Задания к работе №5 по Фундаментальным алгоритмам.

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++.

1. Реализуйте класс *binary_int*, который позволяет выполнять операции с целыми числами, используя только побитовые операции и логические операции для выполнения арифметических задач. Класс должен работать с числами в диапазоне значений типа *int* и представлять их только в виде двоичных значений. Требования к реализации:
    - Запрещено использовать стандартные арифметические операторы +, -, *, /, % для выполнения сложения и вычитания. Все операции реализуются побитовыми манипуляциями с числовыми значениями;
    - Необходимо перегрузить операторы:
        - унарный минус (*operator*-) для возврата числа с противоположным знаком;
        - префиксного и постфиксного инкремента (*operator*++) для увеличения значения числа на 1;
        - префиксного и постфиксного декремента (*operator*--) для уменьшения значения числа на 1;
        - сложения чисел (*operator*+=) с модификацией вызывающего объекта;
        - сложения чисел (*operator*+) с возвратом нового объекта;
        - вычитания чисел (*operator*-=) с модификацией вызывающего объекта;
        - вычитания чисел (*operator*-) с возвратом нового объекта;
        - умножения чисел (*operator**=) с модификацией вызывающего объекта;
        - умножения чисел (*operator**) с возвратом нового объекта;
        - побитового сдвига влево (*operator*<<=) и вправо (*operator*>>=) с модификацией вызывающего объекта;
        - побитового сдвига влево (*operator*<<) и вправо (*operator*>>) с возвратом нового объекта;
        - вставки в поток (*operator*<<) для печати в поток вывода битового представления числа;
    - Необходимо реализовать метод, возвращающий для объекта пару объектов типа *binary_int*, где в первом объекте сохранены только половина старших битов вызывающего объекта, а во втором - только половина младших битов вызывающего объекта; остальные биты необходимо обнулить.

   Продемонстрируйте работу реализованного функционала.

2. Реализовать класс encoder. В классе определить и реализовать:
    - Конструктор, принимающий ключ шифрования (массив байтов типа std::vector<std::byte>)
    - Метод *encode*, который принимает путь ко входному файлу (типа std::string), выходному файлу (типа std::string) и флаг, отвечающий за то, выполнять шифрование или дешифрование (типа bool) и выполняет процесс шифрования/дешифрования файла
    - mutator для значения ключа

   Шифрование/дешифрование файлов выполняется алгоритмом RC4. Структура содержимого файлов произвольна. Продемонстрировать работу класса, шифрование/дешифрование данного файла.

3. Реализовать класс *logical_values_array*. В классе определить и реализовать:
    - поле_value (типа unsigned int), которое хранит значение логической величины
    - 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: инверсия, конъюнкция, дизъюнкция, импликация, коимпликация, сложение по модулю 2, эквивалентность, стрелка Пирса, штрих Шеффера. Примечание: если одну операцию возможно выразить через другую, то необходимо реализовывать одно через другое (например, эквивалентность можно реализовать через сложение по модулю 2 и инверсию)
    - статический метод equals, сравнивающий два значения типа *logical_values_array* по отношению эквивалентности
    - метод get_bit, который возвращает значение бита по его позиции (является параметром)
    - 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.

4. 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)
    - методы, производящие операции сложения, вычитания, умножения и деления комплексных чисел
    - метод, возвращающий модуль комплексного числа
    - метод, возвращающий аргумент комплексного числа

    Продемонстрируйте работу реализованного функционала.

5. Необходимо реализовать функцию *cdecl_translate*, которая принимает строку (*std::string*) с кодом объявления переменной на языке C и возвращает его текстовое описание на английском языке (*std::string*).

   1. Входные и выходные данные:
      - Функция принимает один аргумент — объект класса *std::string*, содержащий строку с объявлением переменной на языке Си (например, “char* b;”).
      - Функция возвращает объект класса *std::string*, представляющий описание данного типа на английском языке (в данном случае результат должен быть: “declare b as pointer to char”).

   2. Требования к переводу:

      - Функция должна распознавать стандартные типы данных, такие как int, char, float, double и производные типы, такие как указатели (*), массивы ([]), функции (например, int f()), и типы со скобками для группировки.
      - Функция должна корректно переводить вложенные и комбинированные типы, например:
        - “char* a;” -> “declare a as pointer to char”
        - “int** b;” -> “declare b as pointer to pointer to int”
        - “float d[10];” -> “declare d as array of 10 elements of float”
        - “int (*func)();” -> “declare func as pointer to function returning int”

   3. Проверка на корректность ввода:

      - Функция должна проверять синтаксическую корректность строки объявления.
      - Типы и идентификаторы переменных должны соответствовать правилам объявления для языка C:
        - Тип данных должен быть допустимым.
        - Название переменной может быть любым подходящим именем, начинающимся с буквы или подчеркивания (_) и содержащим буквы, цифры, и подчеркивания.
      - Если обнаружен некорректный ввод, функция должна вернуть строку с описанием ошибки и указанием её позиции (например, Syntax error at position 4).
      - Для разбора строки и проверки корректности лексем (типа данных, имени переменной, операторов *, [], () и т.д.) допускается использование регулярных выражений.

   Также реализуйте интерпретатор для тестирования функции cdecl_translate. Он должен получать путь к файлу в качестве аргумента командной строки. Файл должен содержать тестовые строки — каждое объявление переменной в новой строке. Ответы на запросы, в том числе описание ошибок также необходимо печатать в стандартный поток вывода.

6. Реализуйте класс vector, который будет представлять динамический массив типа double с переменной длиной и основными методами для работы с ним. Этот класс должен поддерживать типичные операции для контейнеров и обеспечивать совместимость с const.

   Ваш класс должен содержать следующие методы:

   1. at(size_t index) – возвращает ссылку на элемент с индексом index. Должен проверять корректность индекса и выбрасывать исключение, если индекс выходит за пределы массива.
   2. front – возвращает ссылку на первый элемент массива.
   3. back – возвращает ссылку на последний элемент массива.
   4. data – возвращает указатель на внутренний массив данных.
   5. empty – возвращает bool, проверяя, пуст ли массив.
   6. size – возвращает текущее количество элементов в массиве.
   7. reserve(size_t num) – увеличивает вместимость массива до num, если текущая вместимость меньше.
   8. capacity() – возвращает текущую вместимость массива.
   9. shrink_to_fit() – уменьшает вместимость массива до текущего размера, если вместимость больше размера.
   10. clear() – очищает массив, устанавливая его размер в 0.
   11. insert(size_t index, double elem) – вставляет элемент elem на позицию index, сдвигая все последующие элементы вправо. Должен корректно увеличивать вместимость при необходимости.
   12. erase(size_t index) – удаляет элемент на позиции index, сдвигая все последующие элементы влево.
   13. push_back(double elem) – добавляет элемент elem в конец массива.
   14. pop_back() – удаляет последний элемент массива.
   15. resize(size_t size, double elem) – изменяет размер массива. Если новый размер больше текущего, заполняет новые элементы значением elem.
   16. operator<=> – реализует трехстороннее сравнение двух векторов, аналогичное строкам (например, поэлементное сравнение).
   17. operator== – реализует проверку на равенство двух векторов.

   В классе реализуйте следующие конструкторы:

   1. Конструктор из количества элементов и значения по умолчанию – создаёт вектор с заданным количеством элементов, каждый из которых инициализируется значением по умолчанию.
   2. Конструктор из количества элементов – создаёт вектор с заданным количеством элементов, инициализируя их значением 0.0.
   3. Конструктор из пары итераторов – создаёт вектор, копируя элементы из заданного диапазона итераторов.
   4. Конструктор из списка инициализации – создаёт вектор с элементами, переданными в списке инициализации.

   Также реализуйте внутренний класс итератора и методы доступа к ним (begin(), end()). Реализованные итераторы должны обладать функционалом произвольного доступа. Объект вашего класса должен корректно работать с квалификатором const – он должен запрещать изменение себя и хранимых значений, но осуществлять доступ к ним.

7. Создайте класс Warehouse, который управляет запасами на складе, и несколько типов товаров. На складе должны быть разные категории товаров, такие как: продукты питания, бытовая техника, и строительные материалы; каждая со своими особенностями.
Описание классов и их функциональности

1. Базовый класс Product (Товар):
    - Содержит общие характеристики всех товаров: название, уникальный ID, вес, цена, и срок хранения (в днях).
    - Имеет виртуальные методы:
        - calculateStorageFee() — рассчитывает стоимость хранения товара в день на основе веса.
        - displayInfo() — выводит информацию о товаре в структурированном формате.
    - Реализуйте «правило трёх» для класса Product: конструктор копирования, оператор присваивания, и деструктор.
2. Наследники PerishableProduct (Скоропортящийся продукт), ElectronicProduct (Электронный продукт) и BuildingMaterial (Строительный материал):
    - Класс PerishableProduct:
        - Содержит дополнительное поле expirationDate (дата истечения срока годности).
        - Переопределяет calculateStorageFee() для увеличения стоимости хранения по мере приближения к дате истечения срока годности.
    - Класс ElectronicProduct:
        - Включает поле warrantyPeriod (гарантийный срок) и powerRating (мощность).
        - Переопределяет метод displayInfo() для отображения информации о мощности и гарантии.
    - Класс BuildingMaterial:
        - Содержит поле flammability (воспламеняемость), указывающее, может ли материал быть пожароопасным.
        - Переопределяет метод calculateStorageFee() для повышения стоимости хранения при высоком уровне воспламеняемости.
3. Класс Warehouse (Склад):
    - Управляет запасами товаров. Имеет методы для добавления товаров, удаления товаров по ID, поиска товаров по категории и расчёта общей стоимости хранения всех товаров на складе.
    - Метод getExpiringProducts(days) должен возвращать список товаров класса PerishableProduct, срок годности которых истекает в течение указанного количества дней.
    - Метод displayInventory() для отображения всех товаров на складе, отсортированных по категориям.

Требования к перегрузке операторов

1. Оператор += для добавления товаров на склад.
2. Оператор -= для удаления товара по ID (принимает ID товара как аргумент).
3. Оператор [] для доступа к товару по ID. Если товара с данным ID нет на складе, возвращается nullptr.
4. Оператор << для вывода информации о текущем состоянии склада и всех его товарах в консоль.