Software Engineering

Moscow Institute of Physics and Technology

Table of Contents

00. Mandatory Requirements

Требования: часть 1

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

Требования: часть 2

- Ваши решения не должны использовать стандартные символьные потоки ввода и вывода.
- Ваши решения не должны содержать дублирующийся код, неадекватные названия и магические литералы.
- Ваши решения не должны содержать глобальные объекты и неинициализированные переменные.
- Ваши решения не должны содержать символы, не входящие в таблицу ASCII.

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

01. Introduction and Brief Overview

01.01 [06.07]

Напишите программу, которая выводит в стандартный символьный поток вывода `std::cout` любую строку 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 символьный поток ввода `std::cin` для ввода номера N. Используйте стандартный символьный поток вывода `std::cout` для вывода 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 символьный поток ввода `std::cin` для ввода коэффициентов a, b и c. Используйте стандартный символьный поток вывода `std::cout` для вывода корней 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 `pedantic`. Используйте секцию `default` для пятого класса. Используйте стандартный символьный поток ввода `std::cin` для ввода символов. Используйте стандартный символьный поток вывода `std::cout` для вывода названий классов. Не сопровождайте Ваше решение данной задачи тестами.

02.04 [02.20]

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

02.05 [02.24]

Реализуйте алгоритм вычисления числа e на основе суммы членов ряда Маклорена при x равном 1 с точностью, заданной числом epsilon. Используйте тип `double` для промежуточных вычислений и конечного значения числа e. Не вычисляйте факториалы в знаменателях членов ряда Маклорена, чтобы не столкнуться с проблемой переполнения. Используйте известное соотношение между членами ряда Маклорена для оптимизации вычисления каждого нового члена ряда на основе предыдущего члена ряда. Используйте цикл `while` для вычисления членов ряда Маклорена, пока очередной член ряда не станет меньше числа epsilon. Используйте стандартный символьный поток ввода `std::cin` для ввода числа epsilon. Используйте стандартный символьный поток вывода `std::cout` для вывода числа e. Не сопровождайте Ваше решение данной задачи тестами.

02.06 [02.29]

Реализуйте алгоритмы вычисления максимального и минимального значений, среднего арифметического и стандартного отклонения коллекции чисел типа `double`. Используйте встроенный статический массив. Используйте стандартный символьный поток ввода `std::cin` для ввода коллекции чисел. Используйте стандартный символьный поток вывода `std::cout` для вывода максимального и минимального значений, среднего арифметического и стандартного отклонения коллекции чисел. Не сопровождайте Ваше решение данной задачи тестами.

02.07 [02.30]

Доработайте Ваше решение задачи 02.06, используя встроенный динамический массив вместо статического.

02.08 [02.31]

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 псевдоним `std::size_t` для их длин. Используйте кэширование длин последовательностей Коллатца в стандартном контейнере `std::vector` для оптимизации вычисления длины каждой новой последовательности на основе предыдущих последовательностей. Используйте стандартный символьный поток вывода `std::cout` для вывода наибольшей длины последовательности Коллатца среди всех рассмотренных, а также ее начального значения. Не сопровождайте Ваше решение данной задачи тестами.

02.09 [02.41]

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

02.10 [02.42]

Доработайте пример 02.42 таким образом, чтобы вместо алгоритма сортировки слиянием использовался алгоритм быстрой сортировки. Обоснуйте временную сложность полученного гибридного 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` поля типа `double` 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 как свободную функцию. Рассмотрите случаи вырожденного и пустого пересечения. Реализуйте алгоритм вычисления минимального ограничивающего прямоугольника как свободную функцию. Используйте стандартный контейнер `std::vector` для хранения экземпляров структуры `Rectangle`.

03.02 [03.04]

Реализуйте классы `Triangle`, `Square` и `Circle` для представления треугольников, квадратов и окружностей. Реализуйте в классе `Triangle` три поля типа `double` для хранения длин трех сторон, в классе `Square` одно поле типа `double` для хранения длины одной стороны, в классе `Circle` одно поле типа `double` для хранения радиуса. Реализуйте алгоритмы вычисления периметра и площади указанных фигур как публичные функции-члены соответствующих классов. Используйте стандартную константу `std::numbers::pi` для класса `Circle`.

03.03 [03.05]

Реализуйте структуру `Node` для представления узлов односвязного списка. Реализуйте в структуре `Node` поле типа `int` для хранения значения текущего узла списка и поле типа `Node *` для хранения адреса следующего узла списка. Реализуйте класс `List` для представления односвязного списка. Реализуйте в классе `List` два приватных поля типа `Node *` для хранения адресов головного и хвостового узлов списка. Не создавайте в классе `List` поле для хранения текущей длины списка. Реализуйте в классе `List` публичную функцию-член `empty` для проверки наличия узлов в списке. Реализуйте в классе `List` публичную функцию-член `show` для вывода в стандартный символьный поток вывода `std::cout` значений всех текущих узлов списка. Реализуйте в классе `List` публичные функции-члены `push_front` и `push_back` для добавления новых узлов 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`. Реализуйте деструктор, который корректно освободит память, выделенную при создании узлов списка. Используйте литерал `nullptr` для создания нулевых указателей.

03.04 [03.10]

Реализуйте внешнюю систему тестирования приватных функций-членов некоторого класса. Реализуйте класс `Entity`. Реализуйте в классе `Entity` две демонстрационные приватные функции-члены, которые необходимо будет подвергнуть тестированию. Вспомните о принципе SOLID единственности ответственности. Реализуйте два дополнительных самостоятельных класса-тестировщика, каждый из которых будет предназначен для тестирования одной соответствующей ему приватной функции-члена класса `Entity`. Используйте отношение дружбы между классами-тестировщиками и классом `Entity`, чтобы выполнить тестирование приватных функций-членов класса `Entity` без необходимости обращения к его публичному интерфейсу. Используйте паттерн Attorney - Client для ограничения доступа классов-тестировщиков к приватной секции класса `Entity`.

04. Generic Programming

05. Software Architecture Patterns

06. Projects and Libraries

07. Handling Errors and Debugging

08. Instruments of Calculus

08.01 [08.02]

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`.

08.02 [08.05]

Сформулируйте способ хранения полухода шахматной партии с минимально возможными затратами памяти.

09. Detailed Memory Management

09.01 [09.01]

Реализуйте класс `Tracer` для трассировки вызовов функций. Используйте паттерн RAII для вывода парных сообщений в конструкторе и деструкторе класса `Tracer`. Предполагайте, что пользователь самостоятельно создает экземпляр класса `Tracer` в начале каждой собственной функции. Используйте стандартную утилиту `std::source_location` для вывода дополнительной информации в сообщениях. Реализуйте функциональный макрос `trace` по типу `assert` с возможностью отключения всей трассировки при определении макроса `NDEBUG`.

10. Collections and Containers

10.01 [10.29]

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 двумерный стандартный контейнер `std::vector` в качестве симметричной относительно главной диагонали матрицы инцидентности для хранения весов ребер графа. Реализуйте вспомогательный алгоритм для перебора всех перестановок последовательности, включающей в себя каждую вершину графа только один раз. Учтите, что обход вершин графа должен завершаться в начальной вершине.

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 исходной 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. Iterators and Algorithm Libraries

11.01 [11.01]

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

12. Text Data Processing

12.01 [12.02]

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

13. Streams and Data Serialization

14. Concurrent Programming

15. Network Technologies and Tools

15.01 [15.01]

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]

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 двумерный стандартный контейнер `std::array` со спецификатором `constexpr` для генерации и хранения таблицы Виженера.