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

2. Исходный код..... 4

3. Анализ результатов..... 8

1. 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.

2. 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();
}
```

3. Анализ результатов

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

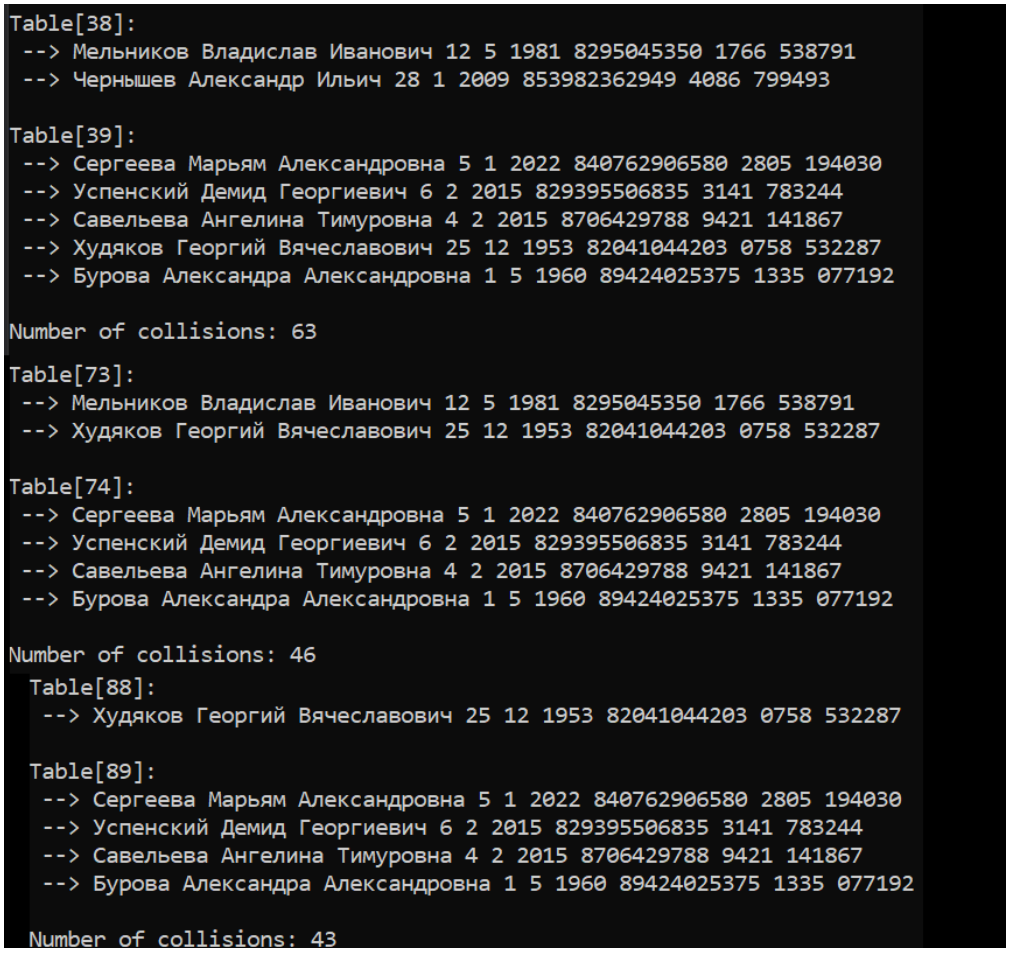

Рисунок 1 — Результаты