++* включает множество разных функций хеширования, в том числе SHA-1, SHA-2 (SHA-224, SHA-256, SHA-384 и SHA-512), SHA-3, Tiger, WHIRLPOOL, RIPEMD-128, RIPEMD-256, RIPEMD-160 и RIPEMD-320. В статической версии библиотеки все они определены в пространстве имен `CryptoPP`, а также MD5 (в пространстве имен `CryptoPP::Weak`).



Все эти хеши наследуют класс `HashTransformation` и являются взаимозаменяемыми. Чтобы вычислить хеш, нужно:

- создать объект класса, наследующего `HashTransformation`, например `SHA512`;
- определить массив байтов достаточной длины для хранения хеш-суммы;
- вызвать `CalculateDigest()`, передав буфер для вывода, текст для хеширования и его длину;
- хеш-сумма оригинального текста возвращается в двоичном виде; ее можно преобразовать в текстовое представление с шестнадцатеричными цифрами с помощью класса `HexEncoder`; для накопления вывода можно использовать приемники, такие как `StringSink` или `FileSink`.



Библиотека *Crypto++* использует идею конвейера, передающего данные из источника в приемник. Внутри этого конвейера могут иметься фильтры, преобразующие данные на пути от источника к приемнику. Объекты внутри конвейера принимают право владения другими объектами, которые передаются им по указателям через конструкторы и автоматически уничтожают их, когда уничтожаются сами. Вот цитата из документации к библиотеке: *«Если конструктор объекта A получает указатель на объект B (кроме простых типов, таких как int и char), тогда A автоматически получает право владения объектом B и уничтожит его, когда будет уничтожаться сам. Если конструктор A получает ссылку на объект B, право владения остается за вызывающим кодом, и этот объект не должен уничтожаться, пока A не закончит пользоваться им».*

Ниже приводится реализация функции `get_hash()`:

```
std::string get_hash(std::string_view password)
{
    CryptoPP::SHA512 sha;
    CryptoPP::byte digest[CryptoPP::SHA512::DIGESTSIZE];

    sha.CalculateDigest(
        digest,
        reinterpret_cast<CryptoPP::byte const*>(password.data()),
        password.length());

    CryptoPP::HexEncoder encoder;
    std::string result;

    encoder.Attach(new CryptoPP::StringSink(result));
```

```

encoder.Put(digest, sizeof(digest));
encoder.MessageEnd();

return result;
}

```



Следующая программа моделирует систему входа, используя класс `user` и функцию `get_hash()`. Как можно догадаться по имени, `users` – это список пользователей. В данном случае список жестко «зашит» в код, но его точно так же можно получить из базы данных. Можете реализовать подобную возможность в качестве самостоятельного упражнения и, например, извлекать список пользователей из базы данных SQLite. Когда пользователь введет свое имя и пароль, программа вычислит хеш SHA512 пароля, проверит по списку совпадение имени и пароля и выведет соответствующее сообщение:

```

int main()
{
    std::vector<user> users
    {
        {
            101, "scarface",
            "07A8D53ADAB635ADDF39BAEACFB799FD7C5BFDEE365F3AA721B7E25B54A4E87D419ADDEA34
            BC3073BAC472DCF4657E50C0F6781DDD8FE883653D10F7930E78FF",
            "Tony", "Montana"
        },
        {
            202, "neo",
            "C2CC277BCC10888ECE90F0F09EE9666199C2699922EFB41EA7E88067B2C075F3DD3FBF3CF
            E9D0EC6173668DD83C111342F91E941A2CADC46A3A814848AA9B05",
            "Thomas", "Anderson"
        },
        {
            303, "godfather",
            "0EA7A0306FE00CD22DF1B835796EC32ACC702208E0B052B15F9393BCCF5EE9ECD8BAAF2784
            0D4D3E6BCC3BB3B009259F6F73CC77480C065DDE67CD9BEA14AA4D",
            "Vito", "Corleone"
        }
    };

    std::string username, password;
    std::cout << "Username: ";
    std::cin >> username;
    std::cout << "Password: ";
    std::cin >> password;

    auto hash = get_hash(password);

    auto pos = std::find_if(

```



```

std::begin(users), std::end(users),
[username, hash](user const & u) {
return u.username == username &&
    u.password == hash; });

if (pos != std::end(users))
    std::cout << "Login successful!" << std::endl;
else
    std::cout << "Invalid username or password" << std::endl;
}

```



## 92. Вычисление хеш-суммы файла

Хеш-сумма файла часто используется, чтобы убедиться в целостности содержимого, например после загрузки файла из сети. Реализации хеш-сумм SHA1 и MD5 можно найти во многих библиотеках, но здесь мы опять используем библиотеку *Crypto++*. Если вы еще не решили предыдущую задачу, «Проверка учетных данных пользователя», сделайте это прямо сейчас, прежде чем продолжить знакомиться с решением данной задачи, потому что некоторая важная информация о библиотеке *Crypto++* здесь не повторяется.

Вычисление хеш-суммы файла с помощью библиотеки *Crypto++* реализуется относительно просто. Код решения использует следующие компоненты:

- *FileSource*, позволяет читать данные из файла с помощью *BufferedTransformation*. По умолчанию он «перекачивает» данные блоками по 4096 байт, но позволяет выбрать другой размер блока. Конструктор, использованный в решении, принимает путь к файлу, логический флаг, определяющий необходимость перекачки всех данных, и объект *BufferTransformation*;
- *HashFilter* использует указанный алгоритм для вычисления хеш-суммы для всех данных, вплоть до первого сигнала *MessageEnd*, и выводит получившееся хеш-значение в подключенный буфер;
- *HexEncoder* преобразует двоичные байты в шестнадцатеричное представление, используя алфавит 0123456789ABCDEF;
- *StringSink* представляет приемник – строку для записи результатов конвейера. Он принимает ссылку на объект строки.

```

template <class Hash>
std::string compute_hash(fs::path const & filepath)
{
    std::string digest;
    Hash hash;

    CryptoPP::FileSource source(
        filepath.c_str(),
        true,

```

```

new CryptoPP::HashFilter(hash,
new CryptoPP::HexEncoder(
new CryptoPP::StringSink(digest)));

return digest;
}

```



**BufferedTransformation** – базовый элемент потока данных в *Crypto++*. Он является обобщением *BlockTransformation*, *StreamTransformation* и *HashTransformation*. Объект *BufferedTransformation* принимает на входе поток байтов (и может делать это поэтапно), производит некоторые вычисления с ними и помещает результат во внутренний буфер для последующего использования. Любой частичный результат, уже имеющийся в выходном буфере, не изменяется при получении очередной порции входных данных. Объекты, наследующие *BufferedTransformation*, могут принимать участие в работе конвейера, передающего данные от источника приемнику.

Шаблонную функцию `compute_hash()` из предыдущего листинга можно использовать для вычисления разных хеш-значений, как показано ниже:

```

int main()
{
    std::string path;
    std::cout << "Path: ";
    std::cin >> path;

    try
    {
        std::cout << "SHA1: "
                  << compute_hash<CryptoPP::SHA1>(path) << std::endl;
        std::cout << "SHA256: "
                  << compute_hash<CryptoPP::SHA256>(path) << std::endl;
        std::cout << "MD5: "
                  << compute_hash<CryptoPP::Weak::MD5>(path) << std::endl;
    }
    catch (std::exception const & ex)
    {
        std::cerr << ex.what() << std::endl;
    }
}

```

Важно отметить, что алгоритм хеширования MD5 считается устаревшим и небезопасным и поддерживается только ради обратной совместимости. Чтобы

воспользоваться им, вы должны определить макрос `CRYPTOPP_ENABLE_NAMESPACE_WEAK` перед подключением заголовка `md5.h`, например:

```
#define CRYPTOPP_ENABLE_NAMESPACE_WEAK 1
#include "md5.h"
```

### 93. Шифрование и расшифровывание файлов

Для решения этой задачи с помощью библиотеки *Crypto++* нам понадобятся следующие компоненты:

- `FileSource`, позволяющий читать данные из файла с помощью `BufferedTransformation`. По умолчанию он «перекачивает» данные блоками по 4096 байт, но позволяет выбрать другой размер блока;
- `FileSink`, позволяющий записывать данные в файл с использованием `BufferedTransformation`. Это парный объект-приемник для объекта-источника `FileSource`;
- `DefaultEncryptorWithMAC` и `DefaultDecryptorWithMAC`, шифрующие и расшифровывающие строки и файлы с использованием опознавательного тега для обнаружения вмешательства. По умолчанию они используют блочный шифр AES и алгоритм хеширования SHA256. Каждый следующий вызов этих классов производит разные результаты из-за использования значения смещения, основанного на времени.

Для шифрования и расшифровывания мы реализуем две перегруженные версии функций:

- первая принимает путь к исходному файлу, путь к целевому файлу и пароль; шифрует или дешифрует исходный файл и записывает результат в целевой;
- вторая версия принимает путь к файлу и пароль; шифрует или дешифрует указанный файл, записывает результат во временный файл, удаляет оригинальный файл и затем переименовывает временный файл, присваивая ему имя оригинального. Эта реализация основана на первой перегруженной версии.

Далее приводится функция, выполняющая шифрование:

```
void encrypt_file(fs::path const & sourcefile,
                 fs::path const & destfile,
                 std::string_view password)
{
    CryptoPP::FileSource source(
        sourcefile.c_str(),
        true,
        new CryptoPP::DefaultEncryptorWithMAC(
            (CryptoPP::byte*)password.data(), password.size()),
```

```

        new CryptoPP::FileSink(
            destfile.c_str())
    );
}

void encrypt_file(fs::path const & filepath,
                 std::string_view password)
{
    auto temp_path = fs::temp_directory_path() / filepath.filename();

    encrypt_file(filepath, temp_path, password);

    fs::remove(filepath);
    fs::rename(temp_path, filepath);
}

```

Расшифровывающая функция выглядит очень похожей, но вместо Default-EncryptorWithMAC она использует DefaultDecryptorWithMAC:

```

void decrypt_file(fs::path const & sourcefile,
                 fs::path const & destfile,
                 std::string_view password)
{
    CryptoPP::FileSource source(
        sourcefile.c_str(),
        true,
        new CryptoPP::DefaultDecryptorWithMAC(
            (CryptoPP::byte*)password.data(), password.size(),
            new CryptoPP::FileSink(
                destfile.c_str())
        )
    );
}

void decrypt_file(fs::path const & filepath,
                 std::string_view password)
{
    auto temp_path = fs::temp_directory_path() / filepath.filename();

    decrypt_file(filepath, temp_path, password);

    fs::remove(filepath);
    fs::rename(temp_path, filepath);
}

```

Следующая программа демонстрирует применение этих функций:

```

int main()
{
    encrypt_file("sample.txt", "sample.txt.enc", "cppchallenger");
    decrypt_file("sample.txt.enc", "sample.txt.dec", "cppchallenger");

    encrypt_file("sample.txt", "cppchallenger");
    decrypt_file("sample.txt", "cppchallenger");
}

```



## 94. Подписывание файлов

Процесс создания и проверки подписи напоминает шифрование и расшифровывание, хотя они и имеют фундаментальные отличия: шифрование производится с помощью открытого, а расшифровывание – закрытого ключа, тогда как создание подписи, наоборот, производится с помощью закрытого ключа, а ее проверка – открытого. Подпись дает получателю возможность с помощью открытого ключа убедиться, что полученный файл не был изменен по пути к нему. Наличие открытого ключа недостаточно, чтобы изменить содержимое файла и вновь подписать его. Для решения этой задачи снова воспользуемся библиотекой *Crypto++*.

Хотя для создания подписи и ее проверки можно использовать любую пару из открытого и закрытого RSA-ключей, в реализации, представленной здесь, ключи случайно генерируются программой в момент запуска. Очевидно, что на практике вы будете генерировать ключи независимо от процедуры создания подписи и ее проверки, а не каждый раз, когда вам потребуется поставить подпись. Функция `generate_keys()`, находящаяся в конце следующего листинга, создает открытый и закрытый RSA-ключи размером 3072 бита. Она применяет несколько вспомогательных функций, которые тоже показаны здесь:

```

void encode(fs::path const & filepath,
            CryptoPP::BufferedTransformation const & bt)
{
    CryptoPP::FileSink file(filepath.c_str());
    bt.CopyTo(file);
    file.MessageEnd();
}

void encode_private_key(fs::path const & filepath,
                       CryptoPP::RSA::PrivateKey const & key)
{
    CryptoPP::ByteQueue queue;
    key.DEREncodePrivateKey(queue);
    encode(filepath, queue);
}

void encode_public_key(fs::path const & filepath,
                      CryptoPP::RSA::PublicKey const & key)

```

```

{
    CryptoPP::ByteQueue queue;
    key.DEREncodePublicKey(queue);
    encode(filepath, queue);
}

void decode(fs::path const & filepath,
           CryptoPP::BufferedTransformation& bt)
{
    CryptoPP::FileSource file(filepath.c_str(), true);
    file.TransferTo(bt);
    bt.MessageEnd();
}

void decode_private_key(fs::path const & filepath,
                       CryptoPP::RSA::PrivateKey& key)
{
    CryptoPP::ByteQueue queue;
    decode(filepath, queue);
    key.BERDecodePrivateKey(queue, false, queue.MaxRetrievable());
}

void decode_public_key(fs::path const & filepath,
                      CryptoPP::RSA::PublicKey& key)
{
    CryptoPP::ByteQueue queue;
    decode(filepath, queue);
    key.BERDecodePublicKey(queue, false, queue.MaxRetrievable());
}

void generate_keys(fs::path const & privateKeyPath,
                  fs::path const & publicKeyPath,
                  CryptoPP::RandomNumberGenerator& rng)
{
    try
    {
        CryptoPP::RSA::PrivateKey rsaPrivate;
        rsaPrivate.GenerateRandomWithKeySize(rng, 3072);

        CryptoPP::RSA::PublicKey rsaPublic(rsaPrivate);

        encode_private_key(privateKeyPath, rsaPrivate);
        encode_public_key(publicKeyPath, rsaPublic);
    }
    catch (CryptoPP::Exception const & e)
    {
        std::cerr << e.what() << std::endl;
    }
}

```







Для создания подписи можно использовать конвейер, начинающийся с `FileSource`, заканчивающийся с `FileSink` и содержащий фильтр `SignerFilter`, который создает сигнатуру сообщения. Для преобразования исходных данных он использует класс создания подписи `RSASSA_PKCS1v15_SHA_Signer`:

```
void rsa_sign_file(fs::path const & filepath,
                 fs::path const & privateKeyPath,
                 fs::path const & signaturePath,
                 CryptoPP::RandomNumberGenerator& rng)
{
    CryptoPP::RSA::PrivateKey privateKey;
    decode_private_key(privateKeyPath, privateKey);

    CryptoPP::RSASSA_PKCS1v15_SHA_Signer signer(privateKey);

    CryptoPP::FileSource fileSource(
        filepath.c_str(),
        true,
        new CryptoPP::SignerFilter(
            rng,
            signer,
            new CryptoPP::FileSink(
                signaturePath.c_str())));
}
```

Обратный процесс – проверка – реализуется аналогично. Роль фильтра в данном случае играет класс `SignatureVerificationFilter`, парный классу `SignerFilter`, а проверку осуществляет `RSASSA_PKCS1v15_SHA_Verifier`, парный классу `RSASSA_PKCS1v15_SHA_Signer`:

```
bool rsa_verify_file(fs::path const & filepath,
                   fs::path const & publicKeyPath,
                   fs::path const & signaturePath)
{
    CryptoPP::RSA::PublicKey publicKey;
    decode_public_key(publicKeyPath.c_str(), publicKey);

    CryptoPP::RSASSA_PKCS1v15_SHA_Verifier verifier(publicKey);

    CryptoPP::FileSource signatureFile(signaturePath.c_str(),
                                       true);

    if (signatureFile.MaxRetrievable() != verifier.SignatureLength())
        return false;

    CryptoPP::SecByteBlock signature(verifier.SignatureLength());
    signatureFile.Get(signature, signature.size());
}
```

```

auto* verifierFilter =
    new CryptoPP::SignatureVerificationFilter(verifier);
verifierFilter->Put(signature, verifier.SignatureLength());

CryptoPP::FileSource fileSource(
    filepath.c_str(),
    true,
    verifierFilter);

return verifierFilter->GetLastResult();
}

```



Программа в следующем листинге генерирует открытый и закрытый RSA-ключи, затем использует закрытый ключ для создания подписи, вызывая `rsa_sign_file()`, и проверяет полученную подпись открытым ключом, вызовом парной функции `rsa_verify_file()`:

```

int main()
{
    CryptoPP::AutoSeededRandomPool rng;

    generate_keys("rsa-private.key", "rsa-public.key", rng);

    rsa_sign_file("sample.txt", "rsa-private.key", "sample.sign", rng);

    auto success =
        rsa_verify_file("sample.txt", "rsa-public.key", "sample.sign");

    assert(success);
}

```



---

# Глава 12

## Сети и службы

---



### Задачи

#### 95. Поиск IP-адреса хоста

Напишите программу, которая определяет и выводит IPv4-адрес хоста. Если программа найдет несколько адресов, она должна вывести их все. Программа должна работать на всех платформах.

#### 96. Клиент-серверная игра Fizz-Buzz

Напишите клиент-серверное приложение, которое можно использовать для игры Fizz-Buzz. Клиент посылает числа на сервер и получает в ответ fizz, buzz, fizz-buzz или само число, согласно правилам игры. Взаимодействие клиента и сервера должно осуществляться посредством протокола TCP. Сервер должен выполнять бесконечный цикл, а клиент должен работать, пока пользователь вводит числа в диапазоне от 1 до 99.

Fizz-Buzz – это игра для детей, обучающая арифметическому делению. Один игрок называет число, а другие должны ответить:

- словом «Fizz», если число кратно 3;
- словом «Buzz», если число кратно 5;
- парой слов «Fizz-Buzz», если число кратно и 3, и 5;
- во всех остальных случаях называется само число.

#### 97. Обменный курс биткойна

Напишите программу, отображающую обменный курс биткойна на основные валюты (такие как USD, EUR или GBP). Обменные курсы должны извлекаться из онлайн-службы, например: <https://blockchain.info>.

#### 98. Получение почты по протоколу IMAP

Напишите программу, которая сможет получать информацию с почтового сервера по протоколу IMAP. Программа должна давать возможность получить:

- список папок в почтовом ящике;
- непочитанные письма из указанной папки.

## 99. Перевод текста на любой язык

Напишите программу, выполняющую перевод текста с одного языка на другой с помощью онлайн-службы. Она должна позволять указать текст для перевода, язык текста и язык перевода.



## 100. Определение лиц на изображениях

Напишите программу, способную определять человеческие лица на изображениях. Как минимум программа должна определять область лица на изображении и пол человека. Информация должна выводиться в консоль. Изображения должны загружаться с диска.

# Решения



## 95. Поиск IP-адреса хоста

Информацию о хосте, включая IP-адреса, можно получить с помощью системных утилит сетевой подсистемы, таких как `gethostbyname()`. Они существуют на всех платформах, но используются по-разному, а нам требуется написать программу, которая работала бы на всех платформах. Существуют разные кроссплатформенные библиотеки с открытым исходным кодом для работы с сетевой подсистемой, такие как *POCO* и *Asio/Boost.Asio*. *POCO* – более сложная библиотека, она включает функции не только для работы с сетью, но и для доступа к данным, криптографии, обработки форматов XML, JSON, Zip и др. *Asio* – автономная библиотека, состоящая только из заголовочных файлов со своей моделью асинхронного ввода/вывода для программирования сетевых приложений. Она также входит в состав библиотеки *Boost*, и в настоящий момент проходит проверку предложение о стандартизации на ее основе. В этой книге я буду использовать автономную версию *Asio*, потому что она состоит только из заголовочных файлов и не имеет дополнительных зависимостей, а значит, проще в использовании.

Получить библиотеку *Asio* можно по адресу: <https://think-async.com/>, одна ко последняя ее версия доступна только в репозитории GitHub: <https://github.com/chriskohlhoff/asio/>. Чтобы воспользоваться ею, нужно лишь загрузить и распаковать содержимое репозитория и подключить заголовок `asio.hpp`. Чтобы избавиться от любых зависимостей в *Boost*, определите макрос `ASIO_STANDALONE` перед подключением заголовка библиотеки.

Функция `get_ip_address()`, показанная в листинге ниже, принимает имя хоста и возвращает список строк с адресами IPv4 для этого хоста. В своей работе она использует несколько компонентов из библиотеки *Asio*.

- `asio::io_context` реализует основные функции асинхронного ввода/вывода.



- asio::ip::tcp::resolver реализует преобразование запроса в список конечных точек. Функция-член resolve() этого класса преобразует имена хостов и служб в список конечных точек. В классе определено несколько перегруженных версий этой функции, но в этом решении используется версия, принимающая протокол (в данном случае IPv4, но протокол IPv6 тоже поддерживается), идентификатор хоста (или числовой адрес в виде строки) и идентификатор службы (может быть номером порта). В случае успеха возвращает список конечных точек, в случае неудачи – возбуждает исключение.
- asio::ip::tcp::endpoint представляет конечную точку, которую можно связать с сокетом TCP.

```
#define ASIO_STANDALONE
#include "asio.hpp"

std::vector<std::string> get_ip_address(std::string_view hostname)
{
    std::vector<std::string> ips;

    try
    {
        asio::io_context context;
        asio::ip::tcp::resolver resolver(context);
        auto endpoints = resolver.resolve(asio::ip::tcp::v4(),
                                         hostname.data(), "");

        for (auto e = endpoints.begin(); e != endpoints.end(); ++e)
            ips.push_back(
                ((asio::ip::tcp::endpoint)*e).address().to_string());
    }
    catch (std::exception const & e)
    {
        std::cerr << "exception: " << e.what() << std::endl;
    }

    return ips;
}
```



Следующий листинг демонстрирует применение этой функции:

```
int main()
{
    auto ips = get_ip_address("packtpub.com");
    for (auto const & ip : ips)
        std::cout << ip << std::endl;
}
```



## 96. Клиент-серверная игра Fizz-Buzz

Для решения данной задачи вновь используем библиотеку *Asio*. Но на этот раз напишем две программы: серверную и клиентскую. Сервер принимает соединения TCP на конкретном порте, открывает сокет соединения и начинает читать данные из сокета. Прочитав данные, она интерпретирует их как число для игры Fizz-Buzz, отправляет обратно ответ и переходит в режим ожидания новых данных. Клиент подключается к хосту по указанному порту, посылает число, прочитанное с клавиатуры, и ждет ответа от сервера, который затем выводит в консоль.

Серверная часть игры Fizz-Buzz реализуется просто и не требует дополнительных пояснений. Функция `fizzbuzz()`, показанная в следующем листинге, принимает число и возвращает результат в виде строки:

```
std::string fizzbuzz(int const number)
{
    if(number != 0)
    {
        auto m3 = number % 3;
        auto m5 = number % 5;
        if(m3 == 0 && m5 == 0) return "fizzbuzz";
        else if(m5 == 0) return "buzz";
        else if(m3 == 0) return "fizz";
    }

    return std::to_string(number);
}
```



На стороне сервера используются два основных компонента. Первый – с именем `session` – читает данные из сокета и отправляет ответ обратно в сокет. Он основан на объекте `asio::ip::tcp::socket` и использует его методы `async_read_some()` и `async_write_some()` для чтения и записи данных. Как можно догадаться по именам методов, они действуют асинхронно и вызывают указанного обработчика по завершении операции. После успешного чтения данных из сокета он посылает обратно результат функции `fizzbuzz()`. После успешной записи данных в сокет запускается очередной цикл чтения. Ниже показана реализация класса `session`:

```
#define ASIO_STANDALONE
#include "asio.hpp"

class session : public std::enable_shared_from_this<session>
{
public:
    session(asio::ip::tcp::socket socket) :
        tcp_socket(std::move(socket))
```



```

{ }

void start()
{
    read();
}

private:
void read()
{
    auto self(shared_from_this());

    tcp_socket.async_read_some(
        asio::buffer(data, data.size()),
        [this, self](std::error_code const ec, std::size_t const length){
            if (!ec)
            {
                auto number = std::string(data.data(), length);
                auto result = fizzbuzz(std::atoi(number.c_str()));
                std::cout << number << " -> " << result << std::endl;
                write(result);
            }
        });
}

void write(std::string_view response)
{
    auto self(shared_from_this());

    tcp_socket.async_write_some(
        asio::buffer(response.data(), response.length()),
        [this, self](std::error_code const ec, std::size_t const) {
            if (!ec)
                read();
        });
}

std::array<char, 1024> data;
asio::ip::tcp::socket tcp_socket;
};

```



Другой компонент, который мы напишем, используется для приема входящих соединений. Он называется `server` и для приема новых соединений использует `asio::ip::tcp::acceptor`. После успешного создания нового сокета он создает объект `session` и вызывает его метод `start()`, чтобы запустить процесс чтения данных, полученных от клиента:



```
class server
{
public:
    server(asio::io_context& context, short const port)
        : tcp_acceptor(context,
            asio::ip::tcp::endpoint(asio::ip::tcp::v4(), port))
        , tcp_socket(context)
    {
        std::cout << "server running on port " << port << std::endl;

        accept();
    }

private:
    void accept()
    {
        tcp_acceptor.async_accept(tcp_socket, [this](std::error_code ec)
        {
            if (!ec)
                std::make_shared<session>(std::move(tcp_socket))->start();

            accept();
        });
    }

    asio::ip::tcp::acceptor tcp_acceptor;
    asio::ip::tcp::socket tcp_socket;
};
```

Следующая далее функция `run_server()` создает объект `asio::io_context` и экземпляр класса `server`, который немедленно начинает принимать соединения, и вызывает метод `run()` объекта контекста. Он выполняет цикл обработки событий до завершения всей работы, пока остается хотя бы один обработчик или пока принудительно не будет остановлен объект `asio::io_context` вызовом метода `stop()`. Функция `run_server()` выполняется бесконечно, пока не возникнет исключение:

```
void run_server(short const port)
{
    try
    {
        asio::io_context context;

        server srv(context, port);

        context.run();
    }
}
```



```

    catch (std::exception& e)
    {
        std::cerr << "exception: " << e.what() << std::endl;
    }
}

int main()
{
    run_server(11234);
}

```



Реализация на стороне клиента немного проще. Для соединения с сервером используется `asio::connect()`. После установки соединения вызываются синхронные методы `write_some()` и `read_some()` объекта `asio::ip::tcp::socket`, чтобы послать данные серверу и прочитать ответ. Они выполняются в цикле, который производит прием данных пользователя с клавиатуры. Цикл длится, пока пользователь вводит числа в диапазоне от 1 до 99. Все это реализует функция `run_client()`:

```

void run_client(std::string_view host, short const port)
{
    try
    {
        asio::io_context context;
        asio::ip::tcp::socket tcp_socket(context);
        asio::ip::tcp::resolver resolver(context);
        asio::connect(tcp_socket,
                      resolver.resolve({ host.data(),
                                         std::to_string(port) }));

        while (true)
        {
            std::cout << "number [1-99]: ";

            int number;
            std::cin >> number;
            if (std::cin.fail() || number < 1 || number > 99)
                break;

            auto request = std::to_string(number);
            tcp_socket.write_some(asio::buffer(request, request.length()));

            std::array<char, 1024> reply;
            auto reply_length = tcp_socket.read_some(
                asio::buffer(reply, reply.size()));

            std::cout << "reply is: ";
        }
    }
}

```

```

        std::cout.write(reply.data(), reply_length);
        std::cout << std::endl;
    }
}
catch (std::exception& e)
{
    std::cerr << "exception: " << e.what() << std::endl;
}
}

int main()
{
    run_client("localhost", 11234);
}

```



Скриншот на рис. 12.1 показывает содержимое консоли на стороне сервера (слева) и на стороне клиента (справа).

cmd.exe C:\WINDOWS\system32\cmd.exe	cmd.exe C:\WINDOWS\system32\cmd.exe
server running on port 11234	number [1-99]: 1
1 -> 1	reply is: 1
2 -> 2	number [1-99]: 2
3 -> fizz	reply is: 2
4 -> 4	number [1-99]: 3
5 -> buzz	reply is: fizz
6 -> fizz	number [1-99]: 4
7 -> 7	reply is: 4
15 -> fizzbuzz	number [1-99]: 5
99 -> fizz	reply is: buzz
	number [1-99]: 6
	reply is: fizz
	number [1-99]: 7
	reply is: 7
	number [1-99]: 15
	reply is: fizzbuzz
	number [1-99]: 99
	reply is: fizz
	number [1-99]:

Рис. 12.1. Вывод на стороне сервера и на стороне клиента

## 97. Обменный курс биткойна

Возможность узнать обменный курс биткойна предоставляется многими веб-службами. Для решения этой задачи мы будем использовать бесплатную службу, доступную по адресу: <https://blockchain.info/ticker>. Запрос GET HTTP возвращает объект JSON с курсом биткойна относительно разных валют. Описание API службы можно найти по адресу: [https://blockchain.info/api/exchange\\_rates\\_api](https://blockchain.info/api/exchange_rates_api). Вот пример объекта JSON:



```
{
  "USD": {
    "15m": 8196.491155299998,
    "last": 8196.491155299998,
    "buy": 8196.491155299998,
    "sell": 8196.491155299998,
    "symbol": "$"
  },
  "GBP": {
    "15m": 5876.884158350099,
    "last": 5876.884158350099,
    "buy": 5876.884158350099,
    "sell": 5876.884158350099,
    "symbol": "£"
  }
}
```

Для передачи данных по сети можно использовать множество разных библиотек. Одна из них – библиотека *curl*, пользующаяся большой популярностью. Проект включает инструмент командной строки (*cURL*) и библиотеку (*libcurl*), написанные на языке С и поддерживающие широкий спектр протоколов, включая HTTP/HTTPS, FTP/FTPS, Gopher, LDAP/LDAPS, POP3/POP3S и SMTP/SMTPS. Главная страница проекта: <https://curl.haxx.se/>. Также есть несколько библиотек на С++, написанных поверх *libcurl*. В том числе открытая и кроссплатформенная библиотека *curlcpp*, которую написал Джузеппе Персико (Giuseppe Persico): <https://github.com/JosephP91/curlcpp>. Мы будем использовать эти две библиотеки для решения данной задачи и одну из них – для решения следующей.

Инструкции по сборке библиотек *libcurl* и *curlcpp* можно найти в документации этих двух проектов. В пакете с исходным кодом примеров к книге вы найдете уже готовые сценарии CMake. Если вы решите собрать библиотеки самостоятельно, для использования в других проектах, тогда вам придется поступить немного иначе, в зависимости от платформы, для которой осуществляется сборка. Далее даются краткие инструкции по сборке библиотек с отладочной информацией для Windows и macOS.

В Windows, используя Visual Studio 2017, выполните следующие шаги:

1. Загрузите исходный код *cURL* (<https://curl.haxx.se/download.html>), распакуйте архив и отыщите файл решения для Visual Studio (`projects\Windows\VC15\curl-all.sln`). Откройте решение и выполните сборку конфигураций LIB Debug - DLL Windows SSPI для целевой платформы (Win32 или x64). В результате будет создан файл статической библиотеки `libcurl.lib`.
2. Загрузите *curlcpp* (<https://github.com/JosephP91/curlcpp>), создайте папку `build` и запустите сценарий CMake из нее, определив переменные `CURL_`

LIBRARY и CURL\_INCLUDE\_DIR. Первая должна содержать путь к `libcurl.lib`, а вторая – путь к папке с заголовками CURL. Откройте сгенерированный проект и соберите его. В результате будет создан файл статической библиотеки `curlcpp.lib`.

3. В проекте для Visual Studio, где вы собираетесь использовать `curlcpp`, добавьте определение `CURL_STATICLIB` для препроцессора, указав путь к папкам `curl\include` и `curlcpp\include` в списке **Additional Include Directories** и пути к папкам с библиотеками в списке **Additional Library Directories**. Наконец, скомпонуйте проект со следующими статическими библиотеками: `libcurl.lib`, `curlcpp.lib`, `Crypt32.lib`, `ws2_32.lib`, `winmm.lib` и `wldap32.lib`.

В macOS, используя Xcode, выполните следующие шаги:

1. Загрузите исходный код `openssl` (<https://www.openssl.org/>), распакуйте архив и выполните следующие команды, чтобы произвести сборку и установку:

```
./Configure darwin64-x86_64-cc shared enableec_
nisp_64_gcc_128 no-ssl2 no-ssl3 no-comp --
openssldir=/usr/local/ssl/macos-x86_64
```

```
make depend
```

```
sudo make install
```



2. Загрузите исходный код `cURL` (<https://curl.haxx.se/download.html>), распакуйте архив и создайте папку `build`, откуда запустите сценарий CMake, определив переменные `OPENSSL_ROOT_DIR` и `OPENSSL_INCLUDE_DIR`, ссылающиеся на `openssl`. Если вы решите запретить компиляцию тестов и создание документации, определите переменные `BUILD_TESTING`, `BUILD_CURL_EXE` и `USE_MANUAL` со значением `OFF`. В результате сборки с отладочной информацией получится файл `libcurl-d.dylib`:

```
cmake -G Xcode .. -DOPENSSL_ROOT_DIR=/usr/local/bin -
DOPENSSL_INCLUDE_DIR=/usr/local/include/
```

3. Загрузите `curlcpp` (<https://github.com/JosephP91/curlcpp>), создайте папку `build` и запустите сценарий CMake из нее, определив переменные `CURL_LIBRARY` и `CURL_INCLUDE_DIR`. Первая должна содержать путь к `libcurl-d.dylib`, а вторая – путь к папке с заголовками CURL. Откройте сгенерированный проект и соберите его. В результате будет создан файл `libcurlcpp.a`:

```
cmake -G Xcode .. -DCURL_LIBRARY=<path>/curl-
7.59.0/build/lib/Debug/libcurl-d.dylib -DCURL_INCLUDE_DIR=
<path>/curl-7.59.0/include
```

4. В проекте для Xcode, где вы собираетесь использовать *cURL* и *curlcpp*, добавьте определение `CURL_STATICLIB` для препроцессора, указав путь к папкам `curl\include` и `curlcpp\include` в списке **Header Search Paths**, пути к папкам с библиотеками в списке **Library Search Paths** и две статические библиотеки, `libcurl-d.dylib` и `libcurlcpp.a`, в список **Link Binary With Libraries**.

Библиотека *libcurl* поддерживает две модели программирования (их еще называют интерфейсами): *easy* и *multi*. Интерфейс *easy* реализует синхронную, эффективную и простую модель программирования для передачи данных. Интерфейс *multi* реализует асинхронную модель передачи данных с использованием одного или нескольких потоков выполнения.

Используя интерфейс *easy*, вы сначала должны инициализировать сеанс, затем определить разные параметры, включая URL и, возможно, функции обратного вызова, которые будут вызываться автоматически, как только данные станут доступны. По завершении настройки можно выполнить передачу данных, которая блокирует выполнение программы до завершения передачи. После передачи данных можно получить информацию о ней, и в конце необходимо остановить сеанс. Инициализация и освобождение ресурсов производятся в соответствии с идиомой RAII библиотеки *curlcpp*.

Следующая функция `get_json_document()` принимает URL и выполняет запрос HTTP GET. Ответ сервера записывается в `std::stringstream` и возвращается вызывающему коду:

```
#include "curl_easy.h"
#include "curl_form.h"
#include "curl_ios.h"
#include "curl_exception.h"

std::stringstream get_json_document(std::string_view url)
{
    std::stringstream str;

    try
    {
        curl::curl_ios<std::stringstream> writer(str);
        curl::curl_easy easy(writer);

        easy.add<CURLLOPT_URL>(url.data());
        easy.add<CURLLOPT_FOLLOWLOCATION>(1L);

        easy.perform();
    }
    catch (curl::curl_easy_exception const & error)
    {
        auto errors = error.get_traceback();
    }
}
```

```

    error.print_traceback();
}

return str;
}

```



В ответ на запрос HTTP GET по адресу <https://blockchain.info/ticker> мы получим объект JSON, как показано выше. Для представления этих данных в программе мы будем использовать следующие типы:

```

struct exchange_info
{
    double    delay_15m_price;
    double    latest_price;
    double    buying_price;
    double    selling_price;
    std::string symbol;
};

```



```
using blockchain_rates = std::map<std::string, exchange_info>;
```

Для интерпретации данных JSON можно использовать библиотеку *nlohmann/json*. Подробнее об этом рассказывается в главе 9 «Сериализация данных». Следующая функция `from_json()` извлекает объект `exchange_info` из данных JSON:

```

#include "json.hpp"

using json = nlohmann::json;

void from_json(const json& jdata, exchange_info& info)
{
    info.delay_15m_price = jdata.at("15m").get<double>();
    info.latest_price    = jdata.at("last").get<double>();
    info.buying_price    = jdata.at("buy").get<double>();
    info.selling_price   = jdata.at("sell").get<double>();
    info.symbol          = jdata.at("symbol").get<std::string>();
}

```

Теперь объединим все вместе и напишем программу, которая получает с сервера информацию об обменном курсе, преобразует ответ в формате JSON и выводит курсы в консоль:

```

int main()
{
    auto doc = get_json_document("https://blockchain.info/ticker");

    json jdata;

```

```

doc >> jdata;

blockchain_rates rates = jdata;
for (auto const & kvp : rates)
{
    std::cout << "1BPI = " << kvp.second.latest_price
                << " " << kvp.first << std::endl;
}
}
}

```

## 98. Получение почты по протоколу IMAP

**Протокол доступа к сообщениям в сети Интернет** (Internet Message Access Protocol, IMAP) – протокол получения сообщений с сервера электронной почты с использованием TCP/IP. Большинство поставщиков услуг электронной почты, включая таких крупных, как Gmail, Outlook.com и Yahoo! Mail, предлагают поддержку этого протокола. Для работы с этим протоколом есть несколько библиотек на C++, таких как VMIME – кроссплатформенная библиотека с открытым исходным кодом и с поддержкой IMAP, POP и SMTP. Однако в этой книге я буду использовать *cURL* (точнее, *libcurl*) и с ее помощью посылать запросы HTTP серверу электронной почты, поддерживающему IMAPS.

Требуемые нам операции можно выполнить с помощью нескольких команд IMAP. В следующем списке адрес `imap.domain.com` служит только примером.

- GET `imaps://imap.domain.com` вернет все папки в почтовом ящике. Чтобы получить подпапки из определенной папки, например `inbox`, нужно выполнить команду GET `imaps://imap.domain.com/<foldername>`.
- SEARCH UNSEEN `imaps://imap.domain.com/<foldername>` извлекает идентификаторы всех неп прочитанных электронных писем из папки `foldername`.
- GET `imaps://imap.domain.com/<foldername>;UID=<id>` извлекает электронное письмо с указанным идентификатором `id` из папки `foldername`.



Клиенты Gmail, Outlook.com или Yahoo! Mail используют очень похожие настройки IMAP: порт 933 с шифрованием TLS; вместо имени пользователя используется электронный адрес, а паролем служит пароль учетной записи. Отличается только имя хоста. Для Gmail – это `imap.gmail.com`, для Outlook.com – `imapmail.outlook.com`, и для Yahoo! Mail – `imap.mail.yahoo.com`. Помните, что если вы используете двухфакторную аутентификацию, тогда вам придется сгенерировать пароль для третьего приложения и использовать его вместо пароля учетной записи.

Описанные возможности реализованы в следующем фрагменте кода в виде функций-членов класса `imap_connection`. При создании экземпляров этого класса конструктору передается URL сервера, номер порта, имя пользователя и пароль. Вспомогательный метод `setup_easy()` инициализирует объект `curl::curl_easy` параметрами аутентификации, такими как порт, имя пользователя и пароль, признаком необходимости использовать шифрование TLS, а также другими типичными настройками, такими как пользовательский агент (необязательно):

```
class imap_connection
{
public:
    imap_connection(std::string_view url,
                    unsigned short const port,
                    std::string_view user,
                    std::string_view pass):
        url(url), port(port), user(user), pass(pass)
    {
    }

    std::string get_folders();
    std::vector<unsigned int> fetch_unread_uids(std::string_view folder);
    std::string fetch_email(std::string_view folder, unsigned int uid);

private:
    void setup_easy(curl::curl_easy& easy)
    {
        easy.add<CURLOPT_PORT>(port);
        easy.add<CURLOPT_USERNAME>(user.c_str());
        easy.add<CURLOPT_PASSWORD>(pass.c_str());
        easy.add<CURLOPT_USE_SSL>(CURLUSESSL_ALL);
        easy.add<CURLOPT_SSL_VERIFYPEER>(0L);
        easy.add<CURLOPT_SSL_VERIFYHOST>(0L);
        easy.add<CURLOPT_USERAGENT>("libcurl-agent/1.0");
    }

private:
    std::string url;
    unsigned short port;
    std::string user;
    std::string pass;
};
```



Метод `get_folders()` возвращает список папок в почтовом ящике в виде строки, полученной от сервера, без всякой дополнительной обработки. Вы можете сами попробовать усовершенствовать ее и возвращать список папок в виде списка. Функция создает объект `curl::curl_easy`, инициализирует его



соответствующими параметрами, такими как URL и сведения для аутентификации, выполняет запрос и возвращает результат, полученный от сервера в форме `std::stringstream`:

```
std::string imap_connection::get_folders()
{
    std::stringstream str;
    try
    {
        curl::curl_ios<std::stringstream> writer(str);

        curl::curl_easy easy(writer);
        easy.add<CURLLOPT_URL>(url.data());
        setup_easy(easy);

        easy.perform();
    }
    catch (curl::curl_easy_exception const & error)
    {
        auto errors = error.get_traceback();
        error.print_traceback();
    }

    return str.str();
}
```



Вот как мог бы выглядеть результат этой функции:

```
* LIST (\HasNoChildren) "/" "INBOX"
* LIST (\HasNoChildren) "/" "Notes"
* LIST (\HasNoChildren) "/" "Trash"
* LIST (\HasChildren \Noselect) "/" "[Gmail]"
* LIST (\All \HasNoChildren) "/" "[Gmail]/All Mail"
* LIST (\Drafts \HasNoChildren) "/" "[Gmail]/Drafts"
* LIST (\HasNoChildren \Important) "/" "[Gmail]/Important"
* LIST (\HasNoChildren \Sent) "/" "[Gmail]/Sent Mail"
* LIST (\HasNoChildren \Junk) "/" "[Gmail]/Spam"
* LIST (\Flagged \HasNoChildren) "/" "[Gmail]/Starred"
```

Метод `fetch_unread_uids()` устроен очень похоже. Эта функция возвращает вектор целых чисел без знака, представляющих идентификаторы непрочитанных электронных писем из указанной папки. Она, как и предыдущая функция, выполняет запрос, но, в отличие от нее, создает список идентификаторов электронных писем. Она также присваивает параметру `CURLLOPT_CUSTOMREQUEST` значение "SEARCH UNSEEN". В результате вместо HTTP-метода GET по умолчанию используется другой HTTP-метод (в данном случае SEARCH):



```

std::vector<unsigned int>
imap_connection::fetch_unread_uids(std::string_view folder)
{
    std::stringstream str;

    try
    {
        curl::curl_ios<std::stringstream> writer(str);

        curl::curl_easy easy(writer);
        easy.add<CURLLOPT_URL>((url.data() + std::string("/") +
                               folder.data() + std::string("/")).c_str());
        easy.add<CURLLOPT_CUSTOMREQUEST>("SEARCH UNSEEN");
        setup_easy(easy);

        easy.perform();
    }
    catch (curl::curl_easy_exception const & error)
    {
        auto errors = error.get_traceback();
        error.print_traceback();
    }

    std::vector<unsigned int> uids;
    str.seekg(8, std::ios::beg);
    unsigned int uid;
    while (str >> uid)
        uids.push_back(uid);

    return uids;
}

```



Последний метод, который мы реализуем, – `fetch_email()`. Он принимает имя папки с идентификатором электронного письма и возвращает письмо в виде строки:

```

std::string imap_connection::fetch_email(std::string_view folder,
                                         unsigned int uid)
{
    std::stringstream str;

    try
    {
        curl::curl_ios<std::stringstream> writer(str);

        curl::curl_easy easy(writer);
        easy.add<CURLLOPT_URL>((url.data() + std::string("/") +
                               folder.data() + std::string("/;UID=") +

```

```

        std::to_string(uid)).c_str());
    setup_easy(easy);

    easy.perform();
}
catch (curl::curl_easy_exception error)
{
    auto errors = error.get_traceback();
    error.print_traceback();
}

return str.str();
}

```



В следующем листинге показано, как использовать этот класс для извлечения контента. Здесь мы читаем имена папок в почтовом ящике, затем получаем идентификаторы всех непрочитанных электронных писем из папки `inbox` и, если таковые существуют, извлекаем и отображаем наиболее свежее:

```

int main()
{
    imap_connection imap("imaps://imap.gmail.com",
                        993,
                        "...(your username)...",
                        "...(your password)...");

    auto folders = imap.get_folders();
    std::cout << folders << std::endl;

    auto uids = imap.fetch_unread_uids("inbox");

    if (!uids.empty())
    {
        auto email = imap.fetch_email("inbox", uids.back());
        std::cout << email << std::endl;
    }
}

```



## 99. Перевод текста на любой язык

Возможность перевода текста с одного языка на другой предоставляется многими облачными службами, включая Microsoft Cognitive Services, Google Cloud Translation API и Amazon Translate. Для решения этой задачи я буду использовать Microsoft Azure Cognitive Services. Azure Cognitive Services – это коллекция алгоритмов машинного обучения и искусственного интеллекта, которые можно использовать для добавления в приложения интеллектуальных функций. Одной из служб в этой коллекции является *Text Translate API*, кото-

рая поддерживает такие возможности, как определение языка, перевод с одного языка на другой и преобразование текста в речь. Для выполнения HTTP-запросов мы снова используем библиотеку *libcurl*.

Служба Text Translate API предлагает разные тарифные планы, но также имеется бесплатный тариф. Он позволяет переводить до двух миллионов символов в месяц, чего вполне достаточно для демонстрационных целей и разработки прототипов. Прежде чем получить доступ к этой службе, вы должны:

1. Создать учетную запись Azure, если у вас ее еще нет.
2. Создать новый ресурс Translator Text API.
3. После создания ресурса перейти в него и скопировать один из двух сгенерированных ключей. Этот ключ необходим для обращений к службе.
4. Конечная точка службы имеет адрес <https://api.microsofttranslator.com/V2/Http.svc>, а не тот, что отображается в форме обзора ресурса.

Документацию с описанием API службы перевода текста можно найти по адресу: <http://docs.microsofttranslator.com/text-translate.html>. Чтобы выполнить перевод текста, необходимо:

1. Создать GET-запрос с адресом [конечная\_точка]/Translate.
2. Определить необходимые параметры запроса (*text* и *to*). Также можно определить необязательный параметр *from*, в котором указать язык оригинала, которым по умолчанию считается английский. Текст для перевода должен содержать не более 10 000 символов и экранировать символы, считающиеся специальными в адресах URL.
3. Создать необходимые заголовки. Обязательно должен быть создан заголовок *Оsc-Apim-Subscription-Key* для передачи ключа приложения.

Например, вот как выглядит GET-запрос для перевода текста "hello world!" с английского на французский:

```
GET /V2/Http.svc/Translate?to=fr&text=hello%20world%21
host: api.microsofttranslator.com
osc-apim-subscription-key: <ключ_приложения>
```

В случае успеха служба вернет документ XML с переведенным текстом. Текст имеет кодировку UTF-8. В настоящее время формат JSON не поддерживается. Ниже приводится ответ сервера на запрос выше:

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">Salut
tout le monde !</string>
```

Мы можем инкапсулировать функции перевода текста в класс, автоматически выполняющий передачу ключа и подставляющий адрес конечной точки. Именно эти возможности предоставляет следующий класс `text_translator`. При



```

{
try
{
using namespace std::string_literals;

std::stringstream str;
std::string text = utf16_to_utf8(wtext);

curl::curl_ios<std::stringstream> writer(str);
curl::curl_easy easy(writer);

curl::curl_header header;
header.add("Ocp-Apim-Subscription-Key:" + app_key);

easy.escape(text);
auto url = endpoint + "/Translate";
url += "?from="s + from.data();
url += "&to="s + to.data();
url += "&text="s + text;

easy.add<CURLLOPT_URL>(url.c_str());
easy.add<CURLLOPT_HTTPHEADER>(header.get());

easy.perform();

auto result = deserialize_result(str.str());
return utf8_to_utf16(result);
}
catch (curl::curl_easy_exception const & error)
{
auto errors = error.get_traceback();
error.print_traceback();
}
catch (std::exception const & ex)
{
std::err << ex.what() << std::endl;
}

return {};
}

```



Вот две вспомогательные функции для преобразования между кодировками UTF-8 и UTF-16:

```

std::wstring utf8_to_utf16(std::string_view text)
{
std::wstring_convert<std::codecvt_utf8_utf16<wchar_t>> converter;
std::wstring wtext = converter.from_bytes(text.data());

```



```

    return wtext;
}

std::string utf16_to_utf8(std::wstring_view wtext)
{
    std::wstring_convert<std::codecvt_utf8_utf16<wchar_t>> converter;
    std::string text = converter.to_bytes(wtext.data());
    return text;
}

```

Следующий листинг демонстрирует применение класса `text_translator` для перевода текста с одного языка на другой:

```

int main()
{
#ifdef _WIN32
    SetConsoleOutputCP(CP_UTF8);
#endif

    set_utf8_conversion(std::wcout);

    text_translator tt(
        "https://api.microsofttranslator.com/V2/Http.svc",
        "...(your app key)...");

    std::vector<std::tuple<std::wstring, std::string, std::string>> texts
    {
        { L"hello world!", "en", "ro"},
        { L"what time is it?", "en", "es" },
        { L"ceci est un exemple", "fr", "en" }
    };

    for (auto const [text, from, to] : texts)
    {
        auto result = tt.translate_text(text, to, from);

        std::cout << from << ": ";
        std::wcout << text << std::endl;
        std::cout << to << ": ";
        std::wcout << result << std::endl;
    }
}

```



Однако вывести в консоль символы в кодировке UTF-8 не так просто. В Windows для этого нужно вызвать `SetConsoleOutputCP(CP_UTF8)`, чтобы включить соответствующую кодовую страницу. Но также можно выбрать соответствующую локаль UTF-8 для потока вывода, что поможет сделать функция `set_utf8_conversion()`:

```
void set_utf8_conversion(std::wostream& stream)
{
    auto codecvt = std::make_unique<std::codecvt_utf8<wchar_t>>();
    std::locale utf8locale(std::locale(), codecvt.get());
    codecvt.release();
    stream.imbue(utf8locale);
}
```



На рис. 12.2 показано, как выглядит вывод этой программы.

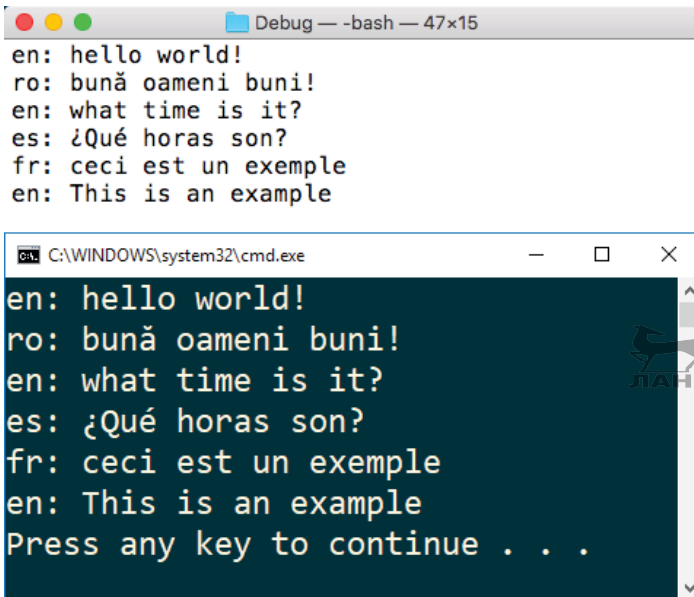


Рис. 12.2. Вывод результата перевода в консоль

## 100. Определение лиц на изображениях

Это еще одна задача, которая легко решается с помощью служб Microsoft Cognitive Services. В частности, в этой группе доступна служба *Face API*, реализующая алгоритмы определения лиц, пола, возраста, эмоций и других признаков, а также позволяющая отыскивать похожие лица, идентифицировать людей, группировать изображения на основе визуальных сходств лиц и других признаков.

По аналогии с Text Translate API, есть бесплатный тариф, допускающий до 30 000 транзакций в месяц, но не более 20 в минуту. Под транзакцией понимается одно обращение к API. Есть также несколько платных тарифов с увеличенным числом транзакций в месяц и в минуту, но для решения этой задачи нам вполне подойдет бесплатный тариф. Есть также тариф с 30-дневным пробным периодом. Чтобы начать пользоваться службой Face API, вы должны:



1. Создать учетную запись Azure, если у вас ее еще нет.
2. Создать новый ресурс Face API.
3. После создания ресурса перейти в него и скопировать один из двух сгенерированных ключей. Они оба потребуются для обращений к службе.

Документация с описанием Face API доступна по адресу: <https://azure.microsoft.com/ru-ru/services/cognitive-services/face/>. Внимательно прочитайте описание метода Detect. Если говорить в двух словах, вот что мы должны сделать:

- создать запрос POST с адресом [конечная\_точка]/Detect;
- определить необходимые параметры запроса (флаги), чтобы получить идентификатор лица, характерные признаки лица и строку, определяющую, какие атрибуты должны анализироваться и возвращаться;
- создать обязательные и необязательные заголовки запроса. Обязательно должен быть создан заголовок Ocp-Apim-Subscription-Key для передачи ключа ресурса в Azure;
- определить изображение для анализа. По желанию можно передать URL изображения в виде объекта JSON (с типом application/json) или само изображение (с типом application/octet-stream). По условиям задачи изображение должно загружаться с диска, поэтому будем использовать второй вариант.

В случае успеха служба вернет объект JSON со всей запрошенной информацией. В случае неудачи в объекте JSON будет возвращена информация с описанием ошибки.

Вот как выглядит запрос на анализ и возврат признаков лица, возраста, пола и эмоций, а также идентификатора лица. Идентификатор лица хранится на сервере 24 часа и может использоваться в других алгоритмах Face API:

```
POST
/face/v1.0/detect?returnFaceId=true&returnFaceLandmarks=true&
returnFaceAttributes=age,gender,emotion
host: westeurope.api.cognitive.microsoft.com
ocp-apim-subscription-key: <ключ_приложения>
content-type: application/octet-stream
content-length: <length>
accept: */*
```

Далее приводится пример объекта JSON, возвращаемого сервером. Имейте в виду, что это лишь часть объекта, потому что полный ответ имеет довольно большой размер. Фактический результат включает 27 разных признаков лица, но в следующем листинге показаны только два из них:

```
[{
  "faceId": "0ddb348a-6038-4cbb-b3a1-86fffe6c1f26",
  "faceRectangle": {
    "top": 86,
    "left": 165,
    "width": 72,
    "height": 72
  },
  "faceLandmarks": {
    "pupilLeft": {
      "x": 187.5,
      "y": 102.9
    },
    "pupilRight": {
      "x": 214.6,
      "y": 104.7
    }
  },
  "faceAttributes": {
    "gender": "male",
    "age": 54.9,
    "emotion": {
      "anger": 0,
      "contempt": 0,
      "disgust": 0,
      "fear": 0,
      "happiness": 1,
      "neutral": 0,
      "sadness": 0,
      "surprise": 0
    }
  }
}]
```



Для десериализации объекта JSON используем библиотеку *nlohmann/json*, а для отправки HTTP-запроса – библиотеку *libcurl*. Следующие классы моделируют результат в случае успешного анализа изображения на стороне сервера:

```
struct face_rectangle
{
  int width = 0;
  int height = 0;
  int left = 0;
  int top = 0;
};

struct face_point
{
```



```
double x = 0;
double y = 0;
};

struct face_landmarks
{
    face_point pupilLeft;
    face_point pupilRight;
    face_point noseTip;
    face_point mouthLeft;
    face_point mouthRight;
    face_point eyebrowLeftOuter;
    face_point eyebrowLeftInner;
    face_point eyeLeftOuter;
    face_point eyeLeftTop;
    face_point eyeLeftBottom;
    face_point eyeLeftInner;
    face_point eyebrowRightInner;
    face_point eyebrowRightOuter;
    face_point eyeRightInner;
    face_point eyeRightTop;
    face_point eyeRightBottom;
    face_point eyeRightOuter;
    face_point noseRootLeft;
    face_point noseRootRight;
    face_point noseLeftAlarTop;
    face_point noseRightAlarTop;
    face_point noseLeftAlarOutTip;
    face_point noseRightAlarOutTip;
    face_point upperLipTop;
    face_point upperLipBottom;
    face_point underLipTop;
    face_point underLipBottom;
};

struct face_emotion
{
    double anger = 0;
    double contempt = 0;
    double disgust = 0;
    double fear = 0;
    double happiness = 0;
    double neutral = 0;
    double sadness = 0;
    double surprise = 0;
};

struct face_attributes
```



```

{
    std::string  gender;
    double      age;
    face_emotion emotion;
};

struct face_info
{
    std::string  faceId;
    face_rectangle  rectangle;
    face_landmarks  landmarks;
    face_attributes  attributes;
};

```



Изображение может содержать несколько лиц, поэтому на самом деле сервер может вернуть массив объектов. Тип `face_detect_response`, показанный ниже, определяет фактический тип ответа:

```
using face_detect_response = std::vector<face_info>;
```

Десериализация выполняется точно так же, как в других задачах в этой книге, с использованием перегруженных функций `from_json()`. Если вы внимательно прочитали решения других задач, где выполняется десериализация JSON, следующий код покажется вам очень знакомым:

```

using json = nlohmann::json;

void from_json(const json& jdata, face_rectangle& rect)
{
    rect.width = jdata.at("width").get<int>();
    rect.height = jdata.at("height").get<int>();
    rect.top = jdata.at("top").get<int>();
    rect.left = jdata.at("left").get<int>();
}

void from_json(const json& jdata, face_point& point)
{
    point.x = jdata.at("x").get<double>();
    point.y = jdata.at("y").get<double>();
}

void from_json(const json& jdata, face_landmarks& mark)
{
    mark.pupilLeft = jdata.at("pupilLeft");
    mark.pupilRight = jdata.at("pupilRight");
    mark.noseTip = jdata.at("noseTip");
    mark.mouthLeft = jdata.at("mouthLeft");
    mark.mouthRight = jdata.at("mouthRight");
    mark.eyebrowLeftOuter = jdata.at("eyebrowLeftOuter");
}

```



```

mark.eyebrowLeftInner = jdata.at("eyebrowLeftInner");
mark.eyeLeftOuter = jdata.at("eyeLeftOuter");
mark.eyeLeftTop = jdata.at("eyeLeftTop");
mark.eyeLeftBottom = jdata.at("eyeLeftBottom");
mark.eyebrowRightInner = jdata.at("eyebrowRightInner");
mark.eyebrowRightOuter = jdata.at("eyebrowRightOuter");
mark.eyeRightInner = jdata.at("eyeRightInner");
mark.eyeRightTop = jdata.at("eyeRightTop");
mark.eyeRightBottom = jdata.at("eyeRightBottom");
mark.eyeRightOuter = jdata.at("eyeRightOuter");
mark.noseRootLeft = jdata.at("noseRootLeft");
mark.noseRootRight = jdata.at("noseRootRight");
mark.noseLeftAlarTop = jdata.at("noseLeftAlarTop");
mark.noseRightAlarTop = jdata.at("noseRightAlarTop");
mark.noseLeftAlarOutTip = jdata.at("noseLeftAlarOutTip");
mark.noseRightAlarOutTip = jdata.at("noseRightAlarOutTip");
mark.upperLipTop = jdata.at("upperLipTop");
mark.upperLipBottom = jdata.at("upperLipBottom");
mark.underLipTop = jdata.at("underLipTop");
mark.underLipBottom = jdata.at("underLipBottom");
}

```



```

void from_json(const json& jdata, face_emotion& emo)
{
    emo.anger = jdata.at("anger").get<double>();
    emo.contempt = jdata.at("contempt").get<double>();
    emo.disgust = jdata.at("disgust").get<double>();
    emo.fear = jdata.at("fear").get<double>();
    emo.happiness = jdata.at("happiness").get<double>();
    emo.neutral = jdata.at("neutral").get<double>();
    emo.sadness = jdata.at("sadness").get<double>();
    emo.surprise = jdata.at("surprise").get<double>();
}

void from_json(const json& jdata, face_attributes& attr)
{
    attr.age = jdata.at("age").get<double>();
    attr.emotion = jdata.at("emotion");
    attr.gender = jdata.at("gender").get<std::string>();
}

void from_json(const json& jdata, face_info& info)
{
    info.faceId = jdata.at("faceId").get<std::string>();
    info.attributes = jdata.at("faceAttributes");
    info.landmarks = jdata.at("faceLandmarks");
    info.rectangle = jdata.at("faceRectangle");
}

```

Но если сервер потерпит неудачу по какой-то причине, он вернет другой объект, описывающий ошибку. Для представления этого описания используется класс `face_error_response`:

```
struct face_error
{
    std::string code;
    std::string message;
};

struct face_error_response
{
    face_error error;
};
```

Нам также потребуются перегруженные версии `from_json()` для десериализации этого описания:

```
void from_json(const json& jdata, face_error& error)
{
    error.code = jdata.at("code").get<std::string>();
    error.message = jdata.at("message").get<std::string>();
}

void from_json(const json& jdata, face_error_response& response)
{
    response.error = jdata.at("error");
}
```

Определив все необходимое, можно написать фактическое обращение к Face API. Так же как в задаче перевода текста, определим класс, инкапсулирующий всю необходимую функциональность (который впоследствии мы сможем расширить и дополнить). Такой подход упростит управление ключами приложения и адресом конечной точки (то есть избавит от необходимости передавать их в каждый вызов функции):

```
class face_manager
{
public:
    face_manager(std::string_view endpoint,
                std::string_view key)
        : endpoint(endpoint), app_key(key)
    {}

    face_detect_response detect_from_file(std::string_view path);

private:
    face_detect_response parse_detect_response(long const status,
```

```
std::stringstream & str);
```

```
std::string endpoint;
std::string app_key;
};
```

Метод `detect_from_file()` принимает путь к файлу с изображением, загружает его, посылает запрос службе Face API, анализирует ответ и возвращает объект `face_detect_response`, содержащий коллекцию объектов `face_info`. Поскольку на сервер передается фактическое изображение, тип содержимого определяется как `application/octet-stream`. Мы должны записать содержимое файла в поле `CURLOPT_POSTFIELDS`, обратившись к функции `curl_easy`, и указать его размер в поле `CURLOPT_POSTFIELDSIZE`:

```
face_detect_response face_manager::detect_from_file(std::string_view path)
{
    try
    {
        auto data = load_image(path);
        if (!data.empty())
        {
            std::stringstream str;
            curl::curl_ios<std::stringstream> writer(str);
            curl::curl_easy easy(writer);
            curl::curl_header header;
            header.add("Ocp-Apim-Subscription-Key:" + app_key);
            header.add("Content-Type:application/octet-stream");

            auto url = endpoint +
                "/detect"
                "?returnFaceId=true"
                "&returnFaceLandmarks=true"
                "&returnFaceAttributes=age,gender,emotion";

            easy.add<CURLOPT_URL>(url.c_str());
            easy.add<CURLOPT_HTTPHEADER>(header.get());

            easy.add<CURLOPT_POSTFIELDSIZE>(data.size());
            easy.add<CURLOPT_POSTFIELDS>(reinterpret_cast<char*>(
                data.data()));

            easy.perform();

            auto status = easy.get_info<CURLINFO_RESPONSE_CODE>();

            return parse_detect_response(status.get(), str);
        }
    }
}
```

```

    }
    catch (curl::curl_easy_exception const & error)
    {
        auto errors = error.get_traceback();
        error.print_traceback();
    }
    catch (std::exception const & ex)
    {
        std::cerr << ex.what() << std::endl;
    }

    return {};
}

```



За десериализацию полученного от сервера объекта JSON отвечает метод `parse_detect_response()`. Он анализирует HTTP-код ответа. В случае успеха сервер вернет код 200. В случае неудачи – код 4xx:

```

face_detect_response face_manager::parse_detect_response(
    long const status, std::stringstream & str)
{
    json jdata;
    str >> jdata;
    try
    {
        if (status == 200)
        {
            face_detect_response response = jdata;

            return response;
        }
        else if (status >= 400)
        {
            face_error_response response = jdata;

            std::cout << response.error.code << std::endl
                << response.error.message << std::endl;
        }
    }
    catch (std::exception const & ex)
    {
        std::cerr << ex.what() << std::endl;
    }

    return {};
}

```





Для чтения файла с диска функция `detect_from_file()` использует еще одну функцию – с именем `load_image()`. Она принимает строку пути к файлу и возвращает его содержимое в виде `std::vector<uint8_t>`:

```
std::vector<uint8_t> load_image(std::string_view filepath)
{
    std::vector<uint8_t> data;

    std::ifstream ifile(filepath.data(), std::ios::binary | std::ios::ate);
    if (ifile.is_open())
    {
        auto size = ifile.tellg();
        ifile.seekg(0, std::ios::beg);

        data.resize(static_cast<size_t>(size));
        ifile.read(reinterpret_cast<char*>(data.data()), size);
    }

    return data;
}
```

Теперь у нас есть все необходимое, чтобы отправить запрос алгоритму Detect службы Face API, проанализировать ответ и вывести его содержимое в консоль. Программа в следующем листинге выводит информацию о лицах, обнаруженных на изображении в файле `albert_and_elsa.jpg` из папки `res` в проекте. Не забудьте подставить фактический адрес конечной точки и ключ приложения для вашего ресурса Face API:

```
int main()
{
    face_manager manager(
        "https://westeurope.api.cognitive.microsoft.com/face/v1.0",
        "...(your api key)...");

#ifdef _WIN32
    std::string path = R"(res\albert_and_elsa.jpg)";
#else
    std::string path = R"(./res/albert_and_elsa.jpg)";
#endif

    auto results = manager.detect_from_file(path);

    for (auto const & face : results)
    {
        std::cout << "faceId: " << face.faceId << std::endl
                  << "age: " << face.attributes.age << std::endl
                  << "gender: " << face.attributes.gender << std::endl
    }
}
```

```

    << "rect: " << "{" << face.rectangle.left
    << "," << face.rectangle.top
    << "," << face.rectangle.width
    << "," << face.rectangle.height
    << "}" << std::endl << std::endl;
  }
}

```



Изображение `albert_and_elsa.jpg` показано на рис. 12.3.



**Рис. 12.3.** Изображение из файла `albert_and_elsa.jpg`

В следующем листинге показан вывод программы. Имейте в виду, что фактические временные идентификаторы лиц будут отличаться от вызова к вызову. Как видите, служба Face API определила на изображении два лица. Первое – это Альберт Эйнштейн, и для него определен возраст 54.9 года. Эта фотография была сделана в 1921 г., когда ему в действительности было 42 года. Второе лицо – это Эльза Эйнштейн, жена Альберта Эйнштейна, в том году ей исполнилось 45. Служба определила ее возраст как 41.6 года. Как видите, возраст определяется довольно грубо:



```
faceId: 77e0536f-073d-41c5-920d-c53264d17b98
age: 54.9
gender: male
rect: {165,86,72,72}
```

```
faceId: afb22044-14fa-46bf-9b65-16d4fe1d9817
age: 41.6
gender: female
rect: {321,151,59,59}
```

В случае неудачи служба вернет сообщение об ошибке (с HTTP-кодом 400). Метод `parse_detect_response()` проанализирует ответ с ошибкой и выведет сообщение в консоль. Например, если передать неверный ключ API, сервер вернет следующее сообщение:

```
Unspecified
Access denied due to invalid subscription key. Make sure you are subscribed
to an API you are trying to call and provide the right key.
```

```
(
Не определено
Доступ запрещен из-за неверного ключа подписки. Проверьте подписку на услуги
API, к которой вы пытаетесь обратиться, и укажите правильный ключ.
)
```

---

# Библиография

## Статьи

- 1337C0D3R, 2011. *Longest Palindromic Substring Part I*, <https://articles.leetcode.com/longest-palindromic-substring-part-1/>;
- Aditya Goel, 2016. *Permutations of a given string using STL*, <https://www.geeksforgeeks.org/permutations-of-a-given-string-using-stl/>;
- Andrei Jakab, 2010. *Using libcurl with SSH support in Visual Studio 2010*, <https://curl.haxx.se/libcurl/c/Using-libcurl-with-SSH-support-in-Visual-Studio-2010.pdf>;
- Ashwani Gautam, 2017. *What is the analysis of quick sort?*, <https://www.quora.com/What-is-the-analysis-of-quick-sort>;
- Ashwin Nanjappa, 2014. *How to build Boost using Visual Studio*, <https://codeyarns.com/2014/06/06/how-to-build-boost-using-visual-studio/>;
- busycrack, 2012. *Telnet IMAP Commands Note*, <https://busylog.net/telnetimap-commands-note/>;
- Dan Madden, 2000. *Encrypting Log Files*, <https://www.codeproject.com/Articles/644/Encrypting-Log-Files>;
- Georgy Gimel'farb, 2016. *Algorithm Quicksort: Analysis of Complexity*, <https://www.cs.auckland.ac.nz/courses/compsci220s1c/lectures/2016S1C/CS220-Lecture10.pdf>;
- Jay Doshi, Chanchal Khemani, Juhi Duseja. *Dijkstra's Algorithm*, <http://codersmaze.com/data-structure-explanations/graphs-data-structure/dijkstras-algorithm-for-shortest-path/>;
- Jeffrey Walton, 2008. *Applied Crypto++: Block Ciphers*, <https://www.codeproject.com/Articles/21877/Applied-Crypto-Block-Ciphers>;
- Jeffrey Walton, 2007. *Product Keys Based on the Advanced Encryption Standard (AES)*, <https://www.codeproject.com/Articles/16465/Product-Keys-Based-on-the-Advanced-Encryption-Stan>;
- Jeffrey Walton, 2006. *Compiling and Integrating Crypto++ into the Microsoft Visual C++ Environment*, [https://www.codeguru.com/cpp/v-s/devstudio\\_macros/openfaq/article.php/c12853/Compiling-and-Integrating-Crypto-into-the-Microsoft-Visual-C-Environment.htm](https://www.codeguru.com/cpp/v-s/devstudio_macros/openfaq/article.php/c12853/Compiling-and-Integrating-Crypto-into-the-Microsoft-Visual-C-Environment.htm);
- Jonathan Boccara, 2017. *How to split a string in C++*, <https://www.fluentcpp.com/2017/04/21/how-to-split-a-string-in-c/>;

- Kenny Kerr, 2013. *Resource Management in the Windows API*, <https://visualstudiomagazine.com/articles/2013/09/01/get-a-handle-on-the-windows-api.aspx>;
- Кенни Керр (Kenny Kerr), 2011. *Windows u C++*. C++ u Windows API, <https://msdn.microsoft.com/ru-ru/magazine/hh288076.aspx?f=255&MSPPErrOr=-2147217396>;
- Marius Bancila, 2015. *Integrate Windows Azure Face APIs in a C++ application*, <https://www.codeproject.com/Articles/989752/Integrate-Windows-Azure-Face-APIs-in-a-Cplusplus-a>;
- Marius Bancila, 2018. *Using Cognitive Services to find your Game of Thrones lookalike*, <https://www.codeproject.com/Articles/1234217/Using-Cognitive-Services-to-find-your-Game-of-Thro>;
- Mary K. Vernon. *Priority Queues*, <http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html>;
- Mathias Bynens. *In search of the perfect URL validation regex*, <https://mathiasbynens.be/demo/url-regex>;
- O. S. Tezer, 2014. *SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems*, <https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems>;
- Robert Nystrom, 2014. *Game Programming patterns: Double Buffer*, <http://gameprogrammingpatterns.com/double-buffer.html>;
- Robert Sedgewick, Philippe Flajolet, 2013. *Introduction to the Analysis of Algorithms*, <http://www.informit.com/articles/article.aspx?p=2017754&seqNum=5>;
- Rosso Salmanzadeh, 2002. *Using libcurl in Visual Studio*, [https://curl.haxx.se/libcurl/c/visual\\_studio.pdf](https://curl.haxx.se/libcurl/c/visual_studio.pdf);
- Sergii Bratus, 2010. *Implementation of the Licensing System for a Software Product*, <https://www.codeproject.com/Articles/99499/Implementation-of-the-Licensing-System-for-a-Softw>;
- Shubham Agrawal, 2016. *Dijkstra's Shortest Path Algorithm using priority queue of STL*, <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority-queue-stl/>;
- Travis Tidwell, 2013. *An Online RSA Public and Private Key Generator*, <http://travistidwell.com/blog/2013/09/06/an-online-rsa-public-and-private-key-generator/>;
- Victor Volkman, 2006. *Crypto++ Holds the Key to Encrypting Your C++ Application Data*, <https://www.codeguru.com/cpp/misc/misc/cryptoapi/article>.

- [php/c11953/Cryptosupregsup-Holds-the-Key-to-Encrypting-Your-C-Application-Data.htm](http://php/c11953/Cryptosupregsup-Holds-the-Key-to-Encrypting-Your-C-Application-Data.htm);
- Yang Song, 2014. *Split a string using C++*, <http://ysonggit.github.io/coding/2014/12/16/split-a-string-using-c.html>;
  - *Decorator Design Pattern*, [https://sourcemaking.com/design\\_patterns/decorator/](https://sourcemaking.com/design_patterns/decorator/);
  - *Composite Design Pattern*, [https://sourcemaking.com/design\\_patterns/composite/](https://sourcemaking.com/design_patterns/composite/);
  - *Template Method Design Pattern*, [https://sourcemaking.com/design\\_patterns/template\\_method/](https://sourcemaking.com/design_patterns/template_method/);
  - *Strategy Design Pattern*, [https://sourcemaking.com/design\\_patterns/strategy/](https://sourcemaking.com/design_patterns/strategy/);
  - *Chain of Responsibility*, [https://sourcemaking.com/design\\_patterns/chain\\_of\\_responsibility/](https://sourcemaking.com/design_patterns/chain_of_responsibility/);
  - *Understanding the PDF File Format: Overview*, <https://blog.idrsolutions.com/2013/01/understanding-the-pdf-file-format-overview/>;
  - *RSA Signing is Not RSA Decryption*, [https://www.cs.cornell.edu/courses/cs5430/2015sp/notes/rsa\\_sign\\_vs\\_dec.php](https://www.cs.cornell.edu/courses/cs5430/2015sp/notes/rsa_sign_vs_dec.php);
  - *RSA Cryptography*, [https://www.cryptopp.com/wiki/RSA\\_Cryptography](https://www.cryptopp.com/wiki/RSA_Cryptography);
  - *Using rand() (C/C++)*, [http://eternallyconfuzzled.com/arts/jsw\\_art\\_rand.aspx](http://eternallyconfuzzled.com/arts/jsw_art_rand.aspx);
  - *Crypto++ Keys and Formats*, [https://www.cryptopp.com/wiki/Keys\\_and\\_Formats](https://www.cryptopp.com/wiki/Keys_and_Formats);
  - *RFC 2060 – Протокол IMAP v.4, rev. 1*, <https://rfc2.ru/2060.rfc>;
  - *Internal Versus External BLOBs in SQLite*, <https://www.sqlite.org/intern-v-extern-blob.html>;
  - *OpenSSL Compilation and Installation*, [https://wiki.openssl.org/index.php/Compilation\\_and\\_Installation](https://wiki.openssl.org/index.php/Compilation_and_Installation).

## Документация к библиотекам

- *C/C++ JSON parser/generator benchmark*, <https://github.com/miloyip/nativejson-benchmark>;
- *Crypto++*, [https://www.cryptopp.com/wiki/Main\\_Page](https://www.cryptopp.com/wiki/Main_Page);
- *Hummus PDF*, <http://pdfhummus.com/How-To>;
- *JSON for Modern C++*, <https://github.com/nlohmann/json>;

- *PDF-Writer*, <https://github.com/galkahana/PDF-Writer>;
- *PNGWriter*, <https://github.com/pngwriter/pngwriter>;
- *pugixml 1.8 quick start guide*, <https://pugixml.org/docs/quickstart.html>;
- *SQLite*, <https://www.sqlite.org/docs.html>;
- *sqlite\_modern\_cpp*, [https://github.com/SQLiteModernCpp/sqlite\\_modern\\_cpp](https://github.com/SQLiteModernCpp/sqlite_modern_cpp);
- *Ziplib wiki*, <https://bitbucket.org/wbenny/ziplib/wiki/Home>.



# Предметный указатель

## А

алгоритм быстрой сортировки 140  
алгоритмы 92  
алгоритм Эвклида 26  
архивы 196

## Б

база данных часовых поясов IANA  
  установка 88  
базы данных 196  
библиотеки  
  Asio/Boost.Asio 259  
  Configuru 186  
  CppSQLite 217  
  Crypto++ 246  
  curlcpp 266  
  FreeType2 207  
  gSOAP 180  
  HaHu 189  
  Info-Zip 199  
  JagPDF 189  
  jsoncpp 186  
  json\_spirit 186  
  libcurl 266  
  libpng 207  
  libxml++ 180  
  lodepng 207  
  LZMA SDK 199  
  MiniZip 199  
  NLOhmann 186  
  PDF-Writer 189  
  png++ 207  
  PNGWriter 207  
  POCO 259  
  PoDoFo 189  
  pugixml 180  
  RapidJSON 186  
  RapidXml 180  
  SQLCipher, расширение 217  
  SQLite 217  
  sqlite3cc 217  
  SQLiteCPP 217  
  sqlite\_modern\_cpp 217  
  taocpp/json 186  
  tinysql 180  
  tinysql2 180  
  Xerces-C++ 180  
  ZipLib 199

Блейз де Виженер (Blaise de Vigenère) 239  
большие объекты, в базе данных 228  
быстрая сортировка, алгоритм 140  
быстрой сортировки, алгоритм 117

## В

Виженера  
  таблицы 239  
  шифр 239  
выражения свертки 49



## Г

графы, поиск путей 121

## Д

дата и время 83  
двойная буферизация  
  в компьютерной графике 103  
двумерные массивы 40  
Дейкстры (Dijkstra) алгоритм 97, 121  
Декоратор, шаблон проектирования 150  
Джованни Батиста Беллазо (Giovan Battista Bellaso) 239  
Джон Хортон Конвей (John Horton Conway) 96  
Джузеппе Персико (Giuseppe Persico) 266  
дружественные числа (amicable numbers)  
  не путать с компанейскими числами (friendly numbers) 30

## Е

европейский номер товара 211

## З

задачи  
  2-мерный массив с поддержкой базовых операций 40  
  алгоритм выбора 94  
  алгоритм объединения в пары 94  
  алгоритм параллельного преобразования 133  
  алгоритм сортировки 95  
  алгоритм шивания 94  
  временные файлы журналов 73  
  время встречи для нескольких часовых поясов 83  
  выборка данных из XML с помощью XPath 177  
  вывод списка фильмов в файл PDF 178  
  вычисление значения числа  $\pi$  25



- вычисление стоимости заказа с учетом скидки 149  
 вычисление хеш-суммы файла 236  
 генератор штрихкодов EAN-13 197  
 генерация всех перестановок символов в строке 94  
 генерация номеров социального страхования 148  
 генерация случайных паролей 148  
 двойной буфер 93  
 день и неделя года 83  
 день недели 83  
 десериализация данных из формата JSON 178  
 добавление диапазона значений в контейнер 40  
 добавление информации о фильмах в базу данных SQLite 198  
 дружественные числа 25  
 игра «Жизнь» 96  
 избыточные числа 24  
 извлечение частей URL 60  
 измерение времени выполнения функции 83  
 календарь на месяц 83  
 клиент/серверная игра Fizz-Buzz 258  
 код Грея 25  
 кодирование и декодирование Base64 236  
 контейнер с наблюдателями 149  
 кратчайший путь между узлами 95  
 литералы разных температурных шкал 41  
 наибольшая подстрока-палиндром 60  
 наибольшая последовательность Коллатца 25  
 наибольшее простое число меньше заданного 24  
 наибольший общий делитель 24  
 наименьшее общее кратное 24  
 обертка для системных дескрипторов 41  
 обменный курс биткойна 258  
 обработка изображений для фильмов в базе данных SQLite 198  
 объединение строк через разделитель 59  
 определение лиц на изображениях 259  
 определение размера каталога 72  
 параллельные алгоритмы поиска максимального и минимального значений с использованием асинхронных функций 133  
 параллельные алгоритмы поиска максимального и минимального значений с использованием потоков 133  
 параллельный алгоритм сортировки 133  
 перевод текста на любой язык 259  
 перечисление адресов IPv4 в заданном диапазоне 40  
 подписывание файлов 237  
 поиск IP-адреса хоста 258  
 поиск файлов в архиве ZIP 196  
 поиск файлов в каталоге, соответствующих регулярному выражению 73  
 получение почты по протоколу IMAP 258  
 потокобезопасное журналирование в консоль 134  
 преобразование в верхний регистр первых букв слов 59  
 преобразование дат в строках 60  
 преобразование десятичных чисел в римские 25  
 преобразование списка телефонных номеров 93  
 преобразование строк в числа 59  
 преобразование чисел в строки 59  
 приоритетная очередь 92  
 проверка действительности номеров ISBN 25  
 проверка наличия в контейнере любого, всех и ни одного из указанных значений 41  
 проверка номерного знака 60  
 проверка пароля 148  
 проверка учетных данных пользователя 236  
 программа Weasel 96  
 простые множители числа 25  
 простые числа, отличающиеся на шесть 24  
 разбиение строк на лексемы по разделителям из списка 60  
 самый часто встречающийся элемент в диапазоне 93  
 сериализация данных в формат JSON 178  
 сериализация и десериализация данных в формате XML 177  
 система обслуживания клиентов 134  
 система одобрений 149  
 создание документа PDF из коллекции изображений 179  
 создание изображения PNG с контрольным текстом 197  
 создание файла PNG с изображением национального флага 196  
 средний рейтинг фильмов 94  
 сумма натуральных чисел, кратных 3 и 5 24  
 табличный вывод списка процессов 72  
 текстовая гистограмма 93  
 тип данных IPv4 40  
 треугольник Паскаля 72  
 удаление пустых строк из текстового файла 72

удаление файлов старше заданной даты 73  
 упаковка и извлечение файлов из архива ZIP 196  
 упаковка и извлечение файлов из архива ZIP с защитой паролем 196  
 фильтрация списка телефонных номеров 93  
 функция выбора минимального значения с переменным числом аргументов 40  
 циклический буфер 92  
 числа Армстронга 25  
 число дней между двумя датами 83  
 чтение информации о фильмах из базы данных SQLite 198  
 шифр Виженера 236  
 шифрование и расшифровывание файлов 237  
 шифр Цезаря 236  
 защита паролем 204

**И**

изображения 196  
 итераторы  
 двунаправленные 65  
 с произвольным доступом 64

**К**

Какутани, проблема 36  
 Кенни Керр (Kenny Kerr) 51  
 Коллатца, гипотеза 36  
 Компоновщик, шаблон проектирования 154  
 конкуренция 133  
 криптография 236

**М**

Майкл Ф. Армстронг (Michael F. Armstrong) 31  
 манипуляторы ввода/вывода 61  
 международный номер товара 211  
 Монте-Карло, метод 37

**Н**

Наблюдатель, шаблон проектирования 166

**О**

обобщенные параллельные алгоритмы в C++17 136  
 оценка по короткой схеме 49

**П**

паролем, защита 204  
 перемещения, семантика 46  
 переполнение 26

перестановки 111  
 пирамидальная сортировка 118  
 потоки данных 72  
 простые близнецы 28  
 простые двоюродные числа 28

**Р**

равномерное статистическое распределение 37  
 регулярные выражения 68  
 тестирование и отладка 69  
 рекурсивные функции 26  
 рекурсия времени компиляции 47  
 рефлексивный двоичный код 33  
 решения  
 2-мерный массив с поддержкой базовых операций 45  
 алгоритм выбора 117  
 алгоритм объединения в пары 114  
 алгоритм параллельного преобразования 134  
 алгоритм сортировки 117  
 алгоритм шивания 115  
 временные файлы журналов 81  
 время встречи для нескольких часовых поясов 88  
 выборка данных из XML с помощью XPath 183  
 вывод списка фильмов в файл PDF 189  
 вычисление значения числа  $\pi$  37  
 вычисление стоимости заказа с учетом скидок 171  
 вычисление хеш-суммы файла 250  
 генератор штрихкодов EAN-13 211  
 генерация всех перестановок символов в строке 111  
 генерация номеров социального страхования 158  
 генерация случайных паролей 154  
 двойной буфер 103  
 день и неделя года 87  
 день недели 86  
 десериализация данных из формата JSON 187  
 добавление диапазона значений в контейнер 48  
 добавление информации о фильмах в базу данных SQLite 223  
 дружественные числа 30  
 игра «Жизнь» 128  
 избыточные числа 29  
 извлечение частей URL 69  
 измерение времени выполнения функции 84  
 календарь на месяц 90  
 клиент/серверная игра Fizz-Buzz 261  
 код Грея 33

- кодирование и декодирование Base64 241  
 контейнер с наблюдателями 166  
 кратчайший путь между узлами 121  
 литералы разных температурных шкал 54  
 наибольшая подстрока-палиндром 66  
 наибольшая последовательность Коллатца 36  
   ограничение 36  
 наибольшее простое число меньше заданного 27  
 наибольший общий делитель 26  
 наименьшее общее кратное 27  
 обертка для системных дескрипторов 50  
 обменный курс биткойна 265  
 обработка изображений для фильмов в базе данных SQLite 227  
 объединение строк через разделитель 64  
 определение лиц на изображениях 279  
 определение размера каталога 77  
 параллельные алгоритмы поиска  
   максимального и минимального значений с использованием асинхронных функций 138  
 параллельные алгоритмы поиска  
   максимального и минимального значений с использованием потоков 136  
 параллельный алгоритм сортировки 140  
 перевод текста на любой язык 274  
 перечисление адресов IPv4 в заданном диапазоне 45  
 подписывание файлов 254  
 поиск IP-адреса хоста 259  
 поиск файлов в архиве ZIP 199  
 поиск файлов в каталоге, соответствующих регулярному выражению 80  
 получение почты по протоколу IMAP 270  
 потокобезопасное журналирование в консоль 142  
 преобразование в верхний регистр первых букв слов 63  
 преобразование дат в строках 71  
 преобразование десятичных чисел в римские 34  
 преобразование списка телефонных номеров 109  
 преобразование строк в числа 61  
 преобразование чисел в строки 60  
 приоритетная очередь 97  
 проверка действительности номеров ISBN 48  
 проверка наличия в контейнере любого, всех и ни одного из указанных значений 49  
 проверка номерного знака 68  
 проверка пароля 150  
 проверка учетных данных пользователя 246  
 программа Weasel 125  
 простые множители числа 32  
 простые числа, отличающиеся на шесть 28  
 разбиение строк на лексемы по разделителям из списка 65  
 самый часто встречающийся элемент в диапазоне 105  
 сериализация данных в формат JSON 185  
 десериализация данных в формате XML 179  
 система обслуживания клиентов 143  
 система одобрений 162  
 создание документа PDF из коллекции изображений 193  
 создание изображения PNG с контрольным текстом 208  
 создание файла PNG с изображением национального флага 207  
 средний рейтинг фильмов 113  
 сумма натуральных чисел, кратных 3 и 5 25  
 табличный вывод списка процессов 74  
 текстовая гистограмма 107  
 тип данных IPv4 41  
 треугольник Паскаля 73  
 удаление пустых строк из текстового файла 76  
 удаление файлов старше заданной даты 78  
 упаковка и извлечение файлов из архива ZIP 201  
 упаковка и извлечение файлов из архива ZIP с защитой паролем 204  
 фильтрация списка телефонных номеров 108  
 функция выбора минимального значения с переменным числом аргументов 47  
 циклический буфер 99  
 числа Армстронга 31  
 число дней между двумя датами 85  
 чтение информации о фильмах из базы данных SQLite 217  
 шифр Виженера 239  
 шифрование и расшифровывание файлов 252  
 шифр Цезаря 237  
 Ричард Докинз (Richard Dawkins) 96, 125
- С**  
 самовлюбленные числа 31  
 семантика перемещения 46  
 сериализация данных 177  
 сети и службы 258  
 синглтон 142  
 сиракузская проблема 36  
 слиянием, сортировка 118



случайные линии разного цвета 209  
 совершенные цифровые инварианты 31  
 сортировка слиянием 118  
 список процессов 74  
 Стратегия, шаблон проектирования 172  
 структура кода EAN-13 213  
 структуры данных 92

## Т

таблицы Виженера 239  
 транзакции 226  
 треугольник Паскаля 73  
 Туэйтса, гипотеза 36

## У

Улама, гипотеза 36

## Ф

файловые системы 72  
 фон с цветной градиентной заливкой 209

## Х

Хассе, алгоритм 36  
 Ховард Хиннант (Howard Hinnant) 85

## Ц

целые числа, 8-битные 59, 62  
 Цепочка обязанностей, шаблон проектирования 162

## Ш

Шаблонный метод, шаблон проектирования 158  
 шаблоны проектирования 148  
 Декоратор 150  
 Компоновщик 154  
 Наблюдатель 166  
 Стратегия 172  
 Цепочка обязанностей 162  
 Шаблонный метод 158

## Э

Эвклида, алгоритм 26  
 нерекурсивная реализация 26  
 рекурсивная реализация 26

## А

Amazon Translate, служба 274

append\_attribute() 181  
 asio/Boost.Asio, библиотека 259  
 asio::connect() 264  
 asio::io\_context 259  
 asio::ip::tcp::acceptor 262  
 asio::ip::tcp::endpoint 260  
 asio::ip::tcp::resolver 260  
 asio::ip::tcp::socket 264

## В

back inserter 80  
 base64, схема кодирования 241  
 BlockTransformation 251  
 boost::uuid, библиотека 81  
 BufferedTransformation 251

## С

chrono, библиотека 78  
 chrono, стандартная библиотека 85  
 СМΥΚ, формат 207  
 Configuru, библиотека 186  
 constexpr gcd(), функция 27  
 constexpr lcm(), функция 27  
 copy if() 80  
 CppSQLite, библиотека 217  
 CrossGuid, библиотека 81  
 CryptoPP::Weak, пространство имен 247  
 Crypto++, библиотека 246  
 две версии 247  
 curlcpp, библиотека 266  
 установка в macOS 267  
 установка в Windows 266



## D

date::iso\_week::year\_weeknum\_weekday 86  
 date::sys\_days, класс 85  
 date::year\_month\_day 86  
 date::year\_month\_day\_last 90  
 date::zoned\_time, класс 89  
 date, библиотека 86  
 DefaultDecryptorWithMAC 252  
 DefaultEncryptorWithMAC 252  
 DES, алгоритм 247  
 directory\_iterator 79

## E

EAN-13, стандарт 211

## F

Face API, служба 279  
 возможности 279

настройка 279  
 FileSource 250  
 filesystem::recursive\_directory\_iterator 77, 80  
 filesystem::temp\_directory\_path() 76  
 filesystem, библиотека 77  
 Fizz-Buzz, игра 261  
 FreeType2, библиотека 207  
 FTP/FTPS, протокол 266  
 future, объект 138

## G

gethostbyname() 259  
 Google Cloud Translation API, служба 274  
 Gopher, протокол 266  
 gSOAP, библиотека 180

## H

HaHu, библиотека 189  
 HashFilter 250  
 HashTransformation 251  
 HexEncoder 250  
 HSV, формат 207  
 HTTP/HTTPS, протокол 266

## I

IANA Time Zone Database  
   установка 88  
 IMAP, протокол 270  
 Info-Zip, библиотека 199  
 IPv4 40  
 IPv4, адреса 41  
   ввод и вывод 41  
   перечисление 43  
   сравнение 41  
 iso\_week.h 86  
 isspace() 110

## J

JagPDF, библиотека 189  
 jsoncpp, библиотека 186  
 json\_spirit, библиотека 186

## L

LDAP/LDAPS, протокол 266  
 libcurl, библиотека 266  
   установка в macOS 267  
   установка в Windows 266  
 libpng, библиотека 207  
 libxml++, библиотека 180  
 lodepng, библиотека 207  
 LZMA SDK, библиотека 199

## M

MD5, алгоритм 247  
 Mersenne Twister, алгоритм 37  
 Microsoft Cognitive Services, служба 274  
 MIME, стандарт 241  
 MiniZip, библиотека 199

## N

nativejson-benchmark, проект 185  
 NLothmann, библиотека 186

## P

PageContentContext::DrawImage() 193  
 PageContentContext::DrawPath() 190  
 PageContentContext::WriteText() 190  
 PDFPage::StartPageContentContext() 189  
 PDFWriter::EndPageContentContext() 190  
 PDFWriter::WritePageAndRelease() 190  
 PDF-Writer, библиотека 189  
 PKWare, шифрование 205  
 PNGWriter, библиотека 207  
 png++, библиотека 207  
 POCO, библиотека 259  
 PoDoFo, библиотека 189  
 POP3/POP3S, протокол 266  
 promise, объект 138  
 pugi::xml\_document 181  
 pugixml? библиотека 180

## R

RapidJSON, библиотека 186  
 RapidXml, библиотека 180  
 recursive\_directory\_iterator  
   неопределенное поведение 79  
 regex 80  
 regex\_match() 80  
 RGB, формат 207  
 RSASSA\_PKCS1v15\_SHA\_Signer 256  
 RSASSA\_PKCS1v15\_SHA\_Verifier 256  
 RSA-ключи 254

## S

SignatureVerificationFilter 256  
 SignerFilter 256  
 SMTP/SMTPS, протокол 266  
 SQLCipher, расширение 217  
 sqlite3cc, библиотека 217  
 SQLiteCPP, библиотека 217  
 sqlite::database 219  
   транзакции 226  
 sqlite\_modern\_cpp, библиотека 217

sqlite::sqlite\_exception 219  
 SQLite, библиотека 217  
     большие объекты в базе данных 228  
 std::accumulate() 77  
 std::accumulate, алгоритм 27  
 std::array 45  
 std::async() 138  
 std::condition\_variable 144  
 std::copy 64  
 std::cout 142  
 std::high\_resolution\_clock 84  
 std::invoke() 84  
 std::launch::async 138  
 std::map 105  
 std::max\_element() 137  
 std::min\_element() 137  
 std::mt19937, генератор псевдослучайных чисел 209  
 std::mutex 142  
 std::next\_permutation() 111  
 std::optional<T> 221  
 std::ostream\_iterator 64  
 std::ostringstream 61, 64  
 std::pair 115  
 std::regex\_match() 68  
 std::regex\_replace() 71  
 std::remove\_if() 110  
 std::set 123  
 std::sregex\_iterator 68  
 std::string::find\_first\_of() 65  
 std::stringstream 63  
 std::string\_view 61  
 std::system\_clock 85  
 std::system\_clock::time\_point 85  
 std::thread::hardware\_concurrency() 134  
 std::transform() 134  
 std::unique\_ptr<T> 221  
 std::uuid, библиотека 81  
 std::vector 46  
 std::wcout 142  
 StreamTransformation 251  
 StringSink 250  
 string\_view 238  
 sum\_proper\_divisors(), функция 31

## Т

taocpp/json, библиотека 186  
 Text Translate API, служба 274  
     возможности 274  
     ключ приложения 274  
 tinymce2, библиотека 180  
 tinymce, библиотека 180

tzdata2017c.tar.gz 88

## U

unsigned long long 77

## W

windowsZones.xml 88

## X

Xerces-C++, библиотека 180

XPath 180, 183

xpath\_exception 183

## Z

ZipArchive::CreateEntry() 201

ZipArchiveEntry::GetFullName() 200

ZipArchiveEntry::GetName() 199

ZipArchiveEntry::SetPassword() 205

ZipArchive::GetEntry() 199

ZipArchive::GetEntryCount() 199

ZipFile::AddEncryptedFile() 205

ZipFile::AddFile() 201

ZipFile::Open() 199

ZipLib, библиотека 199

    PKWare, шифрование 204



---

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «Планета Альянс» наложенным платежом, выслать открытку или письмо по почтовому адресу: **115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.**

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.a-planet.ru.**

Оптовые закупки: **тел. +7 (499) 782-38-89.**

Электронный адрес: **books@aliants-kniga.ru.**

Мариус Бансила

## Решение задач на современном C++

Станьте опытным программистом, решая практические задачи

Главный редактор *Мовчан Д. А.*  
dmkpress@gmail.com

Перевод с английского *Киселев А. Н.*

Корректор *Синяева Г. И.*

Верстка *Паранская Н. В.*

Дизайн обложки *Дукучаева А. С.*

Формат 70×100 1/16.

Печать цифровая. Усл. печ. л. 24,54.

Тираж 200 экз.

Веб-сайт издательства: [www.dmkpress.com](http://www.dmkpress.com)