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.