**ФИТ НГУ, курс ООП, осенний семестр 2015**

**Задача 1f. Словарь на основе дерева**

**История изменений**

* Версия 2. 18.09.2015. Исправлена опечатка - упоминание хэш-таблицы.
* Версия 1. 13.09.2015. Документ создан.

**Общие сведения**

Словарь (ассоциативный контейнер) - это абстракция, позволяющая хранить пары ключ-значение, или, динамическое множество, которое предоставляет операции вставки, удаления и поиска значения по ключу. Реализация словаря с помощью сбалансированного дерева позволяет гарантировать алгоритмическую сложность O(lgN) для этих операций.

Подробно о словарях можно прочитать в части III книги Кормена, о сбалансированных деревьях - главы 12, 13 и другие.

К контрольному сроку должна быть готова основная функциональность без тестов (конструирование/уничтожение, вставка-удаление-поиск).

**Задача**

**Реализовать ассоциативный контейнер с заданным интерфейсом (см. раздел “Реализация”). Используйте любой, вам еще неизвестный алгоритм построения сбалансированного дерева. Например, splay tree или treap.**

1. Тщательно задокументировать публичные члены класса на языке, приближенном к техническому английскому.
2. Написать юнит-тесты на все публичные методы класса с помощью любой специализированной библиотеки (рекомендуется Google Test Framework<http://code.google.com/p/googletest/>), либо без оной (на усмотрение преподавателя). Убедиться в полноте покрытия кода тестами (каждая строчка кода должна исполняться хотя бы одним тестом).

**Методические указания**

* При написании кода особое внимание обращайте на обработку исключительных ситуаций и граничных случаев, в частности, на корректность аргументов методов. Продумывайте и документируйте обработку ошибок в ваших методах.
* Попробуйте часть методов протестировать до их реализации.
* Изучите открытые реализации std::map и других словарей. Сравните с вашей реализацией.

**Реализация**

// В этой задаче для простоты не требуется делать контейнер шаблонным,

// но это вполне допускается по желанию студента. Для нешаблонного решения,

// введем типы ключей и значений: ключом будет выступать строка (например,

// имя студента, значением - произвольная структура (например, численные

// характеристики студента.

typedef std::string Key;

struct Value {

 unsigned age;

 unsigned weight;

};

class Map  
{  
public:  
 Map();

 ~Map();

 Map(const Map& b);

 // Обменивает значения двух словарей.

 // Подумайте, зачем нужен этот метод, при наличии стандартной функции

 // std::swap.  
 void swap(Map& b);  
  
 Map& operator=(const Map& b);

 // Очищает контейнер.  
 void clear();

 // Удаляет элемент по заданному ключу.

 bool erase(const Key& k);    // для чего возвра..для     C

 // Вставка в контейнер. Возвращаемое значение - успешность вставки.

 bool insert(const Key& k, const Value& v);

 // Проверка наличия значения по заданному ключу.

 bool contains(const Key& k) const;

 // Возвращает значение по ключу. Небезопасный метод.

 // В случае отсутствия ключа в контейнера следует вставить в контейнер

 // значение, созданное конструктором по умолчанию и вернуть ссылку на него.   
 Value& operator[](const Key& k);

 // Возвращает значение по ключу. Бросает исключение при неудаче.

 Value& at(const Key& k);

 const Value& at(const Key& k) const;

 size\_t size() const;  
 bool empty() const;

 //что такое “Map == Map”? Что именно учитывать?

 friend bool operator==(const Map & a, const Map & b);  
 friend bool operator!=(const Map & a, const Map & b);

};

**Дополнительные (необязательные) задания**

* Реализуйте конструктор перемещения (нововведение стандарта С++11)
* Сделайте поддержку итераторов.
* Попробуйте сделать класс шаблонным.
* Приблизьте ваш класс к требованиям STL, доделайте недостающие методы.
* Сравните скорость работы вашего словаря со скоростью std::unordered\_map, std::map и другими аналогами. При сравнении производительности не забудьте включить компиляторные оптимизации и отключить добавление отладочной информации.