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: «Информатика»

Тема: Hash-таблицы и Hash-функции

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Тарасов C.В.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**

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

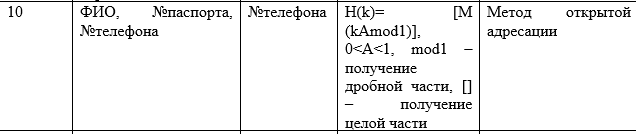
2. Предусмотреть сохранение массива в файл и загрузку массива из файла.

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.





**Анализ задачи**

Хеширование (или хэширование, англ. hashing ) – это преобразование входного массива данных определенного типа и произвольной длины в выходную битовую строку фиксированной длины. Такие преобразования также называются хеш-функциямиили функциями свертки, а их результаты называют хешем, хеш-кодом, хеш-таблицей или дайджестом сообщения (англ. message digest ).

Хеш-таблица – это структура данных, реализующая интерфейс ассоциативного массива, то есть она позволяет хранить пары вида "ключ- значение" и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу. Хеш-таблица является массивом, формируемым в определенном порядке хеш-функцией.

Основная идея базовой структуры при открытом (внешнем) хешировании заключается в том, что потенциальное множество (возможно, бесконечное) разбивается на конечное число классов. Для В классов, пронумерованных от 0 до В-1, строится хеш-функция h(x) такая, что для любого элемента х исходного множества функция h(x) принимает целочисленное значение из интервала 0,1,...,В-1, соответствующее классу, которому принадлежит элемент х. Часто классы называют сегментами, поэтому будем говорить, что элемент х принадлежит сегменту h(x). Массив, называемый таблицей сегментов и проиндексированный номерами сегментов 0,1,...,В-1, содержит заголовки для B списков. Элемент х, относящийся к i -му списку – это элемент исходного множества, для которого h(x)=i.

Если сегменты примерно одинаковы по размеру, то в этом случае списки всех сегментов должны быть наиболее короткими при данном числе сегментов. Если исходное множество состоит из N элементов, тогда средняя длина списков будет N/B элементов. Если можно оценить величину N и выбрать В как можно ближе к этой величине, то в каждом списке будет один или два элемента. Тогда время выполнения операторов словарей будет малой постоянной величиной, не зависящей от N.

Для реализации поставленной задачи понадобится структура Human

struct Human {

Human() {

fio = "NULL";

tele = "NULL";

passport = "NULL";

}

string fio;

string tele;

string passport;

};

Также структура Node для реализации метода цепочек

struct Node {

Human data;

Node\* next = nullptr;

};

А также массивы имен, номеров телефона, номеров паспортов, фамилий и отчеств.

string names[5] = { "Андрей","Сергей","Алексей","Александр","Владимир" };

string surnames[5] = { "Андреев","Сергеев","Алексеев","Александров","Владимиров" };

string patronymics[5] = { "Андреевич","Сергеевич","Алексеевич","Александрович","Владимирович" };

string teles[20] = { "88005553535","89223077612","89028327085","89223076403","89264177054","88005553536","88005553537", "88005553538", "88005553539","88005553530","88005553531","88005553532","88005553533","88005553534","89223077618","89223077617","89223077616","89223077615","89223077611","89223077614" };

string passports[5] = { "23\_32\_675412","99\_11\_231525","44\_65\_654122","12\_44\_342788","32\_32\_564123" };

Массивы понадобятся для создания случайных объектов с разными параметрами

Важной частью программы является структура самой хэш таблицы и методы add(добавление), findindex(поиск индекса), pop(удаление), infile(занесение в файл), outfile(вынесение из файла) реализованные в структуре.

Первая структура хэш таблицы для метода прямой адресации -

struct hashtable {

Human\* array;

int collisioncount = 0;

hashtable(int size) {

array = new Human[size];

}

~hashtable() {

delete[] array;

}

…………………………………………………………..

Вторая структура хэш таблицы для метода цепочек –

struct hashtable\_chain {

vector<Node\*> array;

int collisioncount = 0;

hashtable\_chain(int size) {

array.resize(size);

}

~hashtable\_chain() {

for (auto node : array) {

while (node != nullptr) {

Node\* temp = node;

node = node->next;

delete temp;

}

}

}

…………………………………………………………..

**Код програмы**

#include <iostream>

#include<string>

#include<ctime>

#include<fstream>

#include<windows.h>

#include <vector>

using namespace std;

string names[5] = { "Андрей","Сергей","Алексей","Александр","Владимир" };

string surnames[5] = { "Андреев","Сергеев","Алексеев","Александров","Владимиров" };

string patronymics[5] = { "Андреевич","Сергеевич","Алексеевич","Александрович","Владимирович" };

string teles[20] = { "88005553535","89223077612","89028327085","89223076403","89264177054","88005553536","88005553537", "88005553538", "88005553539","88005553530","88005553531","88005553532","88005553533","88005553534","89223077618","89223077617","89223077616","89223077615","89223077611","89223077614" };

string passports[5] = { "23\_32\_675412","99\_11\_231525","44\_65\_654122","12\_44\_342788","32\_32\_564123" };

struct Human {

Human() {

fio = "NULL";

tele = "NULL";

passport = "NULL";

}

string fio;

string tele;

string passport;

};

struct Node {

Human data;

Node\* next = nullptr;

};

unsigned int HASH(const string key, int table\_size)

{

double A = 0.61803398875;

double hash\_value = 0;

for (size\_t i = 0; i < key.length(); i++) {

hash\_value = hash\_value + (int)key[i] \* pow(A, i + 1);

}

hash\_value = hash\_value - floor(hash\_value);

return (unsigned int)(floor(table\_size \* hash\_value));

}

int getnumber() {

return (rand() % 5);

}

string getfio() {

return(surnames[getnumber()] + " " + names[getnumber()] + " " + patronymics[getnumber()]);

}

string gettele() {

return(teles[(rand() % 20)]);

}

string getpassport() {

return (passports[getnumber()]);

}

Human createhuman() {

Human tmp;

tmp.fio = getfio();

tmp.tele = gettele();

tmp.passport = getpassport();

return tmp;

}

void showinfo(const Human tmp) {

cout << "ФИО:" << tmp.fio << endl;

cout << "Номер телефона:" << tmp.tele << endl;

cout << "Паспорт:" << tmp.passport << endl;

}

void printarray(Human\* array, int size) {

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

showinfo(array[i]);

cout << endl;

}

}

void fillarray(Human\* array, int count) {

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

array[i] = createhuman();

}

}

struct hashtable {

Human\* array;

int collisioncount = 0;

hashtable(int size) {

array = new Human[size];

}

~hashtable() {

delete[] array;

}

void add(Human temp, int size) {

int i = HASH(temp.tele, size);

int j = i;

if (array[i].fio == "NULL") {

array[i] = temp;

return;

}

else {

while (i < size) {

if (array[i].fio == "NULL") {

array[i] = temp;

return;

}

collisioncount++;

i++;

}

if (i >= size) {

i = 0;

collisioncount++;

while (i < j) {

if (array[i].fio == "NULL") {

array[i] = temp;

return;

}

collisioncount++;

i++;

}

}

}

}

void findindex(string temp, int size) {

int hash = HASH(temp, size);

int index = hash;

while (array[index].tele != temp && index < size) {

index++;

}

if (index >= size) {

index = 0;

while (array[index].tele != temp && index < hash) {

index++;

}

if (index >= hash) {

cout << "Человек с номером телефона " << temp << " содержится по индексу " << index << endl;

}

}

else {

cout << "Человек с номером телефона " << temp << " содержится по индексу " << index << endl;

}

}

void pop(string temp, int size) {

int index = HASH(temp, size);

if (array[index].tele == temp) {

array[index] = Human();

return;

}

else {

while (index < size) {

if (array[index].tele == temp) {

array[index] = Human();

return;

}

index++;

}

if (index >= size) {

index = 0;

while (index < size) {

if (array[index].tele == temp) {

array[index] = Human();

return;

}

index++;

}

}

}

}

void infile(Human\* array, int size) {

ofstream fo("file.txt");

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

fo << array[i].fio << endl;

fo << array[i].tele << endl;

fo << array[i].passport << endl;

}

fo.close();

}

void outfile(Human\* array, int size) {

ifstream fi("file.txt");

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

getline(fi, array[i].fio);

getline(fi, array[i].tele);

getline(fi, array[i].passport);

}

fi.close();

}

};

struct hashtable\_chain {

vector<Node\*> array;

int collisioncount = 0;

hashtable\_chain(int size) {

array.resize(size);

}

~hashtable\_chain() {

for (auto node : array) {

while (node != nullptr) {

Node\* temp = node;

node = node->next;

delete temp;

}

}

}

void add(Human temp, int size) {

int i = HASH(temp.tele, size);

Node\* node = new Node();

node->data = temp;

if (array[i] == nullptr) {

array[i] = node;

}

else {

collisioncount++;

Node\* current = array[i];

while (current->next != nullptr) {

current = current->next;

collisioncount++;

}

current->next = node;

}

}

void findindex(string temp, int size) {

int hash = HASH(temp, size);

Node\* current = array[hash];

int index = hash;

while (current != nullptr && current->data.tele != temp) {

current = current->next;

index++;

}

if (current != nullptr) {

cout << "Человек с номером телефона " << temp << " содержится по индексу " << index << endl;

}

else {

cout << "Человек с номером телефона " << temp << " не найден" << endl;

}

}

void pop(string temp, int size) {

int index = HASH(temp, size);

Node\* current = array[index];

Node\* prev = nullptr;

while (current != nullptr && current->data.tele != temp) {

prev = current;

current = current->next;

}

if (current != nullptr) {

if (prev != nullptr) {

prev->next = current->next;

}

else {

array[index] = current->next;

}

delete current;

}

}

void infile(vector<Node\*>& array, int size) {

ofstream fo("file.txt");

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

Node\* current = array[i];

while (current != nullptr) {

fo << current->data.fio << endl;

fo << current->data.tele << endl;

fo << current->data.passport << endl;

current = current->next;

}

}

fo.close();

}

void outfile(vector<Node\*>& array, int size) {

ifstream fi("file.txt");

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

string fio, tele, passport;

getline(fi, fio);

getline(fi, tele);

getline(fi, passport);

Human temp;

temp.fio = fio;

temp.tele = tele;

temp.passport = passport;

int index = HASH(tele, size);

Node\* node = new Node();

node->data = temp;

if (array[index] == nullptr) {

array[index] = node;

}

else {

Node\* current = array[index];

while (current->next != nullptr) {

current = current->next;

}

current->next = node;

}

}

fi.close();

}

};

void showhash(hashtable\* newhash, int size) {

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

showinfo(newhash->array[i]);

cout << endl;

}

}

void showhashtable(hashtable\_chain\* newhash, int size) {

int j = 0;

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

Node\* current = newhash->array[i];

while (current != nullptr) {

j++;

cout << " Номер - " << j << endl;

cout << "ФИО:" << current->data.fio << endl;

cout << "Номер телефона:" << current->data.tele << endl;

cout << "Паспорт:" << current->data.passport << endl;

current = current->next;

cout << endl;

}

}

}

int main()

{

srand(time(0));

setlocale(LC\_ALL, "rus");

int size = 0;

while (size < 10) {

cout << "Введите число элементов массива:";

cin >> size;

}

cout << endl;

Human\* array = new Human[size];

hashtable newtable(size);

hashtable\_chain newtable2(size);

fillarray(array, size);

printarray(array, size);

cout << "=======================================================" << endl;

for (int j = 0;j < size;j++) {

newtable.add(array[j], size);

newtable2.add(array[j], size);

}

showhash(&newtable, size);

newtable.findindex("88005553535", size);

system("pause");

newtable.pop("88005553535", size);

cout << "=======================================================" << endl;

showhash(&newtable, size);

cout << "Количество коллизий на " << size << " элементов - " << newtable.collisioncount << endl;

cout << "Это был метод открытой адресации, сейчас будет метод цепочек " << endl;

system("pause");

system("cls");

showhashtable(&newtable2, size);

newtable2.findindex("88005553535", size);

system("pause");

newtable2.pop("88005553535", size);

cout << "=======================================================" << endl;

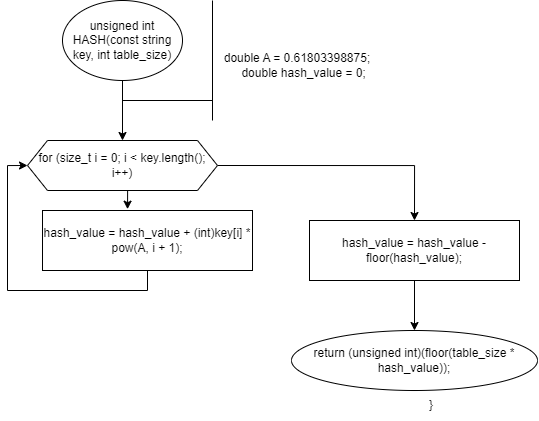
showhashtable(&newtable2, size);

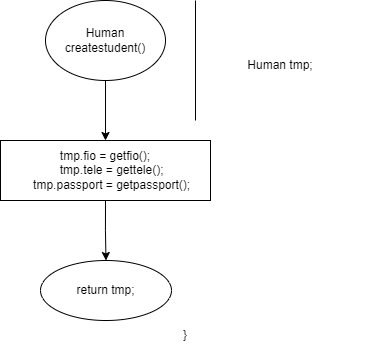
cout << "Количество коллизий на " << size << " элементов - " << newtable2.collisioncount << endl;

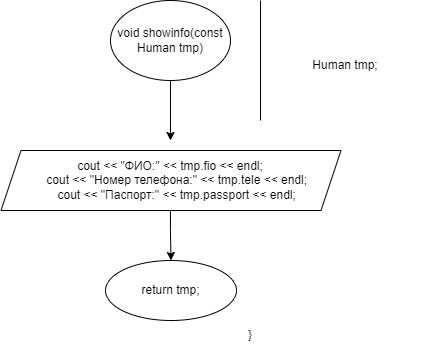
delete[] array;

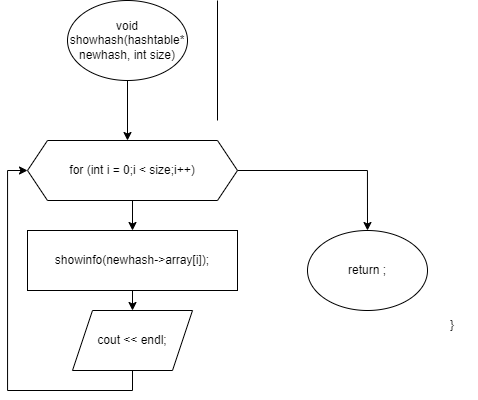
}

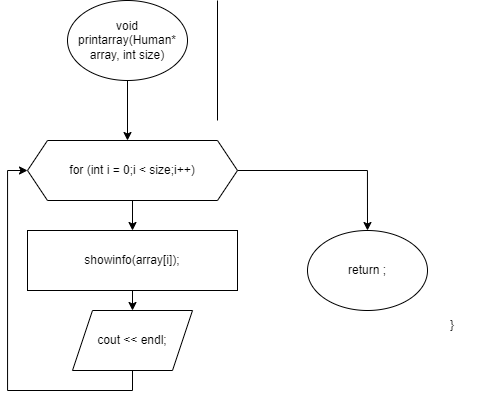
**Блок-схемы**

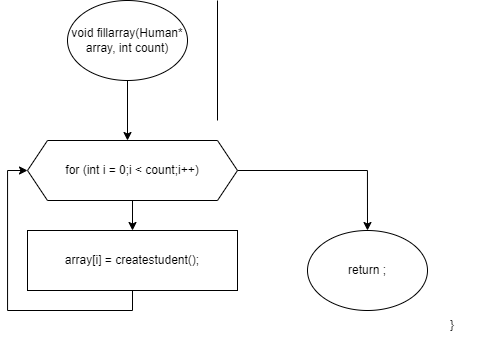
****

****

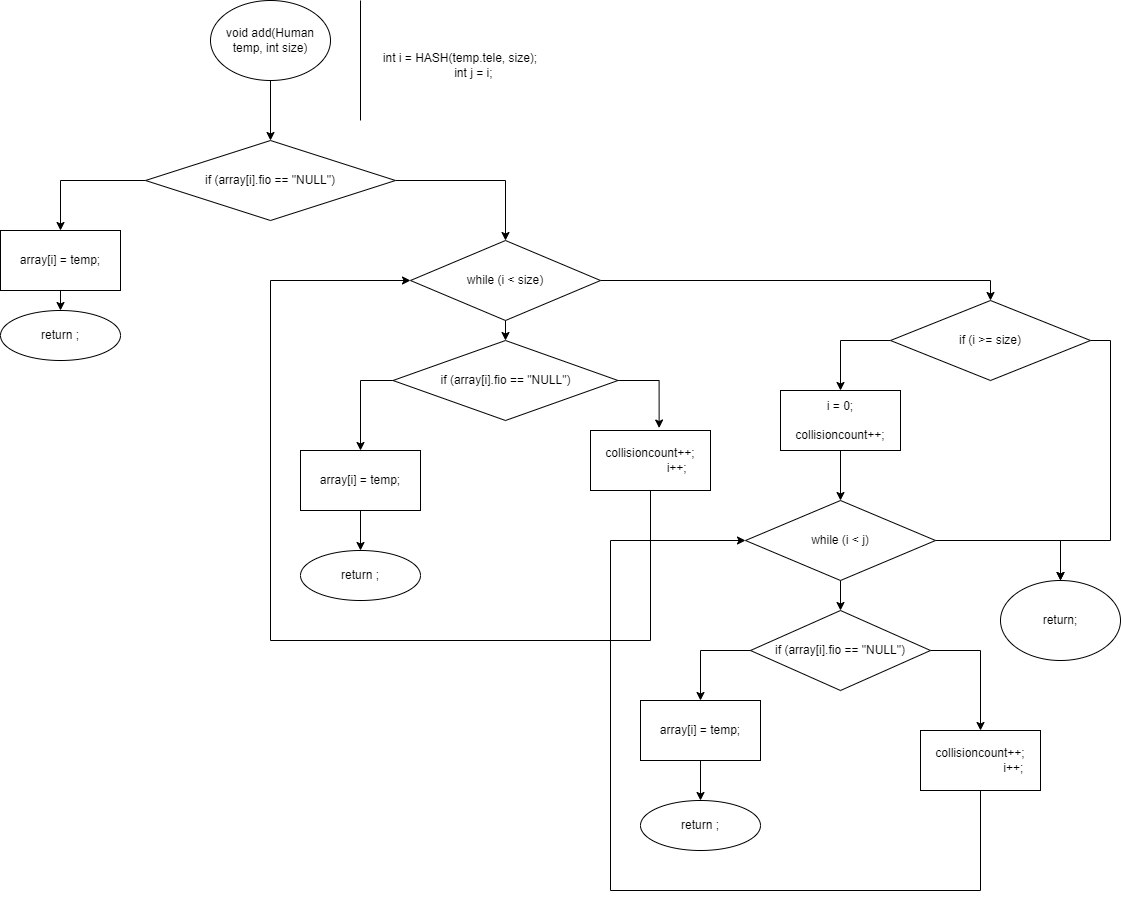
****

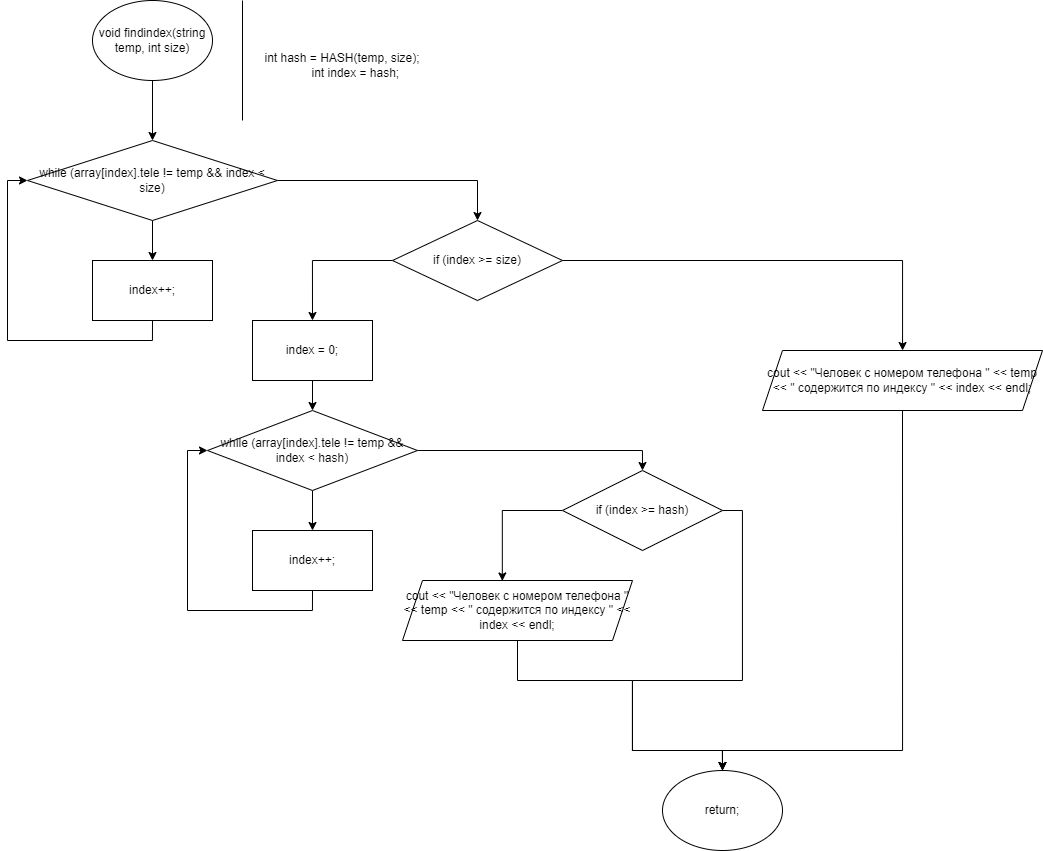
****

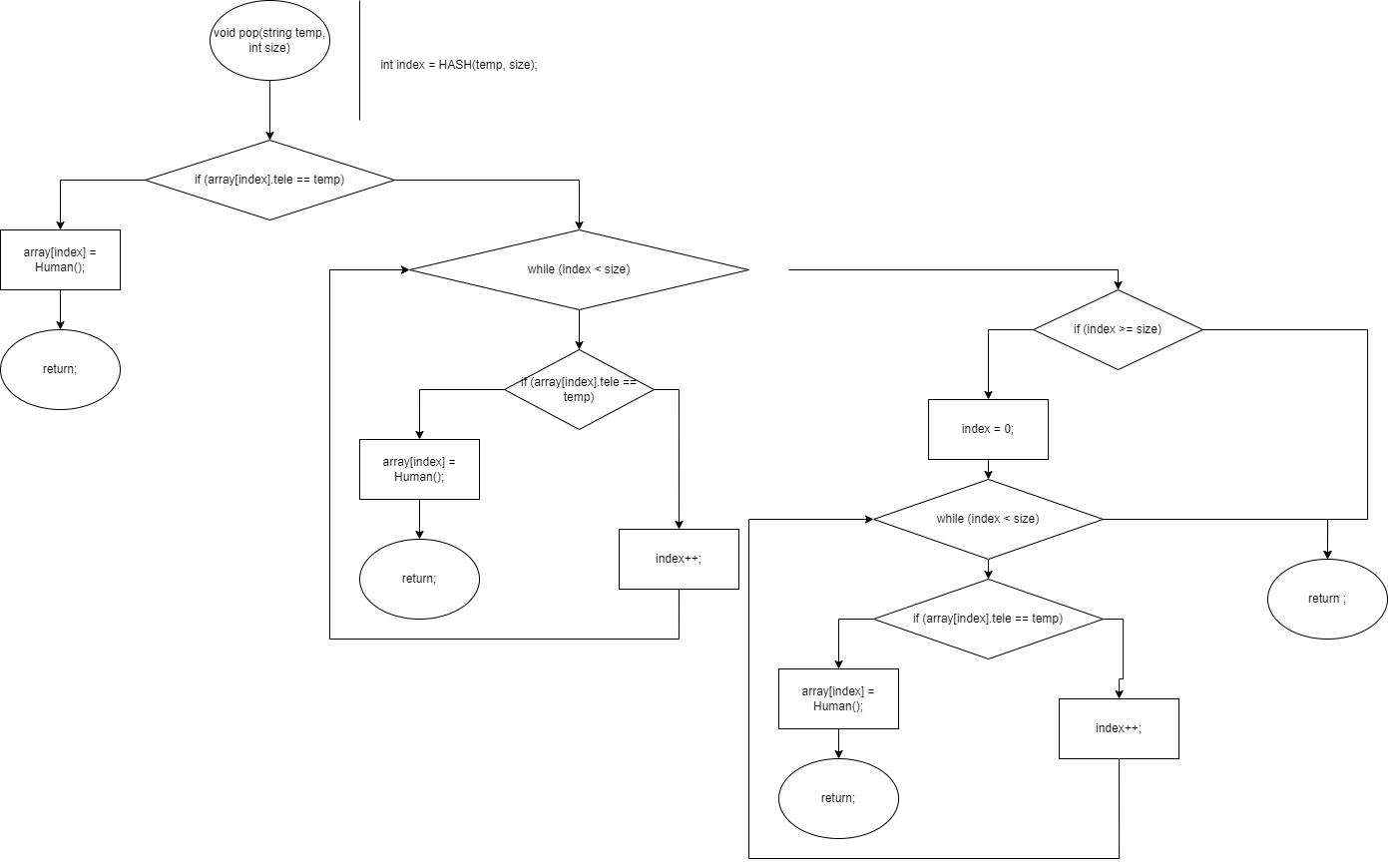
****

****

Функции в структуре hashtable

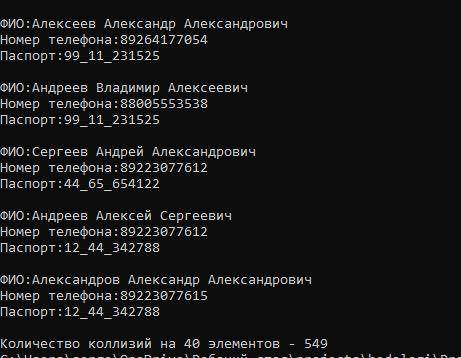


****

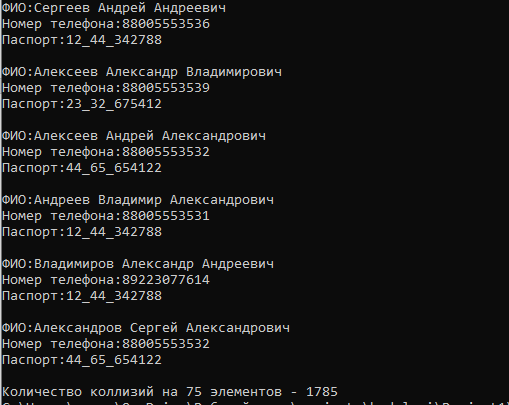
****

**Результат работы программы(метод прямой адресации)**

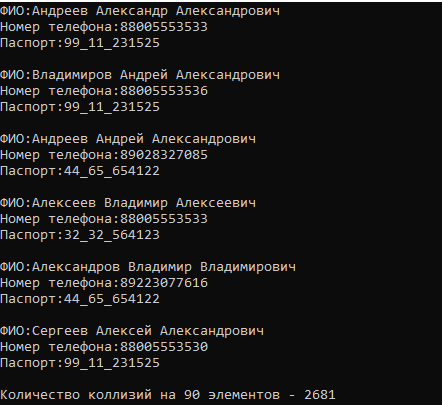
**40**

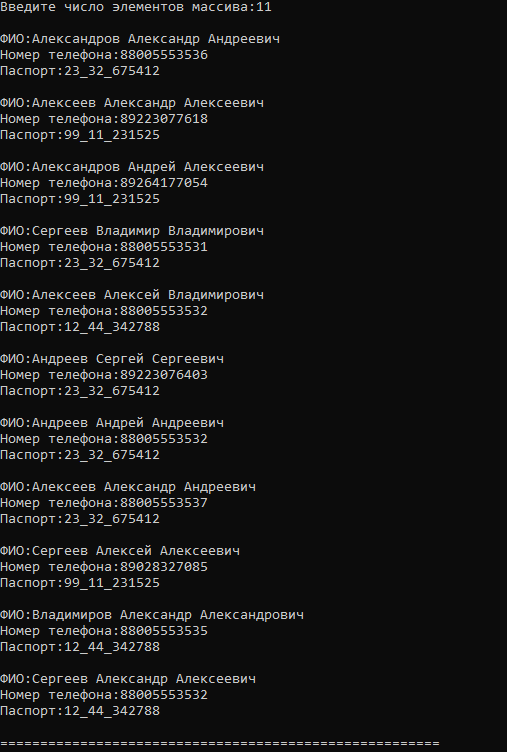


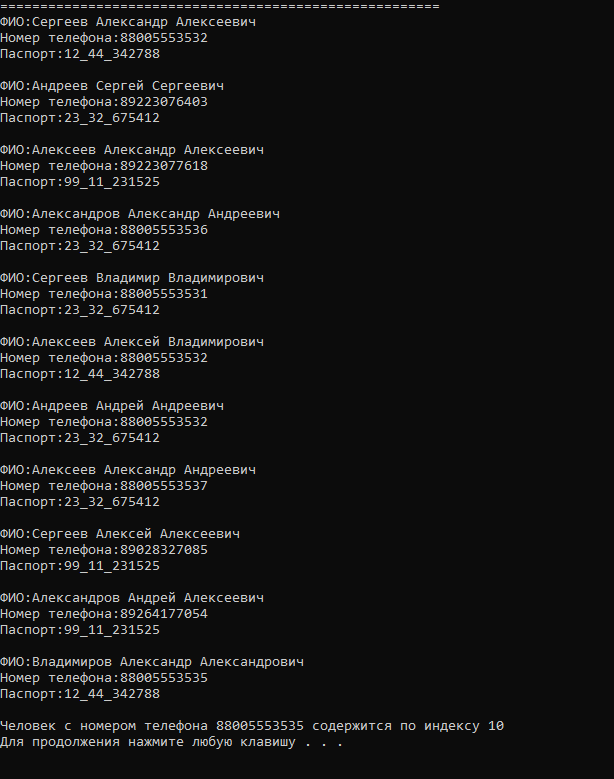
**75**

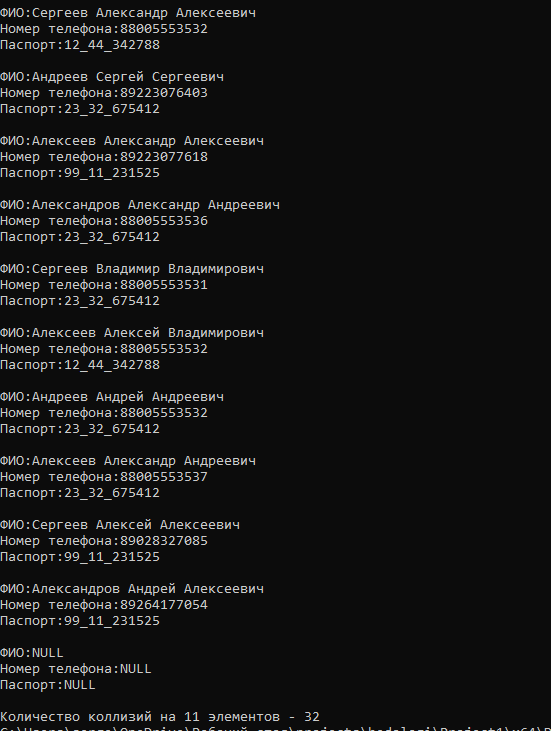


**90**



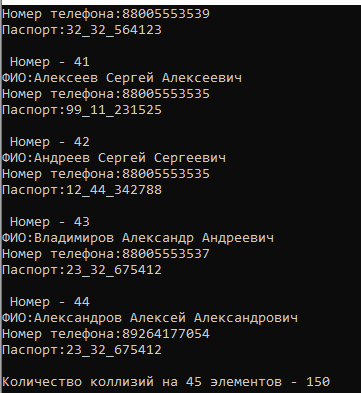


Вывод

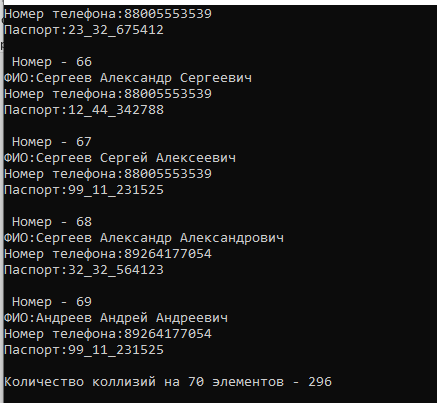
Поиск по заданному ключуУдаление заданного ключа

**Результат работы программы(метод цепочек)**

**45**



**70**



**90**

