# Software Engineering

Moscow Institute of Physics and Technology

# 00. Mandatory Requirements

**Требования: часть 1**

- Ваши решения должны собираться без предупреждений с флагами -Wall, -Wextra и -Wpedantic.
- Ваши решения должны сопровождаться тестами и демонстрационными примерами.
- Ваши решения должны выполняться до конца и проходить тесты без ошибок и неопределенного поведения.
- Ваши решения должны располагаться каждое в отдельном единственном файле исходного кода.
- Ваши решения должны быть адекватно отформатированы в рамках единого стиля.

**Требования: часть 2**

- Ваши решения не должны содержать символы, не представленные в таблице семибитного стандарта ASCII.
- Ваши решения не должны содержать магические литералы и непонятные названия.
- Ваши решения не должны содержать неинициализированные переменные и глобальные объекты.
- Ваши решения не должны содержать дублирующийся код и неиспользуемые части.
- Ваши решения не должны содержать избыточного использования стандартных потоков для ввода и вывода.

В случае конфликтов требований в приоритете является выполнение требований, указанных в условиях задач.

# 01. Introduction and Overview

## 01.01 (06.07)

Напишите программу, которая выводит любую строку в окно терминала через стандартный поток `std::cout` и при этом обладает функцией `main` с единственной инструкцией `return 0`. Предложите по крайней мере четыре разных решения. Впервые я столкнулся с этой задачей на техническом собеседовании в крупную российскую компанию. Для ее решения Вам потребуются технологии, которые будут рассматриваться во втором, третьем и шестом модулях данного курса, поэтому Вы можете пропустить эту задачу и вернуться к ней позже. Возможно, Вы немного удивились тому, что первая же задача данного курса обладает настолько неадекватным уровнем сложности. Это своеобразная дань памяти моему детству. Я начал серьезно изучать компьютерные науки и языки программирования в 12 лет, когда проводил летние школьные каникулы на даче у бабушки с дедушкой. Родители подарили мне две книги: Программирование – принципы и практика с использованием C++ Бьёрна Страуструпа и Язык программирования C Брайана Кернигана и Денниса Ритчи. Также у меня имелся простой ноутбук со средой разработки Code::Blocks, однако не было ни интернета, ни даже мобильной связи, потому что дача находится в низине, а сеть можно поймать только на определенном тайном холмике в лесу. Я решил начать изучение с визуально небольшой книги по языку C, быстро проработать ее, а потом приступить к монографии Страуструпа. Опрометчивое решение! В одном из первых заданий просили написать программу, которая удалила бы все комментарии из исходного кода другой программы на языке C. Предположу, что это весьма сложная задача для третьего дня изучения программирования, но я справился, потому что из-за отсутствия связи с внешним миром я просто не понял, что это сложно. Возможно, именно этот случай помог мне определиться с основным направлением всей дальнейшей деятельности. Любопытно, что случилось, если бы мне тогда подарили монографию Искусство программирования Дональда Кнута? Возможно, я стал бы лучше относиться к математике. Пожалуй, стоит провести небольшой эксперимент над собственными детьми.

# 02. Basics of Programming

## 02.01 ( 02.12 )

Реализуйте алгоритм вычисления N-ого числа ряда Фибоначчи на основе формулы Бине. Используйте тип `double` для промежуточных вычислений и тип `int` для конечного значения числа ряда Фибоначчи. Используйте оператор `static_cast` для преобразования округленного приближенного значения формулы Бине типа `double` к конечному значению типа `int`. Используйте константы для значений в формуле Бине. Используйте стандартные функции `std::sqrt`, `std::pow` и `std::round`. Обоснуйте формулу Бине. Используйте стандартный поток `std::cin` для ввода через терминал номера N. Используйте стандартный поток `std::cout` для вывода через терминал N-ого числа ряда Фибоначчи. Не сопровождайте Ваше решение данной задачи тестами.

## 02.02 ( 02.17 )

Реализуйте алгоритм вычисления корней алгебраического уравнения второй степени с коэффициентами a, b и c типа `double`. Используйте ветвления `if` для проверки значения коэффициента a и значения дискриминанта. Используйте константу epsilon и стандартную функцию `std::abs` для корректного сравнения чисел типа `double` с заданной точностью. Допускайте появление отрицательного нуля. Используйте стандартный поток `std::cin` для ввода через терминал коэффициентов a, b и c. Используйте стандартный поток `std::cout` для вывода через терминал всех корней этого уравнения. Не сопровождайте Ваше решение данной задачи тестами.

## 02.03 ( 02.18 )

Реализуйте алгоритм классификации символов типа `char` из таблицы ASCII с десятичными кодами от 32 до 127 включительно на пять следующих классов: заглавные буквы, строчные буквы, десятичные цифры, знаки препинания, прочие символы. Используйте ветвление `switch` с проваливанием и символьными литералами типа `char` в качестве меток в секциях `case`. Протестируйте нестандартное расширение компилятора g++ для диапазонов и флаг компилятора -Wpedantic. Используйте секцию `default` для пятого класса. Используйте стандартный поток `std::cin` для ввода через терминал символов. Используйте стандартный поток `std::cout` для вывода через терминал всех названий классов. Не сопровождайте Ваше решение данной задачи тестами.

## 02.04 ( 02.20 ) // M. Bancila, The Modern C++ Challenge

Реализуйте алгоритм вычисления всех трехзначных чисел Армстронга. Используйте тройной вложенный цикл `for` для перебора. Не используйте стандартную функцию `std::pow`. Используйте стандартный поток `std::cout` для вывода через терминал всех чисел Армстронга. Не сопровождайте Ваше решение данной задачи тестами.

## 02.05 ( 02.24 )

Реализуйте алгоритмы вычисления числа $\pi$ на основе суммы членов ряда Лейбница и числа $e$ на основе суммы членов ряда Маклорена при x равном 1 с точностью, заданной числом epsilon. Используйте тип `double` для промежуточных вычислений и конечных значений чисел $\pi$ и $e$. Используйте циклы `while` для вычисления членов рядов Лейбница и Маклорена до тех пор, пока очередной член каждого ряда не станет меньше заданного пользователем числа epsilon. Не вычисляйте факториалы в знаменателях членов ряда Маклорена, чтобы избежать проблемы переполнения. Используйте известное соотношение между членами ряда Маклорена для оптимизации вычисления каждого нового члена ряда на основе предыдущего члена ряда. Используйте стандартный поток `std::cin` для ввода через терминал числа epsilon. Используйте стандартный поток `std::cout` для вывода через терминал вычисленных чисел $\pi$ и $e$. Не сопровождайте Ваше решение данной задачи тестами.

## 02.06 ( 02.29 )

Реализуйте алгоритмы вычисления максимального и минимального значений, медианы, среднего арифметического и стандартного отклонения коллекции чисел типа `double`. Используйте встроенный статический массив для хранения и обработки коллекции чисел. Используйте стандартный поток `std::cin` для ввода через терминал коллекции чисел. Используйте любой удобный для Вас способ ввода коллекции чисел, например, с предварительным вводом размера коллекции чисел. Реализуйте алгоритм сортировки пузырьком как дополнительную часть алгоритма вычисления медианы коллекции чисел. Используйте стандартный поток `std::cout` для вывода через терминал максимального и минимального значений, медианы, среднего арифметического, а также стандартного отклонения коллекции чисел. Не сопровождайте Ваше решение данной задачи тестами.

## 02.07 ( 02.30 )

Доработайте Ваше предыдущее решение задачи 02.06 таким образом, чтобы реализация использовала встроенный динамический массив вместо встроенного статического массива. Предполагайте, что размер коллекции чисел неизвестен. Реализуйте алгоритм увеличения емкости встроенного динамического массива в процессе его использования. Реализуйте выделение нового блока памяти размером в два раза больше предыдущего, копирование всех данных из предыдущего блока памяти в новый и освобождение предыдущего блока памяти.

## 02.08 ( 02.31 ) // M. Bancila, The Modern C++ Challenge

Реализуйте алгоритм вычисления наибольшей длины последовательности Коллатца среди всех последовательностей, начинающихся со значений от 1 до 100. Используйте тип `unsigned long long int` для значений последовательностей Коллатца и стандартный тип `std::size_t` для их длин. Используйте стандартный контейнер `std::vector` для кэширование длин последовательностей Коллатца и оптимизации вычисления длины каждой новой последовательности на основе предыдущих последовательностей. Используйте стандартный поток `std::cout` для вывода через терминал наибольшей длины последовательности Коллатца среди рассмотренных, а также значения, с которого она начинается. Не сопровождайте Ваше решение данной задачи тестами.

## 02.09 ( 02.43 ) // M. Bancila, The Modern C++ Challenge

Реализуйте алгоритмы вычисления наибольшего общего делителя двух натуральных чисел типа `int` на основе рекурсивного и итративного подходов, а также алгоритм вычисления наименьшего общего кратного двух натуральных чисел типа `int`. Используйте стандартные функции `std::gcd` и `std::lcm` для валидации результатов.

## 02.10 ( 02.44 )

Доработайте пример `02.44` таким образом, чтобы реализация использовала алгоритм быстрой сортировки вместо алгоритма сортировки слиянием. Реализуйте метод Хоара. Используйте медиану первого, среднего и последнего элементов как опорный элемент. Обоснуйте временную сложность данного алгоритма сортировки.

# 03. Object - Oriented Programming

## 03.01 ( 03.02 )

Реализуйте структуру Rectangle для представления прямоугольников со сторонами, которые параллельны осям координатной плоскости. Реализуйте в структуре Rectangle четыре поля типа int для хранения координат левого верхнего и правого нижнего углов прямоугольника. Используйте систему координат, в которой ось абсцисс направлена вправо, а ось ординат направлена вниз. Реализуйте алгоритм вычисления площади пересечения нескольких прямоугольников. Рассмотрите случаи пустого и вырожденного пересечения прямоугольников. Реализуйте алгоритм вычисления наименьшего ограничивающего несколько прямоугольников прямоугольника. Используйте стандартный контейнер std::vector для хранения экземпляров структуры Rectangle.

## 03.02 ( 03.04 )

Реализуйте класс Circle для представления окружностей. Реализуйте в классе Circle приватное поле типа double для хранения радиуса. Реализуйте класс Triangle для представления треугольников. Реализуйте в классе Triangle три приватных поля типа double для хранения длин трех сторон. Реализуйте класс Square для представления квадратов. Реализуйте в классе Square приватное поле типа double для хранения длины одной стороны. Реализуйте в классах Circle, Triangle и Square необходимые пользовательские конструкторы и публичные функции-члены perimeter и area для вычисления периметра и площади соответственно. Используйте стандартную константу std::numbers::pi в функциях-членах perimeter и area класса Circle.

## 03.03 ( 03.05 )

Реализуйте класс List для представления односвязного списка. Реализуйте структуру Node для представления узлов односвязного списка как приватную вложенную структуру в классе List. Реализуйте в структуре Node поле типа int для хранения значения текущего узла списка и указатель типа Node * для хранения адреса следующего узла списка. Реализуйте в классе List два приватных указателя типа Node * для хранения адресов первого и последнего узлов списка. Не создавайте в классе List поле для хранения текущего размера списка и пользовательские конструкторы. Реализуйте в классе List публичную функцию-член empty для проверки наличия хотя бы одного узла в списке. Реализуйте в классе List публичную функцию-член show для вывода значений всех текущих узлов списка в окно терминала через стандартный поток std::cout. Реализуйте в классе List публичные функции-члены push_front и push_back для вставки новых узлов с пользовательскими значениями в начало и в конец списка соответственно. Используйте оператор new для динамического выделения памяти. Реализуйте в классе List публичные функции-члены pop_front и pop_back для удаления узлов из начала и из конца списка соответственно. Используйте оператор delete для освобождения памяти. Реализуйте в классе List публичную функцию-член get для получения значения текущего среднего узла списка. Используйте только один цикл для обхода списка в функции-члене get класса List. Реализуйте в классе List пользовательский деструктор, который корректно освободит память, выделенную при создании узлов списка. Выполните модульное и интеграционное тестирование для всех реализованных функций-членов класса List.

## 03.04 ( 03.09 )

Реализуйте систему внешнего тестирования приватных функций-членов некоторого класса. Реализуйте класс Entity. Реализуйте в классе Entity приватные функции-члены test_v1 и test_v2, которые необходимо протестировать. Вспомните о первом принципе SOLID - принципе единственности ответственности. Реализуйте дружественные для класса Entity классы Tester_v1 и Tester_v2 для представления тестировщиков функций-членов test_v1 и test_v2 класса Entity соответственно, чтобы получить прямой доступ к приватной секции класса Entity и избежать использования публичного интерфейса класса Entity. Используйте паттерн Attorney - Client для ограничения доступа классов Tester_v1 и Tester_v2 к приватной секции класса Entity.

## 03.05 ( 03.16 ) // H. Sutter, Exceptional C++

Реализуйте систему раздельного переопределения множественно наследуемых виртуальных функций-членов с одинаковыми сигнатурами и разными реализациями. Реализуйте базовые классы Entity_v1 и Entity_v2. Реализуйте в классах Entity_v1 и Entity_v2 виртуальные функции-члены test с одинаковыми сигнатурами и разными реализациями. Реализуйте производный класс Client, который является наследником интерфейсов классов Entity_v1 и Entity_v2. Обратите внимание, что раздельное переопределение наследуемых виртуальных функций-членов test в классе Client невозможно из-за совпадения их сигнатур. Реализуйте производные классы Adapter_v1 и Adapter_v2 для представления посредников, которые являются наследниками интерфейсов классов Entity_v1 и Entity_v2 соответственно. Реализуйте производный класс Client, который является наследником интерфейсов классов Adapter_v1 и Adapter_v2. Реализуйте в классах Adapter_v1 и Adapter_v2 виртуальные функции-члены test_v1 и test_v2 соответственно, которые вызываются из переопределенных наследуемых виртуальных функций-членов test и раздельно переопределяются в классе Client.

## 03.06 ( 03.17 )

Доработайте Ваше предыдущее решение задачи 03.02 таким образом, чтобы реализация использовала иерархию классов геометрических фигур вместо отдельных классов. Реализуйте абстрактный базовый класс Shape для представления интерфейса геометрических фигур. Реализуйте в классе Shape виртуальный деструктор и публичные чисто виртуальные функции-члены perimeter и area для вычисления периметра и площади соответственно. Реализуйте производный класс Circle для представления окружностей, который является наследником интерфейса класса Shape. Реализуйте производный абстрактный базовый класс Polygon для представления многоугольников, который является наследником интерфейса класса Shape. Реализуйте производные классы Triangle и Rectangle для представления треугольников и прямоугольников соответственно, которые являются наследниками интерфейса класса Polygon. Реализуйте производный класс Square для представления квадратов, который является наследником интерфейса класса Rectangle. Не создавайте в классе Square поле для хранения длины стороны квадрата. Реализуйте в классе Square только пользовательский конструктор, который передает аргументы конструктору класса Rectangle. Используйте спецификатор override при переопределении наследуемых виртуальных функций-членов perimeter и area в классах Circle, Triangle и Rectangle. Используйте спецификатор final для запрета наследования от классов Circle и Square и для запрета переопределения наследуемых виртуальных функций-членов perimeter и area в наследниках класса Rectangle. Используйте стандартный контейнер std::vector для хранения экземпляров классов Circle, Triangle и Square через указатели на класс Shape. Продемонстрируйте работу динамического полиморфизма.

## 03.07 ( 03.30 )

Доработайте пример 03.30 таким образом, чтобы пользователь мог вставлять произвольное количество новых элементов в вектор, используя автоматическое увеличение емкости его встроенного динамического массива при нехватке свободных ячеек памяти. Реализуйте в классе Vector два приватных поля стандартного типа std::size_t для хранения значений емкости и размера. Реализуйте в классе Vector публичные функции-члены capacity и size для получения значений емкости и размера вектора соответственно. Реализуйте в классе Vector публичную функцию-член push_back для вставки нового элемента в первую свободную ячейку памяти вектора с возможностью увеличения емкости его встроенного динамического массива в случае нехватки свободных ячеек памяти. Используйте алгоритм увеличения емкости встроенного динамического массива из Вашего предыдущего решения задачи 02.07. Реализуйте в классе Vector публичную функцию-член clear для удаления всех элементов вектора без выполнения освобождения выделенных для них ячеек памяти. Реализуйте в классе Vector публичную функцию-член empty для проверки наличия хотя бы одного элемента в векторе.

## 03.08 ( 03.31 )

|

## 03.09 ( 03.32 ) // M. Bancila, The Modern C++ Challenge

Реализуйте класс IPv4 для представления IP адресов в стандарте IPv4. Реализуйте в классе IPv4 приватный стандартный контейнер std::array с четырьмя элементами стандартного типа std::uint8_t для хранения IP адресов. Реализуйте в классе IPv4 перегруженные операторы префиксного и постфиксного инкремента и декремента IP адресов. Реализуйте дружественные для класса IPv4 перегруженные операторы сравнения, ввода и вывода IP адресов. Используйте стандартный поток std::stringstream для ввода через строку IP адресов в формате четырех целых чисел от 0 до 255 включительно, разделенных точками. Используйте стандартный поток std::stringstream для вывода через строку IP адресов в формате, который использовался для ввода.

## 03.10 ( 03.34 )

Доработайте пример 03.32 таким образом, чтобы реализация использовала перегруженный оператор трехстороннего сравнения, переписывание выражений и перегруженный оператор сравнения на равенство вместо шести перегруженных операторов сравнения. Реализуйте дружественные для класса Rational перегруженные операторы трехстороннего сравнения и сравнения на равенство. Используйте сильный порядок при сравнении.

# 04. Generic Programming

## 04.01 ( 04.01 )

Доработайте Ваше предыдущее решение задачи 02.10 таким образом, чтобы реализация использовала тип `T` в качестве типа сортируемых данных вместо типа `int`. Предполагайте, что для типа `T` определены все необходимые операции. Используйте шаблоны функций. Не изменяйте детали реализации алгоритма сортировки.

## 04.02 ( 04.06 ) // M. Bancila, The Modern C++ Challenge

Реализуйте алгоритмы вычисления максимального и минимального значений, суммы и среднего арифметического пакета аргументов типа `double`. Используйте вариативные шаблоны функций. Используйте рекурсивное инстанцирование вариативных шаблонов функций при вычислении максимального и минимального значений пакета. Используйте выражения свертки при вычислении суммы и среднего арифметического пакета. Используйте оператор `sizeof...` для определения количества аргументов при вычислении среднего арифметического пакета. Предполагайте, что пакет содержит только аргументы типа `double`, при этом их количество ненулевое.

## 04.03 ( 04.07 ) // M. Bancila, The Modern C++ Challenge

Реализуйте алгоритм вставки пакета аргументов типа `int` в произвольный контейнер, обладающий публичной функцией-членом `push_back`. Используйте вариативный шаблон функции. Используйте выражение свертки. Используйте оператор запятая в качестве бинарного оператора в выражении свертки. Предполагайте, что пакет может содержать аргументы других типов, которые следует игнорировать. Реализуйте перегруженный шаблон функции `handle` для вставки аргументов типа `int` в произвольный контейнер посредством функции-члена `push_back` и игнорирования других аргументов. Используйте стандартный контейнер `std::vector` для тестов.

## 04.04 ( 04.08 )

Доработайте Ваше предыдущее решение задачи 03.10 таким образом, чтобы реализация использовала тип `T` в качестве типов числителя и знаменателя дроби вместо типа `int`. Предполагайте, что для типа `T` определены все необходимые операции. Используйте шаблон класса. Не изменяйте детали реализации класса `Rational`.

## 04.05 ( 04.17 )

Реализуйте алгоритм вычисления N-ого числа ряда Фибоначчи на основе рекурсивного подхода. Используйте метапрограммирование шаблонов. Реализуйте базовый шаблон структуры `Fibonacci` для представления N-ого числа ряда Фибоначчи. Реализуйте в шаблоне структуры `Fibonacci` параметр типа `int` для указания номера N на этапе компиляции. Реализуйте в структуре `Fibonacci` статическое константное поле типа `int` для хранения вычисляемого на этапе компиляции N-ого числа ряда Фибоначчи. Реализуйте в структуре `Fibonacci` статическое утверждение `static_assert` для проверки переполнения типа `int` при вычислениях на этапе компиляции. Реализуйте две полные специализации шаблона структуры `Fibonacci` для представления первого и второго чисел ряда Фибоначчи. Реализуйте шаблон константы для сокращения записей обращений к полю шаблона структуры `Fibonacci`. Реализуйте альтернативный алгоритм вычисления N-ого числа ряда Фибоначчи на основе шаблонов констант. Реализуйте тесты на основе статических утверждений `static_assert`.

## 04.06 ( 04.19 )

Доработайте Ваше предыдущее решение задачи 02.05 таким образом, чтобы реализация использовала вычисления на этапе компиляции вместо вычислений на этапе выполнения. Реализуйте мгновенные функции со спецификатором `consteval` для вычисления чисел $\pi$ и $e$ на этапе компиляции. Используйте стандартный контейнер `std::array` со спецификатором `constexpr` для хранения различных значений числа epsilon. Не изменяйте детали реализации алгоритмов. Реализуйте тесты на основе статических утверждений `static_assert`.

## 04.07 ( 04.21 )

Доработайте пример `04.21` таким образом, чтобы пользователь мог вычитать, умножать и делить дроби на этапе компиляции. Реализуйте шаблоны структур `Sub`, `Mul` и `Div` по аналогии с шаблоном структуры `Sum` для представления операций вычитания, умножения и деления дробей соответственно. Используйте структуры `Sum` и `Mul` в структурах `Sub` и `Div` соответственно во избежание дублирования реализаций. Реализуйте в структурах `Sum` и `Mul` алгоритмы сокращения дробей. Используйте стандартную функцию `std::gcd`. Реализуйте в структуре `Div` статическое утверждение `static_assert` для проверки деления на ноль при вычислениях на этапе компиляции. Реализуйте шаблоны псевдонимов для сокращения записей обращений к псевдонимам типов в шаблонах структур `Sub`, `Mul` и `Div` по аналогии с шаблоном псевдонима `sum`. Реализуйте шаблон перегруженного оператора вычитания интервалов со спецификатором `constexpr`, используя внутри него шаблон перегруженного оператора сложения интервалов. Реализуйте тесты на основе статических утверждений `static_assert`.

## 04.08 ( 04.22 )

Доработайте пример `04.22` таким образом, чтобы пользователь мог использовать кортеж на этапе компиляции. Добавьте основному конструктору и шаблону функции-члена `get` класса `Tuple` спецификатор `constexpr`. Реализуйте в классе `Tuple` публичную функцию-член `size` со спецификатором `constexpr` для получения значения размера кортежа. Используйте оператор `sizeof...` для определения размера пакета параметров шаблона класса `Tuple` в функции-члене `size`. Реализуйте тесты на основе статических утверждений `static_assert`.

## 04.09 ( 04.24 )

Доработайте пример `04.24` таким образом, чтобы пользователь мог проверять наличие некоторого заданного типа в очереди. Реализуйте базовый шаблон структуры `Has`, а также две частичные специализации для пустой и непустой очередей соответственно по аналогии с шаблоном класса `Max_Type`. Используйте стандартный шаблон свойств `std::is_same` для сравнения типов. Реализуйте шаблон константы для сокращения записей обращений к полю шаблона структуры `Has`. Реализуйте тесты на основе статических утверждений `static_assert`.

## 04.10 ( 04.35 )

Реализуйте шаблон свойств `is_class` по аналогии со стандартным шаблоном свойств `std::is_class`. Не добавляйте проверку на основе стандартного шаблона свойств `std::is_union` в собственную реализацию шаблона свойств `is_class`. Объясните концепцию указателя на член класса в реализации стандартного шаблона свойств `std::is_class`. Реализуйте метафункции `add_const` и `remove_const` по аналогии со стандартными метафункциями `std::add_const` и `std::remove_const` соответственно. Реализуйте метафункцию `decay` по аналогии со стандартной метафункцией `std::decay`. Реализуйте метафункцию `conditional` по аналогии со стандартной метафункцией `std::conditional`. Реализуйте шаблоны псевдонимов и шаблоны переменных для всех шаблонов свойств и метафункций. Реализуйте тесты на основе статических утверждений `static_assert`.

# 05. Software Design Patterns

## 05.01 ( 05.01 )

Доработайте пример 05.01 таким образом, чтобы пользователь мог выполнять поэтапное создание составного объекта следующим образом: `auto person = builder.name("Ivan").age(25).grade(10).get()`, где `person` является указателем на экземпляр класса `Person`, а `builder` является экземпляром класса `Builder`. Реализуйте класс `Person` для представления составного объекта. Реализуйте класс `Builder` для представления процесса создания составного объекта. Реализуйте в классе `Builder` конструктор по умолчанию для создания составного объекта в начальном состоянии. Реализуйте в классе `Builder` публичные функции-члены для создания составного объекта и публичную функцию-член `get` для получения указателя на составной объект.

## 05.02 ( 05.10 )

Доработайте пример 05.10 таким образом, чтобы реализация использовала шаблон класса вместо ссылки. Реализуйте шаблон производного класса `Decorator`, который является наследником интерфейса класса `Entity` и наследником реализации собственного параметра шаблона. Используйте класс `Entity` как виртуальный базовый класс. Удалите в классе `Decorator` пользовательский конструктор и ссылку на экземпляр класса `Entity`.

## 05.03 ( 05.16 )

Реализуйте систему программного каркаса для некоторой абстрактной стратегической компьютерной игры. Используйте по крайней мере один порождающий паттерн проектирования, например, Builder, для контроля над созданием игровых объектов; один структурый паттерн проектирования, например, Composite, для организации взаимодействия игровых объектов и один поведенческий паттерн проектирования, например, Template Method, для управления поведением игровых объектов в системе. Используйте любые другие паттерны проектирования при необходимости. Поставьте себя на место архитектора или инженера данной программной системы.

## 05.04 ( 05.17 )

Доработайте пример 05.15 таким образом, чтобы реализация использовала статический полиморфизм вместо динамического полиморфизма. Удалите класс `Strategy` и наследование от него всех остальных классов. Реализуйте шаблон производного класса `Entity`, который является наследником реализации собственного параметра шаблона. Удалите в классе `Entity` пользовательский конструктор и ссылку на экземпляр класса `Strategy`.

## 05.05 ( 05.21 )

Доработайте Ваше предыдущее решение задачи 04.04 таким образом, чтобы реализация использовала миксины для подмешивания некоторых дополнительных операторов вместо определения всех операторов непосредственно внутри дроби. Реализуйте шаблоны базовых классов `addable`, `subtractable`, `multipliable`, `dividable`, `incrementable` и `decrementable` для представления подмешиваемых дружественных для указанных базовых классов операторов сложения, вычитания, умножения, деления, префиксного и постфиксного инкремента и декремента дробей соответственно. Реализуйте производный класс `Rational`, который является наследником реализации указанных базовых классов. Используйте документацию библиотеки Boost.Operators.

# 06. Projects and Libraries

## 06.01 (06.05)

Доработайте Ваше предыдущее решение задачи 03.10 таким образом, чтобы реализация использовала заголовочный файл и соответствующий ему дополнительный файл исходного кода вместо одного файла исходного кода. Создайте заголовочный файл, содержащий определение класса `Rational`, файл исходного кода, содержащий реализации некоторых функций-членов класса `Rational`, и файл исходного кода, содержащий реализацию функции `main` с тестами и демонстрационными примерами. Реализуйте защиту от повторого включения заголовочного файла на основе директивы препроцессора `pragma`. Реализуйте небольшие функции-члены класса `Rational` в определении класса `Rational` в заголовочном файле. Не используйте предкомпилированные заголовочные файлы. Не используйте глобальные объекты. Не используйте внутреннее связывание. Продемонстрируйте и объясните основные разновидности ошибок, которые могут возникнуть на этапе компоновки.

## 06.02 (06.13)

Доработайте Ваше предыдущее решение задачи 06.01 таким образом, чтобы реализация использовала модуль вместо заголовочного файла и соответствующего ему файла исходного кода. Создайте юнит интерфейса модуля, содержащий определение класса `Rational`, юнит реализации модуля, содержащий реализации некоторых функций-членов класса `Rational`, и файл исходного кода, содержащий реализацию функции `main` с тестами и демонстрационными примерами. Используйте модули вместо заголовочных файлов стандартной библиотеки. Поместите класс `Rational` и остальные реализованные компоненты в собственное пространство имен. Используйте одиночный, групповой экспорт или экспорт пространства имен при экспорте символов из модуля. Не используйте подмодули. Продемонстрируйте и объясните способ организации модулей на основе разделов.

## 06.03 (06.16)

Сравните время сборки Вашего предыдущего решения задачи 06.01 и время сборки Вашего предыдущего решения задачи 06.02. Определите время сборки каждого объектного файла, время компоновки объектных файлов и полное время сборки обоих решений. Используйте команду time для измерения времени. Используйте флаг компилятора -c для сборки объектных файлов. Добавьте дополнительные заголовочные файлы стандартной библиотеки в Ваши решения. Определите размер каждого объектного файла и полный размер исполняемого файла каждого решения. Прочитайте статью, в которой приводятся результаты подобного сравнения.

## 06.04 (education.sh)

Выполните автоматизированную сборку всех примеров из репозитория Education. Установите глобально библиотеки Boost версии 1.85, выполнив команды из второго закомментированного блока скрипта education.sh. Установите систему автоматизации сборки CMake версии 3.28 или новее, выполнив команду sudo apt install cmake. Установите глобально дополнительные необходимые библиотеки, указанные в командах `find_package` в файле CMakeLists.txt проекта examples из репозитория Education. Используйте любые удобные открытые источники и документации библиотек на GitHub в качестве подкрепления. Обратите внимание, что большинство библиотек собираются и устанавливаются вручную из склонированных локально репозиториев, однако некоторые могут быть установлены через пакетные менеджеры. Например, библиотека TBB устанавливается командой sudo apt install libtbb-dev. Выполните сборку только тех примеров из репозитория Education, которые используют стандартную библиотеку, Boost, TBB, Google.Test или Google.Benchmark. Закомментируйте инструкции сборки остальных примеров в файле CMakeLists.txt проекта examples из репозитория Education. Выполните скрипт education.sh. Приложите в качестве результатов скриншоты терминалов, показывающих содержимое директории libraries и трех директорий output после завершения процесса сборки всех примеров.

## 06.05 (06.17)

Доработайте пример 06.17 таким образом, чтобы пользователь мог указывать версию динамической библиотеки и импортировать из нее необходимую функцию во время выполнения программы без прерывания работы и повторной сборки исполняемого файла. Реализуйте две динамические библиотеки с разными названиями. Реализуйте в библиотеках функции `test` с одинаковыми сигнатурами и разными реализациями. Реализуйте в функции `main` исполняемого файла возможность для пользователя указать версию библиотеки перед импортом функции `test`. Используйте стандартный поток `std::cin` для ввода через терминал названия файла динамической библиотеки. Используйте библиотеку Boost.DLL для импорта функций `test` из динамических библиотек. Используйте систему автоматизации сборки CMake для сборки библиотек и исполняемого файла.

# 07. Debugging and Profiling

## 07.01 (07.09)

Доработайте Ваше предыдущее решение задачи 02.02 таким образом, чтобы реализация использовала для хранения корней опционалы и варианты вместо отдельных переменных. Реализуйте функцию `solve` для вычисления корней, которая принимает в качестве аргументов коэффициенты a, b и c типа `double` и возвращает опционал, содержащий вариант из одного, двух или бесконечного количества корней. Используйте стандартную оболочку `std::optional` для представления опционала из нулевого или ненулевого количества корней. Используйте стандартную оболочку `std::variant` для представления варианта из одного, двух или бесконечного количества корней. Используйте тип `double` для хранения одного корня. Используйте стандартную оболочку `std::pair` для хранения двух корней типа `double`. Используйте стандартный тэг `std::monostate` как тип для обозначения бесконечного количества корней. Не используйте стандартную функцию `std::visit`.

## 07.02 (07.11)

Доработайте Ваше предыдущее решение задачи 05.05 таким образом, чтобы пользователь мог обрабатывать ошибки с помощью исключений. Реализуйте класс `Exception` для представления пользовательского исключения, который является наследником интерфейса стандартного исключения `std::exception`. Переопределите наследуемую виртуальную функцию-член `what` в классе `Exception`. Генерируйте исключение типа `Exception` в конструкторе класса `Rational` при передаче нулевого знаменателя. Реализуйте в функции `main` блок `try` для перехвата исключений и обработчик `catch` для стандартного исключения `std::exception`, а также обработчик `catch` для всех исключений. Используйте стандартный поток `std::cerr` для вывода через терминал всех сообщений об ошибках. Продемонстрируйте в функции `main` генерацию, перехват и обработку стандартных исключений `std::bad_alloc`, `std::bad_variant_access` и `std::bad_optional_access`, а также стандартных исключений `std::length_error` и `std::out_of_range`, генерируемых функциями-членами стандартного контейнера `std::vector`. Объясните причины генерации всех перечисленных выше стандартных исключений.

## 07.03 (07.12) // H. Sutter, Exceptional C++

Исследуйте представленную ниже функцию. Определите в ней такие элементы, которые могут приводить к нормальному ветвлению пути выполнения или генерации некоторого исключения. Считайте, что `Person` является пользовательским классом, а `Status` является пользовательским перечислением с областью видимости.

```
void test(Person const & person)
{
    std::cout << "test : " << person.name() << '\n';

    if (person.grade() == 10 || person.salary() > 1'000'000)
    {
        save(Status::success, person.id());
    }
    else
    {
        save(Status::failure, person.id());
    }
}
```

## 07.04 (07.22)

Доработайте Ваше предыдущее решение задачи 04.01 таким образом, чтобы пользователь мог выполнять тестирование алгоритма сортировки. Используйте автоматизированные тесты на основе библиотеки Google.Test.

## 07.05 (07.23)

Доработайте Ваше предыдущее решение задачи 07.04 таким образом, чтобы пользователь мог выполнять профилирование производительности алгоритма сортировки. Используйте микробенчмарки на основе библиотеки Google.Benchmark. Реализуйте параметризованный тест, передавая в роли аргумента пороговый размер сортируемого сегмента контейнера при выборе между алгоритмом быстрой сортировки и алгоритмом сортировки вставками. Используйте для тестирования контейнер из 10000 обратно сортированных элементов типа `double`.

# 08. Number Processing

## 08.01 ( 08.09 ) // H. Sutter, Exceptional C++ Style

Реализуйте систему мошеннического изменения приватного поля экземпляра некоторого класса внешним пользователем, не являющимся другом этого класса. Реализуйте класс `Entity_v1`. Реализуйте в классе `Entity_v1` приватное поле типа `int`. Реализуйте класс `Entity_v2`. Реализуйте в классе `Entity_v2` публичное поле типа `int`. Используйте оператор `reinterpret_cast` для преобразования ссылки на экземпляр класса `Entity_v1` в ссылку на экземпляр класса `Entity_v2`. Реализуйте хотя бы одну другую систему мошеннического изменения приватного поля экземпляра некоторого класса внешним пользователем, не являющимся другом этого класса.

## 08.02 ( 08.15 )

Доработайте пример `08.15` таким образом, чтобы пользователь мог брать остаток от деления, возводить в степень и вычислять абсолютное значение длинных чисел. Реализуйте в классе `Integer` перегруженный оператор взятия остатка от деления с присваиванием. Реализуйте дружественный для класса `Integer` перегруженный оператор взятия остатка от деления. Реализуйте дружественную для класса `Integer` функцию `pow` для возведения длинного числа в целую степень типа `unsigned int`. Реализуйте в классе `Integer` публичные функции-члены `sign` и `abs` для получения знака и вычисления абсолютного значения длинного числа соответственно.

## 08.03 ( 08.21 )

Реализуйте алгоритмы вычисления целой части двоичного логарифма положительных чисел типа `int` и типа `float`. Предполагайте, что оба этих типа имеют размер 4 байта. Используйте значения типа `unsigned int` для выполнения побитовых операций. Используйте оператор `static_cast` для явного преобразования числа типа `int` к значению типа `unsigned int` и объединение `union` для явного преобразования числа типа `float` к значению типа `unsigned int`. Используйте цикл `while` и оператор побитового сдвига вправо для поиска старшего ненулевого бита в значении типа `unsigned int`. Рассматривайте представление числа типа `float` в соответствии со стандартом IEEE 754. Рассматривайте как нормализованные, так и денормализованные числа типа `float`. Учитывайте, что экспонента числа типа `float` имеет смещение на 127, которое обеспечивает хранение отрицательных степеней без знакового бита. Учитывайте, что максимальное значение экспоненты числа типа `float` используется для представления значения бесконечности inf и неопределенного значения nan. Рассмотрите представление чисел с плавающей точкой в памяти компьютера, используя данную презентацию.

## 08.04 ( 08.28 ) // M. Bancila, The Modern C++ Challenge

Реализуйте алгоритм эволюции Ричарда Докинза, описанный им в третьей главе книги Слепой часовщик. Сгенерируйте начальную строку из 23 случайных букв. Сгенерируйте 100 копий начальной строки, заменяя каждую букву начальной строки другой случайной буквой с вероятностью 0.05. Вычислите для каждой из 100 сгенерированных строк значение метрики, показывающей посимвольное расхождение сгенерированной строки и целевой строкой methinksitislikeaweasel. Завершите работу алгоритма, если для любой из 100 сгенерированных строк значение метрики оказалось равным 0, в противном случае выберите любую сгенерированную строку с наименьшим значением метрики в качестве новой начальной строки и повторите итерацию алгоритма. Используйте только строчные буквы английского алфавита. Используйте стандартный источник энтропии `std::random_device`. Используйте стандартный генератор `std::default_random_engine`. Используйте стандартное распределение `std::uniform_real_distribution`. Используйте стандартный поток `std::cout` для вывода через терминал начальных строк на всех итерациях алгоритма эволюции и конечной целевой строки.

## 08.05 ( 08.37 )

Доработайте пример `08.37` таким образом, чтобы пользователь мог проводить серии измерений временных интервалов с последующим усреднением. Реализуйте в классе `Timer` публичные функции-члены `start` и `stop` для запуска и остановки каждого измерения. Реализуйте в классе `Timer` приватное поле типа `bool` для проверки и обновления состояния таймера в функциях-членах `start` и `stop`. Реализуйте в классе `Timer` приватный стандартный контейнер `std::vector` для хранения стандартных интервалов `std::chrono::duration` с типом `double` в качестве первого параметра шаблона, соответствующих отдельным измерениям. Реализуйте в классе `Timer` публичную функцию-член `average` для вычисления и получения среднего значения времени в секундах.

# 09. Memory Management

## 09.01 ( 09.01 )

Реализуйте класс `Tracer` для представления трассировщика вызовов функций. Реализуйте в классе `Tracer` пользовательский конструктор по умолчанию и пользовательский деструктор, которые будут выводить парные сообщения. Используйте паттерн RAII. Предполагайте, что пользователь самостоятельно создает экземпляр класса `Tracer` в начале каждой собственной функции. Используйте стандартный поток `std::cout` для вывода через терминал всех сообщений. Используйте стандартную утилиту `std::source_location` для вывода дополнительной информации в сообщениях. Реализуйте функциональный макрос `trace` по аналогии со стандартным макросом `assert`, который позволит отключить всю трассировку при определении макроса `NDEBUG`.

## 09.02 ( 09.06 )

Реализуйте класс `Tree` для представления бинарного дерева. Реализуйте структуру `Node` для представления узлов бинарного дерева как публичную вложенную структуру в классе `Tree`. Реализуйте в структуре `Node` поле типа `int` для хранения значения текущего узла дерева, два стандартных указателя `std::shared_ptr` для хранения адресов правого и левого дочерних узлов и стандартный указатель `std::weak_ptr` для хранения адреса родительского узла. Реализуйте в классе `Tree` публичный стандартный указатель `std::shared_ptr` для хранения адреса корневого узла дерева. Реализуйте в классе `Tree` публичные функции-члены `traverse_v1` и `traverse_v2` для вывода значений всех текущих узлов дерева в окно терминала через стандартный поток `std::cout` при обходе дерева алгоритмами поиска в ширину и поиска в глубину соответственно. Сконструируйте в функции `main` экземпляр класса `Tree` таким образом, чтобы дерево содержало корневой узел, два дочерних узла на промежуточном уровне и четыре дочерних узла на последнем уровне. Продемонстрируйте отсутствие неразрывных ассоциативных связей между узлами дерева и корректную работу деструкторов узлов.

## 09.03 ( 09.09 )

Доработайте примеры `05.01`, `05.03`, `05.04`, `05.09` и `05.13` таким образом, чтобы реализации использовали стандартные интеллектуальные указатели вместо обычных указателей. Используйте стандартные указатели `std::shared_ptr` и `std::unique_ptr`. Не используйте стандартный указатель `std::weak_ptr`. Подумайте над тем, какой стандартный интеллектуальный указатель будет уместно использовать в каждом из примеров.

## 09.04 ( 09.11 )

Доработайте Ваше предыдущее решение задачи 04.01 таким образом, чтобы реализация использовала итераторы вместо индексов. Реализуйте передачу коллекции элементов в алгоритм сортировки через передачу по значению двух итераторов начала и конца данной коллекции. Используйте полуоткрытые диапазоны. Предполагайте, что элементы коллекции могут храниться в любом контейнере, который владеет итераторами произвольного доступа. Используйте стандартные функции `std::distance`, `std::advance`, `std::next` и `std::prev`.

## 09.05 ( 09.15 )

Доработайте пример `09.15` таким образом, чтобы реализация использовала двунаправленные итераторы вместо однонаправленных итераторов. Реализуйте в структуре `Node` стандартный указатель `std::weak_ptr` для хранения адреса предыдущего узла списка. Реализуйте в классе `Iterator` перегруженные операторы префиксного и постфиксного декремента. Доработайте в классе `List` текущую реализацию функции-члена `push_back`.

## 09.06 ( 09.16 )

Реализуйте алгоритм вычисления чисел ряда Фибоначчи на основе однонаправленного итератора. Реализуйте класс `Iterator` для представления однонаправленного итератора. Реализуйте в классе `Iterator` два приватных поля типа `int` для хранения двух смежных чисел ряда Фибоначчи. Реализуйте в классе `Iterator` конструктор по умолчанию, перегруженные операторы префиксного и постфиксного инкремента, разыменования и сравнения на равенство. Реализуйте операторы инкремента таким образом, чтобы они вычисляли следующие два смежных числа ряда Фибоначчи на основе двух текущих известных чисел ряда Фибоначчи, а оператор разыменования таким образом, чтобы он возвращал последнее вычисленное число ряда Фибоначчи.

## 09.07 ( 09.21 ) // H. Sutter, Exceptional C++

Доработайте пример `06.09` таким образом, чтобы реализация повторно использовала однократно выделенную неинициализированную память для хранения деталей реализации основного класса вместо многократного выделения и освобождения памяти. Определите накладные расходы и влияние на производительность оригинального паттерна Pimpl. Реализуйте в классе `Entity` приватный стандартный контейнер `std::array` со спецификатором `alignas(std::max_align_t)` и шестнадцатью элементами стандартного типа `std::byte` для хранения экземпляра класса `Implementation`. Реализуйте в конструкторе класса `Entity` два статических утверждения `static_assert` для проверки размера и выравнивания класса `Implementation`. Используйте размещающую версию оператора `new` в конструкторе класса `Entity`. Используйте стандартную функцию `std::destroy_at` в деструкторе класса `Entity`. Реализуйте в классе `Entity` константную и неконстантную публичные функции-члены `get` для получения отмытого указателя на экземпляр класса `Implementation`. Используйте стандартные функции `std::bit_cast` и `std::launder` в обеих функциях-членах `get` класса `Entity`.

## 09.08 ( 09.23 )

Доработайте пример `09.23` таким образом, чтобы пользователь мог обслуживать память встроенных динамических массивов с помощью перегруженных функций выделения и освобождения памяти. Реализуйте в классе `Entity` публичные статические функции-члены `operator new[]` и `operator delete[]` для выделения и освобождения памяти встроенных динамических массивов. Реализуйте в классе `Entity` публичные статические функции-члены `operator new`, `operator delete`, `operator new[]` и `operator delete[]` для выделения и освобождения памяти встроенных динамических массивов без генерации исключений, которые имеют дополнительный второй параметр стандартного типа `std::nothrow_t`, передаваемый с помощью константной lvalue-ссылки. Реализуйте в классе `Client` все необходимые дополнительные публичные объявления `using`.

## 09.09 ( 09.35 )

Доработайте пример `09.35` таким образом, чтобы пользователь мог выбирать алгоритм поиска свободных блоков памяти. Реализуйте в классе `Allocator` приватную функцию-член `find_first` для представления алгоритма поиска первого подходящего свободного блока памяти. Реализуйте в классе `Allocator` приватную функцию-член `find_best` для представления алгоритма поиска лучшего подходящего свободного блока памяти. Реализуйте в конструкторе класса `Allocator` дополнительный параметр для выбора алгоритм поиска свободных блоков памяти. Сравните среднее время работы аллокатора, использующего первый алгоритм поиска свободных блоков памяти, и среднее время работы того же аллокатора, использующего второй алгоритм поиска, в полностью одинаковых тестах. Используйте микробенчмарки на основе библиотеки Google.Benchmark.

## 09.10 ( 09.37 )

# 10. Data Structures

## **10.01** ( 10.29 ) // M. Bancila, The Modern C++ Challenge

Реализуйте алгоритм моделирования игры Жизнь по стандартным правилам на игровом поле размером 10 на 10 клеток. Задайте начальное состояние игрового поля случайным образом. Используйте стандартный двумерный контейнер `std::vector` для хранения текущего состояния игрового поля. Используйте стандартный поток `std::cout` для вывода через терминал состояний игрового поля на всех итерациях моделирования игры.

## **10.02** ( 10.33 )

Реализуйте алгоритм вычисления N - ого числа ряда Фибоначчи на основе метода матричной экспоненциации. Используйте тип `boost::numeric::ublas::matrix` для осуществления вычислений с матрицами. Реализуйте алгоритм быстрого возведения начальной матрицы в степень N. Используйте тип `usigned long long int` для значений элементов матриц. Обоснуйте алгоритмическую сложность реализованного алгоритма и сравните ее с алгоритмической сложностью других известных Вам алгоритмов вычисления N - ого числа ряда Фибоначчи.

# 11. Algorithms and Ranges

## 11.01 (11.01) // H. Sutter, Exceptional C++

Реализуйте функцию, которая возвращает указатель на саму себя так, чтобы можно было написать следующее: `Wrapper function = test(); (*function)();` Используйте класс с перегруженным оператором приведения.

## 11.02 (11.08)

Сравните среднее время вызова лямбда - функции, сохраненной в переменной с местозаменителем типа `auto`, и среднее время вызова той же самой лямбда - функции, сохраненной в стандартной оболочке `std::function`.

## 11.03 (11.27)

Реализуйте алгоритм решения задачи коммивояжера для полносвязного графа, содержащего 10 вершин. Используйте тип `int` для значений весов ребер графа. Инициализируйте веса всех ребер графа случайными значениями от 1 до 10. Используйте стандартный двумерный контейнер `std::vector` в качестве симметричной относительно главной диагонали матрицы инцидентности для хранения весов ребер графа. Используйте стандартный алгоритм `std::next_permutation` для перебора всех перестановок последовательности обхода вершин, включающей в себя каждую вершину графа только один раз. Учтите, что обход вершин графа должен завершаться в начальной вершине. Используйте стандартный поток `std::cout` для вывода через терминал матрицы инцидентности, оптимальной последовательности обхода вершин, а также ее суммарной стоимости.

# 12. String Processing

## 12.01 (12.02)

Напишите программу, которая выводит собственный исходный код в окно терминала через стандартный поток `std::cout`. Не используйте файловые потоки ввода. Предполагайте, что файл с исходным кодом недоступен.

# 13. Streams and Serialization

# 14. Parallel Programming

## 14.01 (14.09)

Доработайте пример 14.09 таким образом, чтобы реализация использовала передаваемый пользователем вызываемый объект и применяла его ко всем элементам коллекции вместо вычисления суммы с помощью стандартной политики `std::plus`. Сравните тип возвращаемого результата реализованного параллельного алгоритма с типом возвращаемого результата стандартного алгоритма `std::for_each` без политики параллельного выполнения. Обоснуйте причины различия типов возвращаемых результатов указанных алгоритмов.

## 14.02 (14.10)

Реализуйте систему обработки исключений, которые могут быть выброшены функциями, выполняемыми в дополнительных стандартных потоках `std::thread`. Используйте обработчик `catch` для перехвата всех исключений в функции дополнительного потока. Используйте стандартный указатель `std::exception_ptr` для хранения исключения, который будет доступен в функции дополнительного потока и в функции основного потока. Используйте стандартную функцию `std::current_exception` для получения указателя на текущее исключение в функции дополнительного потока. Используйте стандартную функцию `std::rethrow_exception` для повторной генерации, перехвата и обработки сохраненного ранее исключения в функции основного потока.

# 15. Computer Network Systems

## **15.01** (15.01) // M. Bancila, The Modern C++ Challenge

Реализуйте алгоритмы шифрования и дешифрования строк, используя шифр Цезаря со сдвигом вправо на заданное количество символов. Предполагайте, что все строки состоят из строчных букв английского алфавита.

## **15.02** (15.02) // M. Bancila, The Modern C++ Challenge

Реализуйте алгоритмы шифрования и дешифрования строк, используя шифр Виженера. Предполагайте, что все строки состоят из строчных букв английского алфавита. Реализуйте алгоритм генерации таблицы Виженера на этапе компиляции. Используйте мгновенную функцию со спецификатором `consteval` и стандартный двумерный контейнер `std::array` со спецификатором `constexpr` для генерации и хранения таблицы Виженера.