Министерство образования и науки Российской Федерации

Федеральное государственное образовательное автономное учреждение высшего образования

"Пермский национальный исследовательский политехнический университет"

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ

Дисциплина: Основы алгоритмизации и программирования

Тема: Хеш функции

|  |  |
| --- | --- |
| Выполнил работу | |
| Студент группы РИС-22-1б | |
| Деревнин И.В. | |
|  | |
| Проверил работу | |
| Доцент кафедры ИТАС | |
| Полякова О.А. | |
|  | |

Пермь – 2023

**Анализ предметной области**

**Постановка задачи**

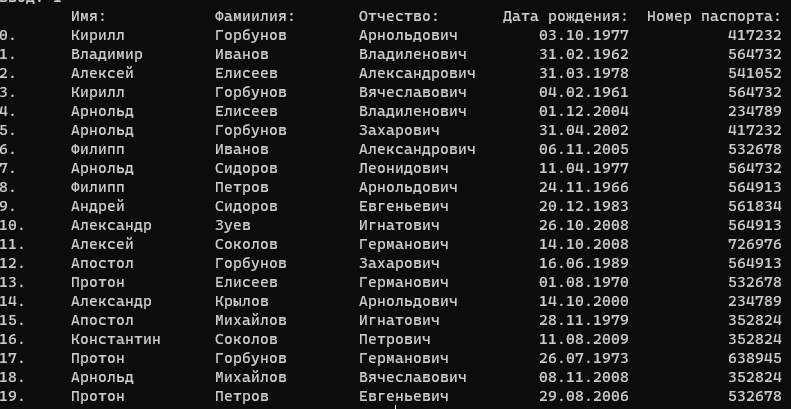
Реализовать программу, которая будет выполнять поиск с помощью хеш-функции, добавлять элементы в изначальный массив данных и удалять элементы по значению, посчитать количество коллизий. Коллизии устранять двумя способами: методом цепочек и методом открытой адресации.

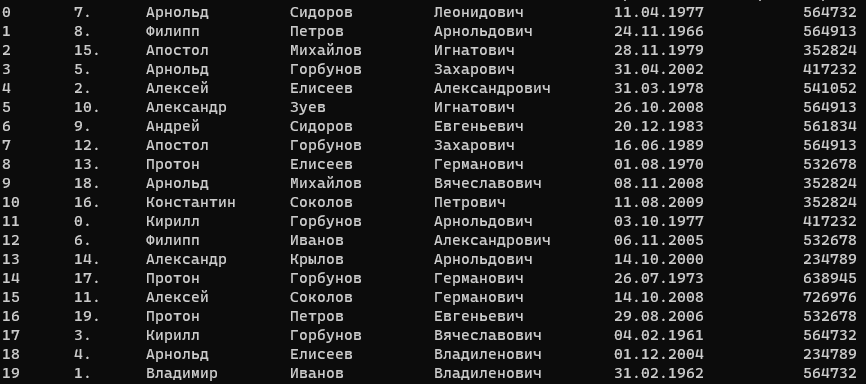
**Анализ задачи**

Для реализации программы будет использована библиотека STL, из нее будут взяты контейнеры: forward\_list – односвязный список для метода цепочек и vector – динамический массив, для хранения данных хешированной таблицы в методе открытой адресации. Для реализации программы необходимо реализовать функцию хеширования, которая будет возвращать индекс, функцию составления хеш-таблицы и поиска в изначальной таблице, для этого необходимо использовать функцию хеширования. Для того, чтобы хранить различный данные необходимо реализовать структуры с полями: фамилия, имя, отчество, дата рождения, которая будет генерироваться случайно, и индекс, чтобы знать положения в изначальной таблице.

**Тестирование программы**

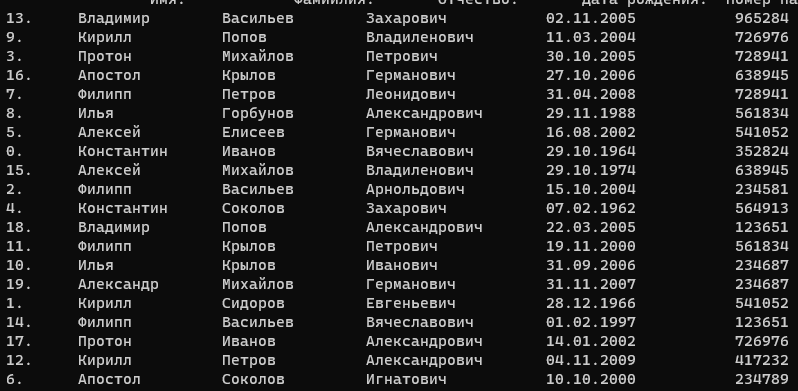
Для того чтобы протестировать программу необходимо сделать тесты для каждой функции.

**

*Рис. 1 – Сгенерированный массив данных.*

*Рис. 2 – Хешированная таблица.*

*Рис. 3 – Количество коллизий.*

*Рис. 4 – Поиск по хеш-таблице.*

*Рис. 5 – Хешированная таблица с применением метода цепочек.*

*Рис. 6 – Количество коллизий с применением метода цепочек.*

*Рис. 7 – Поиск с помощью хеш-функции по односвязному списку.*

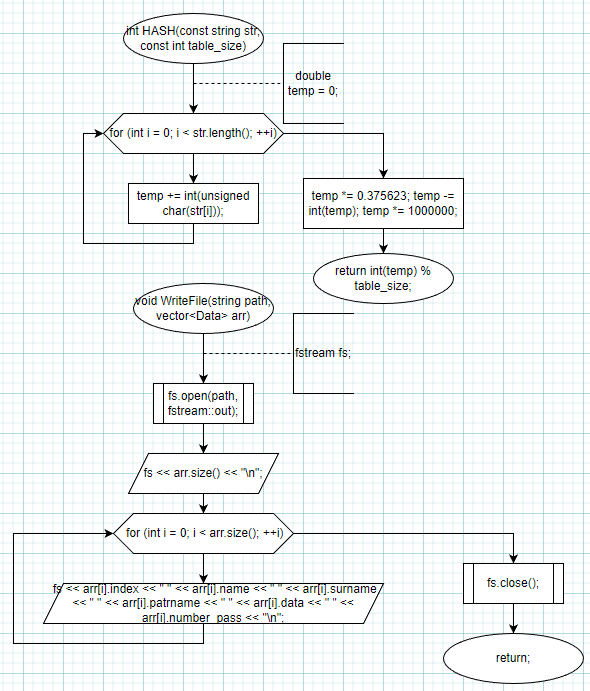
**Заключение**

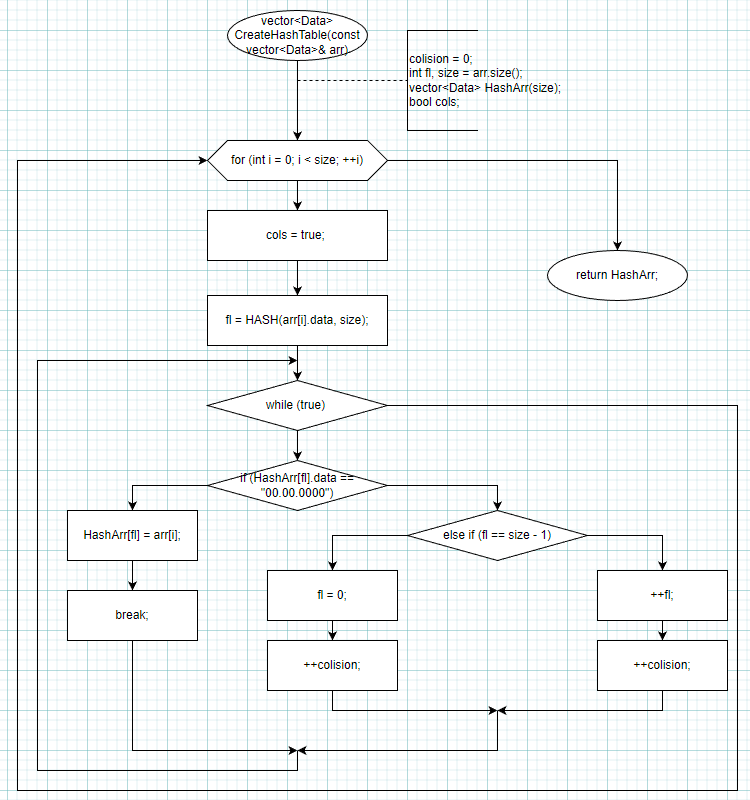
Была разработана программа, которая создает хеш-таблицу, и с помощью нее выполняет поиск, добавление и удаление элементов по значению. Также возможем функционал записи и чтения исходного массива в файл.

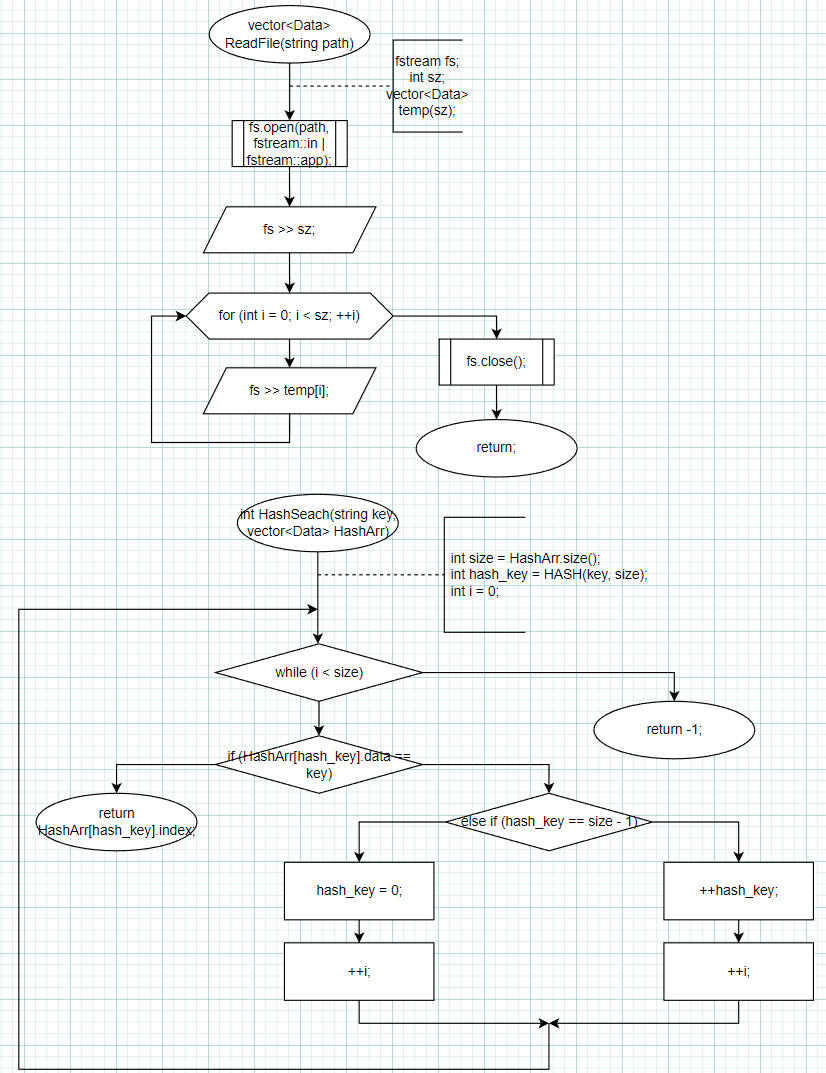
**Приложения**

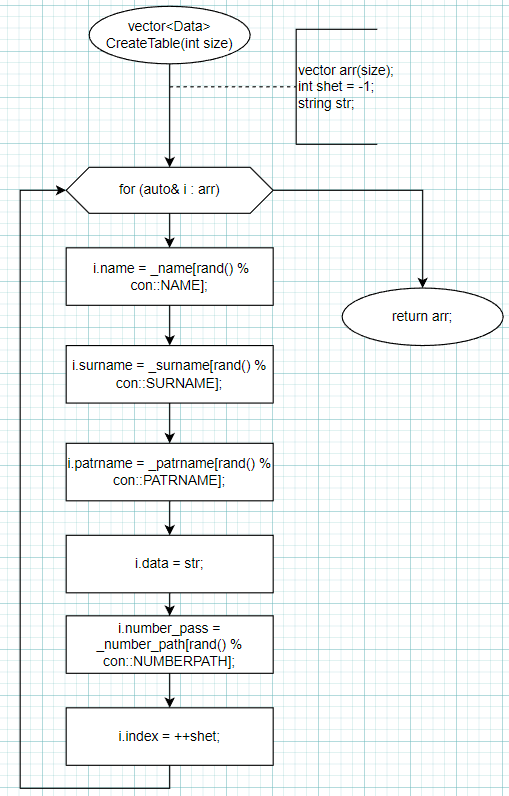
Приложение А – блок схема программы.

Для хеш-поиска с применением метода открытой адресации

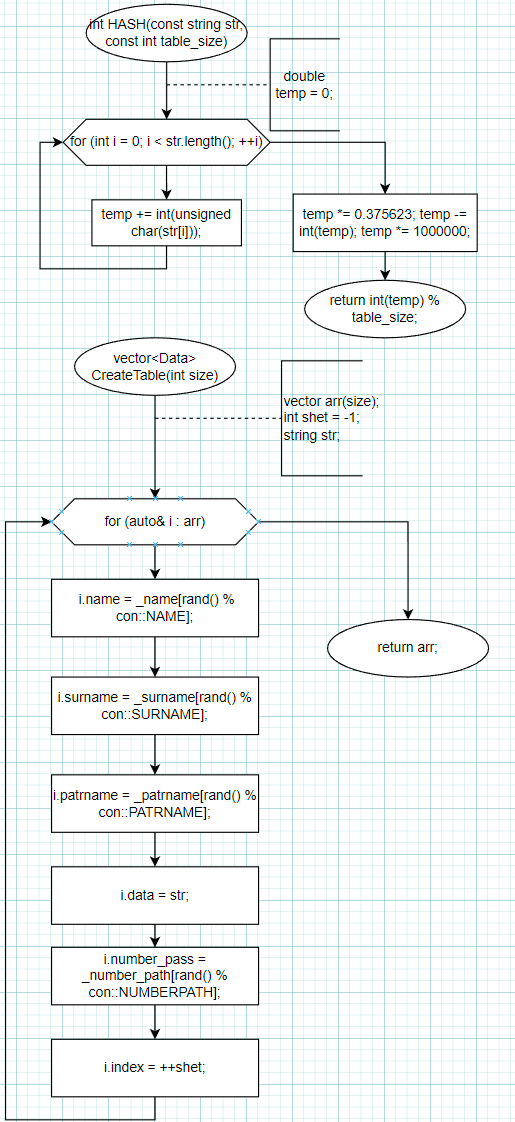


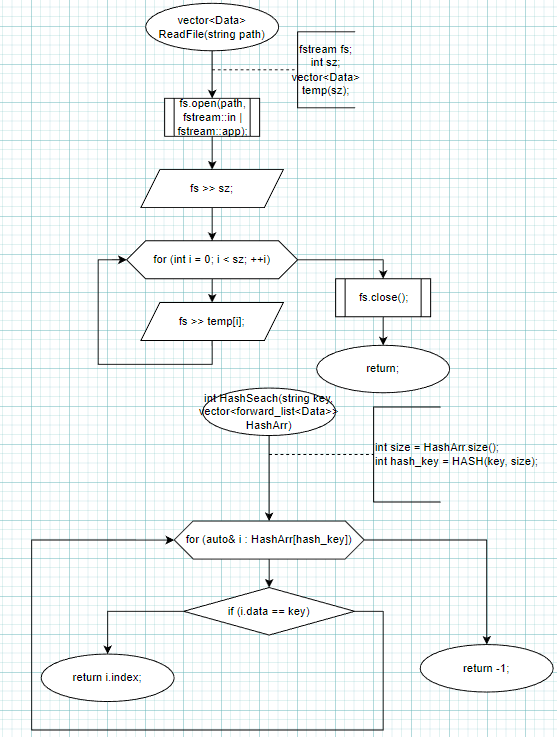


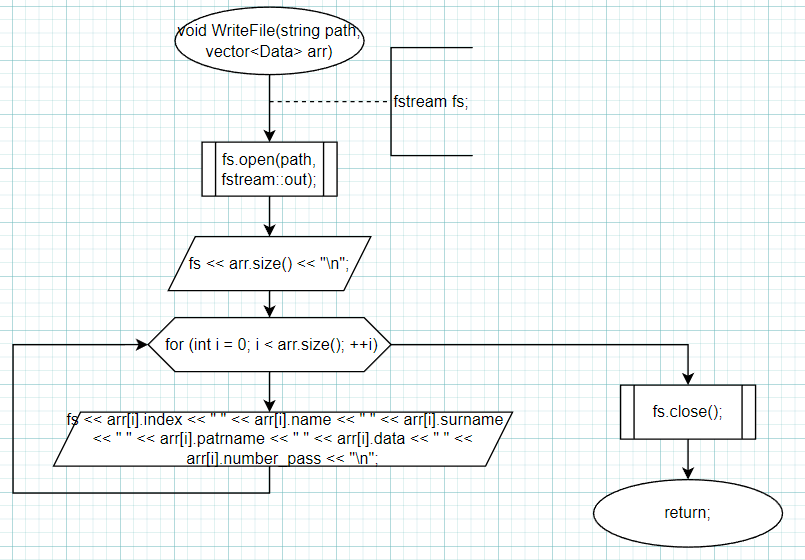


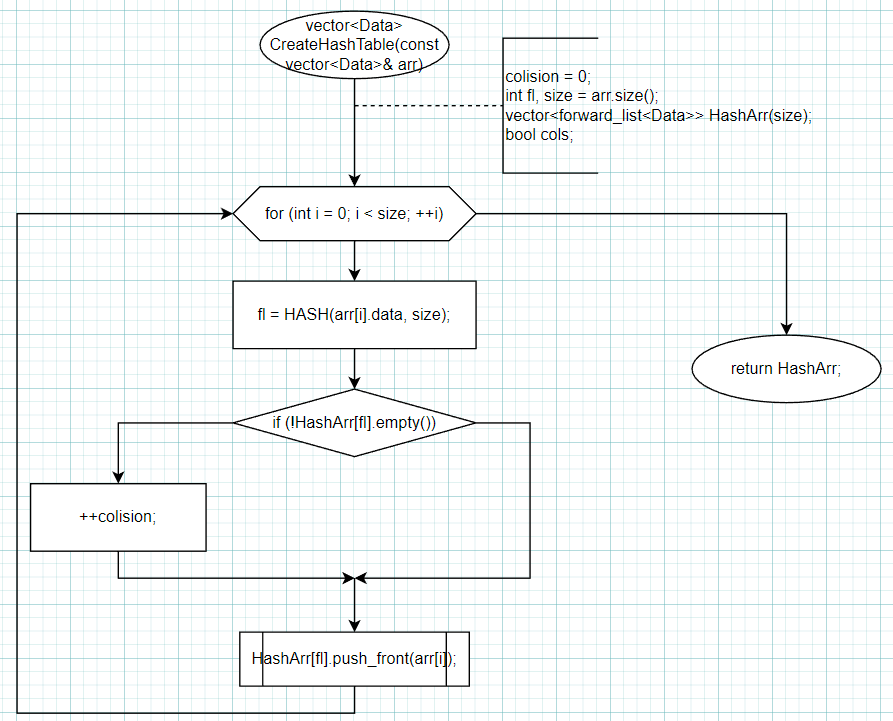


Для хеш-поиска с применением метода цепочек:









Приложение Б – код программы.

Для хеш-поиска с применением метода открытой адресации:

#include <iostream>

#include <vector>

#include <ctime>

#include <Windows.h>

#include <fstream>

constexpr auto tab = "\t";

using namespace std;

enum con //enum для хранения констант, для рандомного заполения данных

{

NAME = 11,

SURNAME = 13,

PATRNAME = 12,

DATA = 11,

NUMBERPATH = 15

};

int colision = 0; //переменная для подсчета коллизий

string \_name[con::NAME] = { "Илья", "Андрей", "Константин", "Александр", "Алексей", "Арнольд", //массивы для рандомного заполения данных о людях

"Кирилл", "Протон", "Владимир", "Апостол", "Филипп" };

string \_surname[con::SURNAME] = { "Иванов", "Петров", "Петров", "Сидоров", "Васильев",

"Попов", "Кузнецов", "Соколов", "Михайлов", "Крылов", "Зуев", "Горбунов", "Елисеев" };

string \_patrname[con::PATRNAME] = { "Вячеславович", "Германович", "Дмитриевич", "Евгеньевич", "Арнольдович",

"Захарович", "Иванович", "Петрович", "Владиленович", "Игнатович", "Леонидович", "Александрович" };

//string \_data[con::DATA] = { "22.01.1979", "25.03.2000", "17.10.1997", "15.06.1980", "03.02.1983",

//"11.11.1999", "15.12.1986", "20.01.2004", "13.12.1993", "12.04.1995", "13.05.1996"};

string \_number\_path[con::NUMBERPATH] = { "564732", "638945", "965284", "532678", "234581", "352824",

"564913", "561834", "234687", "541052", "726976", "417232", "123651", "728941", "234789" };

struct Data //структура для хранения данных о каждом человеке

{

int index; //индекс в начальной таблице

string data; //дата рождения

string name; //имя

string surname; //фамилия

string patrname;//отчество

string number\_pass; //номер пасспорта

Data() //конструктор

{

index = 0;

data = "00.00.0000";

name = "";

surname = "";

patrname = "";

number\_pass = "000000";

}

void Set()

{

string str;

cout << "Имя: "; cin >> name;

cout << "Фамилия: "; cin >> surname;

cout << "Отчество: "; cin >> patrname;

cout << "Дата рождения: "; cin >> data;

cout << "Номер паспорта: "; cin >> number\_pass;

}

private:

friend istream& operator >> (std::istream& in, Data& cl);

friend ostream& operator << (std::ostream& os, Data& cl); //перегруженный оператор вывода

};

int HASH(const string str, const int table\_size) //функция для хеширования данных

{ //логика хеширования из варианта

double temp = 0;

for (int i = 0; i < str.length(); ++i)

{

temp += int(unsigned char(str[i]));

}

temp \*= 0.375623; temp -= int(temp); temp \*= 1000000;

return int(temp) % table\_size;

}

void WriteFile(string path, vector<Data> arr)

{

fstream fs;

fs.open(path, fstream::out);

fs << arr.size() << "\n";

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

{

fs << arr[i].index << " " << arr[i].name << " " << arr[i].surname << " " << arr[i].patrname << " " << arr[i].data << " " << arr[i].number\_pass << "\n";

}

fs.close();

}

vector<Data> ReadFile(string path) //функция записи данных массива в файл

{

fstream fs;

fs.open(path, fstream::in | fstream::app);

int sz;

fs >> sz; //считывания длины массива

vector<Data> temp(sz);

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

{

fs >> temp[i];

}

fs.close();

return temp;

}

vector<Data> CreateTable(int size)

{

vector<Data> arr(size);

int shet = -1;

string str;

char a, b;

for (auto& i : arr)

{

str = ""; //логика генерации рандомной даты рождения.

a = rand() % 4 + 48;

if (a == '3')

b = rand() % 2 + 48;

else if (a == '0')

b = rand() % 9 + 49;

else

b = rand() % 10 + 48;

str = str + a + b + ".";

a = rand() % 2 + 48;

if (a == '1')

b = rand() % 3 + 48;

else

b = rand() % 9 + 49;

str = str + a + b + ".";

a = rand() % 2 + 49;

if (a == '2')

{

b = rand() % 10 + 48;

str = str + a + "00" + b;

}

else

{

str = str + a + '9';

a = rand() % 4 + 54;

b = rand() % 10 + 48;

str = str + a + b;

}

i.name = \_name[rand() % con::NAME]; //заполнение данными структуры

i.surname = \_surname[rand() % con::SURNAME];

i.patrname = \_patrname[rand() % con::PATRNAME];

i.data = str;

i.number\_pass = \_number\_path[rand() % con::NUMBERPATH];

i.index = ++shet;

}

return arr;

}

vector<Data> CreateHashTable(const vector<Data>& arr) //функция генерирущая хешированую таблицу

{

colision = 0;

int fl, size = arr.size();

vector<Data> HashArr(size); //буферный массив

bool cols;

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

{

cols = true; //переменная для подсчета коллизий

fl = HASH(arr[i].data, size);

while (true)

{

if (HashArr[fl].data == "00.00.0000") //поиск свободных ячеек

{

HashArr[fl] = arr[i]; //

break;

}

else if (fl == size - 1) //логика смещения и создания коллизий

{

fl = 0;

if (cols)

{

++colision;

cols = false;

}

}

else

{

++fl;

if (cols)

{

++colision;

cols = false;

}

}

}

}

return HashArr;

}

int HashSeach(string key, vector<Data> HashArr) //поиск по хешированной таблице

{

int size = HashArr.size();

int hash\_key = HASH(key, size);

int i = 0;

while (i < size)

{

if (HashArr[hash\_key].data == key)

{

return HashArr[hash\_key].index; //возврат индекса из основной таблицы

}

else if (hash\_key == size - 1)

{

hash\_key = 0; //смещения в соответствии с колизией

++i;

}

else

{

++hash\_key;

++i;

}

}

return -1; //возврат несуществующего индекса, если ничего не нашли

}

int main()

{

srand(time(0));

system("chcp 1251 >> null");

vector<Data> HashArr;

string key;

int size;

int menu = 1;

cout << "Введите количество людей: ";

cin >> size;

bool Flag = true;

cout << "\033[2J\033[1;1H";

vector<Data> arr = CreateTable(size);

cout << "DONE!" << endl; //меню

do

{

cout << "--------------------МЕНЮ--------------------" << endl;

cout << "1. Вывести таблицу людей" << endl

<< "2. Хешировать таблицу." << endl

<< "3. Вывести хешированную таблицу." << endl

<< "4. Количество коллизий." << endl

<< "5. Поиск." << endl

<< "6. Добавить элемент по индексу." << endl

<< "7. Добавить элемент на место значащего." << endl

<< "8. Удалить элемент по индексу." << endl

<< "9. Удалить элемент по значению." << endl

<< "10. Записать массив в файл." << endl

<< "11. Считать массив из файла." << endl

<< "12. Очистка консоли." << endl

<< "0. Выход." << endl;

cout << "Ввод: "; cin >> menu;

switch (menu)

{

case 1:

{

size = arr.size();

cout << " \tИмя: " << tab << "Фамиилия: " << tab << "Отчество: " << tab << "Дата рождения:" << tab << "Номер паспорта: " << endl;

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

{

cout << arr[i] << endl;

}

system("pause");

break;

}

case 2:

{

size = arr.size();

HashArr = CreateHashTable(arr);

cout << "DONE!" << endl;

break;

}

case 3:

{

size = arr.size();

cout << " \t\tИмя: " << tab << "Фамиилия: " << tab << "Отчество: " << tab << "Дата рождения:" << tab << "Номер паспорта: " << endl;

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

{

cout << i << tab << HashArr[i] << endl;

}

cout << "DONE!" << endl;

break;

}

case 4:

{

cout << "Количество коллизиий: " << colision << endl;

break;

}

case 5:

{

cout << "Введите ключ для поиска: ";

cin >> key;

cout << "Индекс искомого элемента: ";

cout << HashSeach(key, HashArr) << endl;

break;

}

case 6:

{

size = arr.size();

int ind;

cout << "Заполните информацию о новом человеке:" << endl;

Data temp;

temp.Set();

cout << "Введите индекс нового элемента:";

do

{

cin >> ind;

} while (ind > arr.size());

temp.index = ind;

auto i = arr.begin();

advance(i, ind);

arr.insert(i, temp);

++ind;

for (; ind < arr.size(); ++ind)

{

++arr[ind].index;

}

break;

}

case 7:

{

string str\_key;

size = arr.size();

int ind;

cout << "Введите ключ текущего элемента:"; cin >> str\_key;

ind = HashSeach(str\_key, HashArr);

if (ind == -1) { cout << "ERROR!" << endl; break; }

cout << "Заполните информацию о новом человеке:" << endl;

Data temp;

temp.Set();

temp.index = ind;

auto i = arr.begin();

++ind;

advance(i, ind);

arr.insert(i, temp);

for (; ind < arr.size(); ++ind)

{

++arr[ind].index;

}

break;

}

case 8:

{

int ind;

size = arr.size();

cout << "Введите индекс для удаления элемента:";

do

{

cin >> ind;

} while (ind > arr.size());

auto i = arr.begin();

advance(i, ind);

arr.erase(i);

for (; ind < arr.size(); ++ind)

{

--arr[ind].index;

}

break;

}

case 9:

{

int ind;

size = arr.size();

string str\_key;

cout << "Введите ключ для удаления элемента (дата рождения):";

cin >> str\_key;

ind = HashSeach(str\_key, HashArr);

if (ind == -1) { cout << "ERROR!" << endl; break; }

auto i = arr.begin();

if (ind == arr.size()) { arr.pop\_back(); }

else { advance(i, ind); arr.erase(i); }

for (; ind < arr.size(); ++ind)

{

--arr[ind].index;

}

break;

}

case 10:

{

WriteFile("test", arr);

break;

}

case 11:

{

arr = ReadFile("test");

break;

}

case 12: {cout << "\033[2J\033[1;1H"; break;}

case 0:

{

Flag = false;

break;

}

default:

cout << "Попробуйте ввести другую команду." << endl;

break;

}

} while (Flag);

return 0;

}

ostream& operator<<(std::ostream& os, Data& cl)

{

return os << cl.index << "." << tab << cl.name << " " << tab << cl.surname << " " << tab << cl.patrname << tab << " " << cl.data << tab << " " << cl.number\_pass;

}

istream& operator >> (std::istream& in, Data& cl) //перегруженный оператор ввода для моего класса

{

in >> cl.index >> cl.name >> cl.surname >> cl.patrname >> cl.data >> cl.number\_pass; //что и куда читать из потока

return in;

}

Для хеш-поиска с применением метода цепочек:

#include <iostream>

#include <forward\_list>

#include <ctime>

#include <Windows.h>

#include <fstream>

#include <vector>

constexpr auto tab = "\t";

using namespace std;

enum con //enum для хранения констант, для рандомного заполения данных

{

NAME = 11,

SURNAME = 13,

PATRNAME = 12,

DATA = 11,

NUMBERPATH = 15

};

int colision = 0; //переменная для подсчета коллизий

string \_name[con::NAME] = { "Илья", "Андрей", "Константин", "Александр", "Алексей", "Арнольд", //массивы для рандомного заполения данных о людях

"Кирилл", "Протон", "Владимир", "Апостол", "Филипп" };

string \_surname[con::SURNAME] = { "Иванов", "Петров", "Петров", "Сидоров", "Васильев",

"Попов", "Кузнецов", "Соколов", "Михайлов", "Крылов", "Зуев", "Горбунов", "Елисеев" };

string \_patrname[con::PATRNAME] = { "Вячеславович", "Германович", "Дмитриевич", "Евгеньевич", "Арнольдович",

"Захарович", "Иванович", "Петрович", "Владиленович", "Игнатович", "Леонидович", "Александрович" };

//string \_data[con::DATA] = { "22.01.1979", "25.03.2000", "17.10.1997", "15.06.1980", "03.02.1983",

//"11.11.1999", "15.12.1986", "20.01.2004", "13.12.1993", "12.04.1995", "13.05.1996"};

string \_number\_path[con::NUMBERPATH] = { "564732", "638945", "965284", "532678", "234581", "352824",

"564913", "561834", "234687", "541052", "726976", "417232", "123651", "728941", "234789" };

struct Data //структура для хранения данных о каждом человеке

{

int index; //индекс в начальной таблице

string data; //дата рождения

string name; //имя

string surname; //фамилия

string patrname;//отчество

string number\_pass; //номер пасспорта

Data() //конструктор

{

index = 0;

data = "00.00.0000";

name = "";

surname = "";

patrname = "";

number\_pass = "000000";

}

void Set()

{

string str;

cout << "Имя: "; cin >> name;

cout << "Фамилия: "; cin >> surname;

cout << "Отчество: "; cin >> patrname;

cout << "Дата рождения: "; cin >> data;

cout << "Номер паспорта: "; cin >> number\_pass;

}

private:

friend istream& operator >> (std::istream& in, Data& cl);

friend ostream& operator << (std::ostream& os, Data& cl); //перегруженный оператор вывода

};

int HASH(const string str, const int table\_size) //функция для хеширования данных

{ //логика хеширования из варианта

double temp = 0;

for (int i = 0; i < str.length(); ++i)

{

temp += int(unsigned char(str[i]));

}

temp \*= 0.375623; temp -= int(temp); temp \*= 1000000;

return int(temp) % table\_size;

}

vector<Data> CreateTable(int size)

{

vector<Data> arr(size);

int shet = -1;

string str;

char a, b;

for (auto& i : arr)

{

str = ""; //логика генерации рандомной даты рождения.

a = rand() % 4 + 48;

if (a == '3')

b = rand() % 2 + 48;

else if (a == '0')

b = rand() % 9 + 49;

else

b = rand() % 10 + 48;

str = str + a + b + ".";

a = rand() % 2 + 48;

if (a == '1')

b = rand() % 3 + 48;

else

b = rand() % 9 + 49;

str = str + a + b + ".";

a = rand() % 2 + 49;

if (a == '2')

{

b = rand() % 10 + 48;

str = str + a + "00" + b;

}

else

{

str = str + a + '9';

a = rand() % 4 + 54;

b = rand() % 10 + 48;

str = str + a + b;

}

i.name = \_name[rand() % con::NAME]; //заполнение данными структуры

i.surname = \_surname[rand() % con::SURNAME];

i.patrname = \_patrname[rand() % con::PATRNAME];

i.data = str;

i.number\_pass = \_number\_path[rand() % con::NUMBERPATH];

i.index = ++shet;

}

return arr;

}

vector<forward\_list<Data>> CreateHashTable(const vector<Data>& arr) //функция генерирущая хешированую таблицу

{

colision = 0;

int fl, size = arr.size();

vector<forward\_list<Data>> HashArr(size); //запись данных в односвязный список

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

{

fl = HASH(arr[i].data, size); //поиск индекса

if (!HashArr[fl].empty())

++colision;

HashArr[fl].push\_front(arr[i]);

}

return HashArr;

}

int HashSeach(string key, vector<forward\_list<Data>> HashArr) //поиск по хешированной таблице

{

int size = HashArr.size();

int hash\_key = HASH(key, size);

for (auto& i : HashArr[hash\_key])

{

if (i.data == key)

{

return i.index;

}

}

return -1; //возврат несуществующего индекса, если ничего не нашли

}

void WriteFile(string path, vector<Data> arr)

{

fstream fs;

fs.open(path, fstream::out);

fs << arr.size() << "\n";

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

{

fs << arr[i].index << " " << arr[i].name << " " << arr[i].surname << " " << arr[i].patrname << " " << arr[i].data << " " << arr[i].number\_pass << "\n";

}

fs.close();

}

vector<Data> ReadFile(string path) //функция записи данных массива в файл

{

fstream fs;

fs.open(path, fstream::in | fstream::app);

int sz;

fs >> sz; //считывания длины массива

vector<Data> temp(sz);

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

{

fs >> temp[i];

}

fs.close();

return temp;

}

int main()

{

srand(time(0));

system("chcp 1251 >> null");

vector<forward\_list<Data>> HashArr;

string key;

int size;

int menu = 1;

cout << "Введите количество людей: ";

cin >> size;

bool Flag = true;

cout << "\033[2J\033[1;1H";

vector<Data> arr = CreateTable(size);

cout << "DONE!" << endl; //меню

do

{

cout << "--------------------МЕНЮ--------------------" << endl;

cout << "1. Вывести таблицу людей" << endl

<< "2. Хешировать таблицу." << endl

<< "3. Вывести хешированную таблицу." << endl

<< "4. Количество коллизий." << endl

<< "5. Поиск." << endl

<< "6. Добавить элемент по индексу." << endl

<< "7. Добавить элемент на место значащего." << endl

<< "8. Удалить элемент по индексу." << endl

<< "9. Удалить элемент по значению." << endl

<< "10. Записать массив в файл." << endl

<< "11. Считать массив из файла." << endl

<< "12. Очистка консоли." << endl

<< "0. Выход." << endl;

cout << "Ввод: "; cin >> menu;

switch (menu)

{

case 1:

{

size = arr.size();

cout << " \tИмя: " << tab << "Фамиилия: " << tab << "Отчество: " << tab << "Дата рождения:" << tab << "Номер паспорта: " << endl;

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

{

cout << arr[i] << endl;

}

system("pause");

break;

}

case 2:

{

size = arr.size();

HashArr = CreateHashTable(arr);

cout << "DONE!" << endl;

break;

}

case 3:

{

size = arr.size();

cout << " \t\tИмя: " << tab << "Фамиилия: " << tab << "Отчество: " << tab << "Дата рождения:" << tab << "Номер паспорта: " << endl;

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

{

for (auto& i : HashArr[i])

{

cout << i << endl;

}

}

cout << "DONE!" << endl;

break;

}

case 4:

{

cout << "Количество коллизиий: " << colision << endl;

break;

}

case 5:

{

cout << "Введите ключ для поиска: ";

cin >> key;

cout << "Индекс искомого элемента: ";

cout << HashSeach(key, HashArr) << endl;

break;

}

case 6:

{

size = arr.size();

int ind;

cout << "Заполните информацию о новом человеке:" << endl;

Data temp;

temp.Set();

cout << "Введите индекс нового элемента:";

do

{

cin >> ind;

} while (ind > arr.size());

temp.index = ind;

auto i = arr.begin();

advance(i, ind);

arr.insert(i, temp);

++ind;

for (; ind < arr.size(); ++ind)

{

++arr[ind].index;

}

break;

}

case 7:

{

string str\_key;

size = arr.size();

int ind;

cout << "Введите ключ текущего элемента:"; cin >> str\_key;

ind = HashSeach(str\_key, HashArr);

if (ind == -1) { cout << "ERROR!" << endl; break; }

cout << "Заполните информацию о новом человеке:" << endl;

Data temp;

temp.Set();

temp.index = ind;

auto i = arr.begin();

++ind;

advance(i, ind);

arr.insert(i, temp);

for (; ind < arr.size(); ++ind)

{

++arr[ind].index;

}

break;

}

case 8:

{

int ind;

size = arr.size();

cout << "Введите индекс для удаления элемента:";

do

{

cin >> ind;

} while (ind > arr.size());

auto i = arr.begin();

advance(i, ind);

arr.erase(i);

for (; ind < arr.size(); ++ind)

{

--arr[ind].index;

}

break;

}

case 9:

{

int ind;

size = arr.size();

string str\_key;

cout << "Введите ключ для удаления элемента (дата рождения):";

cin >> str\_key;

ind = HashSeach(str\_key, HashArr);

if (ind == -1) { cout << "ERROR!" << endl; break; }

auto i = arr.begin();

if (ind == arr.size()) { arr.pop\_back(); }

else { advance(i, ind); arr.erase(i); }

for (; ind < arr.size(); ++ind)

{

--arr[ind].index;

}

break;

}

case 10:

{

WriteFile("test.txt", arr);

break;

}

case 11:

{

arr = ReadFile("test.txt");

break;

}

case 12: {cout << "\033[2J\033[1;1H"; break;}

case 0:

{

Flag = false;

break;

}

default:

cout << "Попробуйте ввести другую команду." << endl;

break;

}

} while (Flag);

return 0;

}

ostream& operator<<(std::ostream& os, Data& cl)

{

return os << cl.index << "." << tab << cl.name << " " << tab << cl.surname << " " << tab << cl.patrname << tab << " " << cl.data << tab << " " << cl.number\_pass;

}

istream& operator >> (std::istream& in, Data& cl) //перегруженный оператор ввода для моего класса

{

in >> cl.index >> cl.name >> cl.surname >> cl.patrname >> cl.data >> cl.number\_pass; //что и куда читать из потока

return in;

}