Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №17**

Дисциплина: основы алгоритмизации и программирования

Тема: Хеш-функции

Вариант 24

Выполнила:

Студентка группы ИВТ-20-2б

Ананина Арина Юрьевна

Проверил:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь, 2020**

1. Постановка задачи

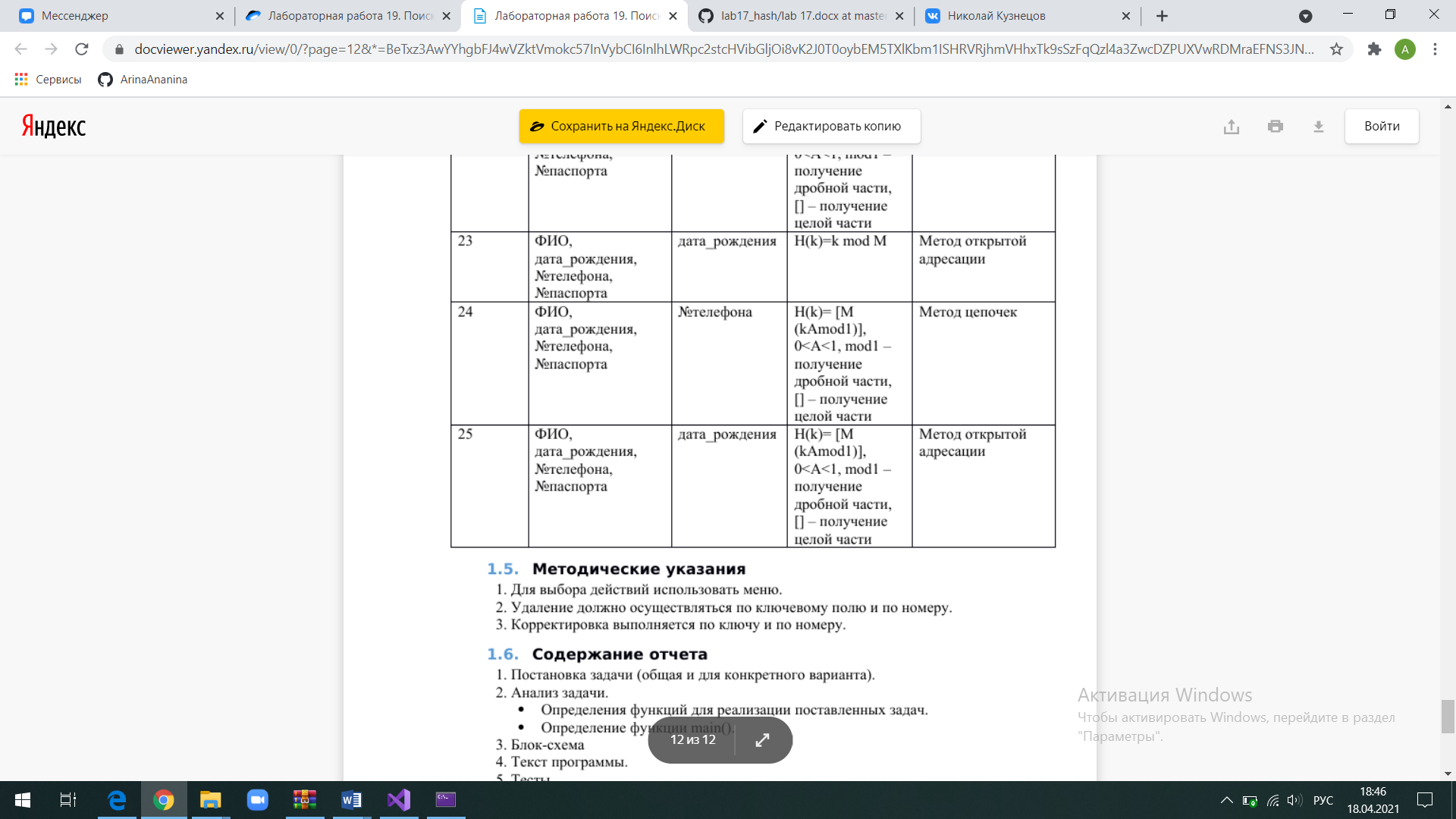
Создать динамический массив из записей (в соответствии с вариантом),

содержащий не менее 100 элементов. Для заполнения элементов массива

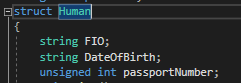
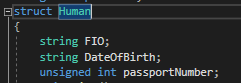
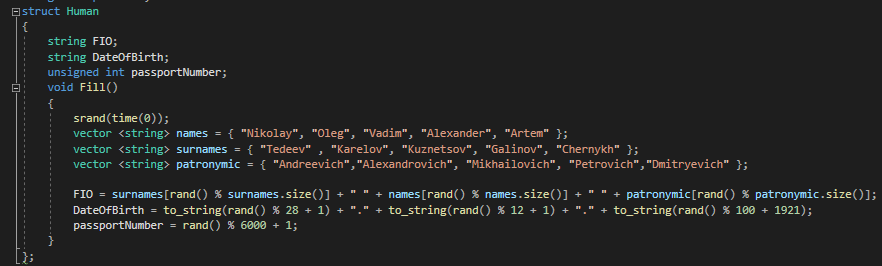
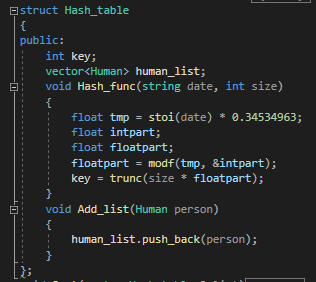
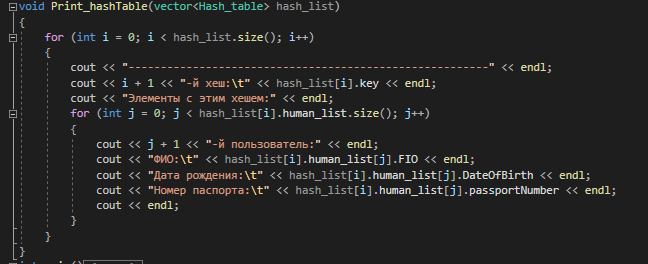
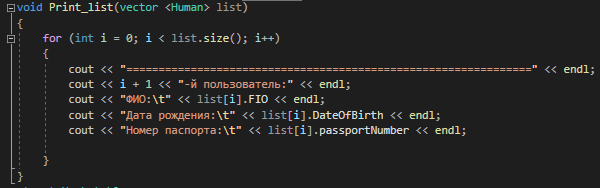
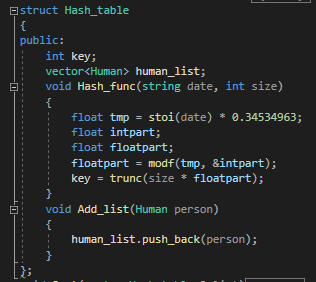
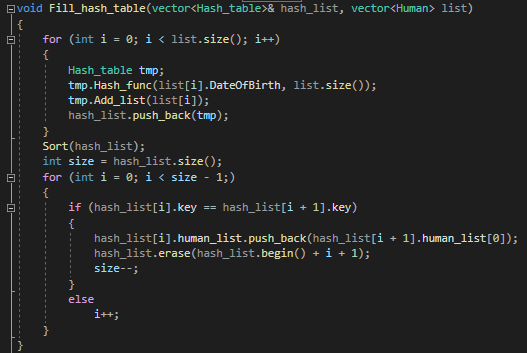
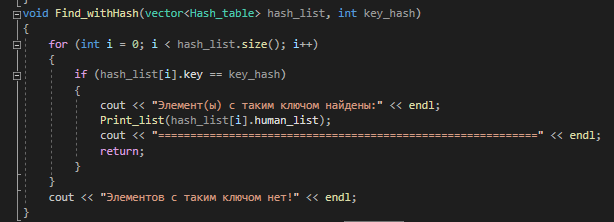
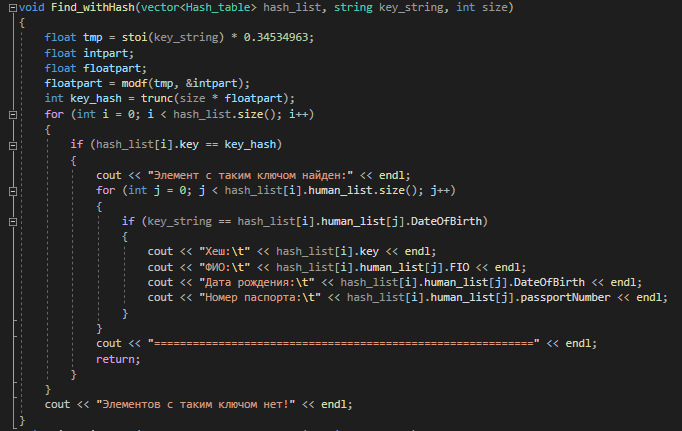
использовать ДСЧ.

Выполнить поиск элемента в массиве по ключу в соответствии с вариантом. Для

поиска использовать хеш-таблицу.



2. Анализ задачи

1. 1 Для решения задачи необходимо:
   1. Реализовать структуру хеш-таблицы и функцию хеширования.
   2. Реализовать функцию, разрешающую коллизии.
   3. Реализовать структуру человека с заданными полями.
   4. Создать динамический массив объектов структуры, заполнение с помощью ДСЧ.
2. В ходе работы были использованы следующие типы данных:
   1. String - поля структуры человека.  
      
   2. Unsigned int - поле номера паспорта человека.  
      
   3. Int - размер динамического массива.
   4. В качестве динамического массива используется класс vector.
3. Для решения задачи данные были представлены в следующем виде:
   1. Динамический массив (vector) типа Human.  
      
4. Для решения задачи использовались следующие структуры:
   1. Human - поля string (ФИО и дата рождения), поле unsigned int - номер паспорта).  
      
   2. Hash\_table - поле int (key - хеш), поле vector <Human> (массив людей, чья дата рождения соответствует хешу key).  
      
5. Для операций ввода и вывода использовались следующие операторы и функции:
   1. cout для вывода таблицы хешей и массива людей в функциях Print\_hashTable и Print\_list.  
      
   2. Cin для ввода ключа поиска.
6. Поставленные задачи будут решены следующими действиями:
   1. Хеш-функция (метод структуры Hash\_table) преобразует строку date, передаваемую как параметр, в значение типа int (хеш). Суть самого алгоритма заключается в том, чтобы привести строку к целочисленному типу и умножить на некую константу (0 < const < 1). Далее выделяется дробная часть от полученного числа, умножается на длину исходного массива (массив типа Human) и дробная часть отбрасывается.  
      
   2. Функция заполнения хеш-таблицы заполняет ее и разрешает коллизии. Функция принимает в качестве параметров ссылки на хеш-таблицу и массив людей. С помощью цикла с автоматическим перебором переменных происходит заполнение таблицы (поле key i-го элемента - результат работы метода Hash\_func, в который передается строка i-го элемента массива людей и размер этого массива; поле human\_list – i-ый человек в массиве людей). Далее для разрешения коллизий хеш-таблица сортируется по полю key.(сортировка Шелла). Алгоритм разрешения коллизий прост: с помощью цикла for осуществляется сравнение текущего элемента таблицы и следующего, если они равны, то в поле human\_list текущего элемента добавляется человек из поля human\_list следующего элемента таблицы, следующий элемент удаляется из таблицы, иначе управляющая переменная инкрементируется.  
      
   3. Функция Find\_withHash перегружена (для поиска по хешу и дате). В случае если функция принимает дату в виде строки, то она хешируется и осуществляется поиск по хешу, далее в подмассиве людей, удовлетворяющему найденному хешу осуществляется поиск по ключу. Если же в функцию передается хеш, то в случае нахождения такого хеша в таблице выводится весь подмассив людей.   
      

3. Код программы

#include <iostream>

#include <string>

using namespace std;

struct Date

{

int Day, Month, Year;

Date(int day, int month, int year)

{

this->Day = day;

this->Month = month;

this->Year = year;

}

};

struct Human

{

string FIO;

Date\* DateOfBirth;

long long PassportNumber, PhoneNumber;

Human()

{

string names[] = { "Иван", "Юрий", "Алексей", "Артём", "Дмитрий" };

string surnames[] = { "Иванов" , "Ананин", "Тарасов" };

string patronymic[] = { "Андреевич", "Вадимович", "Михаилович", "Кирилович", "Ильич" };

this->FIO = surnames[rand() % 3] + " " + names[rand() % 5] + " " + patronymic[rand() % 5];

this->DateOfBirth = new Date(1 + rand() % 28, 1 + rand() % 12, 1900 + rand() % 200);

this->PassportNumber = 5700000000 + (rand() % 100) \* 1000000 + rand() % 1000000;

this->PhoneNumber = 89000000000 + (rand() % 100) \* 10000000 + rand() % 1000000;

}

void Print()

{

cout << endl;

cout << "ФИО: " << FIO << endl;

cout << "Дата рождения: " << DateOfBirth->Day << "." << DateOfBirth->Month << "." << DateOfBirth->Year << endl;

cout << "Номер паспорта: " << PassportNumber << endl;

cout << "Номер телефона: " << PhoneNumber << endl;

}

};

struct HashTableItem

{

Human\* data;

HashTableItem\* next;

HashTableItem(Human\* data, HashTableItem\* next) {

this->data = data;

this->next = next;

}

};

struct HashFunc

{

double A = 0.00000001345666423;

int operator()(long long value, int tableSize) const {

float intprt;

return trunc(tableSize \* modf(value \* this->A, &intprt));

}

};

/\*

const HashFuncByPhone& a = HashFuncByPhone();

a(123, 123);

\*/

template <typename HashFunction>

struct HashTable {

int tableSize;

HashTableItem\*\* hashTable;

HashTable(int tableSize) {

this->tableSize = tableSize;

this->hashTable = new HashTableItem\*[tableSize];

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

this->hashTable[i] = NULL;

}

}

void Add(Human\*& value, const HashFunction& func = HashFunction()) {

int index = func(value->PhoneNumber, this->tableSize);

HashTableItem\* item = new HashTableItem(value, NULL);

if (this->hashTable[index] == NULL) {

this->hashTable[index] = item;

return;

}

HashTableItem\* place = this->hashTable[index];

while (place->next != NULL) {

place = place->next;

}

place->next = item;

}

void Delete(Human\*& value, const HashFunction& func = HashFunction()) {

}

Human\* Find(long long value, const HashFunction& func = HashFunction()) {

int index = func(value, this->tableSize);

if (this->hashTable[index] == NULL) {

return NULL;

}

HashTableItem\* item = this->hashTable[index];

while (item != NULL) {

if (item->data->PhoneNumber == value) {

return item->data;

}

item = item->next;

}

return NULL;

}

void Print() {

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

cout << "INDEX: " << i << endl;

if (this->hashTable[i] == NULL) {

cout << "NULL";

}

else {

HashTableItem\* item = this->hashTable[i];

while (item != NULL) {

cout << item->data->FIO << ", " << item->data->PhoneNumber << endl;

item = item->next;

}

}

cout << endl << endl;

}

}

};

int main()

{

setlocale(LC\_ALL, "rus");

HashTable<HashFunc>\* table = new HashTable<HashFunc>(40);

// Наполнение

for (int i = 0; i < 100; i++) {

Human\* human = new Human();

table->Add(human);

}

table->Print();

// Коллизии

int k = 0;

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

if (table->hashTable[i] != NULL && table->hashTable[i]->next != NULL) {

k++;

}

}

cout << "Кол-во коллизий: " << k;

// Поиск

long long phone;

cout << endl << "Введите номер телефона по которому необходимо осуществить поиск" << endl;

cin >> phone;

Human\* founded = table->Find(phone);

if (founded != NULL) {

founded->Print();

}

else {

cout << "NOT FOUNDED" << endl;

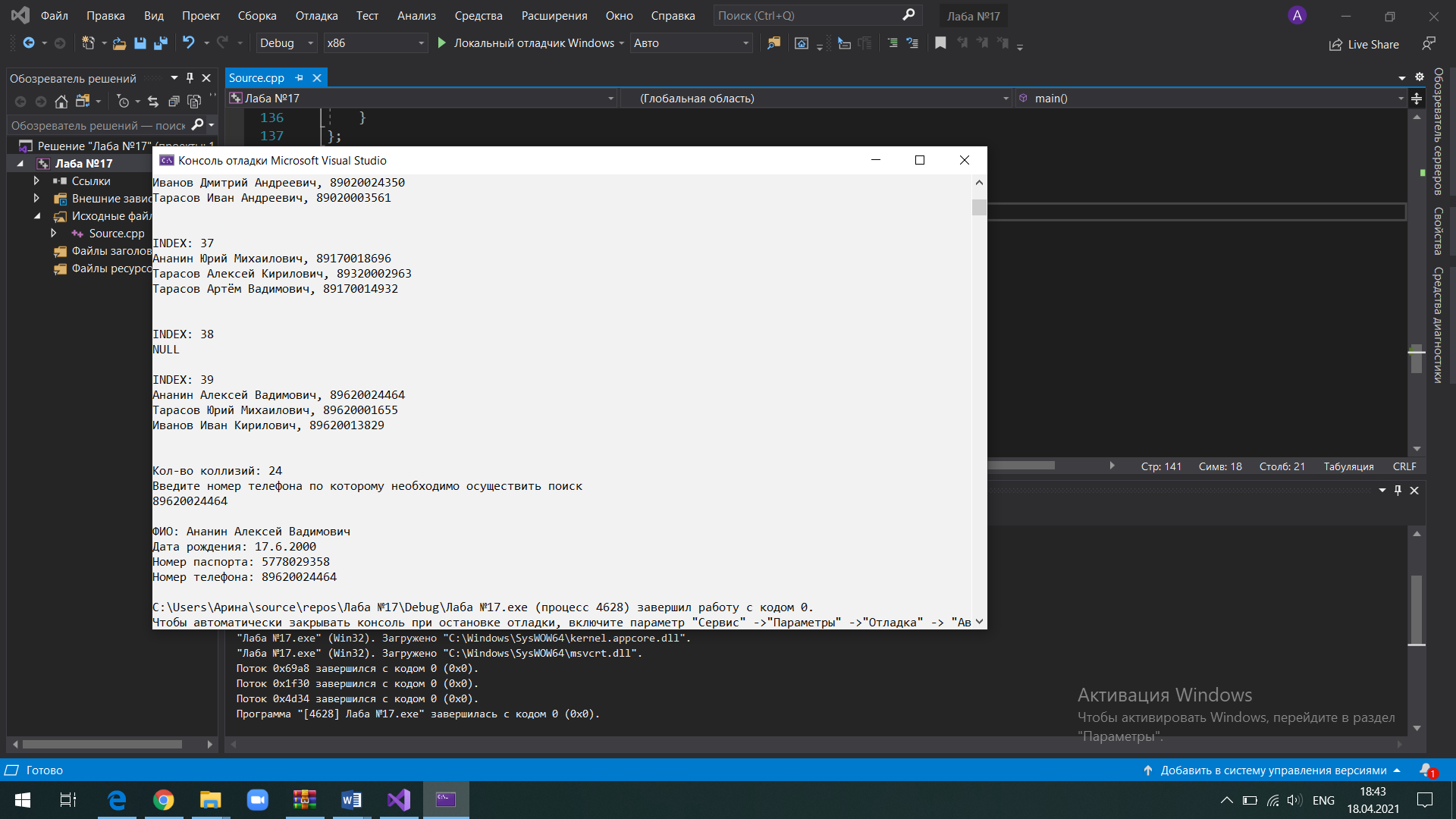
}

return 0;

}

4. Выполнение программы.

Программа сразу же печатает хеш-таблицу и выводит кол-во коллизий.



5. Блок-схема

