Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

# Тема: хэш-таблицы

« Реализовать хеш-таблицу в соответствии с вариантом »

# Семестр 1

Выполнил работу

Студент группы РИС-22-1Б

Зырянов

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**

**Вариант 15**

1. Создать динамический массив из записей (в соответствии с вариантом), содержащий не менее 100 элементов. Для заполнения элементов массива использовать ДСЧ.
2. Предусмотреть сохранение массива в файл и загрузку массива из файла.
3. Предусмотреть возможность добавления и удаления элементов из массива (файла).
4. Выполнить поиск элемента в массиве по ключу в соответствии с вариантом. Для поиска использовать хэш-таблицу.
5. Подсчитать количество коллизий при размере хэш-таблицы 40, 75 и 90 элементов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 15 | ФИО, дата\_рождения, адрес | ФИО | H(k)=k mod M | Метод открытой адресации |

**Алгоритм программы**

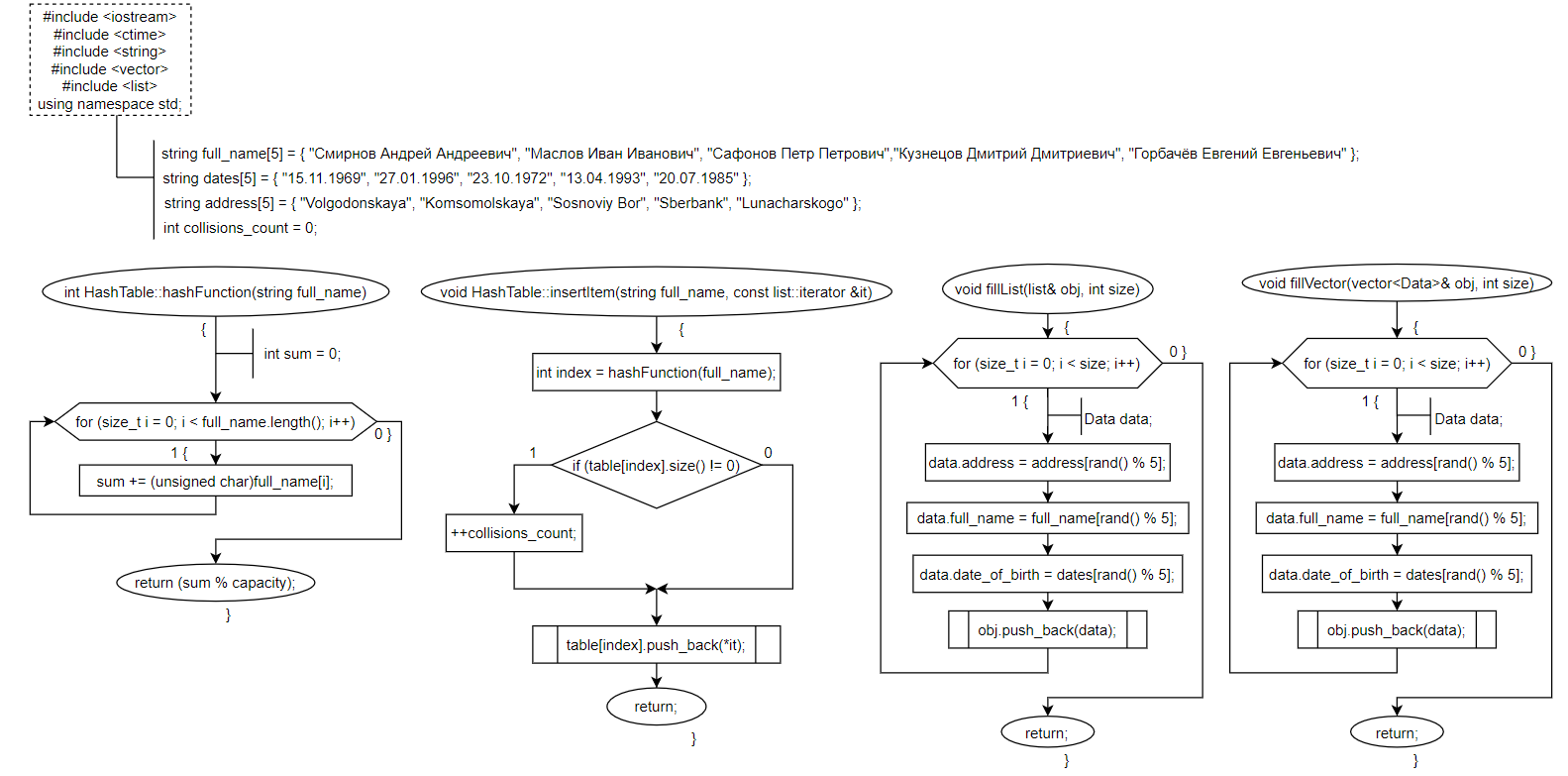


Рисунок 1 – методы теш-таблицы и заполнение листа/вектора

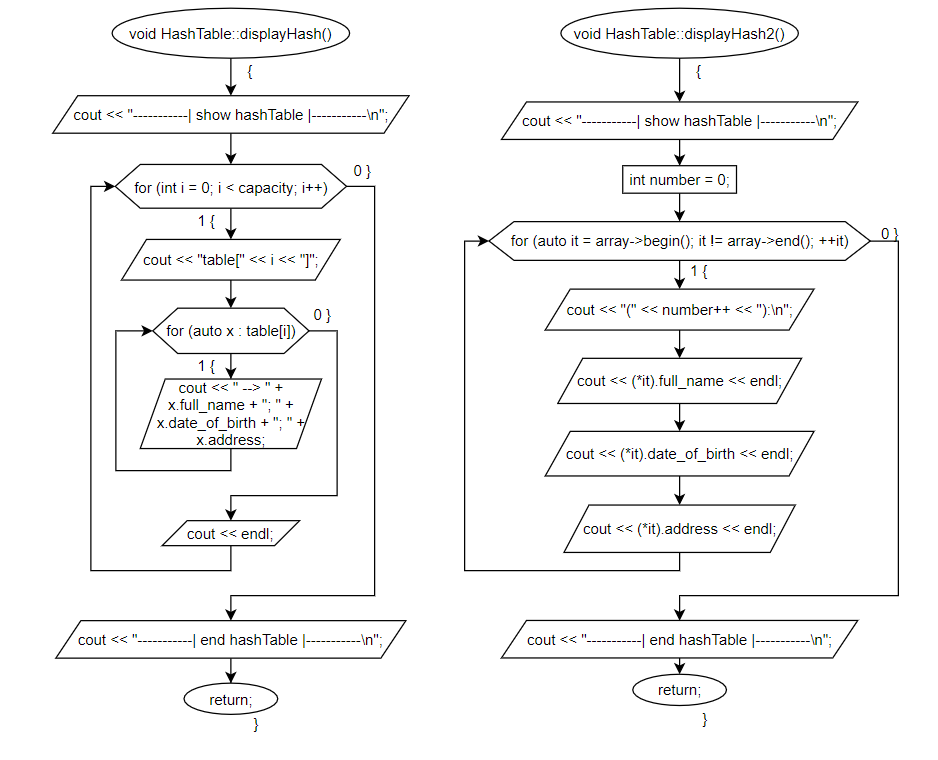


Рисунок 2 – методы вывода данных из хеш-таблицы

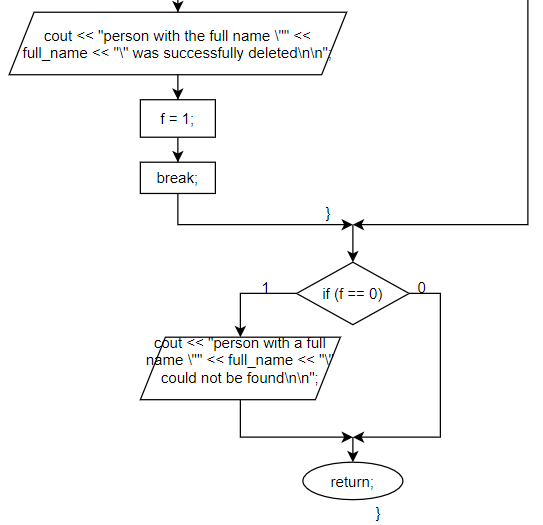
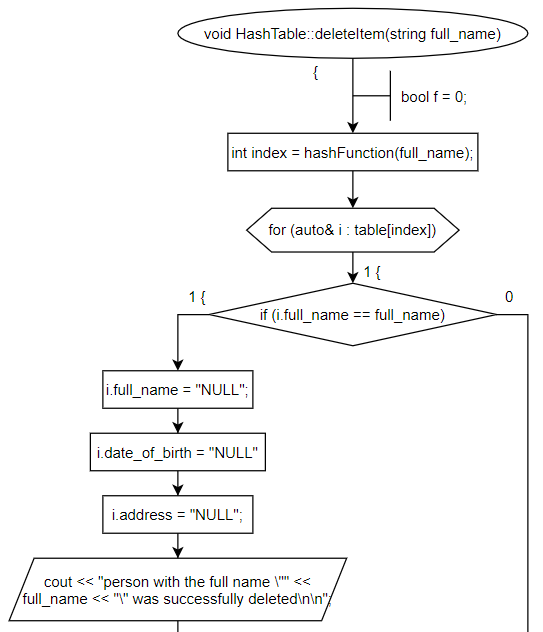
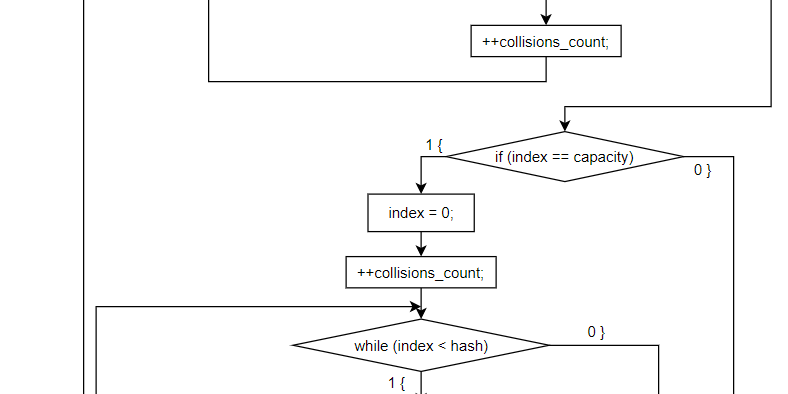
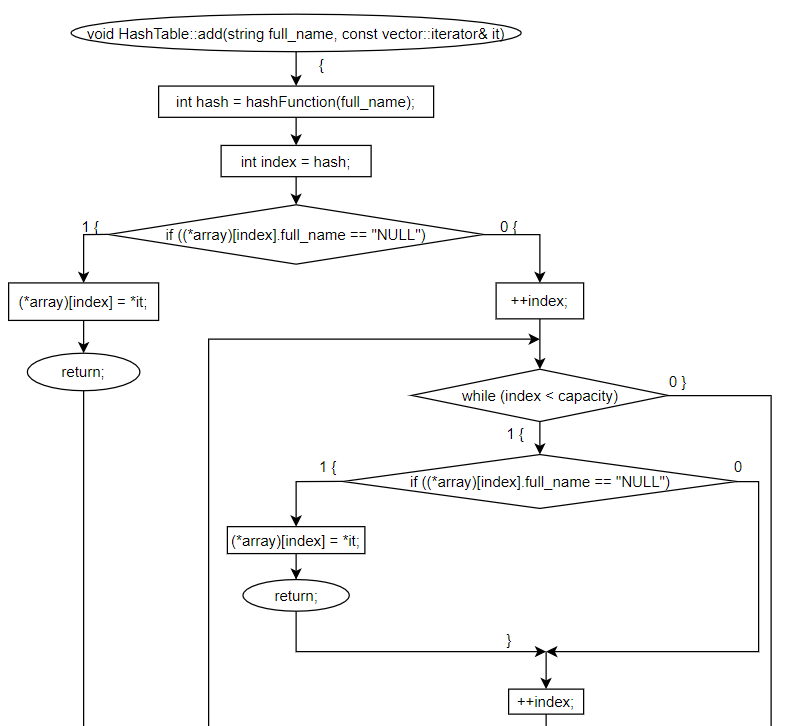


Рисунок 3 – метод удаления элемента хеш-таблицы (цепочки)



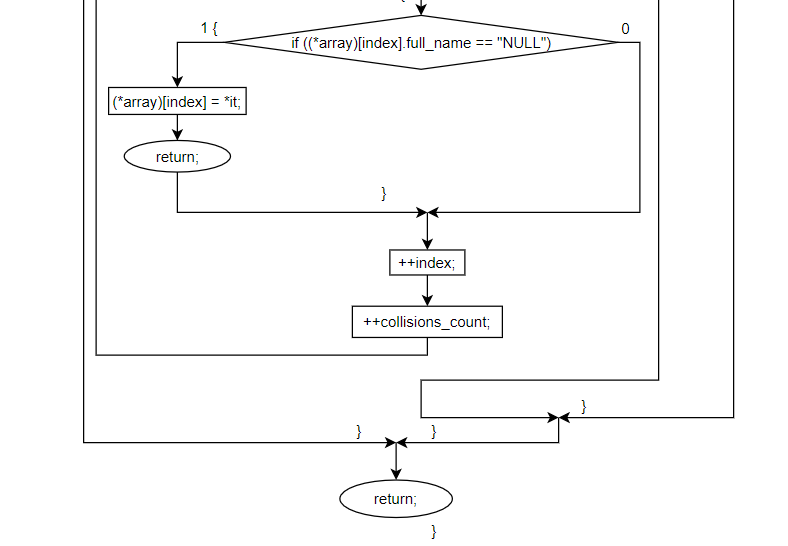


Рисунок 4 – метод добавления данных в хеш-таблицу (открытая адресация)

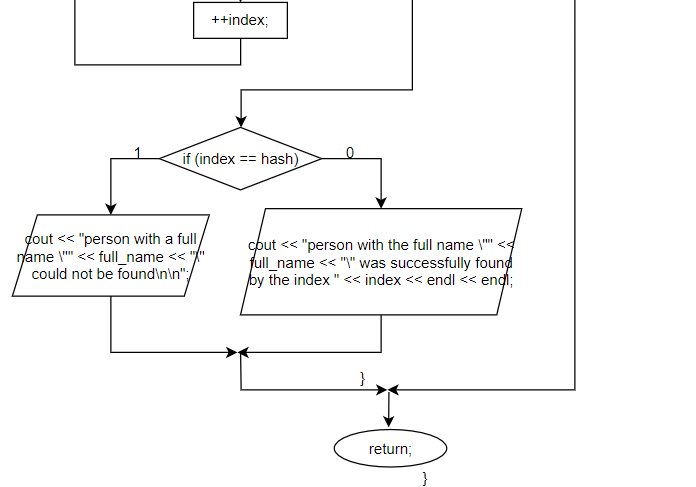
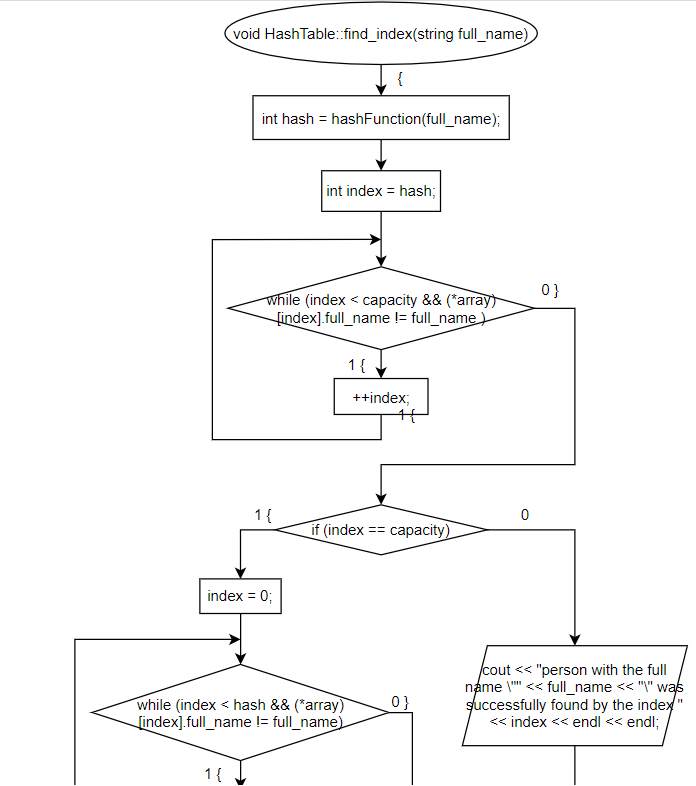
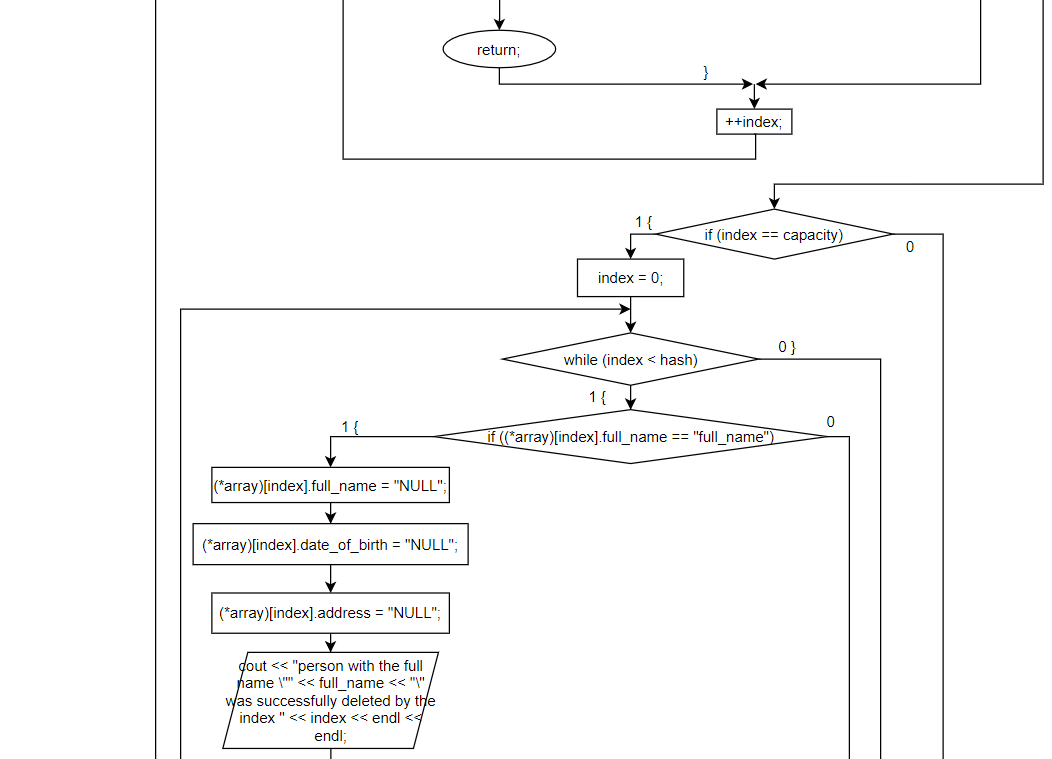
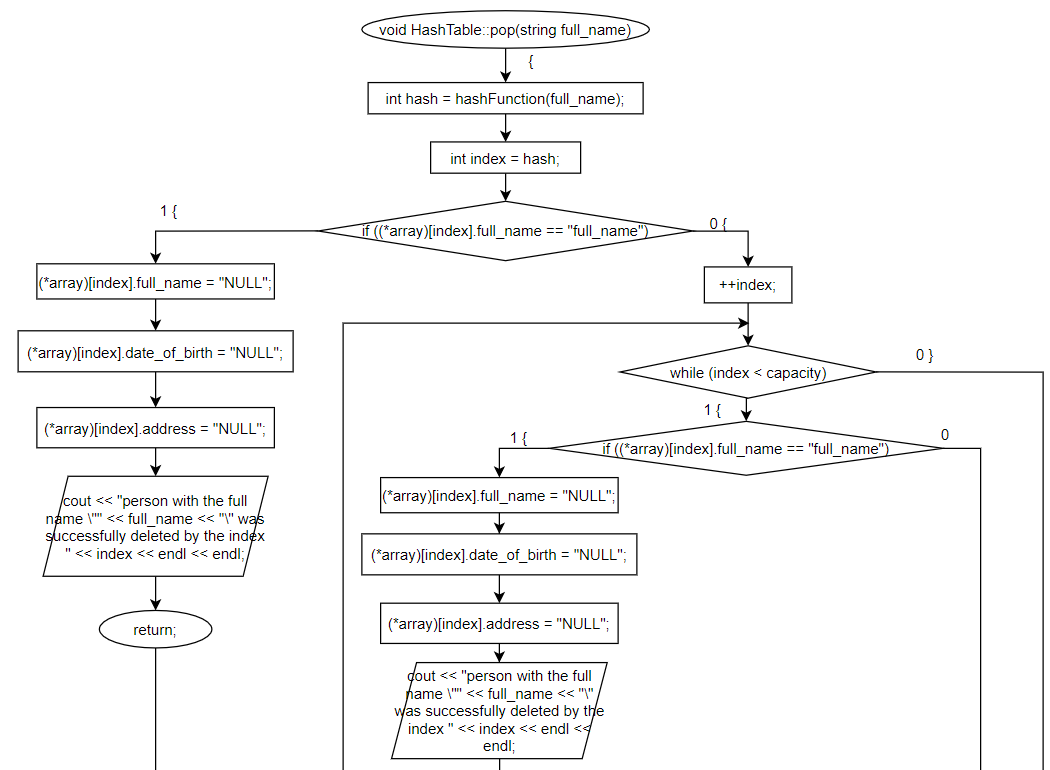


Рисунок 5 – поиск индекса по элементу (открытая адресация)



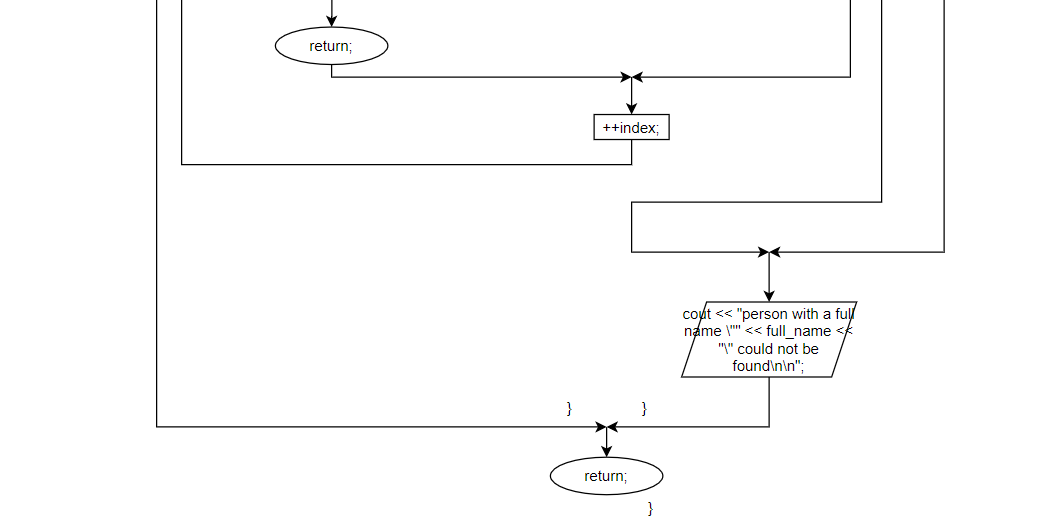


Рисунок 6 – метод удаления (открытая адресация)

**Код программы**

#include <iostream>

#include <ctime>

#include <string>

#include <vector>

#include <list>

using namespace std;

string full\_name[5] = { "Смирнов Андрей Андреевич", "Маслов Иван Иванович", "Сафонов Петр Петрович","Кузнецов Дмитрий Дмитриевич", "Горбачёв Евгений Евгеньевич" };

string dates[5] = { "15.11.1969", "27.01.1996", "23.10.1972", "13.04.1993", "20.07.1985" };

string address[5] = { "Volgodonskaya", "Komsomolskaya", "Sosnoviy Bor", "Sberbank", "Lunacharskogo" };

int collisions\_count = 0;

struct Data

{

string full\_name = "NULL";

string date\_of\_birth = "NULL";

string address = "NULL";

};

class HashTable

{

int capacity;

list<Data>\* table;

vector<Data>\* array;

public:

// generally

HashTable(int V);

int hashFunction(string full\_name);

~HashTable();

// end

// chain methods

void insertItem(string full\_name, const list<Data>::iterator& it);

void deleteItem(string full\_name);

void displayHash();

// end

// open address methods

void add(string full\_name, const vector<Data>::iterator& it);

void displayHash2();

void find\_index(string date\_of\_birth);

void pop(string date\_to\_delete);

// end

};

// generally methods

//------------------------------------------------------------------------------------------------------

HashTable::HashTable(int c)

{

this->capacity = c;

table = new list<Data>[capacity];

array = new vector<Data>(c);

}

int HashTable::hashFunction(string full\_name)

{

int sum = 0;

for (size\_t i = 0; i < full\_name.length(); i++)

sum += (unsigned char)full\_name[i];

return (sum % capacity);

}

HashTable::~HashTable()

{

table->clear();

}

//------------------------------------------------------------------------------------------------------

// end generally methods

// method chains

//------------------------------------------------------------------------------------------------------

void HashTable::insertItem(string full\_name, const list<Data>::iterator &it)

{

int index = hashFunction(full\_name);

if (table[index].size() != 0)

++collisions\_count;

table[index].push\_back(\*it);

}

void HashTable::deleteItem(string full\_name)

{

int index = hashFunction(full\_name);

bool f = 0;

for (auto& i : table[index])

{

if (i.full\_name == full\_name)

{

i.full\_name = "NULL";

i.date\_of\_birth = "NULL";

i.address = "NULL";

cout << "person with the full name \"" << full\_name << "\" was successfully deleted\n\n";

f = 1;

break;

}

}

if (f == 0)

cout << "person with a full name \"" << full\_name << "\" could not be found\n\n";

}

void HashTable::displayHash()

{

cout << "-----------| show hashTable |-----------\n";

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

{

cout << "table[" << i << "]";

for (auto x : table[i])

cout << " --> " + x.full\_name + "; " + x.date\_of\_birth + "; " + x.address;

cout << endl;

}

cout << "-----------| end hashTable |-----------\n";

}

//------------------------------------------------------------------------------------------------------

// end method chains

// method open address

//------------------------------------------------------------------------------------------------------

void HashTable::add(string full\_name, const vector<Data>::iterator& it)

{

int hash = hashFunction(full\_name);

int index = hash;

if ((\*array)[index].full\_name == "NULL")

{

(\*array)[index] = \*it;

return;

}

else

{

++index;

while (index < capacity)

{

if ((\*array)[index].full\_name == "NULL")

{

(\*array)[index] = \*it;

return;

}

++index;

++collisions\_count;

}

if (index == capacity)

{

index = 0;

++collisions\_count;

while (index < hash)

{

if ((\*array)[index].full\_name == "NULL")

{

(\*array)[index] = \*it;

return;

}

++index;

++collisions\_count;

}

}

}

}

void HashTable::find\_index(string full\_name)

{

int hash = hashFunction(full\_name);

int index = hash;

while (index < capacity && (\*array)[index].full\_name != full\_name)

++index;

if (index == capacity)

{

index = 0;

while (index < hash && (\*array)[index].full\_name != full\_name)

++index;

if (index == hash)

cout << "person with a full name \"" << full\_name << "\" could not be found\n\n";

else

cout << "person with the full name \"" << full\_name << "\" was successfully found by the index " << index << endl << endl;

}

else

cout << "person with the full name \"" << full\_name << "\" was successfully found by the index " << index << endl << endl;

}

void HashTable::pop(string full\_name)

{

int hash = hashFunction(full\_name);

int index = hash;

if ((\*array)[index].full\_name == full\_name)

{

(\*array)[index].full\_name = "NULL";

(\*array)[index].date\_of\_birth = "NULL";

(\*array)[index].address = "NULL";

cout << "person with the full name \"" << full\_name << "\" was successfully deleted by the index " << index << endl << endl;

return;

}

else

{

++index;

while (index < capacity)

{

if ((\*array)[index].full\_name == full\_name)

{

(\*array)[index].full\_name = "NULL";

(\*array)[index].date\_of\_birth = "NULL";

(\*array)[index].address = "NULL";

cout << "person with the full name \"" << full\_name << "\" was successfully deleted by the index " << index << endl << endl;

return;

}

++index;

}

if (index == capacity)

{

index = 0;

while (index < hash)

{

if ((\*array)[index].full\_name == full\_name)

{

(\*array)[index].full\_name = "NULL";

(\*array)[index].date\_of\_birth = "NULL";

(\*array)[index].address = "NULL";

cout << "person with the full name \"" << full\_name << "\" was successfully deleted by the index " << index << endl << endl;

return;

}

++index;

}

}

}

cout << "person with a full name \"" << full\_name << "\" could not be found\n\n";

}

void HashTable::displayHash2()

{

cout << "-----------| show hashTable |-----------\n";

int number = 0;

for (auto it = array->begin(); it != array->end(); ++it)

{

cout << "(" << number++ << "):\n";

cout << (\*it).full\_name << endl;

cout << (\*it).date\_of\_birth << endl;

cout << (\*it).address << endl;

}

}

//------------------------------------------------------------------------------------------------------

// end method open address

void fillList(list<Data>& obj, int size)

{

for (size\_t i = 0; i < size; i++)

{

Data data;

data.address = address[rand() % 5];

data.full\_name = full\_name[rand() % 5];

data.date\_of\_birth = dates[rand() % 5];

obj.push\_back(data);

}

}

void fillVector(vector<Data>& obj, int size)

{

for (size\_t i = 0; i < size; i++)

{

Data data;

data.address = address[rand() % 5];

data.full\_name = full\_name[rand() % 5];

data.date\_of\_birth = dates[rand() % 5];

obj.push\_back(data);

}

}

int main()

{

srand(time(0));

system("chcp 1251 >> null");

int size = 0;

cout << "size(for table): "; cin >> size;

int choice = -1;

cout << "\n\t1 - method chain\n\t2 - method open address\n\nyour answer: "; cin >> choice;

switch (choice)

{

case 1:

{

list<Data> lst1;

system("cls"); cout << "----------| show List |----------\n";

fillList(lst1, size);

int number = 0;

for (auto it = lst1.begin(); it != lst1.end(); ++it)

{

cout << "(" << number++ << ")" << endl;

cout << (\*it).full\_name << endl;

cout << (\*it).date\_of\_birth << endl;

cout << (\*it).address << endl;

}

cout << "-----------| end List |-----------\n\n";

HashTable h(size);

for (auto it = lst1.begin(); it != lst1.end(); ++it)

h.insertItem((\*it).full\_name, it);

h.displayHash(); cout << endl;

h.deleteItem("Маслов Иван Иванович");

h.displayHash();

}

break;

case 2:

{

vector<Data> array;

system("cls"); cout << "----------| show Vector |----------\n";

fillVector(array, size);

int number = 0;

for (auto it = array.begin(); it != array.end(); ++it)

{

cout << "(" << number++ << ")" << endl;

cout << (\*it).full\_name << endl;

cout << (\*it).date\_of\_birth << endl;

cout << (\*it).address << endl;

}

cout << "-----------| end Vector |-----------\n\n";

HashTable h2(size);

for (auto it = array.begin(); it != array.end(); ++it)

h2.add((\*it).full\_name, it);

h2.displayHash2(); cout << endl;

h2.find\_index("Маслов Иван Иванович");

h2.pop("Маслов Иван Иванович");

h2.displayHash2();

}

break;

default:

break;

}

cout << endl << "collisions: " << collisions\_count << endl;

}

**Результаты работы программы**

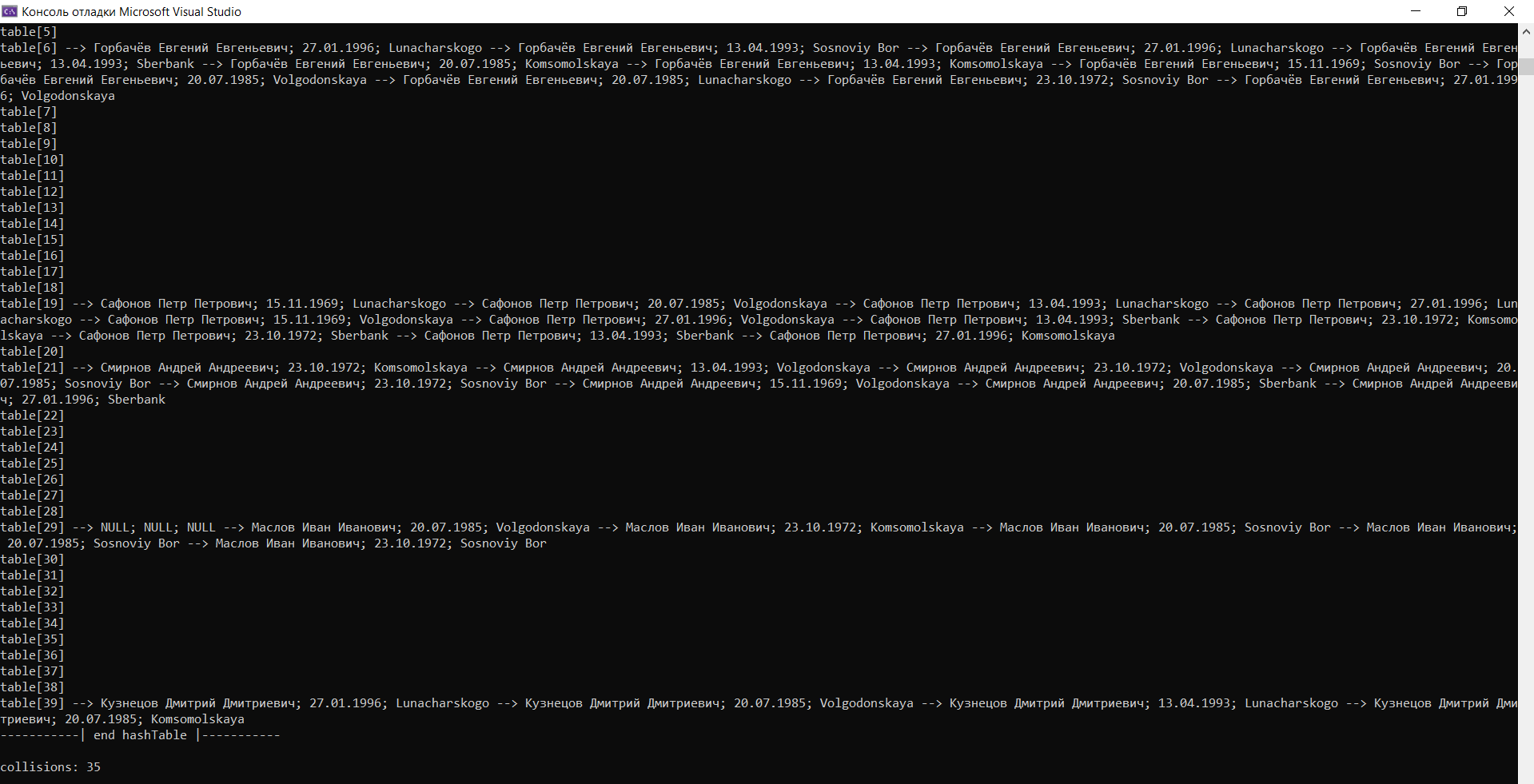


Рисунок 7 – метод цепочек для 40 элементов

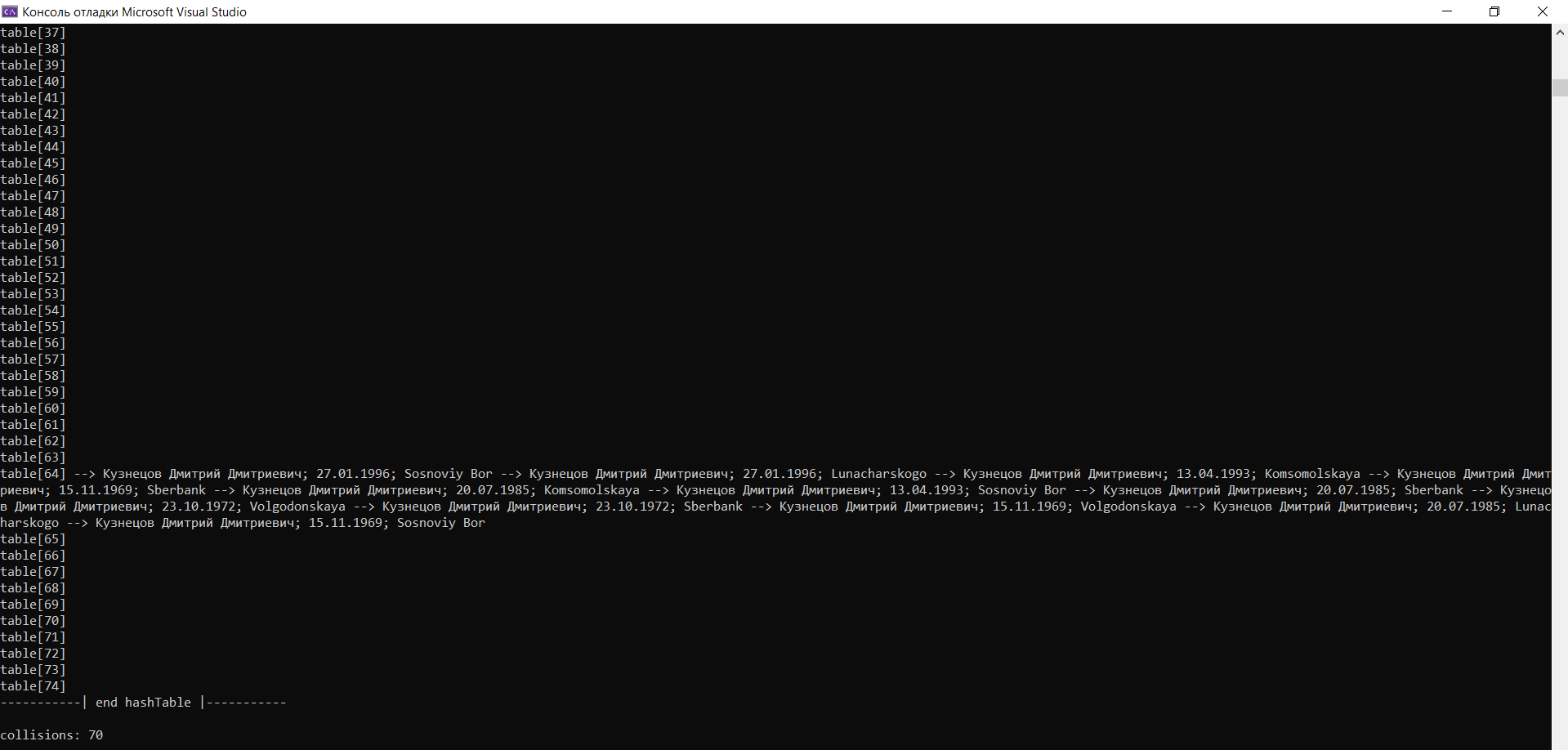


Рисунок 8 – метод цепочек для 75 элементов

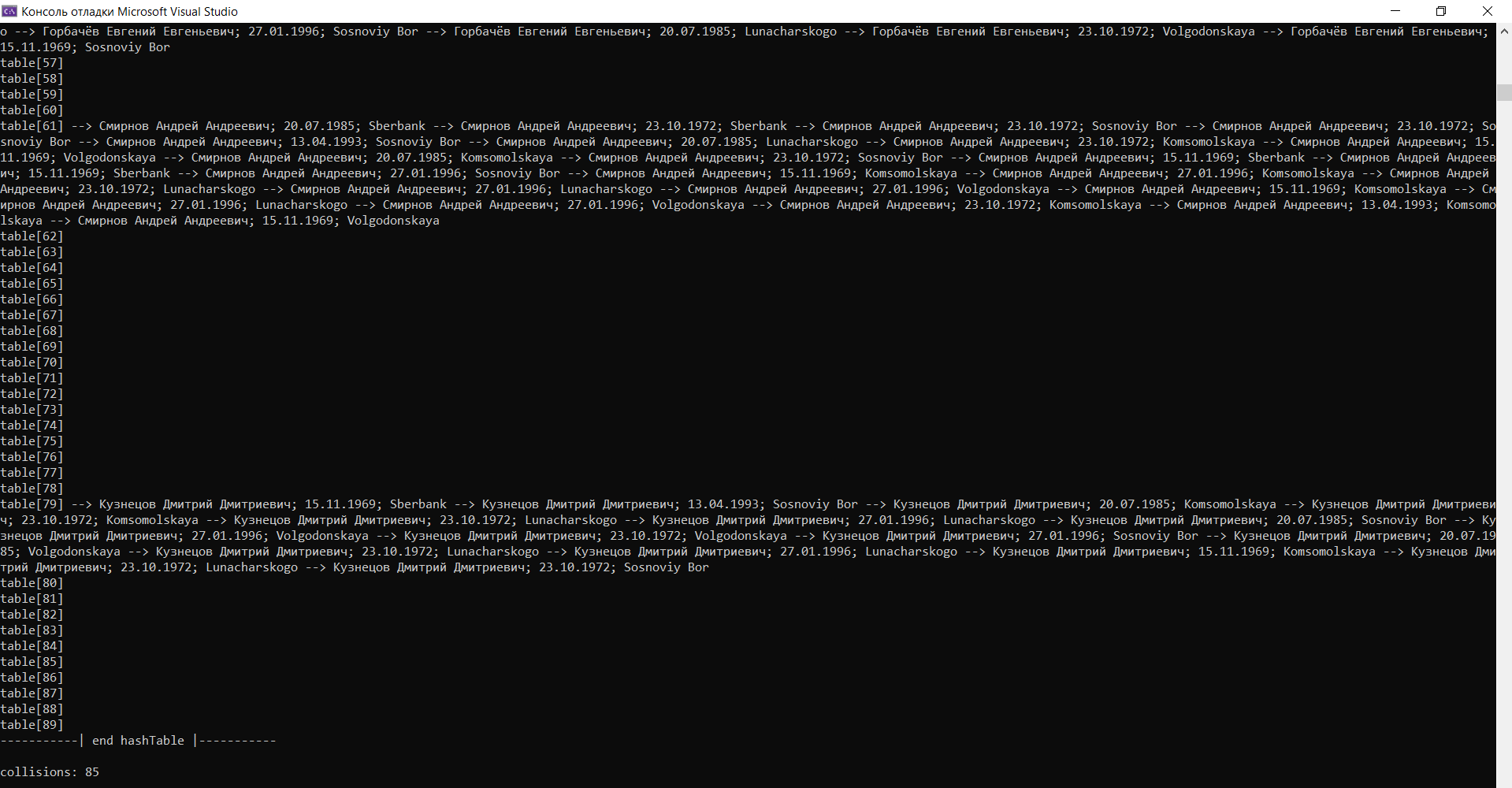


Рисунок 9 – метод цепочек для 90 элементов

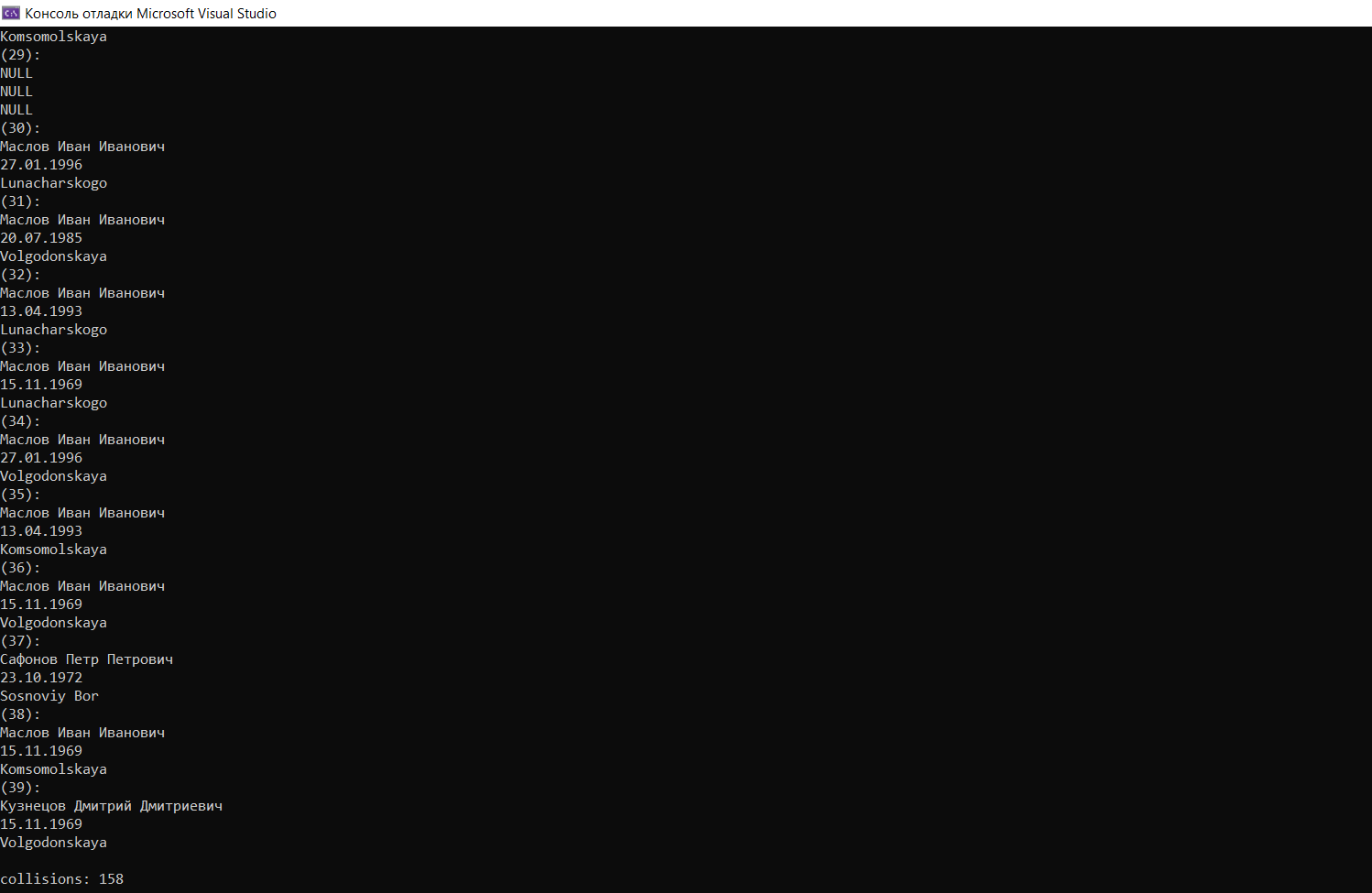


Рисунок 9 – метод открытой адресации для 40 элементов

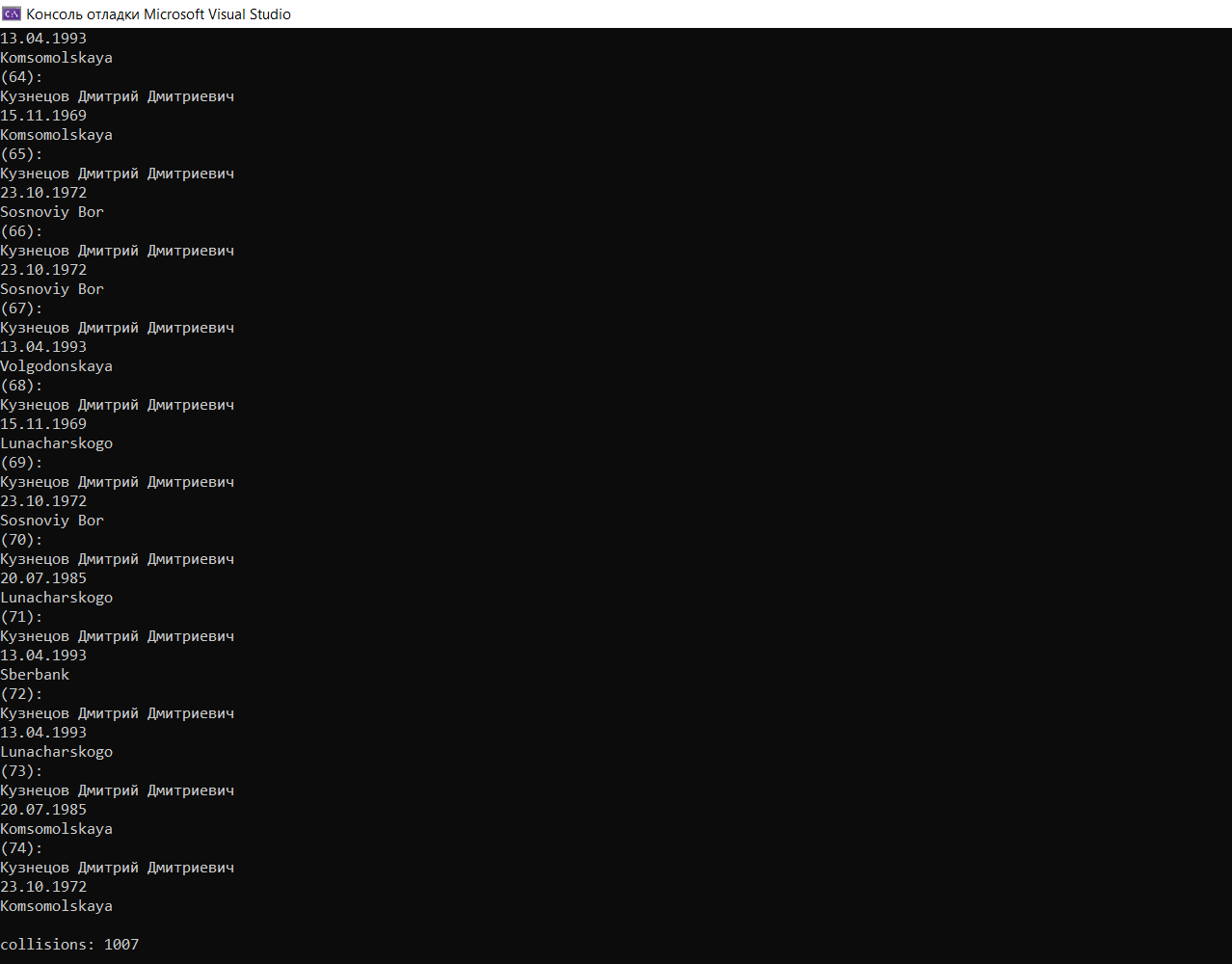


Рисунок 10 – метод открытой адресации для 75 элементов

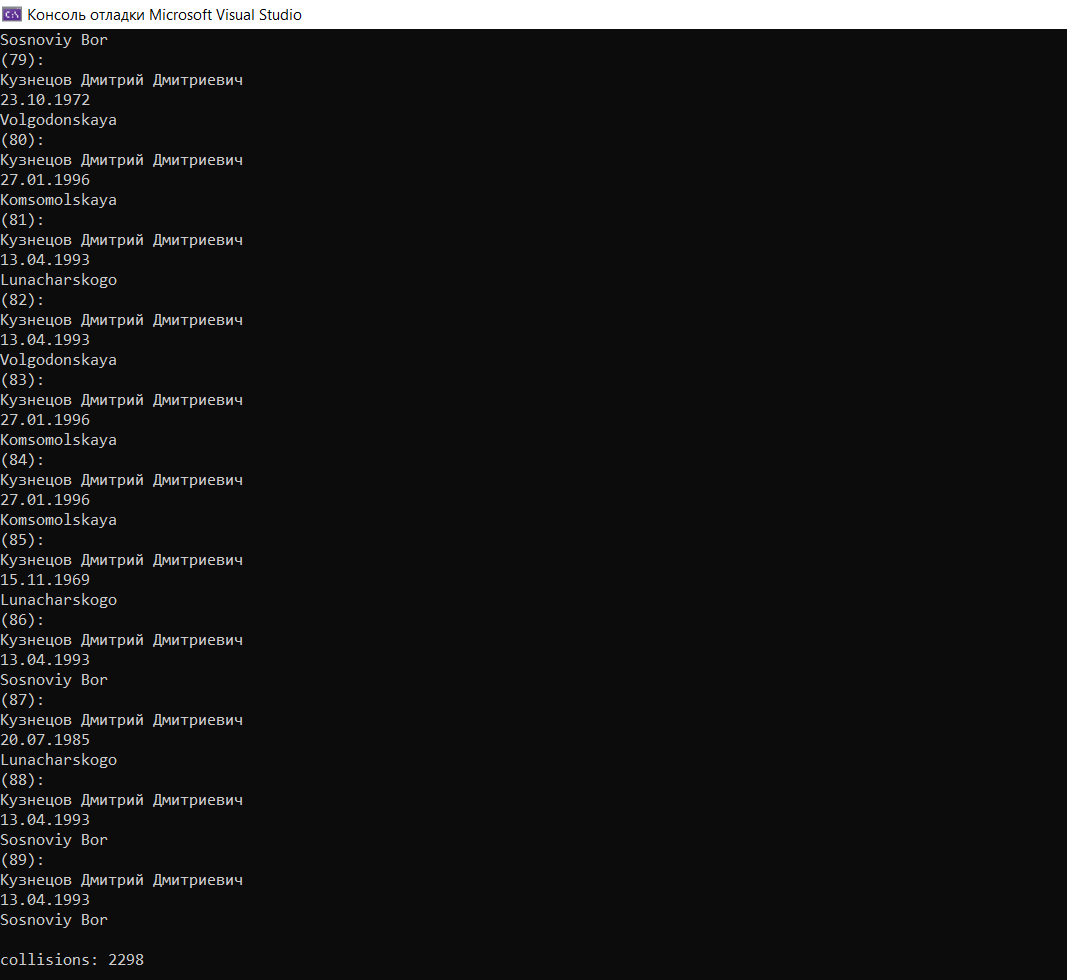


Рисунок 11 – метод открытой адресации для 90 элементов