Задания к работе №2 по языкам и методам программирования.

Все задания реализуются на языке программирования C++ (стандарт C++14 и выше). Реализованные в заданиях приложения не должны завершаться аварийно; все возникающие исключительные ситуации должны быть перехвачены и обработаны.

Во всех заданиях запрещено пользоваться функциями, позволяющими завершить выполнение приложения из произвольной точки выполнения.

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 вводимые (с консоли, файла, командной строки) пользователем данные должны (если не сказано обратное) быть подвергнуты валидации в соответствии с типом валидируемых данных.

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.

Реализованные компоненты должны зависеть от абстракций, а не от конкретных реализаций абстракций. Для реализованных компонентов должны быть переопределены (либо перекрыты - при обосновании) следующие механизмы классов C++: конструктор копирования, деструктор, оператор присваивания, конструктор перемещения, присваивание перемещением.

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

1. Реализуйте родовой класс дерева двоичного поиска (repo path: */associative_container/search_tree/binary_search_tree*) на основе контракта *search_tree* (repo path: */associative_container/search_tree*). Распределение вложенных в объект дерева данных организуйте через объект аллокатора, внедряемый в объект дерева двоичного поиска по указателю интерфейсного типа через конструктор. В узлах дерева запрещено 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: insertion_of_existent_key_attempt_exception, obtaining_of_nonexistent_key_attempt_exception, disposal_of_nonexistent_key_attempt_exception). 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.

2. 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.

3. 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.

4. 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.

5. 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.

6. На основе реализованных в заданиях 1-5 классов деревьев двоичного поиска реализуйте родовой класс хеш-таблицы (repo path: */associative_container/hash_table*) на основе контракта *associative_container* (repo path: */associative_container*). 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 типовой параметр родового класса хеш-таблицы.

7. На основе реализованного в задании 6 класса хеш-таблицы реализуйте приложение, позволяющее организовать макрозамены в тексте. На вход программы подается текстовый файл, который содержит в начале файла набор директив #define и далее обычный текст. Синтаксис директивы соответствует стандарту языка C:

   #define <def_name> <value>

   Аргументов у директивы нет. Ваша программа должна обработать текстовый файл, выполнив замены во всем тексте последовательности символов <def_name> на <value>. Количество директив произвольно, некорректных директив нет, размер текста произволен. В имени <def_name> допускается использование символов латинского алфавита (прописные и строчные буквы не отождествляются) и символов цифр; значение <value> произвольно и завершается символом переноса строки или символом конца файла. Для хранения имен макросов и макроподстановок используйте хеш-таблицу размера HASHSIZE (начальное значение равно 128). Для вычисления хеш-функции интерпретируйте <def_name> как число, записанное в системе счисления с основанием 62 (алфавит этой системы счисления состоит из символов {0, …,9, A, …, Z, a, …, z}). Хеш-значение для <def_name> в рамках хеш-таблицы вычисляйте как остаток от деления эквивалентного для <def_name> числа в системе счисления с основанием 10 на значение HASHSIZE. В ситуациях, когда после модификации таблицы длины самой короткой и самой длинной цепочек в хеш-таблице различаются в 2 раза и более, пересобирайте хеш-таблицу с использованием другого значения HASHSIZE (логику модификации значения HASHSIZE продумайте самостоятельно) до достижения примерно равномерного распределения объектов структур по таблице. Обеспечьте эффективный расчёт хэш-значений при пересборке таблицы при помощи кэширования.