МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

ЭЛЕКТРОТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА «ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И АВТОМАТИЗИРОВАННЫХ СИСТЕМ»

**ОТЧЁТ**

**«ЛАБОРАТОРНАЯ №15:**

**ХЕШ-ТАБЛИЦА»**

Дисциплина: «Программирование»

Выполнил:

Студент группы ИВТ-21-2б

Безух Владимир Сергеевич

Проверил:

Доцент кафедры ИТАС

Полякова Ольга Андреевна

Пермь, 2022

Содержание

[1. Постановка задачи 3](#_Toc99979279)

[2. Исходный код 4](#_Toc99979280)

[3. Анализ результатов 8](#_Toc99979281)

# Постановка задачи

1. Заполнить хеш-таблицу 100 записей (ФИО, дата рождения, номер телефона, номер паспорта).
2. В качестве ключа использовать дату рождения.
3. Подсчитать количество коллизий при размере хеш-таблицы 40, 75, 90. Для разрешения коллизий использовать метод цепочек.

# 

# Исходный код

#include <list>

#include <cmath>

#include <vector>

#include <fstream>

#include <iostream>

struct FullName

{

std::string surname;

std::string forename;

std::string patronym;

bool operator==(const FullName& right) const

{

return (surname == right.surname) &&

(forename == right.forename) &&

(patronym == right.patronym);

}

};

struct Date

{

unsigned short day;

unsigned short month;

unsigned short year;

bool operator==(const Date& right) const

{

return (day == right.day) &&

(month == right.month) &&

(year == right.year);

}

bool operator!=(const Date& right) const {

return !(\*this == right);

}

bool operator<(const Date& right) const {

if (year < right.year) return true;

else if (year == right.year)

{

if (month < right.month) return true;

else if (month == right.month) return day < right.day;

else return false;

}

else return false;

}

};

struct Person

{

FullName personal\_name;

Date date\_of\_birth;

std::string mobile\_number;

std::string passport\_series;

std::string passport\_id;

bool operator==(const Person& right) const

{

return (personal\_name == right.personal\_name) &&

(date\_of\_birth == right.date\_of\_birth) &&

(mobile\_number == right.mobile\_number) &&

(passport\_series == right.passport\_series) &&

(passport\_id == right.passport\_id);

}

friend std::istream& operator>>(std::istream& input, Person& person)

{

input >> person.personal\_name.surname;

input >> person.personal\_name.forename;

input >> person.personal\_name.patronym;

input >> person.date\_of\_birth.day;

input >> person.date\_of\_birth.month;

input >> person.date\_of\_birth.year;

input >> person.mobile\_number;

input >> person.passport\_series;

input >> person.passport\_id;

return input;

}

friend std::ostream& operator<<(std::ostream& output, const Person& person)

{

output << person.personal\_name.surname << ' ';

output << person.personal\_name.forename << ' ';

output << person.personal\_name.patronym << ' ';

output << person.date\_of\_birth.day << ' ';

output << person.date\_of\_birth.month << ' ';

output << person.date\_of\_birth.year << ' ';

output << person.mobile\_number << ' ';

output << person.passport\_series << ' ';

output << person.passport\_id << ' ';

return output;

}

};

template <class K, class D>

class HashTable

{

public:

HashTable(const int& size);

void insertItem(const K& key, const D& data);

void deleteItem(const K& key, const D& data);

void displayHash();

private:

struct HashTablePair

{

HashTablePair(K key = K(), D data = D())

: key\_(key), data\_(data) {}

HashTablePair(const HashTablePair& copy)

: key\_(copy.key\_), data\_(copy.data\_) {}

HashTablePair& operator=(const HashTablePair& right) {

if (this != &right) {

key\_ = right.key\_;

data\_ = right.data\_;

}

return \*this;

}

K key\_;

D data\_;

};

size\_t hashFunction(const K& key) {

return static\_cast<size\_t>(static\_cast<double>(size\_) \* fmod(key \* 0.618033, 1)) % size\_;

}

size\_t size\_;

size\_t number\_of\_collisions\_;

std::list<HashTablePair>\* table\_;

};

template <class K, class D>

HashTable<K, D>::HashTable(const int& size)

{

size\_ = size;

table\_ = new std::list<HashTablePair>[size\_];

number\_of\_collisions\_ = 0;

}

template <class K, class D>

void HashTable<K, D>::insertItem(const K& key, const D& data)

{

size\_t index = hashFunction(key);

if (!table\_[index].empty())

++number\_of\_collisions\_;

table\_[index].emplace\_back(HashTablePair(key, data));

}

template <class K, class D>

void HashTable<K, D>::deleteItem(const K& key, const D& data)

{

int index = hashFunction(key);

std::list<HashTablePair>::template iterator i;

for (i = table\_[index].begin(); i != table\_[index].end(); ++i)

if (i->data\_ == data)

break;

if (i != table\_[index].end())

table\_[index].erase(i);

if (!table\_[index].empty())

--number\_of\_collisions\_;

}

template <class K, class D>

void HashTable<K, D>::displayHash()

{

for (size\_t i = 0; i != size\_; ++i)

{

std::cout << "Table[" << i << "]:\n";

for (auto x : table\_[i])

std::cout << " --> " << x.data\_ << std::endl;

std::cout << std::endl;

}

std::cout << "Number of collisions: " << number\_of\_collisions\_ << "\n\n";

}

int main()

{

setlocale(LC\_ALL, "Russian");

std::vector<size\_t> keys;

std::vector<Person> persons;

std::fstream f\_input("data.txt");

while (!f\_input.eof())

{

Person person;

f\_input >> person;

keys.emplace\_back(person.date\_of\_birth.day \* size\_t{1000000} + person.date\_of\_birth.month \* size\_t{10000} + person.date\_of\_birth.year);

persons.emplace\_back(person);

}

HashTable<int, Person> ht40(size\_t{40});

HashTable<int, Person> ht75(size\_t{75});

HashTable<int, Person> ht90(size\_t{90});

for (size\_t i = 0; i != size\_t{100}; ++i) {

ht40.insertItem(keys[i], persons[i]);

ht75.insertItem(keys[i], persons[i]);

ht90.insertItem(keys[i], persons[i]);

}

ht40.displayHash();

ht75.displayHash();

ht90.displayHash();

}

# Анализ результатов

Результаты работы программы (рис. 1). С увеличением размера хеш-таблицы количество коллизий снижается.

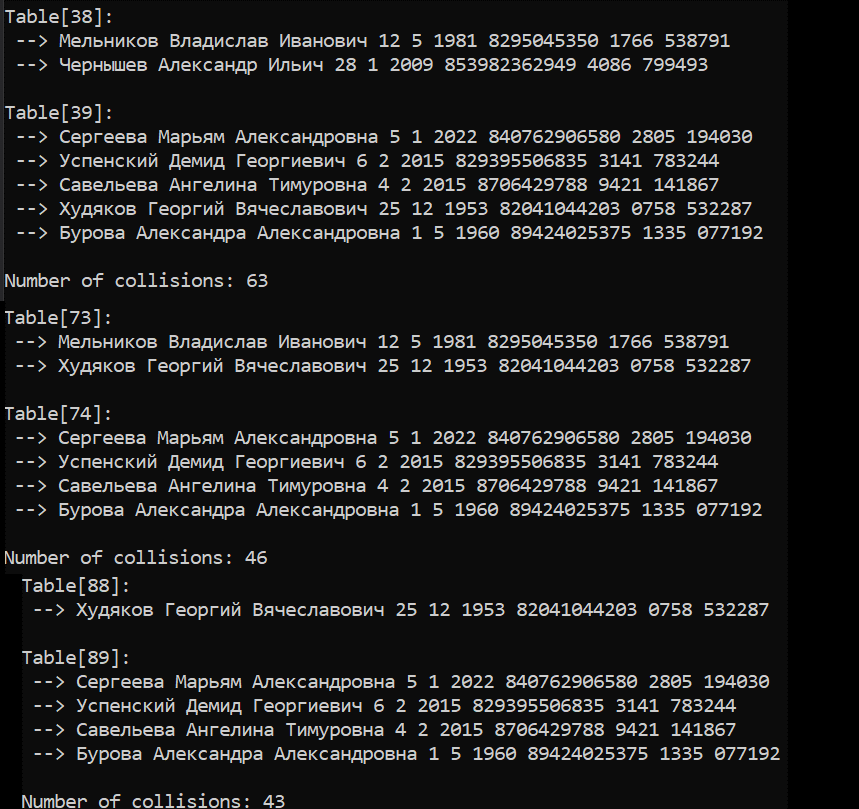


Рисунок — Результаты