ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ОТЧЕТ
О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №1 «РЕАЛИЗАЦИЯ СЛОВАРЯ НА ОСНОВЕ AVL-ДЕРЕВА» ПО ДИСЦИПЛИНЕ «ДИСКРЕТНЫЙ АНАЛИЗ»

Выполнил(а) студент
группы М8О-208Б-23
Жгенти Дарья Никитична

Проверил:
Макаров
Н.К.

Москва, 2025

Вариант: 2

Задача:

Необходимо разработать программную библиотеку, реализующую структуру данных AVL-дерево, и на её основе — программу-словарь.

В словаре каждому ключу (регистронезависимая строка из английских букв длиной не более 256 символов) сопоставляется некоторое целое беззнаковое число (uint64_t).32-разрядные шестнадцатиричные числа).

Программа должна обрабатывать строки входного файла до его окончания.

Формат ввода и вывода: каждая строка может иметь следующий формат:

+ word 34 — добавить слово «word» с номером 34 в словарь. Программа должна вывести строку «OK», если операция прошла успешно, «Exist», если слово уже находится в словаре.

- word — удалить слово «word» из словаря. Программа должна вывести «OK», если слово существовало и было удалено, «NoSuchWord», если слово в словаре не было найдено.

word — найти в словаре слово «word». Программа должна вывести «OK: 34», если слово было найдено; число, которое следует за «OK:» — номер, присвоенный слову при добавлении. В случае, если слово в словаре не было обнаружено, нужно вывести строку «NoSuchWord».

! Save /path/to/file — сохранить словарь в бинарном компактном представлении на диск в файл, указанный парамером команды. В случае успеха, программа должна вывести «OK», в случае неудачи выполнения операции, программа должна вывести описание ошибки (см. ниже).

! Load /path/to/file — загрузить словарь из файла. Предполагается, что файл был ранее подготовлен при помощи команды Save. В случае успеха, программа должна вывести строку «OK», а загруженный словарь должен заменить текущий (с которым происходит работа); в случае неуспеха, должна быть выведена диагностика, а рабочий словарь должен остаться без изменений. Кроме системных ошибок, программа должна корректно обрабатывать случаи несовпадения формата указанного файла и представления данных словаря во внешнем файле.

Для всех операций, в случае возникновения системной ошибки (нехватка памяти, отсутсвие прав записи и т.п.), программа должна вывести строку, начинающуюся с «ERROR:» и описывающую на английском языке возникшую ошибку.

Алгоритм решения:

1. Структура AVL-дерева

AVL-дерево — это самобалансирующееся двоичное дерево поиска, в котором для каждой вершины высота левого и правого поддеревьев отличается не более чем на 1.

- Каждый узел хранит:
  - строковый ключ (в нижнем регистре),
  - значение uint64_t,
  - указатели на левое и правое поддеревья,
  - высоту поддерева.
- Все операции обеспечивают O(log n) сложность.

2. Вставка

- Ключ преобразуется к нижнему регистру (std::tolower для каждой буквы).
- Выполняется стандартная вставка в бинарное дерево поиска.
- Если ключ уже существует — выбрасывается исключение, операция отклоняется.
- После вставки производится балансировка поддерева, с использованием:
  - поворота влево,
  - поворота вправо,
  - двойного поворота.

3. Удаление

- Стандартная логика удаления из бинарного дерева:
  - Если узел имеет двух детей — замена на инфиксного преемника.
  - Затем — балансировка дерева.
- Если ключ не найден — операция отклоняется с выводом NoSuchWord.

4. Поиск

- Обычный спуск по дереву: ключ сравнивается с текущим узлом.
- Сравнение производится в нижнем регистре.
- Если найден — возвращается соответствующее значение.

5. Сохранение в файл

- Используется бинарный формат:
  - сначала записывается число узлов uint64_t,
  - затем по каждому узлу в порядке in-order обхода:
    - длина ключа uint16_t,
    - байты ключа,
    - значение uint64_t.
- Запись производится в std::ofstream в бинарном режиме.

6. Загрузка из файла

- Загружается количество узлов.
- По каждому узлу:
  - читается длина, ключ, значение.
  - добавляется во временное дерево.
- После загрузки основное дерево очищается, и подменяется новым.
- Если файл повреждён или формат неверный — выбрасывается исключение, дерево не заменяется.

Исходный код:

```
#include <iostream>
#include <string>
#include <algorithm>
#include <cctype>       // for std::tolower
#include <cstdint>
#include <fstream>
#include <stdexcept>

class AVLTree {
private:
    struct Node {
        std::string key;
        uint64_t    value;
        int         height;
        Node*       left;
        Node*       right;
        Node(const std::string& k, uint64_t v)
            : key(k), value(v), height(1), left(nullptr),
right(nullptr) {}
    };

    Node* root = nullptr;

    static std::string toLower(const std::string& s) {
        std::string res = s;
        std::transform(res.begin(), res.end(),
res.begin(),
                       [](unsigned char c){ return
std::tolower(c); });
        return res;
    }

    int height(Node* n) const {
        if (n != nullptr) return n->height;
        else              return 0;
```

```
    }

    int balanceFactor(Node* n) const {
        return height(n->right) - height(n->left);
    }

    void updateHeight(Node* n) {
        n->height = 1 + std::max(height(n->left),
height(n->right));
    }

    Node* rotateRight(Node* y) {
        Node* x  = y->left;
        Node* z = x->right;
        x->right = y;
        y->left  = z;
        updateHeight(y);
        updateHeight(x);
        return x;
    }

    Node* rotateLeft(Node* x) {
        Node* y  = x->right;
        Node* z = y->left;
        y->left  = x;
        x->right = z;
        updateHeight(x);
        updateHeight(y);
        return y;
    }

    Node* balance(Node* n) {
        updateHeight(n);
        int bf = balanceFactor(n);
        if (bf < -1) {
            if (balanceFactor(n->left) > 0)
                n->left = rotateLeft(n->left);
            return rotateRight(n);
        }
        if (bf > 1) {
            if (balanceFactor(n->right) < 0)
                n->right = rotateRight(n->right);
            return rotateLeft(n);
        }
        return n;
    }

    Node* insertNode(Node* n, const std::string& k,
```

```
uint64_t v) {
        if (!n) return new Node(k, v);
        if (k < n->key)
            n->left  = insertNode(n->left,  k, v);
        else if (k > n->key)
            n->right = insertNode(n->right, k, v);
        else
            throw std::logic_error("Exist");
        return balance(n);
    }

    Node* minValueNode(Node* n) const {
        while (n->left) n = n->left;
        return n;
    }

    Node* removeNode(Node* n, const std::string& k) {
        if (!n) throw std::logic_error("NoSuchWord");
        if (k < n->key)
            n->left = removeNode(n->left, k);
        else if (k > n->key)
            n->right = removeNode(n->right, k);
        else {
            if (!n->left || !n->right) {
                Node* t = n->left ? n->left : n->right;
                delete n;
                return t;
            }
            Node* succ = minValueNode(n->right);
            n->key   = succ->key;
            n->value = succ->value;
            n->right = removeNode(n->right, succ->key);
        }
        return balance(n);
    }

    Node* findNode(Node* n, const std::string& k) const {
        if (!n) return nullptr;
        if (k < n->key)      return findNode(n->left,  k);
        else if (k > n->key) return findNode(n->right, k);
        else                  return n;
    }

    void clear(Node* n) {
        if (!n) return;
        clear(n->left);
        clear(n->right);
        delete n;
```

```
    }

    uint64_t countNodes(Node* n) const {
        if (!n) return 0;
        return 1 + countNodes(n->left) + countNodes(n-
>right);
    }

    // In-order обход
    void inorderSave(Node* n, std::ofstream& out) const {
        if (!n) return;
        inorderSave(n->left, out);
        uint16_t len = static_cast<uint16_t>(n-
>key.size());
        out.write(reinterpret_cast<const char*>(&len),
sizeof(len));
        out.write(n->key.data(), len);
        out.write(reinterpret_cast<const char*>(&n-
>value), sizeof(n->value));
        inorderSave(n->right, out);
    }

public:
    ~AVLTree() { clear(root); }

    bool insert(const std::string& key, uint64_t value) {
        auto k = toLower(key);
        try {
            root = insertNode(root, k, value);
            return true;
        } catch (std::logic_error&) {
            return false;
        }
    }

    bool remove(const std::string& key) {
        auto k = toLower(key);
        try {
            root = removeNode(root, k);
            return true;
        } catch (std::logic_error&) {
            return false;
        }
    }

    bool find(const std::string& key, uint64_t& out) const
{
        auto k = toLower(key);
```

```
        Node* p = findNode(root, k);
        if (p) { out = p->value; return true; }
        return false;
    }

    // Сохранение дерева в бинарный файл
    void save(const std::string& path) const {
        std::ofstream out(path, std::ios::binary);
        if (!out) throw std::runtime_error("ERROR: cannot
open file for writing");
        // Сначала записываем число узлов
        uint64_t cnt = countNodes(root);
        out.write(reinterpret_cast<const char*>(&cnt),
sizeof(cnt));
        // Потом все пары (len, key, value)
        inorderSave(root, out);
        if (out.fail()) throw std::runtime_error("ERROR:
write failure");
    }

    // Загрузка дерева из бинарного файла
    void load(const std::string& path) {
        std::ifstream in(path, std::ios::binary);
        if (!in) throw std::runtime_error("ERROR: cannot
open file for reading");

        uint64_t cnt;
        in.read(reinterpret_cast<char*>(&cnt),
sizeof(cnt));
        if (!in) throw std::runtime_error("ERROR: invalid
format");

        AVLTree tmp;
        for (uint64_t i = 0; i < cnt; ++i) {
            uint16_t len;
            in.read(reinterpret_cast<char*>(&len),
sizeof(len));
            if (!in) throw std::runtime_error("ERROR:
invalid format");
            std::string key(len, '\0');
            in.read(&key[0], len);
            if (!in) throw std::runtime_error("ERROR:
invalid format");
            uint64_t val;
            in.read(reinterpret_cast<char*>(&val),
sizeof(val));
            if (!in) throw std::runtime_error("ERROR:
invalid format");
```

```
            tmp.root = tmp.insertNode(tmp.root, key, val);
        }

        clear(root);
        root = tmp.root;
        tmp.root = nullptr;
    }
};

int main() {
    AVLTree tree;
    std::string cmd;
    while (std::cin >> cmd) {
        if (cmd == "+") {
            std::string w; uint64_t v;
            std::cin >> w >> v;
            std::cout << (tree.insert(w,v) ? "OK\n" :
"Exist\n");
        }
        else if (cmd == "-") {
            std::string w;
            std::cin >> w;
            std::cout << (tree.remove(w) ? "OK\n" :
"NoSuchWord\n");
        }
        else if (cmd == "!") {
            std::string op, path;
            std::cin >> op >> path;
            try {
                if (op == "Save")      { tree.save(path);
std::cout << "OK\n"; }
                else if (op == "Load") { tree.load(path);
std::cout << "OK\n"; }
                else std::cout << "ERROR: unknown
operation\n";
            } catch (std::runtime_error& e) {
                std::cout << e.what() << "\n";
            }
        }
        else {
            uint64_t v;
            if (tree.find(cmd, v))
                std::cout << "OK: " << v << "\n";
            else
                std::cout << "NoSuchWord\n";
        }
    }
    return 0;
```

}

Тесты:

Ввод	**Вывод**
+ a 1 + A 2 + aaaaaaaaaaaaaaaaaaa... (256 a) 18446744073709551615 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa... (256 a) A - A a	OK Exist OK OK: 18446744073709551615 OK: 1 OK NoSuchWord

Вывод

Разработанная программа реализует полноценный словарь с сохранением и загрузкой на базе сбалансированного AVL-дерева.
Все команды выполняются в логарифмическое время, дерево сохраняется в компактной бинарной форме, устойчивой к ошибкам чтения.

AVL-дерево обеспечивает:

- строгий порядок хранения данных,
- балансировку высот поддеревьев,
- эффективную реализацию операций поиска, вставки и удаления,
- сохранение данных между сессиями.

Временная и пространственная сложность

Операция	**Сложность**
Вставка	O(log n)
Удаление	O(log n)
Поиск	O(log n)
Сохранение/загрузка	O(n)
Память на структуру	O(n)