Universal C++ Engineer's CodeX

est. January, 2018 by Matthias

CodeX Sections

01 General Introduction . . . **3**
01.01 [12] Questions . . . 3
01.02 [75] Functions . . . 3

02 Programming Basics . . . **4**
02.01 [31] Questions . . . 4
02.02 [25] Characters . . . 4
02.03 [50] Numbers (1) . . . 4
02.04 [25] Numbers (2) . . . 4
02.05 [25] Numbers (3) . . . 4
02.06 [25] Numbers (4) . . . 4
02.07 [25] Numbers (5) . . . 4
02.08 [25] Recursion . . . 4
02.09 [25] Sortings (1) . . . 4
02.10 [25] Sortings (2) . . . 4
02.11 [25] Sortings (3) . . . 4
02.12 [25] Sortings (4) . . . 4
02.13 [25] Algorithms (1) . . . 4
02.14 [25] Algorithms (2) . . . 4

03 Classes Development . . . **5**
03.01 [44] Questions . . . 5
03.02 [25] Structures . . . 5
03.03 [25] Classes (1) . . . 5
03.04 [50] Classes (2) . . . 5
03.05 [50] Classes (3) . . . 5
03.06 [25] Classes (4) . . . 5

04 Templates Overview . . . **6**
04.01 [15] Questions . . . 6
04.02 [25] Templates (1) . . . 6
04.03 [25] Templates (2) . . . 6
04.04 [25] Templates (3) . . . 6
04.05 [25] Templates (4) . . . 6
04.06 [25] Templates (5) . . . 6
04.07 [25] Templates (6) . . . 6
04.08 [25] Templates (7) . . . 6
04.09 [25] Templates (8) . . . 6
04.10 [25] Constexprs (1) . . . 6
04.11 [25] Constexprs (2) . . . 6
04.12 [25] Operators . . . 6

05 Patterns and Idioms . . . **7**
05.01 [10] Patterns (1) . . . 7
05.02 [25] Patterns (2) . . . 7
05.03 [10] Patterns (3) . . . 7
05.04 [10] Patterns (4) . . . 7
05.05 [10] Patterns (5) . . . 7
05.06 [10] Patterns (6) . . . 7
05.07 [10] Patterns (7) . . . 7
05.08 [25] Patterns (8) . . . 7
05.09 [10] Patterns (9) . . . 7
05.10 [10] Patterns (a) . . . 7
05.11 [10] Patterns (b) . . . 7
05.12 [10] Patterns (c) . . . 7
05.13 [10] Patterns (d) . . . 7
05.14 [10] Patterns (e) . . . 7
05.15 [25] Patterns (f) . . . 7
05.16 [50] Patterns (g) . . . 7
05.17 [25] Operators (1) . . . 7
05.18 [25] Operators (2) . . . 7

06 Advanced Templates . . . **8**

07 Project Organization . . . **9**
07.01 [25] Headers . . . 9
07.02 [25] Modules . . . 9

08 Errors and Exceptions . . . **10**
08.01 [25] Variants (1) . . . 10
08.02 [25] Variants (2) . . . 10
08.03 [25] Optionals . . . 10
08.04 [25] Exceptions (1) . . . 10
08.05 [25] Exceptions (2) . . . 10
08.06 [25] Exceptions (3) . . . 10

09 Numbers Processing . . . **11**
09.01 [25] Templates . . . 11
09.02 [25] Constants . . . 11
09.03 [20] Numbers (1) . . . 11
09.04 [75] Numbers (2) . . . 11
09.05 [25] Chrono (1) . . . 11
09.06 [25] Chrono (2) . . . 11

10 Memory Management . . . **12**
10.01 [10] Patterns (1) . . . 12
10.02 [25] Patterns (2) . . . 12
10.03 [30] Patterns (3) . . . 12
10.04 [25] Pointers (1) . . . 12
10.05 [25] Pointers (2) . . . 12
10.06 [25] Pointers (3) . . . 12
10.07 [25] Iterators (1) . . . 12
10.08 [25] Iterators (2) . . . 12
10.09 [25] Iterators (3) . . . 12
10.10 [10] Patterns (4) . . . 12
10.11 [10] Patterns (5) . . . 12
10.12 [10] Memory (1) . . . 12
10.13 [10] Memory (2) . . . 12
10.14 [25] Allocators (1) . . . 12
10.15 [25] Allocators (2) . . . 12
10.16 [10] Allocators (3) . . . 12

11 Collections of Objects . . . **13**
11.01 [25] Containers (1) . . . 13
11.02 [10] Containers (2) . . . 13
11.03 [10] Adaptors . . . 13
11.04 [90] Functions . . . 13
11.05 [25] Containers (3) . . . 13

12 Algorithms on Ranges . . . **14**
12.01 [30] Functors . . . 14
12.02 [25] Patterns (1) . . . 14
12.03 [10] Patterns (2) . . . 14
12.04 [10] Patterns (3) . . . 14
12.05 [10] Patterns (4) . . . 14
12.06 [20] Patterns (5) . . . 14
12.07 [30] Templates . . . 14
12.08 [50] Algorithms (1) . . . 14
12.09 [20] Algorithms (2) . . . 14
12.10 [20] Algorithms (3) . . . 14
12.11 [20] Algorithms (4) . . . 14
12.12 [25] Graphs (1) . . . 14
12.13 [25] Graphs (2) . . . 14

13 Text Data Processing . . . **15**
13.01 [20] Encodings . . . 15
13.02 [25] Strings (1) . . . 15
13.03 [25] Strings (2) . . . 15
13.04 [25] Strings (3) . . . 15
13.05 [25] Strings (4) . . . 15
13.06 [25] Strings (5) . . . 15
13.07 [25] Locales (1) . . . 15
13.08 [50] Locales (2) . . . 15
13.09 [25] Regexes (1) . . . 15
13.10 [25] Regexes (2) . . . 15
13.11 [25] Regexes (3) . . . 15
13.12 [25] Regexes (4) . . . 15

14 Streams and Formats . . . **16**
14.01 [25] Streams . . . 16

01 General Introduction

01.01 [12] Questions

- Сформулируйте и прокомментируйте определение языка программирования C++.
- Перечислите и прокомментируйте основные этапы процесса стандартизации C++.
- Перечислите и прокомментируйте основные области применения технологий C++.
- Перечислите и прокомментируйте основные парадигмы программирования в C++.
- Перечислите и прокомментируйте современные инструменты разработчиков C++.
- Перечислите и прокомментируйте основные стили оформления кода программ.
- Перечислите и прокомментируйте основные средства символьного ввода-вывода.
- Объясните предназначение и способы использования системы контроля версий Git.
- Перечислите и прокомментируйте основные команды системы контроля версий Git.
- Объясните необходимость использования отдельных веток в процессе разработки.
- Объясните отличия команд merge и rebase, используемых при обновлении веток.
- Объясните основную суть процессов continuous integration и continuous deployment.

01.02 [75] Functions

Напишите программу, которая выводит в поток вывода любую строку и при этом обладает функцией `main` с пустым телом. Предложите несколько решений. Впервые я столкнулся с этой задачей на собеседовании в одну крайне назойливую компанию. Для ее решения Вам потребуются определенные средства языка, которые будут рассматриваться позже. Сейчас Вы можете пропустить эту задачу и вернуться к ней спустя некоторое время. Возможно, Вы немного удивились тому, что первая же задача данного кодекса обладает настолько неадекватным уровнем сложности. Это своего рода дань памяти моему детству. Я начал серьезно изучать 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 ноутбук со средой разработки Code::Blocks, однако не было ни интернета, ни даже мобильной связи, потому что дача находится в низине, а сеть ловит только на определенном тайном холмике в лесу. Я решил начать с тонкой книги Кернигана, быстро проработать ее, а потом приступить к кирпичу Страуструпа. Опрометчивое решение! В одном из первых заданий просили написать программу, которая удаляла бы все комментарии из другой программы. Наверное, это очень сложно на третий день изучения программирования, но я справился, потому что из-за отсутствия связи с внешним миром я просто не знал, что это сложно. Возможно, именно этот случай помог мне определиться с основным направлением своей жизнедеятельности. Любопытно, что случилось бы со мной, если бы мне тогда попалалась монография Искусство программирования Дональда Кнута? Возможно, я стал бы лучше относиться к математике. Надо будет провести такой эксперимент над собственными детьми.

02 Programming Basics

02.01 [31] Questions

- Объясните понятия объявления, определения, инициализации и присваивания объектов.
- Перечислите и прокомментируйте ключевые особенности арифметических типов данных.
- Прокомментируйте проблемы переполнения, точности и переносимости типов данных.
- Приведите примеры опасного и неправильного использования беззнаковых типов данных.
- Приведите примеры ситуаций, которые могут приводить к неопределенному поведению.
- Перечислите и прокомментируйте особенности способов преобразования типов данных.
- Перечислите и прокомментируйте преимущества, предоставляемые выводом типа `auto`.
- Приведите примеры применения в программах констант, псевдонимов типов и атрибутов.
- Объясните понятия объекта, переменной, литерала, оператора, выражения и инструкции.
- Объясните принцип короткой схемы вычислений, применяемой в логических операторах.
- Объясните отличия префиксных и постфиксных операторов инкремента и декремента.
- Объясните суть принципа максимального куска, которого придерживается компилятор.
- Перечислите и прокомментируйте способы оформления ветвлений в потоке управления.
- Перечислите и прокомментируйте варианты применения атрибутов в инструкции `switch`.
- Перечислите и прокомментируйте способы оформления повторений в потоке управления.
- Объясните причины редкого применения оператора прыжка в современных программах.
- Приведите примеры нежелательных операторов и вариантов организации инструкций.
- Объясните понятия ячейки, адреса и указателя в низкоуровневом программировании.
- Прокомментируйте возможности, которые рождаются из связи указателей и массивов.
- Прокомментируйте особенности работы со статическими и динамическими массивами.
- Приведите примеры неправильного или опасного использования указателей и массивов.
- Сформулируйте 2 определения ссылок и обоснуйте их преимущества перед указателями.
- Приведите примеры создания массивов или контейнеров, хранящих ссылки на объекты.
- Объясните понятия объявления и определения, рассматривая использование функций.
- Приведите примеры нескольких рекомендаций по разработке качественных функций.
- Прокомментируйте особенности передачи аргументов по значению, указателю и ссылке.
- Прокомментируйте особенности передачи встроенных массивов и контейнеров в функции.
- Прокомментируйте особенности работы механизма аргументов функций по умолчанию.
- Объясните особенности использования локальных и статических объектов в функциях.
- Прокомментируйте особенности организации и работы механизма перегрузки функций.
- Перечислите и прокомментируйте моменты, когда возможно осуществление встраивания.

02.02 [25] Characters

Реализуйте алгоритм классификации введенных пользователем символов. Выделите четыре следующих класса: десятичная цифра – 10 символов, арифметический оператор – 4 символа, круглая скобка – 2 символа, остальные символы. Используйте ветвление типа `switch` с проваливанием для избежания дублирования кода.

02.03 [50] Numbers (1)

Реализуйте алгоритм для вычисления всех чисел Армстронга, всех чисел Фибоначчи, всех избыточных чисел и всех дружественных чисел, меньших числа N, заданного пользователем. Используйте оптимальные способы.

02.04 [25] Numbers (2)

Реализуйте алгоритм для вычисления всех простых множителей некоторого числа, заданного пользователем.

02.05 [25] Numbers (3)

Реализуйте рекурсивную и нерекурсивную версии алгоритма Евклида для вычисления наибольшего общего делителя. Также дополнительно реализуйте алгоритм вычисления наименьшего общего кратного двух чисел. Проверьте правильность работы Ваших реализаций, сравнив их результаты с результатами, полученными в результате использования численных алгоритмов `std::gcd` и `std::lcm` из стандартной библиотеки `numeric`.

02.06 [25] Numbers (4)

Реализуйте алгоритм для вычисления числа e через сумму ряда Маклорена при `x = 1`. Используйте числа с плавающей точкой типа `double`. Оптимизируйте вычисление членов ряда таким образом, чтобы использовалась только операция деления. Умножение при вычислении факториала может привести к переполнению. Завершайте вычисление, когда очередной член ряда окажется меньше некоторой заданной константы `epsilon`. Сравните Ваше получившееся значение с константой `std::numbers::e` из стандартной библиотеки `numbers`.

02.07 [25] Numbers (5)

Гипотеза Коллатца – одна из нерешенных проблем современной математики. Возьмем любое положительное целое число `N`. Если число четное, то определим следующий элемент последовательности как `N / 2`, в противном случае определим его как `3 * N + 1`. Утверждается, что вычисляемая таким образом последовательность всегда достигает значения 1. Ваша задача – сгенерировать последовательности Коллатца для всех целых чисел от 1 до 10000 включительно, выбрать наибольшую и вывести ее длину и ее начальное значение. Оптимизируйте вычисления, используя кэширование длин последовательностей для уже проверенных чисел.

02.08 [25] Recursion

Реализуйте рекурсивный алгоритм генерации списка инструкций для решения задачи о Ханойских башнях.

02.09 [25] Sortings (1)

Реализуйте алгоритм сортировки выбором контейнера `std::vector` целочисленных значений через индексы.

02.10 [25] Sortings (2)

Реализуйте алгоритм сортировки Шелла контейнера `std::vector` целочисленных значений через индексы.

02.11 [25] Sortings (3)

Реализуйте алгоритм сортировки кучей контейнера `std::vector` целочисленных значений через индексы.

02.12 [25] Sortings (4)

Реализуйте алгоритм быстрой сортировки контейнера `std::vector` целочисленных значений через индексы.

02.13 [25] Algorithms (1)

Реализуйте алгоритм удаления дубликатов из отсортированного по возрастанию контейнера `std::vector` целочисленных значений. В качестве основы используйте приведенную в справочнике реализацию алгоритма `std::unique`, однако не используйте итераторы и семантику перемещения с `std::move`, используйте индексы и копирование. Обратите внимание, данный алгоритм не удаляет дубликаты, он лишь перемещает в начало уникальные элементы и возвращает позицию конца. Постарайтесь реализовать максимально компактный код.

02.14 [25] Algorithms (2)

Реализуйте алгоритм обращения порядка элементов контейнера `std::vector` целочисленных значений. Используйте функцию `std::swap` для почленного обмена местами элементов с двух противоположных концов.

03 Classes Development

03.01 [44] Questions

- Объясните, в каких ситуациях допускается использовать структуры вместо классов.
- Перечислите и прокомментируйте способы инициализации данных-членов структур.
- Приведите примеры нескольких рекомендаций по разработке качественных классов.
- Объясните распределение внутренностей классов на публичную и приватную секции.
- Объясните, реализацию каких функций-членов классов следует выносить наружу.
- Прокомментируйте особенности применения константных функций-членов классов.
- Объясните понятия побитовой и логической константности экземпляров классов.
- Приведите пример корректных геттера и сеттера приватных данных-членов классов.
- Прокомментируйте предназначение и особенности работы конструкторов классов.
- Объясните, почему в конструкторах следует использовать списки инициализации.
- Объясните, почему практически никогда не следует вызывать деструкторы классов.
- Прокомментируйте, когда допускается использование статических членов классов.
- Объясните, из-за чего использование отношения дружбы ухудшает инкапсуляцию.
- Прокомментируйте отношения композиции, агрегации, ассоциации и зависимости.
- Перечислите основные концепции объектно-ориентированного программирования.
- Перечислите и прокомментируйте основные разновидности наследования классов.
- Прокомментируйте ход работы конструкторов и деструкторов в иерархиях классов.
- Объясните особенности переопределения и вызова унаследованных функций-членов.
- Объясните понятия статического и динамического типов полиморфного объекта.
- Объясните явление срезки полиморфных объектов при работе с ними по значению.
- Перечислите условия, необходимые для корректной работы виртуальных функций.
- Перечислите и прокомментируйте достоинства и недостатки виртуальных функций.
- Объясните необходимость реализации виртуальных деструкторов в базовых классах.
- Объясните суть работы таблиц виртуальных функций и виртуальных указателей.
- Объясните концепцию абстракции и предназначение абстрактных базовых классов.
- Прокомментируйте проблему бриллианта при множественном наследовании классов.
- Объясните, почему классы без данных-членов занимают непустую ячейку в памяти.
- Объясните, как компиляторы осуществляют оптимизацию пустого базового класса.
- Приведите примеры ситуаций, в которых допускаются понижающие преобразования.
- Прокомментируйте, как устроена реализация гетерогенного хранилища `std::any`.
- Приведите примеры ситуаций, в которых перемещение эффективнее копирования.
- Прокомментируйте классификацию выражений и их фундаментальные свойства.
- Объясните, какие возможности предоставляет разделение ссылок на lvalue и rvalue.
- Перечислите и прокомментируйте все существующие специальные функции-члены.
- Прокомментируйте особенности поверхностного и глубокого копирования данных.
- Объясните роль идиомы копирования с обменом в специальных функциях-членах.
- Прокомментируйте основное действие, выполняющееся внутри функции `std::move`.
- Перечислите и прокомментируйте правила генерации специальных функций-членов.
- Сформулируйте и прокомментируйте правила 0, 3 и 5 специальных функций-членов.
- Приведите примеры ситуаций, в которых компилятор может сократить копирование.
- Прокомментируйте преобразования, которые актуальны при перегрузке операторов.
- Приведите примеры операторов, которые можно, нельзя и не следует перегружать.
- Приведите примеры операторов, которые можно перегружать как функции-друзья.
- Прокомментируйте особенность вызова перегруженной версии оператора стрелочка.

03.02 [25] Structures

Реализуйте структуру `Rectangle`, хранящую целочисленные координаты левого нижнего угла прямоугольника и две длины его сторон. Будем предполагать, что все подобные прямоугольники имеют стороны, параллельные координатным осям. Реализуйте функцию, которая будет принимать на вход контейнер `std::vector` экземпляров структуры `Rectangle` и вычислять площадь их общего пересечения. В функции `main` продемонстрируйте использование реализованных Вами компонент. Прямоугольники и их пересечения могут быть вырожденными.

03.03 [25] Classes (1)

Реализуйте классы, описывающие основные геометрические фигуры, а именно треугольник, окружность и квадрат. Вы должны реализовать 3 независимых класса. В качестве приватных данных-членов рассматривайте только те, которые необходимы для вычисления периметра и площади фигур. Реализуйте необходимые конструкторы и публичные функции-члены, например, геттеры. Реализуйте публичные функции-члены для вычисления периметра и площади каждой фигуры. Для числа Пи в классе окружносте создайте статическую константу. Убедитесь, что Вы рассмотрели все необходимые аспекты проектирования классов, которые мы затрагивали на семинаре. Продемонстрируйте некоторые варианты использования классов в функции `main`.

03.04 [50] Classes (2)

Реализуйте иерархию классов, описывающих геометрические фигуры. Реализуйте классы, описывающие прямоугольник, квадрат, треугольник, эллипс и окружность. Реализуйте наследование квадрата от прямоугольника, окружности от эллипса, а для треугольника и прямоугольника реализуйте дополнительный промежуточный базовый класс многоугольника. Реализуйте во всех классах необходимые конструкторы. В качестве данных-членов добавляйте только те, которые необходимы для вычисления периметра и площади фигур. Например, для эллипса это две полуоси, для прямоугольника две стороны, для треугольника три стороны. Не нужно создавать отдельные данные-члены для радиуса окружности и стороны квадрата. Используйте наследование интерфейса и делегирующие конструкторы. Реализуйте две ветви виртуальных функций-членов – одну для вычисление площади, вторую для вычисление периметра фигур. Не создавайте собственную константу для числа Пи на этот раз. Воспользуйтесь существующей константой `std::numbers::pi` из стандартной библиотеки `numbers`. Реализуйте абстрактный базовый класс `Shape` фигур с интерфейсом чисто виртуальных функций-членов. В функции `main` создайте контейнер `std::vector` фигур, используя указатели на базовый класс. Добавьте в контейнер разные фигуры и продемонстрируйте работу всех виртуальных функций-членов.

03.05 [50] Classes (3)

Реализуйте класс контейнера, который будет являться высокоуровневой оберткой вокруг встроенного динамического массива. Используйте в качестве основы рассмотренную реализацию, в которой реализованы необходимые данные-члены и специальные функции-члены класса. Добавьте в класс дополнительное поле `capacity`, чтобы хранить количество выделенных ячеек памяти. Поле `size` при этом остается для хранения количества элементов массива, т.е. количества занятых ячеек памяти. Реализуйте константную и неконстантную версии функций-членов `front` и `back`, предоставляющих доступ к первому и последнему элементу массива соответственно. Реализуйте константную и неконстантную версии оператора доступа по индексу. Здесь можно не беспокоиться о дублировании кода, реализации будут тривиальными. Реализуйте геттеры для получения текущих значений полей `size` и `capacity`. Реализуйте функцию-член `empty`, проверяющую, является ли массив пустым. Реализуйте функцию-член `clear`, удаляющую все элементы массива, но не освобождающую выделенную память. Реализуйте функцию-член `push_back`, добавляющую новый элемент в первую свободную ячейку памяти внутреннего динамического массива. Тщательно продумайте последовательность действий на тот случай, когда свободных ячеек памяти нет, т.е. когда значения полей `size` и `capacity` равны. Выделяйте дополнительную память оптимальным образом, минимизируя количество операций с диспетчером памяти.

03.06 [25] Classes (4)

Реализуйте класс, представляющий собой сетевые IP адреса в формате IPv4. Реализуйте перегруженные операторы ввода-вывода для экземпляров данного класса. Пользователь должен иметь возможность вводить в консоли компоненты адреса через точку. В таком же формате сетевые адреса должны выводиться в консоль.

04 Templates Overview

04.01 [15] Questions

- Перечислите существующие виды шаблонов и приведите примеры их использования.
- Приведите примеры ошибок, выявляемых на каждом из этапов трансляции шаблонов.
- Приведите примеры использования полных или частичных специализаций шаблонов.
- Объясните, чему отдает предпочтение компилятор при перегрузке шаблонов функций.
- Приведите примеры применения типовых и нетиповых параметров шаблонов классов.
- Прокомментируйте особенности инстанцирования функций-членов шаблонов классов.
- Приведите примеры ситуаций, когда можно не указывать явно аргументы шаблонов.
- Прокомментируйте особенности оформления функций-друзей для шаблонов классов.
- Приведите примеры ситуаций, в которых можно использовать вариативные шаблоны.
- Объясните процесс рекурсивного инстанцирования вариативных шаблонов функций.
- Приведите примеры использования шаблонов псевдонимов и шаблонов переменных.
- Объясните, почему шаблоны являются Тьюринг-полным языком программирования.
- Приведите примеры оптимизаций и вычислений, выполняемых на этапе компиляции.
- Приведите примеры использования в коде программ спецификаторов `constexpr`.
- Приведите примеры использования гибридного метапрограммирования в шаблонах.

04.02 [25] Templates (1)

Доработайте пример `optimized_search.cpp` так, чтобы тип элементов контейнера мог быть произвольным.

04.03 [25] Templates (2)

Доработайте пример `combined_sorting.cpp` так, чтобы тип элементов контейнера мог быть произвольным.

04.04 [25] Templates (3)

Доработайте Ваше предыдущее решение задачи Classes (3) из модуля 3 так, чтобы контейнер стал шаблоном.

04.05 [25] Templates (4)

Реализуйте вариативный шаблон функции, которая принимает произвольное количество аргументов произвольных типов и возвращает сумму всех аргументов, которые обладают целочисленным типом `int`. Используйте рекурсивное инстанцирование и отдельный шаблон функции для извлечения целочисленных значений.

04.06 [25] Templates (5)

Реализуйте вариативный шаблон функции, которая определяет минимальное значение из пакета своих аргументов, используя для этого оператор меньше. Предполагается, что все аргументы имеют одинаковые типы.

04.07 [25] Templates (6)

Реализуйте вариативный шаблон функции, которая добавляет произвольное количество элементов в конец произвольного контейнера, обладающего функцией-членом `push_back`. Используйте здесь выражение свертки.

04.08 [25] Templates (7)

Реализуйте алгоритм вычисления N-ого числа ряда Фибоначчи, используя метапрограммирование шаблонов.

04.09 [25] Templates (8)

Реализуйте алгоритм вычисления биномиального коэффициента, используя метапрограммирование шаблонов.

04.10 [25] Constexprs (1)

Реализуйте алгоритм вычисления числа e через ряд Маклорена, используя `constexpr`-возможности языка.

04.11 [25] Constexprs (2)

Реализуйте алгоритм вычисления простого числа с номером N, используя `constexpr`-возможности языка.

04.12 [25] Operators

Доработайте пример `hybrid_durations.cpp` так, чтобы работали операции умножения, вычитания и деления.

05 Patterns and Idioms

05.01 [10] Patterns (1)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Builder.

05.02 [25] Patterns (2)

Реализуйте вариацию паттерна Builder, допускающего поэтапное контруирование объекта следующим образом: `Person p = builder.name("Ivan").age(26).height(180).get()` Здесь предполагается, что `builder` является экземпляром некоторого класса `Builder`, предназначенного для поэтапного конструирования экземпляров класса `Person`. Изначально в объекте `builder` создается «пустой» экземпляр класса `Person`, который последовательно дополняется необходимыми атрибутами в процессе вызова функций-членов класса `Builder`.

05.03 [10] Patterns (3)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Factory.

05.04 [10] Patterns (4)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Prototype.

05.05 [10] Patterns (5)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Composite.

05.06 [10] Patterns (6)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Decorator.

05.07 [10] Patterns (7)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Adapter.

05.08 [25] Patterns (8)

Реализуйте вариацию паттерна Adapter, используя закрытое наследование вместо отношения композиции.

05.09 [10] Patterns (9)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Bridge.

05.10 [10] Patterns (a)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна State.

05.11 [10] Patterns (b)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Template.

05.12 [10] Patterns (c)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Strategy.

05.13 [10] Patterns (d)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Observer.

05.14 [10] Patterns (e)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Memento.

05.15 [25] Patterns (f)

Реализуйте вариацию паттерна Bridge, используя статический полиморфизм вместо динамического.

05.16 [50] Patterns (g)

Реализуйте алгоритм, который будет рассчитывать время, затрачиваемое для получения желаемых оценок на различных учебных курсах. В данном алгоритме будут присутствовать общие компоненты, например, учет халявности преподавателя, однако для разных курсов будут и отличающиеся условия, например, на одних курсах необходимо сдавать еженедельные домашние задания, а на других можно сдать тетрадь соседа. Для начала реализуйте несколько классов, каждый из которых будет соответствовать определенному курсу и будет по-своему определять вычисление некоторой компоненты алгоритма. Далее создайте общий для них базовый класс, который будет задавать структуру всего алгоритма и вычислять общие компоненты. В данной ситуации Вы должны реализовать паттерн шаблонный метод и идиому NVI. Далее переработайте реализованную Вами иерархию при помощи паттерна CRTP. В результате должен получиться так называемый перевернутый миксин. Сдавать следует оба решения: шаблонный метод на виртуальных функциях и миксин на основе паттерна CRTP.

05.17 [25] Operators (1)

Доработайте пример `operator_overloading.cpp` так, чтобы все необходимые операторы подмешивались за счет использования базовых классов на основе миксинов из Boost.Operators. Из собственных реализаций должен остаться минимальный набор операций, например, из шести операций сравнения должна быть реализована только одна – оператор меньше. Остальные пять операций сравнения подмешиваются с помощью миксинов.

05.18 [25] Operators (2)

Доработайте пример `operator_overloading.cpp` так, чтобы для сравнения использовался оператор spaceship.

06 Advanced Templates

07 Project Organization

07.01 [25] Headers

Доработайте пример `class_basics.cpp` так, чтобы компоненты класса были корректно разделены между заголовочным файлом и файлом исходного кода. Не забудьте продемонстрировать использование Вашего класса в функции `main`. В сумме в Вашем проекте должны получиться два файла исходного кода и один заголовочный файл. Дополнительно поместите все компоненты Вашего класса в отдельное пространство имен.

07.02 [25] Modules

Доработайте Ваше предыдущее решение задачи Classes, используя модули вместо заголовочных файлов.

08 Errors and Exceptions

08.01 [25] Variants (1)

Реализуйте алгоритм решения квадратного уравнения, принимающий в качестве аргументов коэффициенты уравнения a, b и c. Гарантируется, что коэффициент a не равен нулю. Квадратное уравнение может иметь 2, 1 или 0 решений. Для хранения решений используйте класс-хранилище `std::variant`, в котором для представления 0 решений будет использоваться тип `std::monostate`. Для представления двух решений рекомендуется использовать тип `std::pair`, а для представления одного решения достаточно одиночного `double`. Точность сравнения чисел с плавающей точкой в данной задаче можно не затрагивать, в случае с дискриминантом допускается проверка условия `d == 0.0`. Также не используйте функцию `std::visit` из публичных решений.

08.02 [25] Variants (2)

Реализуйте алгоритм нахождения точки пересечения двух прямых на плоскости. Каждая прямая задается каноническим образом через три коэффициента a, b и c. Как известно, две прямые могут пересекаться, совпадать или быть параллельны, таким образом можно получить 0, 1 или бесконечно много точек пересечения. Для хранения результата используйте класс-хранилище `std::variant`. Подумайте, какие альтернативные варианты стоит организовать в данном хранилище. Для лучшей организации Вашего кода рекомендую создать вспомогательные классы `Point` и `Line`, но также не используйте функцию `std::visit` из публичных решений.

08.03 [25] Optionals

Реализуйте класс человека, обладающий опциональными данными на основе класса-хранилища `std::optional`.

08.04 [25] Exceptions (1)

Продемонстрируйте на простых примерах обработку исключений `std::bad_alloc`, `std::bad_variant_access`, `std::bad_optional_access`, а также исключений `std::length_error` и `std::out_of_range`, генерируемых функциями-членами класса-контейнера `std::vector`. Объясните, кто и почему выбрасывает эти исключения.

08.05 [25] Exceptions (2)

Доработайте пример `operator_overloading.cpp` так, чтобы ошибки обрабатывались с помощью исключений.

08.06 [25] Exceptions (3)

Рассмотрим представленную ниже функцию. Вам необходимо выделить в ней все точки, в которых возможно ветвление пути выполнения. Например, в данной функции представлен оператор `if`, который может выполняться по одному или другому пути. Эти пути мы будем называть нормальными путями выполнения. На первый взгляд кажется, что их два, но на самом деле больше. Далее рассмотрим, например, оператор вывода. В соответствии со стандартом C++ каждый оператор вывода может выбрасывать исключения, таким образом каждый оператор вывода в приведенном коде порождает точку ветвления пути выполнения – один путь будет являться нормальным, а другой ненормальным с исключением. Всего в данном коде представлено 6 операторов вывода, которые в совокупности обеспечивают 6 точек ветвления. Составьте полный список точек ветвления пути выполнения для данного кода. Примечание: `String` и `Employee` – это какие-то пользовательские классы.

```
String evaluate_salary_and_return_name(Employee e)
{
  if (e.title() == "CEO" || e.salary() > 100000)
  {
    std::cout << e.name() << " " << e.surname() << " is overpaid.\n";
  }
  else
  {
    std::cout << e.name() << " is not overpaid.\n";
  }
  return e.name() + " " + e.surname();
}
```

09 Numbers Processing

09.01 [25] Templates

Реализуйте вариативный шаблон функции, вычисляющей суммарный объем памяти в байтах, занимаемый пакетом ее параметров. Внутри функции используйте выражение свертки, обычный оператор `sizeof` и оператор сложения в выражении свертки. Продемонстрируйте альтернативу без использования выражения свертки.

09.02 [25] Constants

Доработайте пример `class_basics.cpp` так, чтобы можно было задавать месяц через именованную константу перечисления с областью видимости. Это позволит избежать потенциальной путаницы при указании дня и месяца. Реализуйте перечисление с областью видимости для месяцев и продемонстрируйте его использование.

09.03 [20] Numbers (1)

Доработайте Ваши предыдущие решения задач Variants (1) и Variants (2) из модуля 8 так, чтобы в коде учитывались все необходимые аспекты правильной работы с числами с плавающей точкой, в частности, точность, значение бесконечности и знаковый ноль. Позаботьтесь о том, чтобы в консоль не выводился минус для нуля.

09.04 [75] Numbers (2)

Рассмотрим окружность и случайный вписанный в нее треугольник. С какой вероятностью центр окружности окажется внутри данного треугольника? Вычислите приблизительную вероятность, используя метод Монте-Карло. Для этого сгенерируйте большое количество случайных вписанных в окружность треугольников и определите, сколько треугольников содержат центр окружности. Рассмотрим окружность O с радиусом `r = 1` и центром в точке P с координатами `(1; 1)`. Благодаря симметрии можно зафиксировать одну из вершин каждого случайно сгенерированного вписанного в окружность O треугольника ABC. Пусть точка A имеет фиксированные координаты `(1; 0)`. Для вершин треугольника B и C достаточно случайно сгенерировать величину угла w на отрезке `[0; 2Pi]`, а затем вычислить координаты по следующим формулам: `x = 1 + cos(w)` и `y = 1 + sin(w)`. Для проверки принадлежности точки P треугольнику ABC следует использовать метод барицентрических координат. Площадь S со знаком треугольника ABC можно вычислить как `[AB * AC] / 2`, где `[AB * AC]` означает величину векторного произведение векторов AB и AC. Барицентрические координаты точки P вычисляются следующим образом: `a = [PA * PB] / 2S`, `b = [PB * PC] / 2S` и `c = [PC * PA] / 2S`. Точка P находится внутри треугольника ABC, если все три барицентрические координаты a, b и c неотрицательны. Постарайтесь максимально оптимизировать код. Допускается использование типа данных `float`. Для генерации случайных чисел используйте вихрь Мерсенна `std::mt19937`. Программу следует запускать в режиме Release. После вычисления приблизительной вероятности с помощью метода Монте-Карло предоставьте точный математически обоснованный ответ. В трехмерном случае рассмотрим сферу и случайный вписанный в нее тетраэдр. С какой вероятностью центр сферы окажется внутри данного тетраэдра? Вычислите точную вероятность, используя математические рассуждения. В N-мерном случае рассмотрим гиперсферу и случайный вписанный в нее симплекс. С какой вероятность центр гиперсферы окажется внутри данного симплекса? Достаточно предъявить без обоснования общую формулу вероятности, параметризованную только числом N.

09.05 [25] Chrono (1)

Доработайте пример `chrono_management.cpp` так, чтобы с помощью хронометра можно было бы проводить серии измерений. Предполагается, что хронометр можно будет останавливать и перезапускать, а внутри себя он будет содержать контейнер `std::vector`, в котором будут храниться несколько временных интервалов, соответствующих отдельным измерениям. Добавьте функции-члены `start` и `stop`, которые будут вызываться в начале и конце каждого измерения в серии. Функция `start` сохраняет время запуска, функция `stop` вызывает функцию `elapsed` и записывает замеренный интервал во внутренний контейнер. Также добавьте булевский флаг в хронометр, показывающий, идет ли сейчас замер или нет. В функциях `start` и `stop` следует проверять и обновлять данный флаг. Если хронометром пользуются неправильно необходимо генерировать исключение. В качестве итогового результата хронометр должен выдавать для серии измерений среднее значение времени.

09.06 [25] Chrono (2)

Реализуйте алгоритм, который сможет вычислить временной интервал в днях между двумя заданными датами.

10 Memory Management

10.01 [10] Patterns (1)

Приведите пример программной задачи, в решении которой будет уместно использование идиомы RAII.

10.02 [25] Patterns (2)

Реализуйте класс логгера-трассировщика, используя идиому RAII и информацию из `std::source_location`. Предполагается, что пользователь данного класса в начале каждой функции будет создавать экземпляр логгера, конструктор которого выведет в консоль сообщение о начале выполнения функции с указанием всей необходимой информации. Деструктор совершит аналогичное действие после завершения работы данной функции. Оформите инструкции вывода трассировочных сообщений в отдельной функции, доступной для пользователя.

10.03 [30] Patterns (3)

Доработайте рассмотренные в модуле 5 паттерны Factory, Prototype, Composite, Decorator, Strategy и Observer, используя интеллектуальные указатели вместо обычных. Подумайте, какой тип интеллектуальных указателей, а именно `std::shared_ptr` или `std::unique_ptr`, следует использовать. `std::weak_ptr` Вам не потребуется.

10.04 [25] Pointers (1)

Доработайте пример `custom_unique_ptr.cpp` так, чтобы дополнительные шаблонные версии перемещающего конструктора и оператора перемещающего присваивания позволяли использовать в качестве аргумента `other` объекты типа `Unique < U >` , то есть указатели на другой тип `U`, отличный от типа `T`. Предполагается, что тип `U` является производным от типа `T`, поэтому допускается сохранять указатель на производный тип в указателе на базовый тип. Для более надежной работы шаблонов добавьте им ограничения через `requires`. В ограничениях должно быть указано, что тип `U` обязан являться производным от типа `T`. Используйте шаблон свойств `std::is_convertible` через шаблон переменной. Также дополните интерфейс класса `Unique` оператором стрелочка, функцией-членом `get` и оператором неявного приведения класса `Unique` к типу `bool`.

10.05 [25] Pointers (2)

Доработайте пример `custom_shared_ptr.cpp` так, чтобы дополнительные шаблонные версии перемещающих и копирующих конструкторов и операторов присваивания позволяли использовать в качестве аргумента `other` объекты типа `Shared < U >` , то есть указатели на другой тип `U`, отличный от типа `T`. Предполагается, что тип `U` является производным от типа `T`, поэтому допускается сохранять указатель на производный тип в указателе на базовый тип. Для более надежной работы шаблонов добавьте им ограничения через `requires`. В ограничениях должно быть указано, что тип `U` обязан являться производным от типа `T`. Используйте шаблон свойств `std::is_convertible` через шаблон переменной. Также дополните интерфейс класса `Shared` оператором стрелочка, функцией-членом `get` и оператором неявного приведения класса `Shared` к типу `bool`.

10.06 [25] Pointers (3)

Реализуйте класс бинарного дерева, используя интеллектуальные указатели для связи родителей и потомков. Реализуйте структуру `Node`, содержащую указатели типа `std::shared_ptr` на правого и левого потомков, указатель типа `std::weak_ptr` на родителя и данные типа `T`. Вырастите в функции `main` небольшое дерево и продемонстрируйте отсутствие неразрушимых циклических связей по аналогии с примером `smart_pointers.cpp`.

10.07 [25] Iterators (1)

Доработайте пример `optimized_search.cpp` так, чтобы вместо обычных указателей использовались итераторы. Для работы с итераторами вместо арифметических операторов следует использовать обобщенные функции, например, `std::next` и `std::prev`. Итераторы могут быть любой категории, не обязательно произвольного доступа. Также итераторы могут задавать начало и конец коллекции, хранящейся в произвольном контейнере, поэтому функция должна стать шаблоном. В качестве подсказки можете использовать приведенную в справочнике реализацию алгоритма `std::lower_bound`. Вместо `nullptr` возвращайте итератор конца.

10.08 [25] Iterators (2)

Доработайте пример `combined_sorting.cpp` так, чтобы вместо обычных указателей использовались итераторы. Для работы с итераторами вместо арифметических операторов следует использовать обобщенные функции, например, `std::distance` и `std::iter_swap`. Итераторы могут быть любой категории, не обязательно произвольного доступа. Также итераторы могут задавать начало и конец коллекции, хранящейся в произвольном контейнере, поэтому функции должны стать шаблонами. В качестве подсказки можете использовать приведенную в справочнике реализацию алгоритма `std::sort`. Пользовательский компаратор здесь не нужен.

10.09 [25] Iterators (3)

Доработайте пример `pattern_iterator.cpp` так, чтобы односвязный список превратился в двусвязный, а итераторы стали двунаправленными. Для этого в структуру `Node` необходимо добавить указатель на предыдущий узел, а для итератора оформить перегруженные версии префиксного и постфиксного операторов декремента.

10.10 [10] Patterns (4)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Iterator.

10.11 [10] Patterns (5)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Facade.

10.12 [10] Memory (1)

Сформулируйте принципы функционирования и использования таблиц страниц для отображения адресов.

10.13 [10] Memory (2)

Определите среднее время выделения диспетчером памяти блоков памяти размерами в 1 Кб, 1 Мб и 1 Гб.

10.14 [25] Allocators (1)

Доработайте пример `block_allocator.cpp` так, чтобы данный аллокатор можно было бы использовать в контейнере `std::vector` вместо стандартного `std::allocator` по аналогии с примером `arena_adaptor.cpp`.

10.15 [25] Allocators (2)

Доработайте пример `block_allocator.cpp` так, чтобы уменьшилась фрагментация памяти. Для этого алгоритм `find_first` следует заменить на алгоритм `find_best`, определяющий среди блоков списка свободный фрагмент равный или наиболее близкий по размеру требуемому. Сравните производительность обеих версий.

10.16 [10] Allocators (3)

Исследуйте публичные реализации Buddy аллокатора и опишите его предназначение и особенности работы.

11 Collections of Objects

11.01 [25] Containers (1)

Проанализируйте систему выделения памяти в векторе. Для этого определите с помощью вызовов функции-члена `capacity`, во сколько раз изменяется емкость вектора при нехватке памяти для размещения новых элементов. Также определите, как увеличивается емкость вектора, если задать ее начальное значение вручную с помощью вызова функции-члена `reserve`. Дополнительно определите, как осуществляется выделение памяти в предельном случае, когда вектор уже запросил большой объем памяти и на выполнение следуещего запроса у ОС может не хватить ресурсов. В комментариях в коде подробно опишите полученные наблюдения.

11.02 [10] Containers (2)

Сформулируйте идею однопроходного алгоритма поиска среднего узла односвязного или двусвязного списков.

11.03 [10] Adaptors

Доработайте пример `stack_with_minimum.cpp` так, чтобы контейнеры хранили и обрабатывали максимумы.

11.04 [90] Functions

Исследуйте 9 хэш-функций из этой статьи на равномерность и количество возникающих коллизий. Данные хэш-функции предназначены для строк, поэтому Вам потребуется написать генератор множества случайных строк без дубликатов. Рекомендую ограничиться английским алфавитом в нижнем регистре и строками из 10 символов. Множество строк нужно подготовить один раз. Для всех хэш-функций оно должно быть одним и тем же. Для исключения наличия дубликатов можно добавлять новые строки в контейнеры типа `std::set` или `std::unordered_set` в процессе генерации. Определите количество коллизий дял каждой функции на различных количествах хэшируемых строк, например, от 100000 до 2000000 с шагом 100000 и постройте графики полученных зависимостей. В данной задаче используйте режим Release x86 из-за особенностей хэш-функций.

11.05 [25] Containers (3)

Реализуйте многофункциональный контейнер для хранения записей телефонного справочника, которым разные клиенты смогут пользоваться по-разному. Например, городская типография собирается напечатать справочник, и ей нужны записи в отсортированном по фамилии человека порядке. Рекламное агентство нуждается в произвольном доступе к записям справочника. Регулярный пользователь хочет за максимально короткое время находить нужную ему запись. Удовлетворите желания клиентов, реализовав контейнер на базе Boost.MultiIndex.

12 Algorithms on Ranges

12.01 [30] Functors

Доработайте пример `combined_sorting.cpp` так, чтобы пользователь мог задавать критерий сортировки через функцию, передаваемую в алгоритм сортировки как дополнительный аргумент, тип которого является указателем на функцию. Продемонстрируйте использование собственных критериев сортировки по возрастанию и по убыванию. Доработайте тот же пример так, чтобы пользователь мог задавать критерий сортировки через функциональный объект с перегруженным оператором вызова, передаваемый в алгоритм сортировки как дополнительный аргумент, тип которого задается дополнительным параметром шаблона. Продемонстрируйте использование собственных критериев сортировки по возрастанию и по убыванию, а также функциональных объектов из стандартной библиотеки `std::less` и `std::greater`. Доработайте тот же пример так, чтобы пользователь мог задавать критерий сортировки через лямбда-выражение, передаваемое в алгоритм сортировки как дополнительный аргумент, тип которого задается дополнительным параметром шаблона. Продемонстрируйте использование собственных критериев сортировки по возрастанию и по убыванию. Сдавайте три файла.

12.02 [25] Patterns (1)

Реализуйте паттерн Factory в виде класса, содержащего ассоциативный контейнер, связывающий тип или код создаваемого объекта и пакет инструкций для создания этого объекта. Пакет инструкций следует оформить в виде лямбда-выражения. Необходимо реализовать способ хранения лямбда выражений по аналогии с примером `lambda_expressions.cpp`. Разместите ассоциативный контейнер в приватной секции, инициализируйте его и реализуйте необходимый публичный интерфейс. Продемонстрируйте варианты использования Вашей фабрики.

12.03 [10] Patterns (2)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Command.

12.04 [10] Patterns (3)

Приведите пример программной задачи, в решении которой будет уместно использование паттерна Visitor.

12.05 [10] Patterns (4)

Изучите публичные реализации компьютерной игры Asteroids. Во время игры программа должна обрабатывать столкновения объектов, например, астероидов различных типов, космических кораблей и пуль. Объясните, из-за чего может возникнуть дублирование кода и как оно устраняется благодаря использованию паттерна Visitor.

12.06 [20] Patterns (5)

Доработайте Ваши предыдущие решения задач Variants (1) и Variants (2) из модуля 8 так, чтобы упакованный в `std::variant` результат работы алгоритма выводился в консоль одним из посетителей. Посетителей можно реализовать как функциональные объекты с перегруженными операторами вызова или как лямбда-выражения. Обязательно используйте в решении функцию `std::visit` по аналогии с примером `advanced_visitors.cpp`.

12.07 [30] Templates

Реализуйте вариативные шаблоны функций `any_of`, `all_of` и `none_of`, проверяющие наличие в коллекции объектов хотя бы одного из заданных объектов, всех заданных объектов и ни одного из заданных объектов соответственно. Количество заданных объектов может быть произвольным. Используйте выражения свертки. Для сокращения дублирования кода проверок рекомендуется реализовать дополнительный невариативный шаблон функции `has_one`, которая будет проверять наличие в коллекции объектов одного заданного значения.

12.08 [50] Algorithms (1)

Продемонстрируйте использование алгоритмов `std::count`, `std::mismatch`, `std::equal`, `std::search`, `std::replace`, `std::remove`, `std::unique`, `std::rotate`, `std::partition` и `std::nth_element` из стандартной библиотеки на простейших тестовых примерах. Не используйте здесь диапазоны и средства просмотра.

12.09 [20] Algorithms (2)

Продемонстрируйте использование алгоритмов `std::accumulate`, `std::inner_product`, `std::partial_sum` и `std::adjacent_difference` из стандартной библиотеки работы с числами на простейших тестовых примерах.

12.10 [20] Algorithms (3)

Доработайте Ваше предыдущее решение задачи Algorithms (1), используя диапазоны и средства просмотра.

12.11 [20] Algorithms (4)

Продемонстрируйте использование инструментов просмотра `views::istream`, `views::counted`, `views::join` и `views::stride` из стандартной библиотеки диапазонов и наблюдателей на простейших тестовых примерах.

12.12 [25] Graphs (1)

Продемонстрируйте использование алгоритма поиска путей Беллмана-Форда из библиотеки Boost.Graph.

12.13 [25] Graphs (2)

Продемонстрируйте использование алгоритма поиска путей Флойда-Уоршелла из библиотеки Boost.Graph.

13 Text Data Processing

13.01 [20] Encodings

Сформулируйте полные алгоритмы формирования кодировок символов UTF-16 в версиях big и little endian.

13.02 [25] Strings (1)

Реализуйте алгоритм преобразования строки шестнадцатиричных цифр в коллекцию 8-битных целых чисел. Например, строка `"BAAD"` должна быть преобразована в коллекцию `0xBA, 0xAD`. Для хранения коллекции можно использовать контейнер `std::vector`. Предполагается, что в исходной строке четное количество цифр.

13.03 [25] Strings (2)

Реализуйте алгоритм преобразования начальных символов слов некоторого текста в верхний регистр, а остальных символов в нижний регистр. Например, текст `"heLLo, woRLD"` должен превратиться в `"Hello, World"`.

13.04 [25] Strings (3)

Реализуйте алгоритм конкатенации коллекции строк в одну строку через заданный пользователем разделитель.

13.05 [25] Strings (4)

Реализуйте алгоритм разбиения строки на лексемы по заданному списку возможных символов-разделителей.

13.06 [25] Strings (5)

Реализуйте алгоритм поиска внутри строки подстроки-палиндрома наибольшей длины. Если таковых подстрок несколько, достаточно найти первую из них. Для уменьшения сложности алгоритма используйте кэширование.

13.07 [25] Locales (1)

Реализуйте алгоритм конвертации валют, обеспечивающий ввод и вывод денежных сумм с учетом национальных стандартов. Например, Вы можете реализовать перевод EUR в RUB. В таком случае Вам потребуется использовать недружественную европейскую, например, немецкую локаль для настройки ввода суммы в EUR и русскую локаль для настройки вывода суммы в RUB. Для наиболее удобной реализации предлагаю воспользоваться манипуляторами из стандартной библиотеки `std::get_money` и `std::put_money`. Основная сложность в данной задаче заключается в выявлении корректных названий доступных в рамках операционной системы локалей. На операционных системах семейства Windows локали с высокой долей вероятности будут иметь названия вида `ru_RU.cp1251` и `de_DE.cp1251`. Для установки кодировки cp1251 в консоли на Windows следует в начале функции `main` добавить инструкцию `system("chcp 1251")`. На Linux по умолчанию используется кодировка UTF-8, поэтому дополнительные действия не потребуются. Список установленных в системе локалей можно просмотреть, набрав в терминале команду `locale -a`. Их названия будут записаны в виде `ru_RU.utf8`.

13.08 [50] Locales (2)

Реализуйте алгоритм побуквенной транслитерации кириллических символов русского алфавита. Предположим, что пользователь вводит символы в консоли в кодировке системной локали по умолчанию, например, cp1251. Вы должны конвертировать кодировку символов из cp1251 в UTF-8, а затем в UTF-32 наиболее удобным способом. Далее замените все кириллические символы латинскими в соответствии с правилами транслитерации. Для программной реализации используйте заранее подготовленное отображение в виде хэш-таблицы. Конвертируйте кодировку обратно в UTF-8, а затем в cp1251 и выведите результат преобразования в консоль. Преобразование кодировки cp1251 актуально только для Windows, на Linux данные действия не потребуются.

13.09 [25] Regexes (1)

Реализуйте регулярное выражение, описывающее автомобильные номера в России. Для избежания возможных проблем с кириллицей допускается использовать латиницу. Продемонстрируйте реализованное регулярное выражение в составе функции, осуществляющей поиск всех корректных автомобильных номеров в заданном тексте. Текст подготовьте самостоятельно и внедрите его в исходный код с помощью сырых строк. В тексте должны присутсвовать как корректные, так и некорректные автомобильные номера, а также мусорный текст.

13.10 [25] Regexes (2)

Реализуйте регулярное выражение, описывающее адреса электронных почт. Допускается использовать публичное регулярное выражение из интернета, но, желательно, в упрощенном виде, чтобы не получилось спагетти-кода и чтобы Вы понимали все составляющие его компоненты. Продемонстрируйте реализованное регулярное выражение в составе функции, осуществляющей поиск всех корректных адресов электроных почт в заданном тексте. Текст подготовьте самостоятельно и внедрите его в исходный код с помощью сырых строк. В тексте должны присутсвовать как корректные, так и некорректные адреса электроных почт, а также мусорный текст.

13.11 [25] Regexes (3)

Реализуйте функцию, которая принимает на вход строку с URL и извлекает из него отдельные компоненты, такие как протокол, домен, порт и путь. Используйте группы в круглых скобках внутри регулярного выражения.

13.12 [25] Regexes (4)

Реализуйте функцию, которая принимает на вход текст с датами в одном формате, например, с использованием слэша в качестве разделителя, а возвращает копию этого текста с датами, оформленными в другом формате.

14 Streams and Formats

14.01 [25] Streams

Реализуйте алгоритм преобразования коллекции 8-битных целых чисел в строку шестнадцатиричных цифр. Например, коллекция `0xBA, 0xAD` должна быть преобразована в строку `"BAAD"`. Для хранения коллекции можно использовать контейнер `std::vector`. Используйте строковый поток вывода и манипуляторы потоков.