Задания к работе 1 по алгоритмам и структурам данных.

Все задания реализуются на языке программирования C++ (стандарт C++14 и выше). Реализованные в заданиях приложения не должны завершаться аварийно.

Во всех заданиях запрещено использование глобальных переменных (включая errno).

Во всех заданиях запрещено использование оператора безусловного перехода (goto).

Во всех заданиях запрещено пользоваться функциями, позволяющими завершить выполнение приложения из произвольной точки выполнения, вне контекста исполнения функции main.

Во всех заданиях при реализации необходимо разделять контексты работы с данными (поиск, сортировка, добавление/удаление, модификация и т. п.) и отправка данных в поток вывода / выгрузка данных из потока ввода.

Во всех заданиях все параметры функций и вводимые (с консоли, файла, командной строки) пользователем данные должны подвергаться валидации в соответствии с типом валидируемых данных, если не сказано обратное; валидация должна зависеть от типа данных и логики применения этих данных для выполнения целевой подзадачи. При передаче аргументов приложению в командную строку, их количество также должно валидироваться.

Во всех заданиях необходимо контролировать ситуации с невозможностью [пере]выделения памяти; во всех заданиях необходимо корректно освобождать всю выделенную динамическую память.

Все ошибки, связанные с операциями открытия файла, должны быть обработаны; все открытые файлы должны быть закрыты.

Во всех заданиях запрещено использование глобальных переменных. Во всех заданиях при реализации функций необходимо обеспечить возможность обработки ошибок различных типов на уровне вызывающего кода.

Во всех заданиях сравнение (на предмет эквивалентности или отношения порядка) вещественных чисел на уровне функции должно использовать значение эпсилон, которое является параметром этой функции.

Во всех заданиях при реализации функций необходимо максимально ограничивать возможность модификации (если она не подразумевается) передаваемых в функцию параметров (используйте ключевое слово const), а также вызывающего объекта, в случае вызова его методов.

Для реализованных компонентов должны быть переопределены (либо перекрыты / оставлены реализации по умолчанию - при обосновании) следующие механизмы классов C++: конструктор копирования, деструктор, оператор присваивания.

Во всех заданиях необходимо уменьшать количество копирований нетривиально копируемых объектов.

Во всех заданиях необходимо проектировать компоненты с учетом SOLID принципов. Компонент не должен управлять ресурсом, если это не является его единственной задачей.

Запрещается пользоваться элементами стандартной библиотеки языка C, если существует их аналог в стандартной библиотеке языка C++.

Запрещается использование STL.

1. Реализовать класс *encoder*. В классе определить и реализовать:

- конструктор, принимающий ключ шифрования в виде массив байтов типа *unsigned char const* * и размер этого массива
- mutator для значения ключа
- метод encode, который принимает путь ко входному файлу (типа *char const* *), выходному файлу (типа *char const* *) и флаг, отвечающий за то, выполнять шифрование или дешифрование (типа *bool*) и выполняет процесс шифрования/дешифрования файла

Шифрование/дешифрование файлов выполняется алгоритмом RC4. Структура содержимого файлов произвольна. Продемонстрировать работу класса, произведя шифрование/дешифрование различных файлов: текстовых, графических, аудио, видео, исполняемых.

2. Реализовать класс *logical_values_array*. В классе определить и реализовать:

- поле *_value* (типа *unsigned int*), которое хранит значение массива логических величин
- accessor для поля *_value*
- конструктор, принимающий значение типа *unsigned int* (равное по умолчанию 0) и инициализирующий переданным значением поле *_value*
- методы, соответствующие всем стандартным логическим операциям, и (для бинарных операций) выполняющиеся между каждой парой битов полей *_value* объектов, над которыми выполняется преобразование: инверсия, конъюнкция, дизъюнкция, импликация, коимпликация, сложение по модулю 2, эквивалентность, стрелка Пирса, штрих Шеффера; в результате выполнения метода результат сохраняется в новый возвращаемый объект. При реализации необходимо использовать набор следующих базисных операций: конъюнкция, дизъюнкция, инверсия, сложение по модулю 2.
- статический метод *equals*, сравнивающий два объекта по отношению эквивалентности
- метод *get_bit*, который возвращает значение бита по его позиции (позиция является параметром типа *size_t*)
- перегруженный оператор *[]*, делегирующий выполнение на метод *get_bit*
- метод, принимающий значение типа *char* *; по значению адреса в параметре должно быть записано двоичное представление поля *_value* в виде строки в стиле языка программирования *C*. Примечание: конвертация должна быть основана на использовании битовых операций.

Продемонстрируйте работу реализованного функционала.

3. Реализовать класс комплексного числа. В классе определить и реализовать:

- поля, соответствующие действительной и мнимой части комплексного числа (типа *double*)
- конструктор, который принимает значения действительной и мнимой части (оба параметра по умолчанию равны 0)
- операторные методы, производящие операции сложения, вычитания, умножения и деления комплексных чисел (с модификацией и без модификации вызывающего объекта: +=/+, ...)
- метод, возвращающий модуль комплексного числа
- метод, возвращающий аргумент комплексного числа
- перегруженный оператор вставки в поток
- перегруженный оператор выгрузки из потока

Продемонстрируйте работу реализованного функционала.

4. Реализовать класс матрицы с вещественными значениями. В классе определить и реализовать:

- поля для хранения элементов матрицы в динамической памяти (типа double **) и размерностей матрицы (типа size_t)
- конструктор, инициализирующий состояние матрицы на основе размерностей, с заполнением всех элементов значением *0*
- операторные методы, осуществляющие сложение матриц, умножение матриц, умножение матрицы на число, умножение числа на матрицу, вычитание матриц
- метод, возвращающий транспонированную матрицу
- метод, возвращающий значение определителя матрицы, вычисленный при помощи метода Гаусса
- метод, возвращающий обратную матрицу
- * перегруженный оператор [], возвращающий значение элемента по его индексам строки и столбца (индексирование начинается с 0), с возможностью модификации возвращённого элемента в вызывающем коде

В случае невозможности её вычисления, должна быть сгенерирована исключительная ситуация типа, nested по отношению к типу матрицы.
Продемонстрируйте работу реализованного функционала.

5. Опишите интерфейс структуры данных вида приоритетная очередь. Интерфейс должен предоставлять следующие методы:

- добавление значения типа *char* * (строка в стиле C) по ключу типа *int* в приоритетную очередь, с копированием строки в контекст структуры
- поиск значения по наиболее приоритетному ключу
- удаление значения по наиболее приоритетному ключу
- слияние двух приоритетных очередей в вызывающий объект приоритетной очереди, с поддержкой fluent API

6. На основе интерфейса из задания 5 реализуйте класс двоичной приоритетной очереди. Продемонстрируйте работу реализованного функционала.

7. На основе интерфейса из задания 5 реализуйте класс биномиальной приоритетной очереди. Продемонстрируйте работу реализованного функционала.

8. На основе интерфейса из задания 5 реализуйте класс фибоначчиевой приоритетной очереди. Продемонстрируйте работу реализованного функционала.

9. На основе интерфейса из задания 5 реализуйте класс левосторонней приоритетной очереди. Продемонстрируйте работу реализованного функционала.

10. На основе интерфейса из задания 5 реализуйте класс косой приоритетной очереди. Продемонстрируйте работу реализованного функционала.

11. На основе интерфейса из задания 5 реализуйте класс декартова дерева. Продемонстрируйте работу реализованного функционала.

Задания к работе 2 по алгоритмам и структурам данных.

Все задания реализуются на языке программирования C++ (стандарт C++14 и выше). Реализованные в заданиях приложения не должны завершаться аварийно.

Во всех заданиях запрещено использование глобальных переменных (включая errno).

Во всех заданиях запрещено использование оператора безусловного перехода (goto).

Во всех заданиях запрещено пользоваться функциями, позволяющими завершить выполнение приложения из произвольной точки выполнения, вне контекста исполнения функции main.

Во всех заданиях при реализации необходимо разделять контексты работы с данными (поиск, сортировка, добавление/удаление, модификация и т. п.) и отправка данных в поток вывода / выгрузка данных из потока ввода.

Во всех заданиях все параметры функций и вводимые (с консоли, файла, командной строки) пользователем данные должны подвергаться валидации в соответствии с типом валидируемых данных, если не сказано обратное; валидация должна зависеть от типа данных и логики применения этих данных для выполнения целевой подзадачи. При передаче аргументов приложению в командную строку, их количество также должно валидироваться.

Во всех заданиях необходимо контролировать ситуации с невозможностью [пере]выделения памяти; во всех заданиях необходимо корректно освобождать всю выделенную динамическую память.

Все ошибки, связанные с операциями открытия файла, должны быть обработаны; все открытые файлы должны быть закрыты.

Во всех заданиях запрещено использование глобальных переменных. Во всех заданиях при реализации функций необходимо обеспечить возможность обработки ошибок различных типов на уровне вызывающего кода.

Во всех заданиях сравнение (на предмет эквивалентности или отношения порядка) вещественных чисел на уровне функции должно использовать значение эпсилон, которое является параметром этой функции.

Во всех заданиях при реализации функций необходимо максимально ограничивать возможность модификации (если она не подразумевается) передаваемых в функцию параметров (используйте ключевое слово const), а также вызывающего объекта, в случае вызова его методов.

Для реализованных компонентов должны быть переопределены (либо перекрыты / оставлены реализации по умолчанию - при обосновании) следующие механизмы классов C++: конструктор копирования, деструктор, оператор присваивания.

Во всех заданиях необходимо уменьшать количество копирований нетривиально копируемых объектов.

Во всех заданиях необходимо проектировать компоненты с учетом SOLID принципов. Компонент не должен управлять ресурсом, если это не является его единственной задачей.

Запрещается пользоваться элементами стандартной библиотеки языка C, если существует их аналог в стандартной библиотеке языка C++.

Запрещается использование STL (за вычетом случаев применения по заданию).

1. Реализуйте класс длинного целого числа. Данными объекта числа является: информация о знаке числа (хранится как бит значения типа *int*); динамический массив цифр в системе счисления с основанием $2^{8\times sizeof(int)}$. Обеспечьте эффективность по памяти для чисел в диапазоне

$$[-2^{8*sizeof(int)-1} \ldots 2^{8*sizeof(int)-1} - 1]$$

засчёт хранения значения в знаковом поле. Порядок хранения цифр числа - little endian. Формат хранения - знаковый (для отрицательных чисел хранение числа необходимо организовать при помощи дополнительного кода). Для класса реализуйте два конструктора: от динамического массива цифр (*int* *) в формате little endian и размера этого массива; от строкового представления числа (значение типа *char const* *) в формате big endian и основания системы счисления (значение типа *size_t*). Также для класса реализуйте операторы для: сложения длинных целых чисел (+=, +); вычитания длинных целых чисел (-=, -); умножения в столбик длинных целых чисел (*=, *); целочисленного деления в столбик длинных целых чисел (/=, /); взятия остатка от деления длинных целых чисел (%=, %); отношения эквивалентности на множестве длинных целых чисел (==, !=); отношения порядка на множестве длинных целых чисел (<, <=, >, >=); поразрядные (~, &, &=, |, |=, ^, ^=); битового сдвига (<<, <<=, >>, >>=); вставки в поток (friend <<, вывод значения числа в системе счисления с основанием 10); выгрузки из потока (friend >>, ввод значения числа в системе счисления с основанием 10).

   Продемонстрируйте работу реализованного функционала.

2. На основе реализованного в задании 1 функционала реализуйте класс дроби, хранящей в себе значения числителя (длинное целое число) и знаменателя (длинное целое число). Знак дроби должен располагаться в знаменателе дроби; в произвольный момент времени модули числителя и знаменателя любого объекта дроби должны быть взаимно простыми числами. Для класса реализуйте операторы для: сложения дробей (+=, +); вычитания дробей (-=, -); умножения дробей (*=, *); деления дробей (/=, /); отношения эквивалентности на множестве дробей (==, !=); отношения порядка на множестве дробей (<, <=, >, >=); вставки в поток (friend <<, вывод значения в формате: “<знак><числитель>/<модуль знаменателя>”); выгрузки из потока (friend >>, ввод значения в формате “<знак><числитель>/<модуль знаменателя>” или в формате строкового представления числа в системе счисления с основанием 10). Также реализуйте функционал для:
   - вычисления тригонометрических функций (sin, cos, tg, ctg, sec, cosec, arcsin, arccos, arctg, arcctg, arcsec, arccosec) над объектом дроби с заданной точностью ε (задаётся как параметр метода в виде объекта дроби);
   - возведения дроби в целую неотрицательную степень;
   - вычисления корня натуральной степени из дроби с заданной точностью ε (задаётся как параметр метода в виде объекта дроби);
   - вычисления двоичного, натурального и десятичного логарифмов из дроби с заданной точностью ε (задаётся как параметр метода в виде объекта дроби).

   Продемонстрируйте работу реализованного функционала.

3. На основе реализованного в задании 2 функционала реализуйте класс полинома от одной переменной. Полином должен быть реализован на базе структуры данных вида односвязный список (тип списка для хранения данных реализуйте самостоятельно) структур, хранящих степень переменной (целое неотрицательное число типа *unsigned int*) и коэффициент при степени переменной (объект дроби, тип которой реализован в задании 2); в произвольный момент времени список не должен содержать две одинаковых степени переменной, а также список должен быть отсортирован по значению степени переменной по убыванию. Для класса реализуйте операторы для: сложения полиномов (+=, +); вычитания полиномов (-=, -); умножения полиномов (*=, *); деления полиномов (/=, /); отношения эквивалентности на множестве полиномов (==, !=); вставки в поток (friend <<, вывод значения в формате: “({[<знак>][<коэффициент]x^[<степень>]} [...])”); выгрузки из потока (friend >>, ввод значения в формате представления, определяемом самостоятельно). Также реализуйте методы для: дифференцирования полинома по переменной; интегрирования полинома по переменной (для свободного коэффициента $c = 0$).

   Продемонстрируйте работу реализованного функционала, реализовав приложение, принимающее на вход путь к текстовому файлу, каждая строка которого описывает примеры в форматах:

   <unary operation> <polynomial>

   <polynomial1> <binary operation> <polynomial2>

   , где polynomial[1][2] - строковое представление полинома; <unary operation> - одна из операций: “diff”, “intgr”, соответствующая реализованным для типа полинома операторам/методам; <binary operation> - одна из операций: “+”, “-”, “*”, “/”, “%”, “==”, “!=”, соответствующая реализованным для типа полинома операторам. Приложение должно для каждого примера вывести результат вычисления выражения в консольный поток вывода.

4. На основе реализованного в задании 3 функционала реализуйте прикладное приложение для решения следующего набора задач с применением многочленов:
   - Реализовать проверку принадлежности многочлена $f(x)$ линейной оболочке, которая порождена многочленами $g_1(x),\ g_2(x), ..., g_k(x)$. Если ответ положительный, то получите представление вида:

$$f(x) = \sum_{l=1}^{k} A_l g_l(x).$$

   - Пусть дан многочлен $f(x), \deg f(x) = n - 1$. Далее дан набор из n различных чисел $x_0,\ x_1,\ x_2, ...,\ x_{n-1}$, по которому строятся полиномы $l_k(x)$:

$$l_k(x) = \prod_{j=0,\, j\neq k}^{n-1} \frac{x-x_j}{x_k-x_j}.$$

   Получите представление

$$f(x) = \sum_{k=0}^{n-1} B_k l_k(x).$$

   Проверьте, что для $\forall x_0,\ x_1,\ x_2, ...,\ x_{n-1}$ коэффициенты B_k определяются однозначно.
   - Для заданного многочлена $f(x)$ и числа $a \in R$ получить его представление в виде линейной комбинации степеней $(x - a)^k$.
   - Пусть многочлен $f(x) = f_0 + f_1(x - a) + ... + f_k(x - a)^k$, где $f_j,\ a \in R$. Получить его представление по степеням $(x - B)$, где $B \in R$ – параметр.
   - Рассмотрим рациональную функцию $R(x) = \frac{f(x)}{g(x)}$, где $f(x), g(x)$ – многочлены известной степени. Реализуйте методы нахождения

$$\lim_{x \to A} R(x), \quad \lim_{x \to \pm\infty} R(x).$$

   - Рассмотрим рациональную функцию $T(x) = \frac{f_1\left(s_1(x)\right)^k}{f_2\left(s_2(x)\right)^l}$, где $f_1,\ s_1, f_2, s_2$ –многочлены, $k, l \in N$. Реализуйте методы нахождения

$$\lim_{x \to A} T(x), \quad \lim_{x \to \pm\infty} T(x).$$

5. На основе реализованного в задании 2 функционала реализуйте класс комплексного числа, в котором вещественная и мнимая части должны являться числами, репрезентируемыми объектами дробей. Для класса реализуйте операторы: сложения комплексных чисел (+=, +); вычитания комплексных чисел (-=, -); умножения комплексных чисел (*=, *); деления комплексных чисел (/=, /); отношения эквивалентности на множестве комплексных чисел (==, !=); вставки в поток (friend <<, вывод значения в формате “<вещественная часть> +/- <мнимая часть>i”); выгрузки из потока (friend >>, ввод значения в формате “<вещественная часть> +/- <мнимая часть>i”). Также на уровне типа комплексного числа реализуйте объектный функционал для вычисления аргумента комплексного числа, модуля комплексного числа, корня n -й степени из комплексного числа (все вычисления выполнять с заданной точностью ε (задаётся как параметр метода в виде объекта дроби).

   Продемонстрируйте работу реализованного функционала.

6. На основе реализованного в задании 2 класса дроби реализуйте приложение-интерпретатор, решающее задачи линейной алгебры.

   Приложение принимает на вход путь к файлу с заданиями трёх типов:

   - задачи из раздела векторной алгебры:
     - скалярное произведение двух векторов в n-мерном пространстве;
     - векторное произведение двух векторов в трёхмерном и семимерном пространствах;
     - смешанное произведение трёх векторов в трёхмерном и семимерном пространствах;
     - стандартные арифметические операции над векторами в n-мерном пространстве;
     - нахождение модуля вектора;
     - процесс ортогонализации переданного множества векторов;
   - задачи из раздела матричной алгебры:
     - стандартные арифметические операции над матрицами (сложение матриц, умножение матриц, умножение матрицы на число);
     - вычисление определителя матрицы;
     - нахождение обратной матрицы;
     - решение СЛАУ методами: Гаусса, Жордана-Гаусса;
     - нахождение собственных чисел матрицы;
     - нахождение собственных векторов матрицы;
     - нахождение ранга матрицы;
     - вычисление размера линейной оболочки по заданным векторам;
     - проверка принадлежности вектора заданной линейной оболочке;
   - задачи из раздела прямых и плоскостей в пространстве:
     - уравнением задается прямая на плоскости - необходимо вывести остальные уравнения этой этой прямой на плоскости;
     - уравнениями задаются две прямые на плоскости - необходимо найти точку пересечения прямых;
     - уравнением задаётся прямая в n-мерном пространстве, а также на вход подаётся точка - необходимо найти расстояние от точки до прямой;
     - уравнением задаётся прямая в n-мерном пространстве, а также на вход подаётся точка, не лежащая на прямой - необходимо найти симметричную относительно прямой точку для данной;
     - уравнением задаётся плоскость в трёхмерном пространстве - необходимо вывести остальные уравнения этой плоскости в трёхмерном пространстве;
     - уравнением задана прямая в n-мерном пространстве - необходимо вывести остальные уравнения этой прямой в n-мерном пространстве;
     - уравнениями задаются две плоскости в трёхмерном пространстве - вывести все уравнения прямой в трёхмерном пространстве, являющейся их пересечением;
     - уравнением задана плоскость в трёхмерном пространстве, а также уравнением задана не параллельная заданной плоскости прямая - необходимо найти проекцию прямой на плоскость.

   Входные данные поступают из файла. Формат данных в файле определите самостоятельно. Файл считается корректным и валидация содержимого файла не требуется. Для обработки входного файла реализуйте класс. Под каждую задачу должен быть реализован отдельный объектный метод класса, который её решает. Для вещественной арифметики используйте реализацию дробей из задания 2. Результатом решения задачи является pdf-файл, содержащий подробное пошаговое решение с трассировкой промежуточных состояний решения, построенный в результате компиляции TeX-файла, который должен быть построен Вашим приложением.

7. Реализуйте приложение, генерирующее файлы со входными данными для интерпретатора задач линейной алгебры, реализованного в задании 6, по предоставленным ответам к этим задачам (пример: если требуется составить 10 задач на тему “вычисление скалярного произведения двух векторов в n-мерном пространстве” и в качестве ответа указано значение $\frac{125}{3}$, генератор должен составить 10 различных задач с такими векторами, чтобы их скалярное произведение равнялось $\frac{125}{3}$). Формат представления ответа и типа задачи, для которой предоставлен ответ, предусмотрите самостоятельно. Генерируемые задачи аналогичны задачам, представленным в задании 5. Ответы на задачи считаются согласованными.

   Продемонстрируйте работу вашего приложения, сгенерировав для каждого из 23 типов задач, для каждой задачи для 10 различных ответов, по 25 задач, и решите их с помощью вашей реализации интерпретатора из задания 5, сравните указанные при генерации и полученные при решении ответы.

8. На основе реализованного в задании 2 класса дроби реализуйте сервис, предоставляющий функционал для:
   - вычисления коэффициентов цепной дроби по значению обыкновенной дроби и наоборот;
   - построения коллекции подходящих дробей для цепной дроби, для обыкновенной дроби;
   - построения представления значения обыкновенной дроби в виде пути (значения типа $std::vector < bool >$) в дереве Штерна-Броко и наоборот;
   - построения представления значения обыкновенной дроби в виде пути (значения типа $std::vector < bool >$) в дереве Калкина-Уилфа и наоборот.

   Продемонстрируйте работу реализованного функционала.

9. При помощи класса дроби из задания 2 реализуйте методы для вычисления числа π с точностью 500000 знаков после запятой в системе счисления с основанием 10. Вычисление организовать четырьмя различными способами:
   - по формуле Бэйли-Боруэйна-Плаффа:

$$\pi = \sum_{n=0}^{\infty} \frac{1}{16^n}\left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6}\right);$$

   - по формуле Беллара:

$$\pi = \frac{1}{2^6}\sum_{n=0}^{\infty} \frac{(-1)^n}{2^{10n}}\left(-\frac{2^5}{4n+1} - \frac{1}{4n+3} + \frac{2^8}{10n+1} - \frac{2^6}{10n+3} - \frac{2^2}{10n+5} - \frac{2^2}{10n+7} + \frac{1}{10n+9}\right);$$

   - по формуле Чудновских:

$$\frac{1}{\pi} = \frac{1}{426880\sqrt{10005}}\sum_{n=0}^{\infty} \frac{(6n)!(13591409+545140134n)}{(3n)!(n!)^3(-640320)^{3n}};$$

   - по формуле Рамануджана:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801}\sum_{n=0}^{\infty} \frac{(4n)!(1103+26390n)}{(n!)^4 396^{4n}}.$$

   Результатами работы методов должны являться вычисленные значения и количество затраченных на нахождение итераций.

Задания к работе 3 по алгоритмам и структурам данных.

Все задания реализуются на языке программирования C++ (стандарт C++14 и выше). Реализованные в заданиях приложения не должны завершаться аварийно.

Во всех заданиях запрещено использование глобальных переменных (включая errno).

Во всех заданиях запрещено использование оператора безусловного перехода (goto).

Во всех заданиях запрещено пользоваться функциями, позволяющими завершить выполнение приложения из произвольной точки выполнения, вне контекста исполнения функции main.

Во всех заданиях при реализации необходимо разделять контексты работы с данными (поиск, сортировка, добавление/удаление, модификация и т. п.) и отправка данных в поток вывода / выгрузка данных из потока ввода.

Во всех заданиях все параметры функций и вводимые (с консоли, файла, командной строки) пользователем данные должны подвергаться валидации в соответствии с типом валидируемых данных, если не сказано обратное; валидация должна зависеть от типа данных и логики применения этих данных для выполнения целевой подзадачи. При передаче аргументов приложению в командную строку, их количество также должно валидироваться.

Во всех заданиях необходимо контролировать ситуации с невозможностью [пере]выделения памяти; во всех заданиях необходимо корректно освобождать всю выделенную динамическую память.

Все ошибки, связанные с операциями открытия файла, должны быть обработаны; все открытые файлы должны быть закрыты.

Во всех заданиях запрещено использование глобальных переменных. Во всех заданиях при реализации функций необходимо обеспечить возможность обработки ошибок различных типов на уровне вызывающего кода.

Во всех заданиях сравнение (на предмет эквивалентности или отношения порядка) вещественных чисел на уровне функции должно использовать значение эпсилон, которое является параметром этой функции.

Во всех заданиях при реализации функций необходимо максимально ограничивать возможность модификации (если она не подразумевается) передаваемых в функцию параметров (используйте ключевое слово const), а также вызывающего объекта, в случае вызова его методов.

Для реализованных компонентов должны быть переопределены (либо перекрыты / оставлены реализации по умолчанию - при обосновании) следующие механизмы классов C++: конструктор копирования, деструктор, оператор присваивания.

Во всех заданиях необходимо уменьшать количество копирований нетривиально копируемых объектов.

Во всех заданиях необходимо проектировать компоненты с учетом SOLID принципов. Компонент не должен управлять ресурсом, если это не является его единственной задачей.

Запрещается пользоваться элементами стандартной библиотеки языка C, если существует их аналог в стандартной библиотеке языка C++.

Запрещается использование STL.

1. Разработайте приложение для моделирования автоматизированной системы контроля работы лифтов.
   Основные требования к системе контроля движения лифтов:
   Время в системе дискретное с шагом 1 минута.
   Имеется n−этажное здание. В здании имеется k лифтов. Один лифт может находиться в следующих состояниях:
   - стоит с закрытыми дверями;
   - стоит с открытыми дверями;
   - движется вверх;
   - движется вниз.

   Для всех состояний важным представляется значение номера этажа, на котором лифт в данный момент находится. Двигаться лифт может только закрытым.
   В лифте имеется n кнопок с номерами этажей. Кнопки с номерами этажей могут находиться в двух положениях: “Нажата” и “Отжата”. Нажатая кнопка с номером этажа означает, что имеется задание на движение лифта в направлении к данному этажу. Направление движения должно определяться положением лифта в текущий момент относительно целевого этажа. В каждый момент времени в состоянии “Нажата” может находиться несколько кнопок с номерами этажей. При движении в каком-либо направлении по достижении следующего нужного этажа лифт останавливается, и соответствующая кнопка принимает положение “Отжата”.
   На каждом этаже имеется кнопка вызова лифта. После нажатия кнопки вызова лифта кнопка остается в нажатом состоянии до остановки лифта на данном этаже. Если кнопка вызова нажимается на этаже по пути следования лифта, лифт должен остановиться на данном этаже, после чего кнопка вызова на данном этаже должна быть автоматически отжата. Для обслуживания этажа выбирается любой из лифтов, наименее удаленных от этажа, которым нет необходимости менять направление движения. Иными словами, если в системе два лифта: первый лифт на первом этаже, а второй с пассажирами на 7 этаже поднимается вверх и нажата кнопка вызова лифта на 6 этаже, то на 6 этаж поедет первый лифт, а не второй.
   Расстояние между соседними этажами лифт преодолевает за $3 + 5 \times m_i$ секунд, где m_i − текущая загруженность лифта (отношение массы перевозимого груза к максимальной нагрузке M_i), с округлением в бóльшую сторону. В лифте одновременно может находиться несколько человек (также лифт может двигаться пустым). Лифт не должен начинать движение, если суммарный перемещаемый вес превышает M_i кг ($i = 1, 2, \ldots, k$). Если суммарный вес людей, находящихся в лифте, превышает M_i кг, в лифте возникает ситуация перегрузки, при этом последний зашедший должен покинуть кабину лифта (при этом в лифт могут зайти другие пассажиры, ожидающие лифт на этом же этаже) и, после того, как лифт уедет, повторно вызвать лифт.
   Для каждого пассажира должны быть определены следующие атрибуты: уникальный идентификатор, время появления (с точностью до минут), исходный этаж, целевой этаж, вес (в кг) - вещественное число двойной точности.
   Для моделирования работы системы лифтов разработайте интерфейс, обеспечивающий обработку набора заданий. Задания описываются текстовыми файлами, пути к которым передаются приложению в виде аргументов командной строки; в файлах указаны идентификаторы пассажиров, время появления, их вес, исходный этаж (этаж, на котором пассажир вызывает лифт), целевой этаж. Входные файлы считаются корректными. Пример файла:

   ```
   1 80 3 00:01 8
   11 65 2 00:15 6
   25 65 2 00:15 6
   ```

   Ещё один текстовый файл (путь к нему передаётся первым аргументом командной строки) содержит значения параметров n, k, $M_1 \ldots M_k$.
   Результатом работы моделирующего приложения должны являться два текстовых файла, в которых предоставлены отчеты о пассажирах и работе лифтов соответственно:
   - для каждого пассажира: время его появления, исходный этаж, целевой этаж, время погрузки в лифт, общее время движения, список пассажиров, с которыми он пересекался во время своего пути, покидал ли кабину лифта в связи с перегрузкой;
   - для каждого лифта: время простоя, время в пути, количество пройденных пролётов между этажами, суммарный перевезенный груз (значение в кг), максимальная выполненная нагрузка, количество перегрузок.

   Содержимое полученных отчётов должно быть сгруппированно по пассажирам и лифтам соответственно (порядок групп может быть произволен).

2. Некоторое инфекционное заболевание от человека к человеку передается контактным путем. Вероятность заразиться им при контакте равна $p_1 \in (0; 1]$; инкубационный период заболевания равен I дней; заразившийся может излечиться с вероятностью $p_2 \in [0; 1]$; при этом излечение начинается сразу после инкубационного периода и требует $[T \ldots 3T]$ дней ($T > 3$); во время излечения заразившийся "самоизолирован" и не контактирует со своим окружением (с теми, с кем он непосредственно знаком), в остальное время контактирование с каждым из своего окружения постоянно; в случае неуспешного излечения заболевание становится хроническим, при этом носитель хронического заболевания не является его распространителем. На иммунитет к повторному заболеванию после успешного излечения влияет параметр r (логическое значение: $true$ или $false$). Вспышкой болезни будем называть случайное возникновение заболевания у какого-либо псевдослучайного человека. Реализуйте приложение, моделирующее распространение заболевания в течение d дней. Время в системе дискретно (шаг - 1 день).
В первый день моделирования происходит вспышка болезни.
Во входном файле построчно содержатся данные о человеке и его знакомствах. Формат одной записи:

<id>,<name>,<surname>,[<familiar people id's>,...]

На последней строке входного файла указываются значения параметров p_1, p_2, d, r, T, I.

Пример файла:

1,Иван,Иванов,2,3
2,Юрий,Андреев,1,4
3,Александр,Васильев,1,4
4,Учи,Матчасть,2,3
0.15 0.825 50 true 7 4

В выходной файл (используя класс логгера из задания 2) необходимо вывести информацию, сгруппированную следующим образом:

- фамилии и имена всех не заразившихся в течение моделирования людей;
- фамилии и имена всех исцелившихся в течении моделирования людей;
- фамилии и имена исцелившихся людей, окружение которых не исцелилось;
- фамилии и имена всех не заразившихся людей, у которых всё их окружение заразилось более одного раза;
- фамилии и имена всех людей, переболевших более одного раза (только в случае, если r имеет значение $true$);
- фамилии и имена людей, у которых заболевание перешло в хроническую форму;

Для демонстрации работы реализуйте приложение-генератор входного файла, который будет подан на вход основному приложению. Выходной файл приложения должен содержать записи о 10'000-1'000'000 людях с псевдослучайно сгенерированными фамилиями и именами, а также знакомствами (граф знакомств должен быть связным). Информация о связях между людьми в выходном файле должна быть согласована (если человек с id = 52 знает человека с id = 1089, то и человек с id = 1089 знает человека с id = 52).

3. Инфекционное отделение поликлиники имеет смотровую на *N* человек и *M* докторов, которые готовы оказать помощь заболевшим или проконсультировать здоровых людей. По правилам инфекционного отделения в смотровую может зайти здоровый человек, если в ней есть свободное место и в ней находятся только здоровые люди, и наоборот: при наличии свободных мест заболевший человек может войти в смотровую, если там только заболевшие. Как только человек вошел в смотровую, он занимает свое место и ожидает доктора. Незанятый доктор выбирает пациента, пришедшего раньше других и проводит приём, который длится от 1 до *T* временных единиц. В особых случаях, доктор может попросить у другого не занятого доктора помощи, которая также может длиться от 1 до *T* временных единиц. Если все места в смотровой заняты, то пришедшие пациенты встают в очередь. При этом в очереди спустя некоторое время, при наличии заболевшего человека, заражается вся очередь. Количество пациентов и интервал их появления произволен и случаен.
   Реализуйте приложение, моделирующее работу инфекционного отделения поликлиники. Ваша программа должна вести лог-файл с подробной историей работы инфекционного отделения. Необходимо сохранять информацию о:
   - пришедших пациентах и их состоянии;
   - времени работы докторов и времени оказания помощи каждому отдельному пациенту;
   - количестве временных единиц, проведённых пациентом с момента поступления в отделение до момента окончания его приёма;
   - заражении всей очереди с указанием количества новых заболевших.

   Продемонстрируйте работу вашей программы при различных значениях параметров *N*, *M*, *T*. Подберите параметры так, чтоб показать особые случаи, которые могут возникнуть в инфекционном отделении.

4. Реализовать систему усиления игровых персонажей с помощью случайных элементов экипировки для некоторой MMORPG.

   Каждый персонаж может иметь некоторое количество ячеек для установки усиления (очень часто под ячейками имеют в виду слоты для элементов одежды, например, шлем, наплечники, перчатки, плащ и др.). Для каждого персонажа определены главные его характеристики: здоровье, броня и, в зависимости от класса персонажа: сила, ум, ловкость, и вторичные характеристики: меткость, везение, мастерство. Персонажи классифицируются следующим образом: защитники, целители, бойцы ближнего боя и бойцы дальнего боя. В зависимости от класса персонажа для него имеет значение та или иная главная характеристика. Для защитников – это здоровье, для целителей – ум, для бойцов ближнего боя – ловкость или сила, для бойцов дальнего боя – ум.

   Вторичные характеристики действуют одинаково для всех классов: меткость определяет возможность промаха при атаке, везение определяет возможность нанесения двойного урона, мастерство увеличивает наносимый урон.

   Продумайте и реализуйте закон для расчета урона (или лечения), наносимого персонажем в зависимости от его главных и вторичных характеристик.

   Продумайте и реализуйте для каждого класса персонажа его способности, а также систему генерации усилений. При создании персонажа никаких усилений (экипировки) у него нет. Продумайте возможность генерации необходимых элементов. Обратите внимание что на каждом предмете экипировки могут быть следующие характеристики: здоровье, броня, ум или сила, или ловкость, и одна или две вторичные характеристики.

   Общая характеристика персонажа определяется как суммарная совокупность значений характеристик на всех его элементах экипировки.

   При демонстрации работы приложения Вам необходимо показать генерацию элементов экипировки созданного персонажа, наносимый урон способностями в зависимости от текущих характеристик. Также необходимо иметь возможность создать произвольное количество случайных персонажей (со случайной экипировкой). Использование различных контейнеров (массивы, списки, хеш-таблицы, приоритетные очереди, деревья поиска, стеки, очереди, деки и т. д.) для обеспечения хранения данных в системе должно быть обосновано.

5. Реализовать класс монома от нескольких переменных, содержащий в качестве полей значение коэффициента (объект типа дроби из работы 2 по АиСД), а также ассоциативный контейнер вида АВЛ-дерево, ключом которого является имя переменной (непустая строка, состоящая исключительно из символов букв латинского алфавита), а значением - степень этой переменной. В классе должны быть определены и реализованы:
   - конструкторы (один из которых конструктор от char const *: например, “<5/8*x^2*yy^3*zzz^0>”);
   - методы доступа к членам класса;
   - перегруженные операторы для выполнения стандартных арифметических операций: сложение (+=, +), вычитание (-=, -), умножение (*=, *);
   - перегруженные операторы для выполнения операций отношения эквивалентности на множестве мономов от нескольких переменных (==, !=);
   - перегруженные операторы выгрузки из потока / вставки в поток для корректного ввода / вывода монома.

   На основе реализованного класса монома необходимо реализовать класс полинома от многих переменных. Класс полинома представляет собой контейнер мономов, основанный на структуре данных вида двусвязный список. В классе полинома необходимо реализовать:
   - конструкторы (один из которых конструктор от char const *: например, “<5*xy^2*yz^3*zz^0-1/2*xy^3>”);
   - методы доступа к членам класса;
   - перегруженные операторы для выполнения стандартных арифметических операций: сложение (+=, +), вычитание (-=, -), умножение (*=, *);
   - перегруженные операторы для выполнения операций отношения эквивалентности на множестве мономов от нескольких переменных (==, !=);
   - перегруженные операторы выгрузки из потока / вставки в поток для корректного ввода / вывода монома.

   Для демонстрации работы вашей программы реализуйте возможность обработки текстового файла следующего вида:

   <4x^2y^3-xy^2+4xy+1>+<x^2y^3+2xy^2+3x^2y^2-4xy+5>;
   <4x^2y^3 +4xy+1>*< x^2y^3+2xy^2+3x^2y^2-4xy+5>;
   < x^2y^3+2xy^2+3x^2y^2-4xy+5>-<4x^2y^3-xy^2+4xy+1>;
   < x^2y^3+2xy^2>==< x^2y^3+2xy^2>;
   < x^2y^3+2xy^2>!=< x^2y^3+2xy^2>;

   Замечание. Арифметические операции, возвращающие новый объект, необходимо реализовать с помощью соответствующих им операций, модифицирующих вызывающий объект: например, операция + должна быть реализована с помощью операции +=. Продемонстрировать передачу аргументов в функции по значению и по ссылке. Также необходимо продемонстрировать возврат объекта из функции. Реализуйте возможность сохранения и восстановления ваших объектов в/из потоков. Формат строкового представления для монома и полинома продумайте самостоятельно.

6. На вход программе подается набор путей к текстовым файлам, содержащим инструкции вида:

```
{
        ShowVar;
        var1=new(<size>);
        {
                var2=new(<size1>);
                ShowVar;
                {
                        {
                                var3=new(<size2>);
                        }
                        var5=new(<size3>);
                        ShowVar;
                        var21=new(<size12>);
                }
                var22=new(<size21>);

        }
        var2=new(<size1>);
        ShowVar;
}
```

Количество инструкций и их порядок произволен. Приложение должно поочерёдно обработать инструкции из каждого переданного файла. Оператор *new* запрашивает нужный размер оперативной памяти (указывается в круглых скобках в виде целого неотрицательного числа, записанного в системе счисления с основанием 16). Оператор *ShowVar* выводит в стандартный поток вывода все видимые из данного места кода переменные, с указанием размеров памяти, который ими занят. Максимально доступный объем оперативной памяти определяется параметром *N*, являющимся аргументом командной строки. Фигурные скобки определяют блоки кода, которые определяют области видимости переменных. Если переменная покидает область видимости, она становится недоступной. При покидании области видимости, занятая память не возвращается автоматически. При этом учтите, что, если очередной вызов оператора new не может быть выполнен, то необходимо запустить операцию освобождения памяти под покинувшие свою область определения переменные, а затем вновь попытаться выделить требуемый блок. При невозможности выделения блока, необходимо завершить выполнение приложения. Для распределения динамической памяти используйте алгоритмы системы двойников.

7. Разработайте приложение для проведения лотереи (аналог Русского лото). Ваше приложение должно обеспечивать генерацию билетов для очередного тиража лотереи. Количество генерируемых билетов произвольно и может быть велико (> 20’000’000 шт.). Учтите ситуацию, что не все сгенерированные билеты могут участвовать в тираже (это типичная ситуация, которая возникает при неполной реализации билетов к тиражу). Смоделируйте проведение розыгрыша: на каждом ходе проверяйте, появился ли победитель; предусмотрите систему выигрышей; предоставьте возможность поиска билетов по заданным критериям: номеру билета, величине выигрыша, и т. д. Сохраняйте информацию о проведенных тиражах для обеспечения поиска данных в будущем. Реализуйте функционал обработки данных таким образом, чтобы тип коллекции/адаптера, в котором размещаются Ваши данные, являлся параметром шаблона. Продемонстрируйте обработку данных с подстановкой в качестве параметра шаблона собственных реализаций косого дерева и двусвязного списка.

8. Разработайте приложение для обработки данных логистической компании. Информация о доставке груза должна содержать информацию о грузе (название, вес, отправитель, получатель, стоимость, стоимость доставки, содержимое и т. д.), а также информацию об участках маршрута доставки (для участка необходимо хранить: пункт отправления, время отправления, пункт назначения, время доставки в пункт назначения, вид транспорта (автомобильный, железнодорожный, морской, авиационный)). Ваше приложение должно обеспечить возможность хранения и обработки данных о доставках (поиск по заданным критериям, добавление/удаление доставок). Для демонстрации работы реализуйте генератор (на базе паттерна “фабричный метод”), выдающий случайным образом сформированную информацию о доставке (необходимо проследить за тем, чтобы пункт назначения n −ного участка и пункт отправления $(n + 1)$ −го участка совпадали). Реализуйте функционал обработки данных таким образом, чтобы тип коллекции/адаптера, в котором размещаются Ваши данные, являлся параметром шаблона. Продемонстрируйте обработку данных с подстановкой в качестве параметра шаблона собственных реализаций красно-чёрного дерева и хеш-таблицы, разрешающей коллизии методом цепочек.

9. Разработайте приложение для моделирования проведения лотереи (аналог Спортлото (6 из 49 или 5 из 36)). Ваше приложение должно обеспечивать генерацию билетов для очередного тиража лотереи. Количество генерируемых билетов произвольно и может быть велико (> 20’000’000 шт.). Учтите ситуацию, что не все сгенерированные билеты могут участвовать в тираже (это типичная ситуация, которая возникает при неполной реализации билетов к тиражу). Смоделируйте проведение розыгрыша: на каждом ходе проверяйте, появился ли победитель; предусмотрите систему выигрышей; предоставьте возможность поиска в интерактивном диалоге всех билетов по заданным критериям: номеру билета, величине выигрыша, и т. д. Сохраняйте информацию о проведенных тиражах для обеспечения поиска данных в будущем. Реализуйте функционал обработки данных таким образом, чтобы тип коллекции/адаптера, в котором размещаются Ваши данные, являлся параметром шаблона. Продемонстрируйте обработку данных с подстановкой в качестве параметра шаблона собственных реализаций стека и очереди на базе односвязного списка.

Задания к работе 4 по алгоритмам и структурам данных.

Все задания реализуются на языке программирования C++ (стандарт C++14 и выше). Реализованные в заданиях приложения не должны завершаться аварийно.

Во всех заданиях запрещено использование глобальных переменных (включая errno).

Во всех заданиях запрещено использование оператора безусловного перехода (goto).

Во всех заданиях запрещено пользоваться функциями, позволяющими завершить выполнение приложения из произвольной точки выполнения, вне контекста исполнения функции main.

Во всех заданиях при реализации необходимо разделять контексты работы с данными (поиск, сортировка, добавление/удаление, модификация и т. п.) и отправка данных в поток вывода / выгрузка данных из потока ввода.

Во всех заданиях все параметры функций и вводимые (с консоли, файла, командной строки) пользователем данные должны подвергаться валидации в соответствии с типом валидируемых данных, если не сказано обратное; валидация должна зависеть от типа данных и логики применения этих данных для выполнения целевой подзадачи. При передаче аргументов приложению в командную строку, их количество также должно валидироваться.

Во всех заданиях необходимо контролировать ситуации с невозможностью [пере]выделения памяти; во всех заданиях необходимо корректно освобождать всю выделенную динамическую память.

Все ошибки, связанные с операциями открытия файла, должны быть обработаны; все открытые файлы должны быть закрыты.

Во всех заданиях запрещено использование глобальных переменных. Во всех заданиях при реализации функций необходимо обеспечить возможность обработки ошибок различных типов на уровне вызывающего кода.

Во всех заданиях сравнение (на предмет эквивалентности или отношения порядка) вещественных чисел на уровне функции должно использовать значение эпсилон, которое является параметром этой функции.

Во всех заданиях при реализации функций необходимо максимально ограничивать возможность модификации (если она не подразумевается) передаваемых в функцию параметров (используйте ключевое слово const), а также вызывающего объекта, в случае вызова его методов.

Для реализованных компонентов должны быть переопределены (либо перекрыты / оставлены реализации по умолчанию - при обосновании) следующие механизмы классов C++: конструктор копирования, деструктор, оператор присваивания.

Во всех заданиях необходимо уменьшать количество копирований нетривиально копируемых объектов.

Во всех заданиях необходимо проектировать компоненты с учетом SOLID принципов. Компонент не должен управлять ресурсом, если это не является его единственной задачей.

Запрещается пользоваться элементами стандартной библиотеки языка C, если существует их аналог в стандартной библиотеке языка C++.

Запрещается использование STL.

1. Реализуйте логгер (repo path: */logger/client_logger*) на основе контракта *logger* (repo path: */logger/logger*). Ваша реализация логгера должна конфигурироваться на основе реализации порождающего паттерна проектирования “строитель”. Поэтапное построение объекта логгера предполагает следующие возможности:
    - настройка потоков вывода (файловые и консольный) с заданием для каждого потока вывода множества severity; созданный реализацией строителя объект логгера должен выводить сообщения с заданным severity только в те настроенные потоки вывода, в множестве severity которых присутствует переданное методу log значение severity;
    - настройка структуры лога, печатаемого логгером в потоки вывода, в виде форматной строки в стиле C (в форматной строке допустимо использование следующих флагов: %d - текущая дата (григорианский календарь, GMT+0); %t - текущее время (GMT+0); %s - строковое представление уровня жёсткости логгирования; %m - логгируемое сообщение;
    - настройка информации о потоках вывода, их severity и структуры лога на основе содержимого конфигурационного файла (в качестве параметров подаются путь к конфигурационному файлу и путь поиска (path) уровня конфигурационного файла, где находится информация о наполнении логгера). Структуру конфигурационного файла определите самостоятельно; предполагается, что найденное содержимое конфигурационного файла с вышеописанной информацией валидно, однако не гарантируется, что это содержимое будет найдено в файле, а также не не гарантируется существование самого файла;
    - удаление всех настроенных параметров с возможностью дальнейшей работы со строителем.

    Потоки вывода, используемые объектом логгера, должны быть открыты во время жизни объекта логгера. Учтите, что один и тот же поток вывода может использоваться единовременно различными объектами логгеров (и множества *severity* для этого потока вывода на уровне различных объектов логгера также могут различаться). При разрушении последнего объекта логгера, связанного с заданным файловым потоком вывода, этот файловый поток вывода должен быть закрыт (реализуйте механизм подсчёта ссылок).

2. Реализуйте (repo path: */allocator/allocator_global_heap*) аллокатор на основе контракта *allocator* (repo path: */allocator/allocator*). Выделение и освобождение динамической памяти реализуйте посредством глобальных операторов *new* и *delete* соответственно. В типе аллокатора допускается единственное поле типа *logger* * - указатель на объект логгера, используемый объектом аллокатора в процессе работы. При невозможности выделения памяти должна быть сгенерирована исключительная ситуация типа *std::bad_alloc*, а также сгенерированный объект исключения должен быть перехвачен и обработан в вызывающем коде. При освобождении памяти должна осуществляться проверка на принадлежность освобождаемого блока к текущему объекту аллокатора (если это не так, необходимо сгенерировать исключительную ситуацию типа *std::logic_error*, а также перехватить и обработать объект исключения в вызывающем коде). Продемонстрируйте работу объекта аллокатора, разместив в нём объекты различных типов данных. Предусмотрите логгирование (на уровне объекта реализованного аллокатора) следующих данных/ситуаций:
   - проброс исключительной ситуации - приоритет *logger::severity::error*;
   - переопределение запроса пользователя на выделение памяти - приоритет *logger::severity::warning*;
   - состояние блока (без служебных данных) перед освобождением (в виде массива байт) - приоритет *logger::severity::debug*;
   - начало/окончание вызова любого метода уровня интерфейса аллокатора - приоритет *logger::severity::debug*;
   - начало/окончание вызова любого метода уровня реализованного компонента - приоритет *logger::severity::trace*.

3. Реализуйте (repo path: */allocator/allocator_sorted_list*) аллокатор на основе контракта *allocator_with_fit_mode* (repo path: */allocator/allocator*). Выделение памяти реализуйте при помощи методов (с возможностью конфигурации конкретной реализации через конструктор объекта и через метод *set_fit_mode* контракта *allocator_with_fit_mode*) первого подходящего, лучшего подходящего, худшего подходящего, а освобождение памяти - при помощи метода освобождения в рассортированном списке. В типе аллокатора допускается единственное поле типа *void ** - указатель на доверенную объекту аллокатора область памяти. Служебные данные для работы аллокатора размещайте в доверенной ему области памяти (размер доверенной области памяти задаётся на уровне конструктора объекта аллокатора и доверенная память при этом запрашивается из объекта аллокатора, передаваемого как параметр по умолчанию конструктору (если объект аллокатора отсутствует, память запрашивается из глобальной кучи)). При освобождении памяти в объект аллокатора должна осуществляться проверка на принадлежность освобождаемого блока к текущему объекту аллокатора. Обращения к объекту аллокатора должны быть синхронизированы (должно гарантироваться, что в произвольный момент времени жизни объекта аллокатора выделение/освобождение памяти в нём выполняется максимум в одном потоке исполнения). Продемонстрируйте работу объекта аллокатора, разместив в нём объекты различных типов данных. Предусмотрите логгирование (на уровне объекта реализованного типа аллокатора) следующих данных/ситуаций:
   - проброс исключительной ситуации - приоритет *logger::severity::error*;
   - переопределение запроса пользователя на выделение памяти - приоритет *logger::severity::warning*;
   - после выполнения операции выделения/освобождения памяти: объём доступной для выделения памяти в байтах - приоритет *logger::severity::information*;
   - состояние блока (без служебных данных) перед освобождением (в виде массива байт) - приоритет *logger::severity::debug*;
   - после выполнения операции выделения/освобождения памяти: состояние всей неслужебной памяти, управляемой объектом аллокатора (формат строкового представления блока: “<*block availability*> <*block size*>”, где <*block availability*> - признак свободности/занятости блока (для свободного блока - строка “avail”, для занятого - строка “occup”), <*block size*> - размер текущего блока в байтах (без учёта служебной памяти уровня блока; строковые представления блоков сепарированы символом ‘|’); блоки в строковом представлении отсортированы по возрастанию по ключу адреса байта памяти начала) - приоритет *logger::severity::debug*;
   - начало/окончание вызова любого метода уровня интерфейса аллокатора - приоритет *logger::severity::debug*;
   - начало/окончание вызова любого метода уровня реализованного компонента - приоритет *logger::severity::trace*.

4. Реализуйте (repo path: */allocator/allocator_boundary_tags*) аллокатор на основе контракта *allocator_with_fit_mode* (repo path: */allocator/allocator*). Выделение памяти реализуйте при помощи методов (с возможностью конфигурации конкретной реализации через конструктор объекта и через метод *set_fit_mode* контракта *allocator_with_fit_mode*) первого подходящего, лучшего подходящего, худшего подходящего, а освобождение памяти - при помощи метода освобождения с дескрипторами границ. В типе аллокатора допускается единственное поле типа *void* * - указатель на доверенную объекту аллокатора область памяти. Служебные данные для работы аллокатора размещайте в доверенной ему области памяти (размер доверенной области памяти задаётся на уровне конструктора объекта аллокатора и доверенная память при этом запрашивается из объекта аллокатора, передаваемого как параметр по умолчанию конструктору (если объект аллокатора отсутствует, память запрашивается из глобальной кучи)). При освобождении памяти в объект аллокатора должна осуществляться проверка на принадлежность освобождаемого блока к текущему объекту аллокатора. Обращения к объекту аллокатора должны быть синхронизированы (должно гарантироваться, что в произвольный момент времени жизни объекта аллокатора выделение/освобождение памяти в нём выполняется максимум в одном потоке исполнения). Продемонстрируйте работу объекта аллокатора, разместив в нём объекты различных типов данных. Предусмотрите логгирование (на уровне объекта реализованного типа аллокатора) следующих данных/ситуаций:
   - проброс исключительной ситуации - приоритет *logger::severity::error*;
   - переопределение запроса пользователя на выделение памяти - приоритет *logger::severity::warning*;
   - после выполнения операции выделения/освобождения памяти: объём доступной для выделения памяти в байтах - приоритет *logger::severity::information*;
   - состояние блока (без служебных данных) перед освобождением (в виде массива байт) - приоритет *logger::severity::debug*;
   - после выполнения операции выделения/освобождения памяти: состояние всей неслужебной памяти, управляемой объектом аллокатора (формат строкового представления блока: “<*block availability*> <*block size*>”, где <*block availability*> - признак свободности/занятости блока (для свободного блока - строка “avail”, для занятого - строка “occup”), <*block size*> - размер текущего блока в байтах (без учёта служебной памяти уровня блока; строковые представления блоков сепарированы символом ‘|’); блоки в строковом представлении отсортированы по возрастанию по ключу адреса байта памяти начала) - приоритет *logger::severity::debug*;
   - начало/окончание вызова любого метода уровня интерфейса аллокатора - приоритет *logger::severity::debug*;
   - начало/окончание вызова любого метода уровня реализованного компонента - приоритет *logger::severity::trace*.

5. Реализуйте (repo path: */allocator/allocator_buddies_system*) аллокатор на основе контракта *allocator_with_fit_mode* (repo path: */allocator/allocator*). Выделение памяти реализуйте при помощи методов (с возможностью конфигурации конкретной реализации через конструктор объекта и через метод *set_fit_mode* контракта *allocator_with_fit_mode*) первого подходящего, лучшего подходящего, худшего подходящего, а освобождение памяти - при помощи метода освобождения в системе двойников. В типе аллокатора допускается единственное поле типа *void* * - указатель на доверенную объекту аллокатора область памяти. Служебные данные для работы аллокатора размещайте в доверенной ему области памяти (размер доверенной области памяти задаётся на уровне конструктора объекта аллокатора и доверенная память при этом запрашивается из объекта аллокатора, передаваемого как параметр по умолчанию конструктору (если объект аллокатора отсутствует, память запрашивается из глобальной кучи)). При освобождении памяти в объект аллокатора должна осуществляться проверка на принадлежность освобождаемого блока к текущему объекту аллокатора. Обращения к объекту аллокатора должны быть синхронизированы (должно гарантироваться, что в произвольный момент времени жизни объекта аллокатора выделение/освобождение памяти в нём выполняется максимум в одном потоке исполнения). Продемонстрируйте работу объекта аллокатора, разместив в нём объекты различных типов данных. Предусмотрите логгирование (на уровне объекта реализованного типа аллокатора) следующих данных/ситуаций:
   - проброс исключительной ситуации - приоритет *logger::severity::error*;
   - переопределение запроса пользователя на выделение памяти - приоритет *logger::severity::warning*;
   - после выполнения операции выделения/освобождения памяти: объём доступной для выделения памяти в байтах - приоритет *logger::severity::information*;
   - состояние блока (без служебных данных) перед освобождением (в виде массива байт) - приоритет *logger::severity::debug*;
   - после выполнения операции выделения/освобождения памяти: состояние всей неслужебной памяти, управляемой объектом аллокатора (формат строкового представления блока: “<*block availability*> <*block size*>”, где <*block availability*> - признак свободности/занятости блока (для свободного блока - строка “avail”, для занятого - строка “occup”), <*block size*> - размер текущего блока в байтах (без учёта служебной памяти уровня блока; строковые представления блоков сепарированы символом ‘|’); блоки в строковом представлении отсортированы по возрастанию по ключу адреса байта памяти начала) - приоритет *logger::severity::debug*;
   - начало/окончание вызова любого метода уровня интерфейса аллокатора - приоритет *logger::severity::debug*;
   - начало/окончание вызова любого метода уровня реализованного компонента - приоритет *logger::severity::trace*.

6. Реализуйте (repo path: */allocator/allocator_red_black_tree*) аллокатор на основе контракта *allocator_with_fit_mode* (repo path: */allocator/allocator*). Выделение памяти реализуйте при помощи методов (с возможностью конфигурации конкретной реализации через конструктор объекта и через метод *set_fit_mode* контракта *allocator_with_fit_mode*) первого подходящего, лучшего подходящего, худшего подходящего, а освобождение памяти - при помощи алгоритмов вставки/удаления для красно-чёрного дерева. В типе аллокатора допускается единственное поле типа *void* * - указатель на доверенную объекту аллокатора область памяти. Служебные данные для работы аллокатора размещайте в доверенной ему области памяти (размер доверенной области памяти задаётся на уровне конструктора объекта аллокатора и доверенная память при этом запрашивается из объекта аллокатора, передаваемого как параметр по умолчанию конструктору (если объект аллокатора отсутствует, память запрашивается из глобальной кучи)). При освобождении памяти в объект аллокатора должна осуществляться проверка на принадлежность освобождаемого блока к текущему объекту аллокатора. Обращения к объекту аллокатора должны быть синхронизированы (должно гарантироваться, что в произвольный момент времени жизни объекта аллокатора выделение/освобождение памяти в нём выполняется максимум в одном потоке исполнения). Продемонстрируйте работу объекта аллокатора, разместив в нём объекты различных типов данных. Предусмотрите логгирование (на уровне объекта реализованного типа аллокатора) следующих данных/ситуаций:
   - проброс исключительной ситуации - приоритет *logger::severity::error*;
   - переопределение запроса пользователя на выделение памяти - приоритет *logger::severity::warning*;
   - после выполнения операции выделения/освобождения памяти: объём доступной для выделения памяти в байтах - приоритет *logger::severity::information*;
   - состояние блока (без служебных данных) перед освобождением (в виде массива байт) - приоритет *logger::severity::debug*;
   - после выполнения операции выделения/освобождения памяти: состояние всей неслужебной памяти, управляемой объектом аллокатора (формат строкового представления блока: “<*block availability*> <*block size*>”, где <*block availability*> - признак свободности/занятости блока (для свободного блока - строка “avail”, для занятого - строка “occup”), <*block size*> - размер текущего блока в байтах (без учёта служебной памяти уровня блока; строковые представления блоков сепарированы символом ‘|’); блоки в строковом представлении отсортированы по возрастанию по ключу адреса байта памяти начала) - приоритет *logger::severity::debug*;
   - начало/окончание вызова любого метода уровня интерфейса аллокатора - приоритет *logger::severity::debug*;
   - начало/окончание вызова любого метода уровня реализованного компонента - приоритет *logger::severity::trace*.