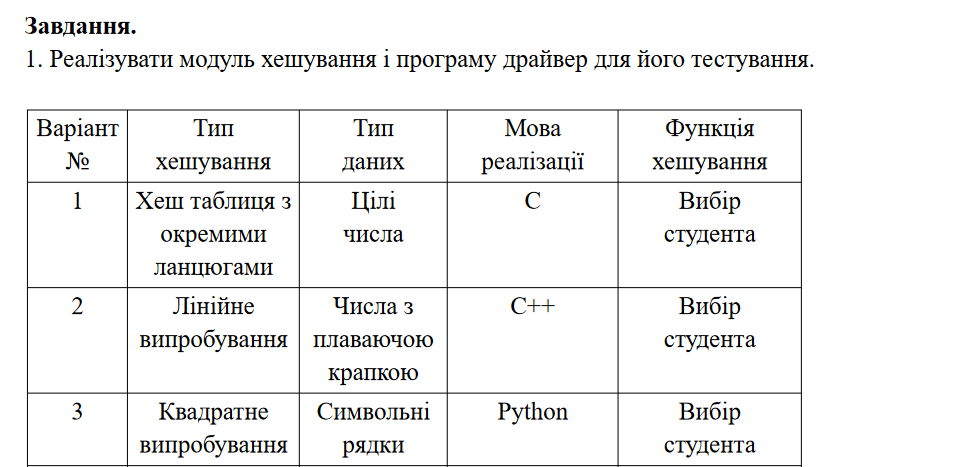
Завдання



#include <iostream>

#include <vector>

#include <string>

using namespace std;

// Функція хешування - простий приклад, можна вибрати інший варіант для оптимізації

int hashFunction(const string& key, int tableSize) {

    int hashVal = 0;

    for (char ch : key) {

        hashVal += ch;

    }

    return hashVal % tableSize;

}

class HashTable {

private:

    struct HashNode {

        string key;

        bool deleted;

        HashNode(const string& k) : key(k), deleted(false) {}

    };

    vector<HashNode\*> table;

    int size;

    int capacity;

public:

    HashTable(int capacity) : capacity(capacity), size(0) {

        table.resize(capacity, nullptr);

    }

    ~HashTable() {

        for (int i = 0; i < capacity; ++i) {

            delete table[i];

        }

    }

    void insert(const string& key) {

        if (size >= capacity) {

            cout << "Хеш-таблиця переповнена!" << endl;

            return;

        }

        int index = hashFunction(key, capacity);

        while (table[index] != nullptr && !table[index]->deleted) {

            index = (index + 1) % capacity; // квадратне випробування

        }

        if (table[index] == nullptr) {

            table[index] = new HashNode(key);

        } else {

            table[index]->key = key;

            table[index]->deleted = false;

        }

        size++;

    }

    bool search(const string& key) {

        int index = hashFunction(key, capacity);

        int startIndex = index;

        while (table[index] != nullptr) {

            if (!table[index]->deleted && table[index]->key == key) {

                return true;

            }

            index = (index + 1) % capacity; // квадратне випробування

            if (index == startIndex) {

                break; // пройшли всю таблицю

            }

        }

        return false;

    }

    void remove(const string& key) {

        int index = hashFunction(key, capacity);

        int startIndex = index;

        while (table[index] != nullptr) {

            if (!table[index]->deleted && table[index]->key == key) {

                table[index]->deleted = true;

                size--;

                return;

            }

            index = (index + 1) % capacity; // квадратне випробування

            if (index == startIndex) {

                break; // пройшли всю таблицю

            }

        }

    }

};

int main() {

    HashTable hashTable(10); // створення хеш-таблиці розміром 10

    // Додавання елементів

    hashTable.insert("apple");

    hashTable.insert("banana");

    hashTable.insert("orange");

    // Пошук елементів

    cout << "Пошук 'apple': " << (hashTable.search("apple") ? "Знайдено" : "Не знайдено") << endl;

    cout << "Пошук 'banana': " << (hashTable.search("banana") ? "Знайдено" : "Не знайдено") << endl;

    cout << "Пошук 'grape': " << (hashTable.search("grape") ? "Знайдено" : "Не знайдено") << endl;

    // Видалення елементів

    hashTable.remove("apple");

    cout << "Пошук 'apple' після видалення: " << (hashTable.search("apple") ? "Знайдено" : "Не знайдено") << endl;

    return 0;

}

