ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Лабораторная работа

Хэш таблицы

Вариант №20

Выполнил студент группы РИС-23-3Б

Блинов А. Е.

Проверила доцент кафедры ИТАС

О. А. Полякова

2024 г.

**Постановка задача**

1. Создать динамический массив из записей (в соответствии с вариантом), содержащий не менее 100 элементов. Для заполнения элементов массива использовать ДСЧ.

2. Выполнить поиск элемента в массиве по ключу в соответствии с вариантом. Для поиска использовать хэш-таблицу.

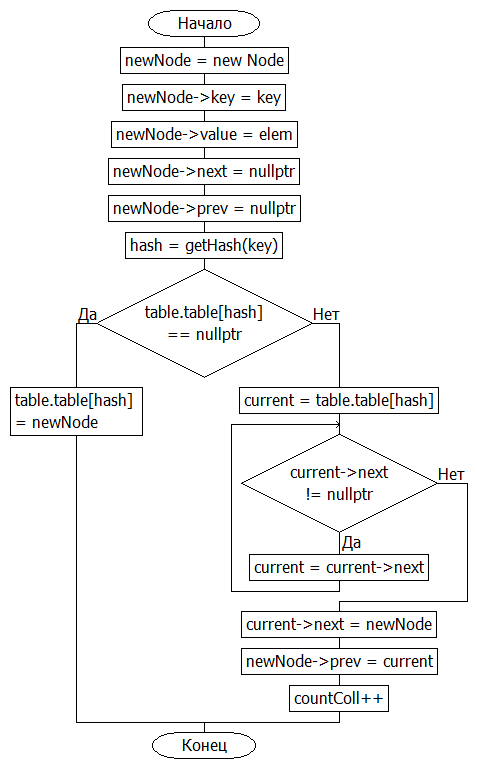
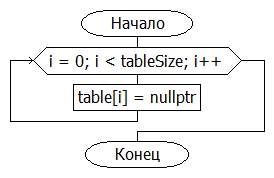
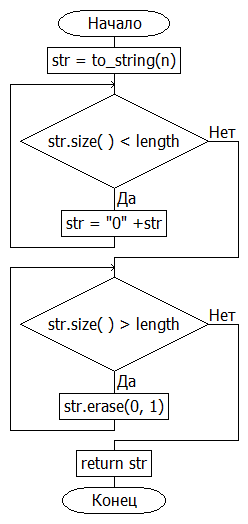
3. Подсчитать количество коллизий при размере хэш-таблицы 40, 75 и 90 элементов.

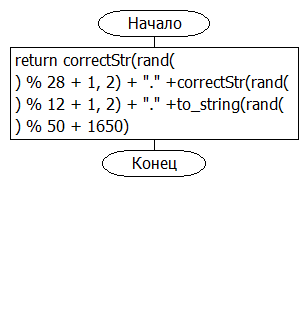
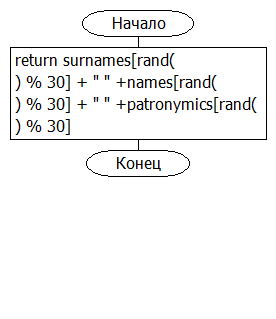
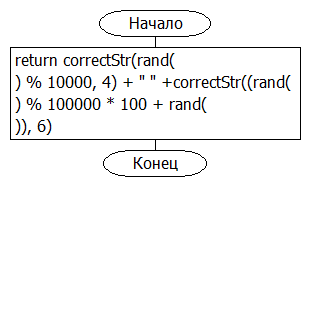
**Данные**: ФИО, дата\_рождения, №паспорта

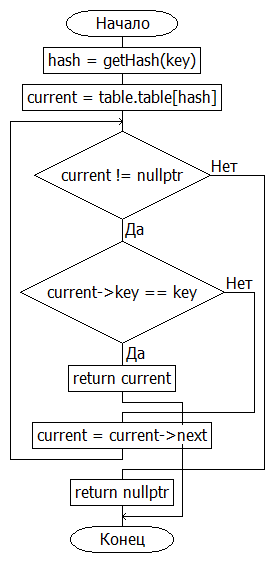
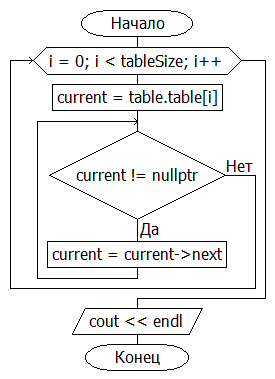
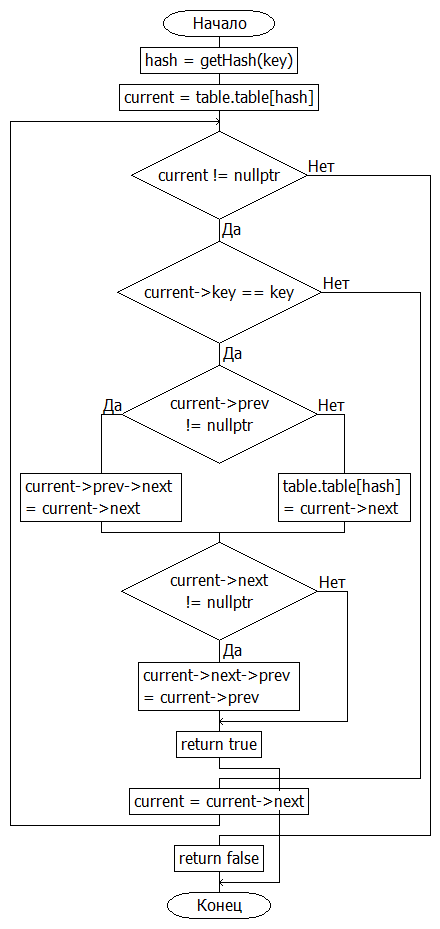
**Ключ (string):** дата\_рождения,

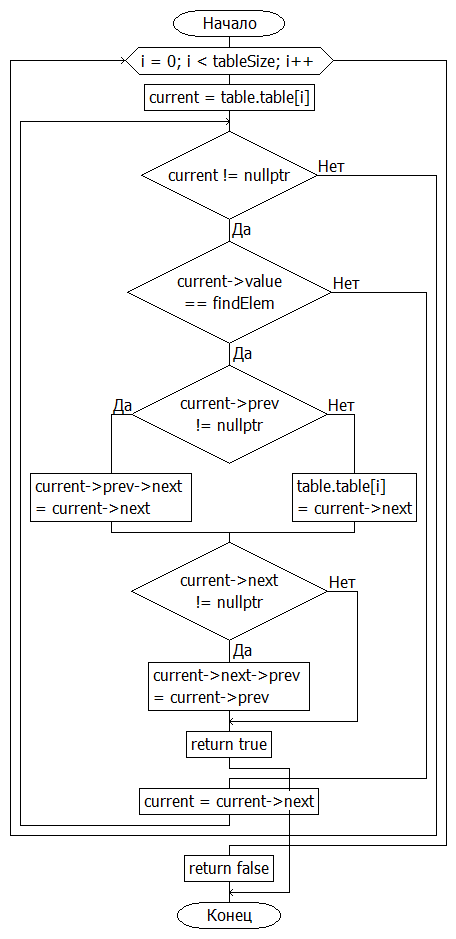
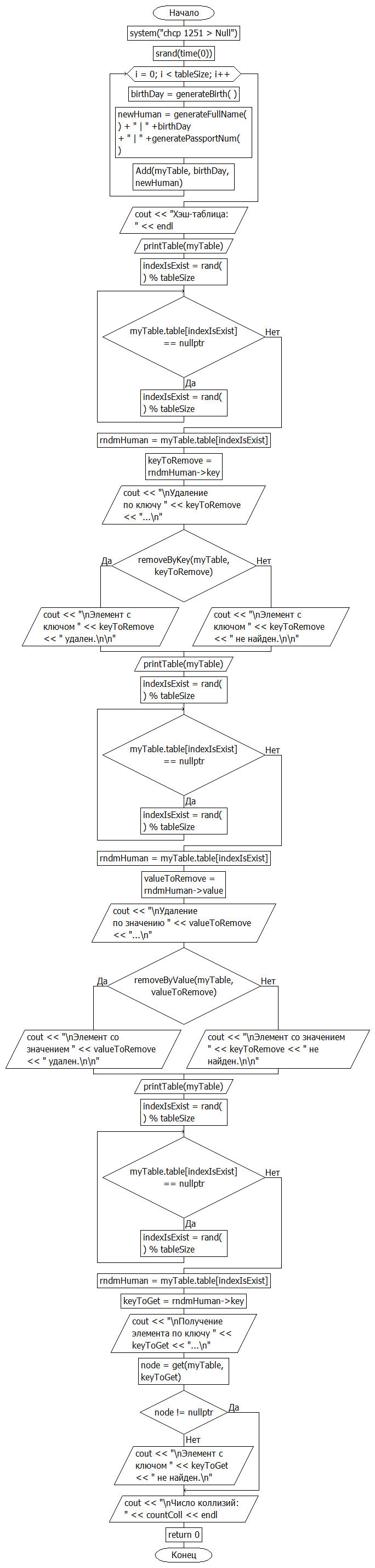
**Хэш-функция:** H(k)= [M (kAmod1)], 0<A<1, mod1 – получение дробной части, [] – получение целой части

**Блок-схема**

**** **** 

** ** ****

** **

**Код на языке C++**

#include <iostream>

#include <string>

#include <ctime>

#include <cmath>

#define \_USE\_MATH\_DEFINES

#include <cmath>

using namespace std;

const int tableSize = 150;

const double A = 1.44269504088896340736; // коэффициент для формулы

int countColl = 0;

struct Node {

string key = "", value = "";

Node\* next = nullptr;

Node\* prev = nullptr;

};

struct HashTable {

Node\* table[tableSize];

HashTable() {

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

table[i] = nullptr;

}

}

};

double modForFunc(double k) {

int intPart = static\_cast<int>(k);

return k - intPart;

}

int getHash(double k) {

return static\_cast<int>(tableSize \* modForFunc(k \* A));

}

int getHash(string line) {

int n = 0;

for (int i = 0; i < line.size(); i++) {

n += static\_cast<int>(pow(line[i], 2) \* 0.636619772367581343076 + abs(line[i]) \* 0.318309886183790671538);

}

return getHash(abs(n));

}

void Add(HashTable& table, string key, string elem) {

Node\* newNode = new Node;

newNode->key = key;

newNode->value = elem;

newNode->next = nullptr;

newNode->prev = nullptr;

int hash = getHash(key);

if (table.table[hash] == nullptr) {

table.table[hash] = newNode;

}

else {

Node\* current = table.table[hash];

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

newNode->prev = current;

countColl++;

}

}

bool removeByKey(HashTable& table, string key) {

int hash = getHash(key);

Node\* current = table.table[hash];

while (current != nullptr) {

if (current->key == key) {

if (current->prev != nullptr) {

current->prev->next = current->next;

}

else {

table.table[hash] = current->next;

}

if (current->next != nullptr) {

current->next->prev = current->prev;

}

delete current;

return true;

}

current = current->next;

}

return false;

}

bool removeByValue(HashTable& table, string findElem) {

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

Node\* current = table.table[i];

while (current != nullptr) {

if (current->value == findElem) {

if (current->prev != nullptr) {

current->prev->next = current->next;

}

else {

table.table[i] = current->next;

}

if (current->next != nullptr) {

current->next->prev = current->prev;

}

delete current;

return true;

}

current = current->next;

}

}

return false;

}

Node\* get(HashTable& table, string key) {

int hash = getHash(key);

Node\* current = table.table[hash];

while (current != nullptr) {

if (current->key == key) {

return current;

}

current = current->next;

}

return nullptr;

}

void printTable(HashTable& table) {

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

Node\* current = table.table[i];

while (current != nullptr) {

cout << "[" << current->key << ": " << current->value << "]\n";

current = current->next;

}

}

cout << endl;

}

string surnames[] = {

"Lee", "Zhang", "Wang", "Kim", "Hwang", "Park", "Nguyen", "Huang", "Zhou", "Garcia",

"Miyagi", "Bhatt", "Munoz", "Hernandez", "Gonzalez", "Roberto", "Rodriguez", "Brown", "Smith", "Johnson",

"Williams", "Jackson", "Thompson", "Bell", "Davis", "Moore", "Anderson", "Thomas", "Perez", "Carter"

};

string names[] = {

"Aiden", "Sophia", "Ethan", "Isabella", "Mason", "Olivia", "Noah", "Ava", "Liam", "Emma",

"Caden", "Madison", "Grayson", "Aria", "Zoe", "Elijah", "Charlotte", "Carter", "Amelia", "Harper",

"Lucas", "Avery", "Mia", "Logan", "Evelyn", "Jackson", "Luna", "Daniel", "Scarlett", "Layla"

};

string patronymics[] = {

"Иванович", "Петрович", "Александрович", "Сергеевич", "Дмитриевич", "Николаевич", "Андреевич", "Егорович", "Максимович", "Артемович",

"Семенович", "Владимирович", "Степанович", "Алексеевич", "Юрьевич", "Федорович", "Леонидович", "Игнатьевич", "Борисович", "Григорьевич",

"Олегович", "Павлович", "Константинович", "Михайлович", "Романович", "Станиславович", "Денисович", "Аркадьевич", "Тимофеевич", "Евгеньевич"

};

string generateFullName() {

return surnames[rand() % 30] + " " + names[rand() % 30] + " " + patronymics[rand() % 30];

}

string correctStr(int n, int length) {

string str = to\_string(n);

while (str.size() < length) {

str = "0" + str;

}

while (str.size() > length) {

str.erase(0, 1);

}

return str;

}

string generateBirth() {

return correctStr(rand() % 28 + 1, 2) + "." + correctStr(rand() % 12 + 1, 2) + "." + to\_string(rand() % 50 + 1650);

}

string generatePassportNum() {

return correctStr(rand() % 10000, 4) + " " + correctStr((rand() % 100000 \* 100 + rand()), 6);

}

int main() {

system("chcp 1251 > Null");

srand(time(0));

HashTable myTable;

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

string birthDay = generateBirth();

string newHuman = generateFullName() + " | " + birthDay + " | " + generatePassportNum();

Add(myTable, birthDay, newHuman);

}

cout << "Хэш-таблица: " << endl;

printTable(myTable);

int indexIsExist = rand() % tableSize;

while (myTable.table[indexIsExist] == nullptr) {

indexIsExist = rand() % tableSize;

}

Node\* rndmHuman = myTable.table[indexIsExist];

string keyToRemove = rndmHuman->key;

cout << "\nУдаление по ключу " << keyToRemove << "...\n";

if (removeByKey(myTable, keyToRemove)) {

cout << "\nЭлемент с ключом " << keyToRemove << " удален.\n\n";

}

else {

cout << "\nЭлемент с ключом " << keyToRemove << " не найден.\n\n";

}

printTable(myTable);

indexIsExist = rand() % tableSize;

while (myTable.table[indexIsExist] == nullptr) {

indexIsExist = rand() % tableSize;

}

rndmHuman = myTable.table[indexIsExist];

string valueToRemove = rndmHuman->value;

cout << "\nУдаление по значению " << valueToRemove << "...\n";

if (removeByValue(myTable, valueToRemove)) {

cout << "\nЭлемент со значением " << valueToRemove << " удален.\n\n";

}

else {

cout << "\nЭлемент со значением " << keyToRemove << " не найден.\n\n";

}

printTable(myTable);

indexIsExist = rand() % tableSize;

while (myTable.table[indexIsExist] == nullptr) {

indexIsExist = rand() % tableSize;

}

rndmHuman = myTable.table[indexIsExist];

string keyToGet = rndmHuman->key;

cout << "\nПолучение элемента по ключу " << keyToGet << "...\n";

Node\* node = get(myTable, keyToGet);

if (node != nullptr) {

cout << "\nНайден элемент: " << node->value << endl;

}

else {

cout << "\nЭлемент с ключом " << keyToGet << " не найден.\n";

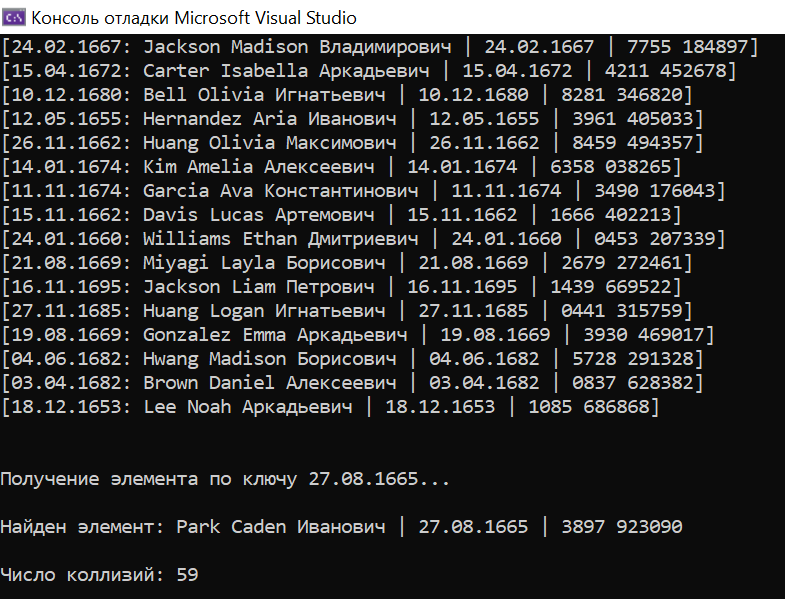
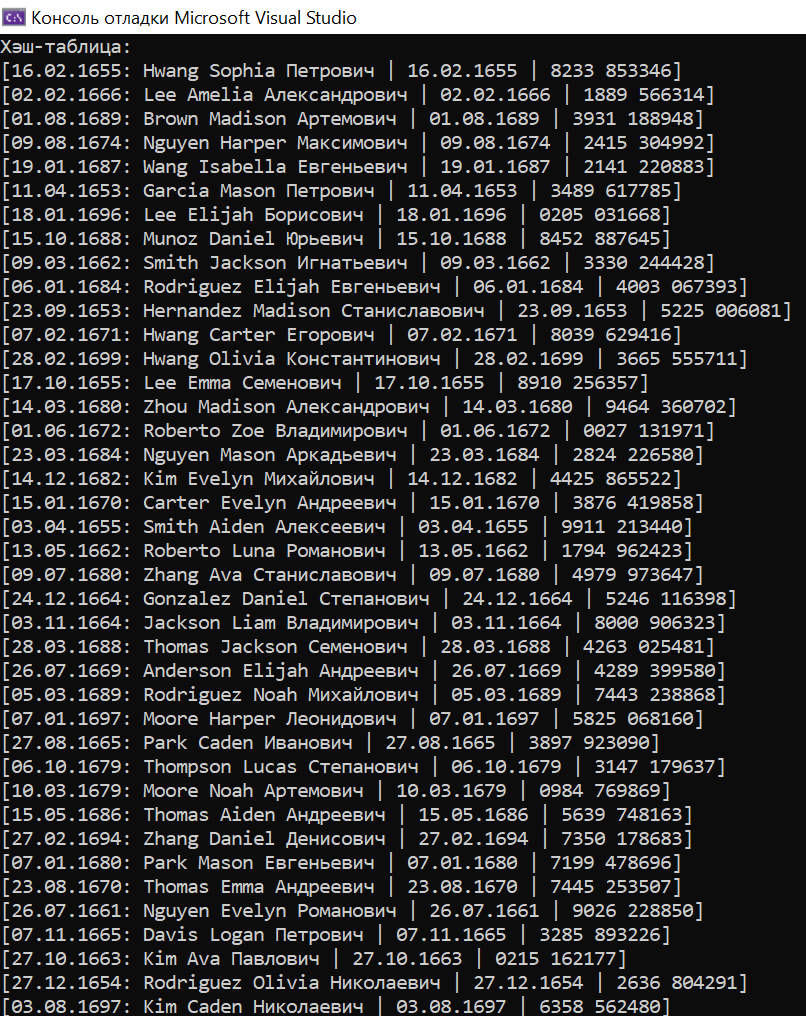
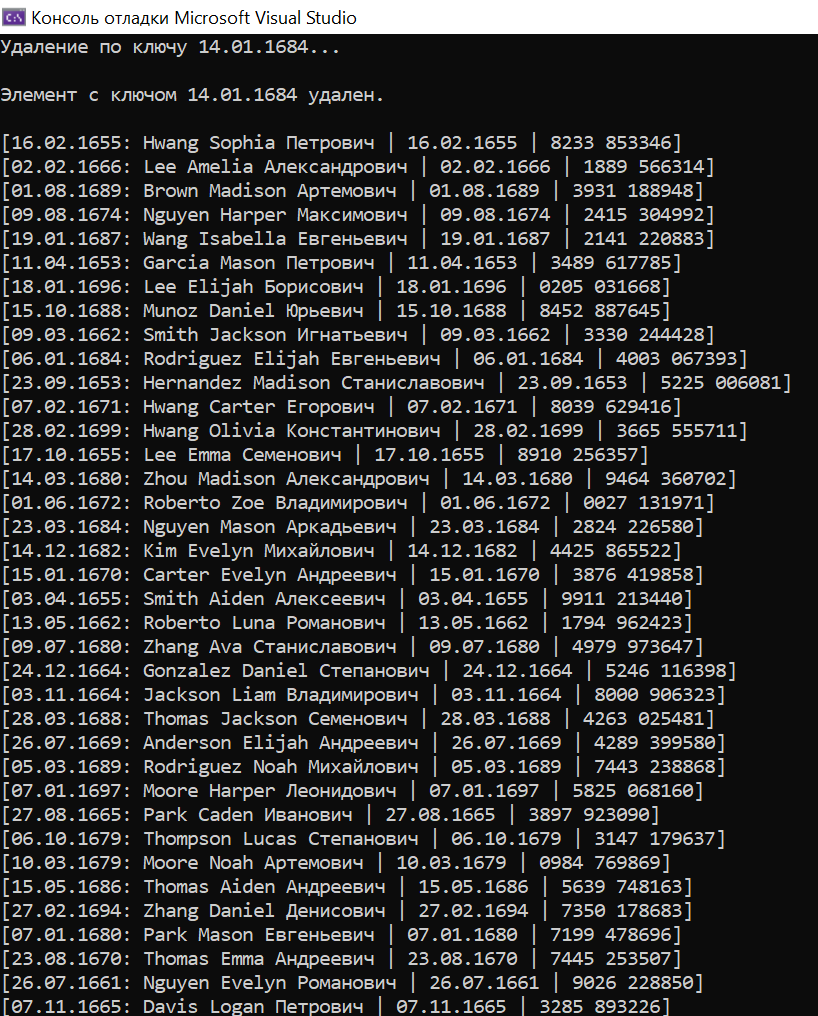
}

cout << "\nЧисло коллизий: " << countColl << endl;

return 0;

}

**Результаты работы программы**

****

**Выводы**

Программа успешно работает

GitHub: