Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа**

Выполнил:   
студент группы РИС-23-1б   
Молодых Никита Андреевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

Пермь, 2024 г.

**«HashTable**.**»**

**Условие:**

**Код на языке C++:**

#include <iostream>

#include <string>

#define \_USE\_MATH\_DEFINES

#include <math.h>

using namespace std;

#include <ctime>

const int M = 50;

const double A = M\_PI\_4;

int countercol = 0;

double mod1(double k) {

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

return k - intPart;

}

struct Node {

string key = "";

string value = "";

Node\* next = nullptr;

Node\* prev = nullptr;

};

struct HashTable {

Node\* table[M];

HashTable() {

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

table[i] = nullptr;

}

}

};

int getHash(double k) {

return static\_cast<int>(M \* mod1(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) \* M\_2\_SQRTPI + abs(line[i]) \* M\_SQRT1\_2);

}

return getHash(abs(n));

}

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 print(HashTable& table) {

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

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

while (current != nullptr) {

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

current = current->next;

}

}

}

bool 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;

return true;

}

else {

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

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

current->prev = current;

countercol++;

}

}

bool removeKey(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->prev->next = current->prev;

}

delete current;

return true;

}

current = current->next;

}

return false;

}

bool removeValue(HashTable& table, string elem) {

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

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

while (current != nullptr) {

if (current->value == elem) {

if (current->prev != nullptr) {

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

}

else {

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

}

if (current->next != nullptr) {

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

}

delete current;

return true;

}

current = current->next;

}

}

return false;

}

string Familia[]{

"Иванов", "Петров", "Сидоров", "Козлов", "Смирнов",

"Кузнецов", "Васильев", "Попов", "Алексеев", "Лебедев",

"Соколов", "Козлов", "Новиков", "Морозов", "Павлов",

"Волков", "Богданов", "Воронин", "Зайцев", "Фролов",

"Никитин", "Соловьев", "Мельников", "Калашников", "Орлов",

"Киселев", "Макаров", "Андреев", "Захаров", "Григорьев",

"Ильин", "Романов", "Максимов", "Быков", "Трофимов",

"Тимофеев", "Никифоров", "Савельев", "Панов", "Ларионов",

"Федоров", "Тихонов", "Устинов", "Львов", "Афанасьев",

"Гаврилов", "Платонов", "Семенов", "Цветков", "Исаев"

};

string Imya[] = {

"Алексей", "Мария", "Иван", "Екатерина", "Дмитрий",

"Анна", "Владимир", "Ольга", "Павел", "Наталья",

"Сергей", "Елена", "Антон", "София", "Артем",

"Анастасия", "Михаил", "Кристина", "Александр", "Татьяна",

"Николай", "Алиса", "Денис", "Виктория", "Василий",

"Юлия", "Станислав", "Людмила", "Артур", "Евгения",

"Роман", "Вероника", "Максим", "Евгений", "Валентина",

"Игорь", "Дарья", "Григорий", "Лариса", "Владислав",

"Олег", "Елизавета", "Даниил", "Марина", "Арсений",

"Полина", "Константин", "Милана", "Тимур", "Лилия"

};

string Otchestvo[] = {

"Алексеевич", "Ивановна", "Дмитриевич", "Петровна", "Сергеевич",

"Владимировна", "Андреевич", "Михайловна", "Николаевич", "Артемовна",

"Павлович", "Григорьевна", "Викторович", "Александровна", "Степанович",

"Игоревна", "Федорович", "Валерьевна", "Юрьевич", "Васильевна",

"Максимович", "Анатольевна", "Тимофеевич",

"Борисовна", "Семенович",

"Владиславовна", "Олегович", "Юрьевна", "Артурович", "Станиславовна",

"Ильич", "Валентиновна", "Арсеньевич", "Геннадьевна", "Аркадьевич",

"Сергеевна", "Данилович", "Леонидовна", "Витальевич", "Георгиевна",

"Афанасьевич", "Антоновна", "Тихонович", "Евгеньевна", "Кириллович",

"Александрович", "Эдуардовна", "Валентинович", "Аркадьевна", "Савельевич"

};

string generateFullName() {

return Familia[rand() % 50] + ' ' + Imya[rand() % 50] + ' ' + Otchestvo[rand() % 50];

}

string correctStr(int n, int lenght) {

string strn = to\_string(n);

while (strn.size() < lenght) {

strn = '0' + strn;

}

while (strn.size() > lenght) {

strn.erase(0, 1);

}

return strn;

}

string generateBR() {

return correctStr(rand() % 28 + 1, 2) + '.' + correctStr(rand() % 12 + 1, 2) + '.' + to\_string(rand() % 74 + 1950);

}

string generatePassportNum() {

return correctStr(rand() % 10000, 4) + ' ' + correctStr((rand() % 1000000 \* 100 + rand()), 6);

}

int main() {

system("chcp 1251>NULL");

system("cls");

srand(time(NULL));

HashTable table;

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

string birthDay = generateBR();

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

add(table, birthDay, newHuman);

}

cout << "Хеш таблица: " << endl;

print(table);

int Ind = rand() % M;

while (table.table[Ind] == nullptr) {

Ind = rand() % M;

}

Node\* random = table.table[Ind];

string Key = random->key;

cout << "Удаление по key: " << Key << " : " << endl;

if (removeKey(table, Key)) {

cout << "Элемент " << Key << " удалён" << endl;

}

else {

cout << "Элемент не найден" << endl;

}

print(table);

Ind = rand() % M;

while (table.table[Ind] == nullptr) {

Ind = rand() % M;

}

random = table.table[Ind];

string value = random->value;

cout << "Удаление по value: " << value << " : " << endl;

if (removeValue(table, value)) {

cout << "Элемент " << value << " удалён" << endl;

}

else {

cout << "Элемент не найден" << endl;

}

print(table);

Ind = rand() % M;

while (table.table[Ind] == nullptr) {

Ind = rand() % M;

}

random = table.table[Ind];

string KeytoGet = random->key;

cout << "Получение элемента: " << KeytoGet << " : " << endl;

Node\* node = get(table, KeytoGet);

if (node != nullptr) {

cout << "Найден элемент: " << node->value << endl;

}

else {

cout << "Элемент не найден" << endl;

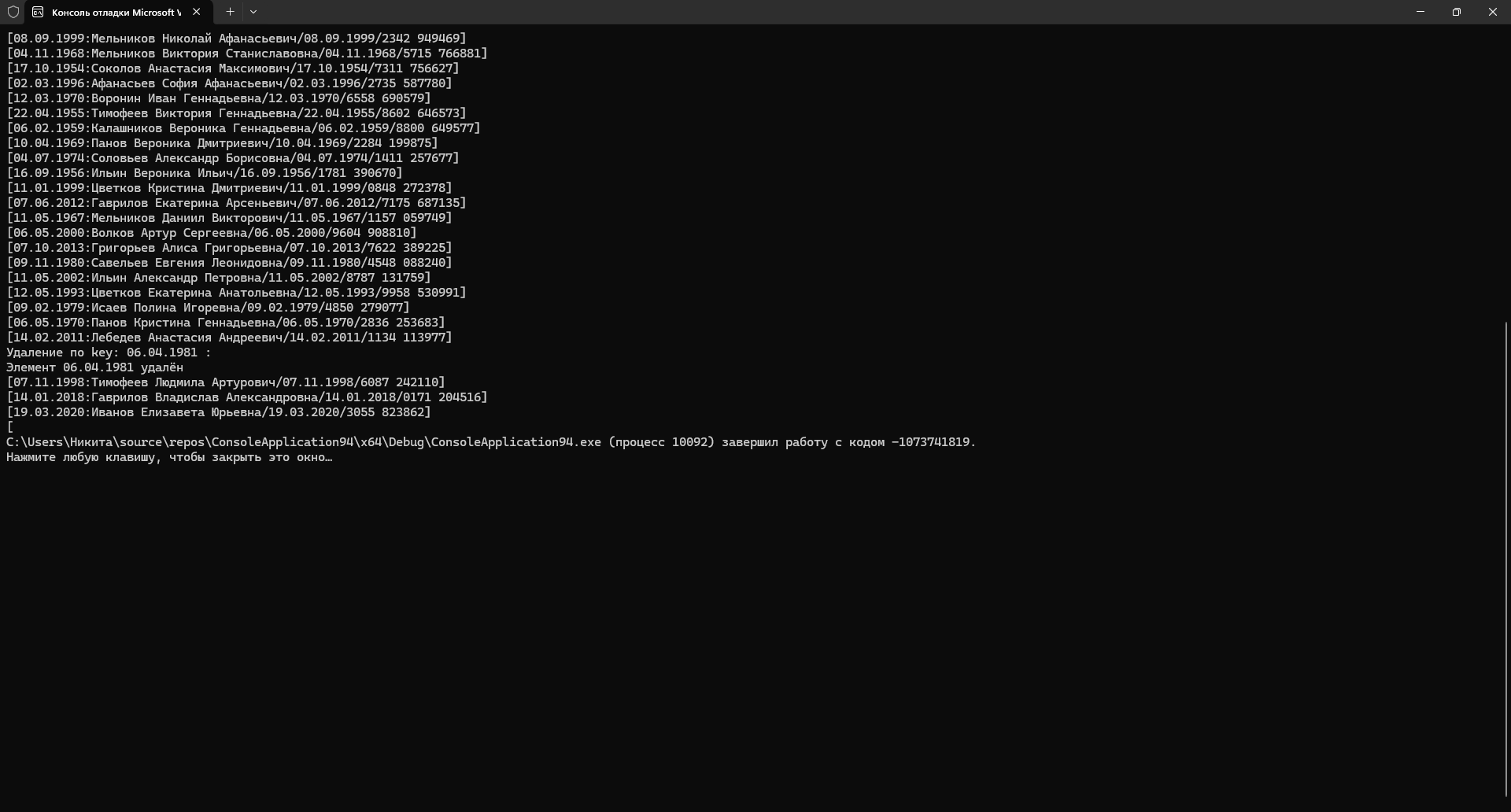
}

cout << "Число коллизий: " << countercol << endl;

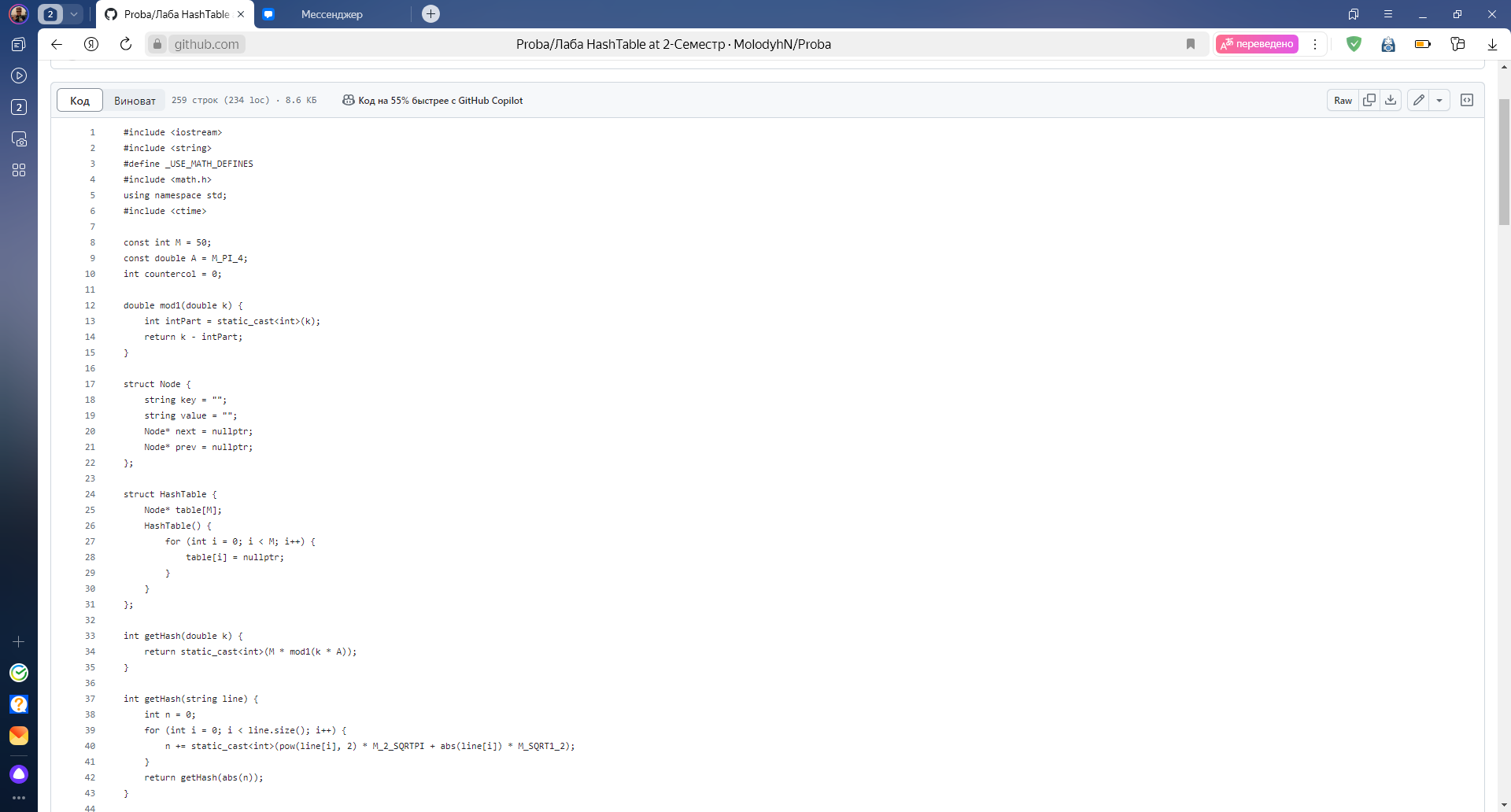
return 0;

}

**Работа программы:**

****

**Cкрины из гита:**

****

<https://github.com/MolodyhN/Proba>

**Вывод:** Задача была выполнена. Всё получилось.