



#include <iostream>

#include <vector>

#include <list>

#include <string>

#include <chrono>

// Каб было меньш писаць, выкарыстоўваю стандратны спісак і вектар замест масіва

struct School {

int schoolNumber;

std::string directorLastName;

};

const int TABLE\_SIZE = 10; // Памер хэш-табліцы

// Функцыя ўніверсальнага хэшавання

int universalHash(int key) {

// Выбіраем адвольныя каэфіцыенты a і b

const int a = 3;

const int b = 7;

const int p = 31; // Просты лік, большы за памер табліцы

return ((a \* key + b) % p) % TABLE\_SIZE; // Вылічаем хэш з дапамогай універсальнай функцыі хэшавання

}

// Функцыя для ўстаўкі элемента ў хэш-табліцу

void insert(std::vector<std::list<School>>& table, const School& school) {

int index = universalHash(school.schoolNumber); // Вылічаем індэкс устаўкі ў табліцу

table[index].push\_back(school); // Дадаем элемент у адпаведны спіс хэш-табліцы

}

// Функцыя для пошуку элемента па ключы

std::string search(const std::vector<std::list<School>>& table, int key) {

int index = universalHash(key); // Вылічаем індэкс пошуку ў табліцы

auto start = std::chrono::high\_resolution\_clock::now(); // Пачатак засечкі часу пошуку

for (const auto& entry : table[index]) { // Праходзім па спісе элементаў па вылічаным індэксе

if (entry.schoolNumber == key) { // Калі знойдзены элемент з указаным ключом

auto end = std::chrono::high\_resolution\_clock::now(); // Канец засечкі часу пошуку

std::chrono::nanoseconds duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start); // Вылічэнне працягласці часу пошуку ў мікрасекундах

std::cout << "Пошук заняў: " << duration.count() << " нанасекунд\n"; // Выводзім працягласць часу пошуку

return entry.directorLastName; // Вяртаем прозвішча дырэктара знойдзенай школы

}

}

return "Школа не знойдзена"; // Калі школа не знойдзена, вяртаем паведамленне пра памылку

}

// Функцыя для вываду ўсёй хэш-табліцы

void printHashTable(const std::vector<std::list<School>>& table) {

for (int i = 0; i < TABLE\_SIZE; ++i) { // Праходзім па ўсіх элементах табліцы

std::cout << "Хэш" << i << ": "; // Выводзім індэкс хеша

if (!table[i].empty()) { // Калі спіс не пусты

for (const auto& entry : table[i]) { // Праходзім па ўсіх элементах спісу

std::cout << "[" << entry.schoolNumber << ", " << entry.directorLastName << "] "; // Выводзім дадзеныя элемента

}

}

else {

std::cout << "Пуста"; // Калі спіс пусты, выводны паведамленне аб пустэчы

}

std::cout << std::endl; // Пераходзім на новы радок

}

}

int main() {

// Ініцыялізуем хэш-табліцу

std::vector<std::list<School>> hashTable(TABLE\_SIZE);

// Дадаем дадзеныя аб школах у табліцу

insert(hashTable, { 8, "Янаў" });

insert(hashTable, { 18, "Пятроў" }); // Гэты элемент будзе мець той жа хэш, што і элемент з нумарам 101

insert(hashTable, { 25, "Сідараў" });

insert(hashTable, { 32, "Сідараў" });

// Выводзім усю хэш-табліцу

std::cout << "Хэш-табліца:" << std::endl;

printHashTable(hashTable);

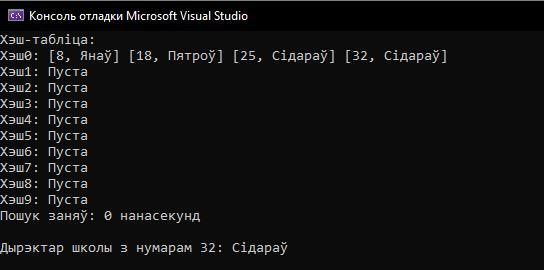
// Выконваем пошук па нумары школы

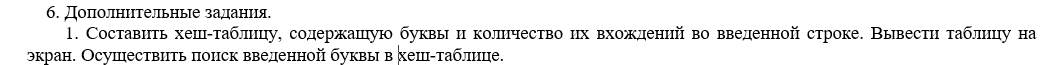
int searchKey = 32;

std::cout << "\nДырэктар школы з нумарам " << searchKey << ": " << search(hashTable, searchKey) << std::endl;

return 0;

}





#include <iostream>

#include <vector>

#include <list>

#include <string>

// Структура для захоўвання літар і іх колькасці ўваходжанняў

struct Entry {

char letter;

int count;

};

// Структура для захоўвання хэш-табліцы

struct HashTable {

std::vector<std::list<Entry>> table; // Захоўванне элементаў хэш-табліцы ў выглядзе спісу спісаў

int size; // Памер хэш-табліцы

// Канструктар для стварэння хэш-табліцы з зададзеным памерам

HashTable(int size) : size(size) {

table.resize(size); // Змяняем памер вектара спісаў

}

// Функцыя для дадання элемента ў хэш-табліцу

void insert(char letter) {

int index = hashFunction(letter); // Вылічаем індэкс хеша для літары

for (auto& entry : table[index]) { // Правяраем, ці ёсць ужо такая літара ў табліцы

if (entry.letter == letter) { // Калі літара ўжо ёсць

entry.count++; // Павялічваем колькасць уваходжанняў

return; // Завяршаем функцыю

}

}

// Калі літары няма ў табліцы, дадаем новы элемент

Entry newEntry = { letter, 1 }; // Ствараем новы запіс з літарай і адным уваходжаннем

table[index].push\_back(newEntry); // Дадаем элемент у спіс

}

// Функцыя для пошуку літары ў хэш-табліцы

int search(char letter) {

int index = hashFunction(letter); // Вылічаем індэкс хеша для літары

for (const auto& entry : table[index]) { // Праходзім па спісе элементаў па вылічаным індэксе

if (entry.letter == letter) { // Калі знойдзена літара

return entry.count; // Вяртаем колькасць уваходжанняў

}

}

return 0; // Калі літара не знойдзена, вяртаем 0

}

// Простая хэш-функцыя для літар

int hashFunction(char letter) {

return letter % size; // Простая хэш-функцыя, рэшту ад дзялення знака на памер табліцы

}

};

int main() {

std::string input;

std::cout << "Увядзіце радок: ";

std::getline(std::cin, input); // Уводзім радок

HashTable hashTable(26); // Ствараем хэш-табліцу памерам 26 для захоўвання літар

// Запаўняем хэш-табліцу літарамі з уведзенага радка

for (char c : input) {

if (std::isalpha(c)) { // Правяраем, ці з'яўляецца сімвал літарай

c = std::tolower(c); // Прыводзім літару да ніжняга рэгістра

hashTable.insert(c); // Дадаем літару ў хэш-табліцу

}

}

// Выводзім на экран змесціва хэш-табліцы

std::cout << "Хэш-табліца:" << std::endl;

for (int i = 0; i < 26; ++i) {

std::cout << static\_cast<char>('a' + i) << ": " << hashTable.search('a' + i) << std::endl;

}

// Пошук літары ў хэш-табліцы

char searchLetter;

std::cout << "Увядзіце літару для пошуку: ";

std::cin >> searchLetter; // Уводзім літару для пошуку

int count = hashTable.search(std::tolower(searchLetter)); // Шукаем літару ў хэш-табліцы (прыводзім да ніжняга рэгістра для пошуку)

if (count > 0) {

std::cout << "Літара '" << searchLetter << "' знойдзена. Колькасць уваходжанняў: " << count << std::endl;

}

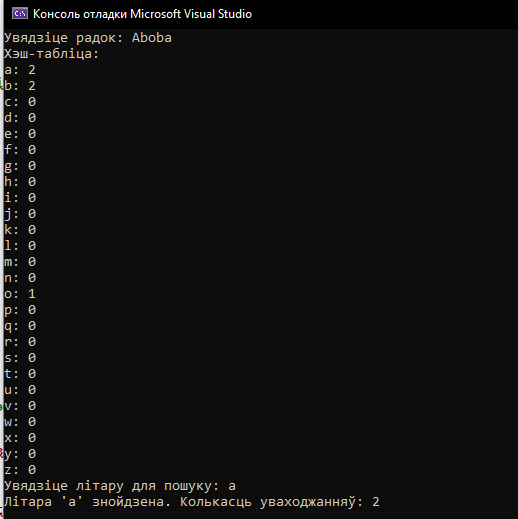
else {

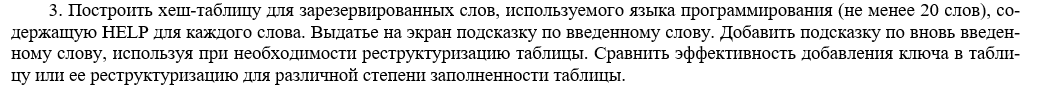
std::cout << "Літара '" << searchLetter << "' не знойдзена." << std::endl;

}

return 0;

}





#include <iostream>

#include <vector>

#include <list>

#include <string>

#include <Windows.h>

// Структура для захоўвання пар ключ-значэнне (слова - падказка)

struct Entry {

std::string word;

std::string help;

};

// Структура для захоўвання хэш-табліцы

struct HashTable {

std::vector<std::list<Entry>> table; // Захоўванне элементаў хэш-табліцы ў выглядзе спісу спісаў

int size; // Памер хэш-табліцы

int count; // Колькасць элементаў у табліцы

// Канструктар для стварэння хэш-табліцы з зададзеным памерам

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

table.resize(size); // Змяняем памер вектара спісаў

}

// Функцыя для дадання элемента ў хэш-табліцу

void insert(const std::string& word, const std::string& help) {

int index = hashFunction(word); // Вылічаем індэкс хеша для слова

for (auto& entry : table[index]) { // Правяраем, ці ёсць ужо такое слова ў табліцы

if (entry.word == word) { // Калі слова ўжо ёсць

entry.help = help; // Абнаўляем падказку

return; // Завяршаем функцыю

}

}

// Калі слова няма ў табліцы, дадаем новы элемент

Entry newEntry = { word, help }; // Ствараем новы запіс са словам і падказкай

table[index].push\_back(newEntry); // Дадаем элемент у спіс

count++; // Павялічваем колькасць элементаў у табліцы

// Правяраем, калі каэфіцыент запоўненасці табліцы перавышае 75%, то перабудоўваем табліцу

if (static\_cast<double>(count) / size > 0.75) {

rehash();

}

}

// Функцыя для пошуку падказкі па слове ў хэш-табліцы

std::string search(const std::string& word) {

int index = hashFunction(word); // Вылічаем індэкс хеша для слова

for (const auto& entry : table[index]) { // Праходзім па спісе элементаў па вылічаным індэксе

if (entry.word == word) { // Калі знойдзена слова

return entry.help; // Вяртаем падказку

}

}

return "Слова не знойдзена"; // Калі слова не знойдзена, вяртаем паведамленне пра памылку

}

// Функцыя для перабудовы табліцы (павелічэнне памеру ўдвая і перахэшавання элементаў)

void rehash() {

size \*= 2; // Павялічваем памер табліцы ўдвая

std::vector<std::list<Entry>> newTable(size); // Ствараем новую табліцу з павялічаным памерам

// Перехешируем элементы са старой табліцы ў новую

for (const auto& list : table) {

for (const auto& entry : list) {

int index = hashFunction(entry.word); // Вылічаем новы індэкс хеша для слова

newTable[index].push\_back(entry); // Дадаем элемент у спіс новай табліцы

}

}

table = std::move(newTable); // Замяняем старую табліцу новай

}

// Простая хэш-функцыя для слоў

int hashFunction(const std::string& word) {

int sum = 0;

for (char c : word) {

sum += static\_cast<int>(c); // Сумуем коды сімвалаў слова

}

return sum % size; // Простая хэш-функцыя, рэшту ад дзялення сумы кодаў сімвалаў на памер табліцы

}

};

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

// Ствараем хэш-табліцу для зарэзерваваных слоў

HashTable reservedWords(20);

// Запаўняем табліцу зарэзерваванымі словамі і іх падказкамі

reservedWords.insert("int", "HELP: Вызначэнне цэлалікавай зменнай");

reservedWords.insert("if", "HELP: Умоўны аператар");

reservedWords.insert("else", "HELP: Галінаванне ў выпадку ілжывай умовы");

reservedWords.insert("for", "HELP: Цыкл з прадумовай");

reservedWords.insert("while", "HELP: Цыкл з паставай");

reservedWords.insert("switch", "HELP: Множны выбар");

reservedWords.insert("case", "HELP: Галінка ўнутры аператара switch");

reservedWords.insert("break", "HELP: Перапыненне цыкла або аператара switch");

reservedWords.insert("continue", "HELP: Працяг выканання цыкла");

reservedWords.insert("return", "HELP: Зварот значэння з функцыі");

reservedWords.insert("void", "HELP: Вызначэнне функцыі без якое вяртаецца значэння");

reservedWords.insert("include", "HELP: Уключэнне загалоўкавых файлаў");

reservedWords.insert("namespace", "HELP: Вызначэнне вобласці імёнаў");

reservedWords.insert("using", "HELP: Імпарт азначэнняў з прасторы імёнаў");

reservedWords.insert("const", "HELP: Вызначэнне канстанты");

reservedWords.insert("static", "HELP: Выкарыстанне статычнага мадыфікатара");

reservedWords.insert("class", "HELP: Вызначэнне класа");

reservedWords.insert("struct", "HELP: Вызначэнне структуры");

reservedWords.insert("public", "HELP: Вобласць бачнасці для ўсіх");

reservedWords.insert("private", "HELP: Вобласць бачнасці толькі для класа");

// Выводзім меню для карыстальніка

std::cout << "Даступныя каманды:\n";

std::cout << "1. Пошук падказкі па слове\n";

std::cout << "2. Даданне новай падказкі\n";

std::cout << "0. Выхад\n";

int choice;

std::string inputWord, inputHelp;

while (true) {

std::cout << "\nУвядзіце нумар каманды: ";

std::cin >> choice;

if (choice == 0) {

break; // Выхад з праграмы

}

else if (choice == 1) {

std::cout << "Увядзіце слова для пошуку падказкі: ";

std::cin >> inputWord;

std::cout << reservedWords.search(inputWord) << std::endl;

}

else if (choice == 2) {

std::cout << "Увядзіце новае слова:";

std::cin >> inputWord;

std::cout << "Увядзіце падказку для гэтага слова: ";

std::cin.ignore(); // Чысцім буфер пасля ўводу ліку

std::getline(std::cin, inputHelp);

reservedWords.insert(inputWord, inputHelp); // Дадаем новае слова і падказку ў табліцу

}

else {

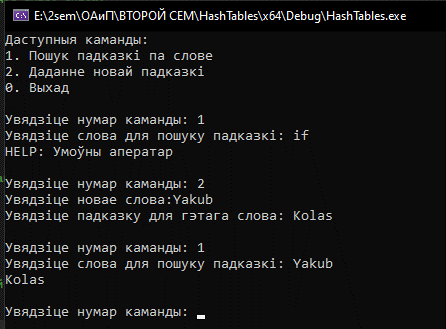
std::cout << "Няслушны нумар каманды. Паспрабуйце зноў." << std::endl;

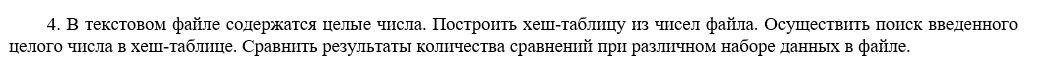
}

}

return 0;

}





#include <iostream>

#include <fstream>

#include <vector>

#include <list>

#include <string>

// Структура для захоўвання пар ключ-значэнне (лік - статут)

struct Entry {

int number;

std::string status;

};

// Структура для захоўвання хэш-табліцы

struct HashTable {

std::vector<std::list<Entry>> table; // Захоўванне элементаў хэш-табліцы ў выглядзе спісу спісаў

int size; // Памер хэш-табліцы

int count; // Колькасць элементаў у табліцы

// Канструктар для стварэння хэш-табліцы з зададзеным памерам

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

table.resize(size); // Змяняем памер вектара спісаў

}

// Функцыя для дадання элемента ў хэш-табліцу

void insert(int number) {

int index = hashFunction(number); // Вылічаем індэкс хеша для ліку

for (auto& entry : table[index]) { // Правяраем, ці ёсць ужо такі лік у табліцы

if (entry.number == number) { // Калі лік ужо ёсць

return; // Завяршаем функцыю

}

}

// Калі лікі няма ў табліцы, дадаем новы элемент

Entry newEntry = { number, "FOUND" }; // Ствараем новы запіс з лікам і статусам "FOUND"

table[index].push\_back(newEntry); // Дадаем элемент у спіс

count++; // Павялічваем колькасць элементаў у табліцы

}

// Функцыя для пошуку колькасці ў хэш-табліцы

int search(int number, int& comparisons) {

int index = hashFunction(number); // Вылічаем індэкс хеша для ліку

comparisons = 0; // Лічыльнік параўнанняў

for (const auto& entry : table[index]) { // Праходзім па спісе элементаў па вылічаным індэксе

comparisons++; // Павялічваем лічыльнік параўнанняў

if (entry.number == number) { // Калі знойдзены лік

return comparisons; // Вяртаем колькасць параўнанняў

}

}

return comparisons; // Калі колькасць не знойдзена, вяртаем колькасць параўнанняў

}

// Простая хэш-функцыя для лікаў

int hashFunction(int number) {

return number % size; // Простая хэш-функцыя, рэшту ад дзялення ліку на памер табліцы

}

};

int main() {

// Ствараем хэш-табліцу для лікаў

HashTable numbersTable(100); // Мяркуем, што файл змяшчае не больш за 100 лікаў

// Счытваем лікі з файла і дадаем іх у хэш-табліцу

std::ifstream inputFile("numbers.txt");

if (!inputFile.is\_open()) {

std::cerr << "Памылка адкрыцця файла!" << std::endl;

return 1;

}

int number;

while (inputFile >> number) {

numbersTable.insert(number); // Дадаем лік у хэш-табліцу

}

inputFile.close();

// Пошук колькасці ў хэш-табліцы

int searchNumber;

std::cout << "Увядзіце цэлы лік для пошуку: ";

std::cin >> searchNumber;

int comparisons;

int result = numbersTable.search(searchNumber, comparisons);

if (result != 0) {

std::cout << "Лічба знойдзена. Колькасць параўнанняў: " << comparisons << std::endl;

}

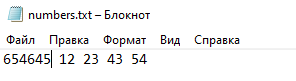
else {

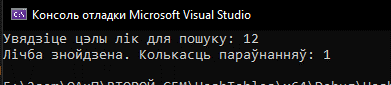
std::cout << "Лік не знойдзены." << std::endl;

}

return 0;

}





Вывад заўседы будзе 1, бо хэш функцыя вылічвае індекс, а лікі у спісах будуць аднолькавыя(у выпадку паўтарэння), таму першы лік у спісе і будзе роўны шуканаму