Задания к работе №5 по Фундаментальным Алгоритмам.

Все задания реализуются на языке программирования C++ (стандарт C++20 и выше).

Реализованные в заданиях приложения не должны завершаться аварийно; все возникающие исключительные ситуации должны быть перехвачены и обработаны.

Во всех заданиях запрещено использование: глобальных переменных (включая errno), 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 системных ресурсов уровня ОС (файлы, средства синхронизации, etc.), должны быть обработаны; все открытые системные ресурсы должны быть возвращены ОС.

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 объекта, в случае метода.

Для реализованных компонентов должны быть переопределены (либо перекрыты - при обосновании) следующие механизмы классов C++: конструктор копирования, деструктор, оператор присваивания, конструктор перемещения, присваивание перемещением.

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 Си, если существует их аналог в стандартной библиотеке языка C++.

Для задач, каталоги которых в репозитории содержат папку tests, требуется демонстрация прохождения всех описанных тестов для реализованных компонентов. Модификация кода тестов запрещена.

1. Реализуйте логгер (repo path: */logger/client_logger*) на основе базового класса *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 класс дерева двоичного поиска (repo path: */associative_container/search_tree/binary_search_tree*). Распределение памяти для вложенных в объект дерева данных организуйте через объект аллокатора, внедряемый в объект дерева двоичного поиска через конструктор. В узлах дерева необходимо хранение указателя на родительский узел. Операции уровня CRUD-классификации функционала взаимодействия с данными реализуйте на основе поведенческого паттерна проектирования “шаблонный метод”, предусмотрев вспомогательный функционал для выполнения в типах-наследниках реализованного типа дерева двоичного поиска операций балансировки после выполнения основного алгоритма. При невозможности выполнения операции, генерируйте исключительную ситуацию (типы исключительных ситуаций являются nested по отношению к типу дерева). Traverse-взаимодействие с объектом дерева двоичного поиска реализуйте на основе 12 типов итераторов, синтезируемых свёрткой агрегатной функцией вида декартово произведение множеств для следующих множеств:
   - Вид обхода: префиксный, инфиксный, постфиксный;
   - Направление обхода: прямой, обратный;
   - Иммутабельность обходимых данных: мутабельны, иммутабельны.

   Для каждого обходимого узла итератор должен обеспечить функционал доступа к содержимому узла: ключу, значению, глубине (относительно корня; глубина корня дерева равна нулю). Также реализуйте методы поворотов: малого левого/правого, двойного левого/правого, большого левого/правого.
   Продемонстрируйте работу реализованного функционала.

3. На основе реализованного класса из задания 2 реализуйте родовой класс АВЛ-дерева (repo path: */associative_container/search_tree/binary_search_tree/AVL_tree*). Для реализации пронаследуйте тип узла дерева двоичного поиска с добавлением необходимой информации и переопределите/доопределите функционал шаблонных методов для обеспечения выполнения балансировки дерева после выполнения основного алгоритма.

4. На основе реализованного класса из задания 2 реализуйте родовой класс красно-чёрного дерева (repo path: */associative_container/search_tree/binary_search_tree/red_black_tree*). Для реализации пронаследуйте тип узла дерева двоичного поиска с добавлением необходимой информации и переопределите/доопределите функционал шаблонных методов для обеспечения выполнения балансировки дерева после выполнения основного алгоритма.

5. На основе реализованного класса из задания 2 реализуйте родовой класс косого дерева (repo path: */associative_container/search_tree/binary_search_tree/splay_tree*). Для реализации пронаследуйте тип узла дерева двоичного поиска с добавлением необходимой информации и переопределите/доопределите функционал шаблонных методов для обеспечения выполнения балансировки дерева после выполнения основного алгоритма.

6. На основе реализованного класса из задания 2 реализуйте родовой класс scapegoat-дерева (repo path: */associative_container/search_tree/binary_search_tree/scapegoat_tree*). Для реализации пронаследуйте тип узла дерева двоичного поиска с добавлением необходимой информации и переопределите/доопределите функционал шаблонных методов для обеспечения выполнения балансировки дерева после выполнения основного алгоритма.

7. На основе реализованных в заданиях 2-6 классов деревьев двоичного поиска реализуйте родовой класс хэш-таблицы (repo path: */associative_container/hash_table*). Для разрешения коллизий используйте метод цепочек, где в качестве цепочек выступают реализации сбалансированных деревьев поиска; используемый в качестве типа цепочки тип сбалансированного дерева поиска конфигурируется как шаблонный параметр родового класса хеш-таблицы.

   На основе реализованного типа хеш-таблицы реализуйте приложение, осуществляющее подмену слов в тексте. На вход программе через аргументы командной строки подаются пути к текстовым файлам, и на основании словаря синонимов, в каждом текстовом файле происходит подмена синонимичного понятия на эталонное. Словарь синонимов – это набор записей вида <эталонное слово> {список синонимов, разделённых запятой}. При обработке текста выполняется поиск слова по словарю: если текущее слово - эталонное, то замену его производить не требуется; если же оно является синонимом, то необходимо выполнить его замену на соответствующее ему эталонное слово.

   Ваше приложение должно быть устойчивым: ни при каких обстоятельствах оно не должно аварийно завершаться. Для демонстрации работы используйте содержательный словарь синонимов (> 1000 записей), по вашему выбору, входные файлы могут быть с английским текстом или с русским.