**Министерство образования и науки Российской Федерации**

**Пермское федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский Национальный Исследовательский Политехнический Университет»**

**Электротехнический факультет**

**Кафедра «Информационные технологии и автоматизированные системы»**

**ОТЧЁТ**

По лабораторной работе №17 на тему

«Алгоритмы хеширования»

Вариант №11

Выполнил студент группы ИВТ-20-2б

Сабуров Павел Алексеевич

Проверил доцент кафедры ИТАС

Полякова О.А.

Пермь 2021

**Цель работы** –изучить построение функции хеширования и алгоритмов хеширования данных, научиться разрабатывать алгоритмы открытого хеширования при решении задачи (в соответствии с вариантом) на языке C++.

**Постановка задачи**

Задача – реализовать на языке программирования C++ программу, содержащую следующие элементы функции:

1. Функцию хеширования;
2. Множество элементов;
3. Хеш-таблицу;
4. Функцию поиска в хеш-таблице;
5. Возможность ввода элементов с клавиатуры или с файла;
6. Меню управления программой;

**Исходные данные для варианта №11:**

* Метод хеширования – остаток от деления;
* Метод разрешения коллизий – метод открытой адресации;
* Структура «Человек»

Структура «Человек»:

* ФИО (Полное имя);
* Номер паспорта;
* Номер телефона;

Хеширование и поиск осуществляется по номеру паспорта.

**Анализ задачи**

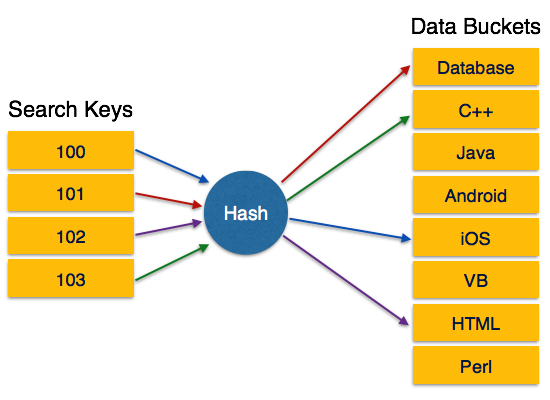
Для решения задачи были использованы следующие средства:

1. Язык программирования C++ (Microsoft Visual C++)
2. Текстовый редактор Microsoft Visual Studio Code

**Хеширование:**

Хеширование – это процесс превращения некоторых данных в некоторую битовую последовательность фиксированной длины на выходе (например: слово переводится в число). Такое слово можно быстро отличить от других, составленных с помощью того же алгоритма хеширования.

Таким образом, результат работы хеш-функции помогает различать одни данные от других, приводя в сравнение только значения функций хеширования, обработавших намного меньше информации. То есть, зная ключ поиска, (если элемент существует) можно мгновенно найти искомый элемент.



Однако, поскольку длина входных данных, как правило больше, чем длина выходных данных, на выходе могут получиться одни и те же выходные данные для разных входных данных. Такое явление называют *коллизией*.

В таком случае алгоритм хеширования должен стремиться к тому, чтобы разные данные имели разные значения – свести количество коллизий к нулю. В редких случаях коллизии в принципе неизбежны.

**Метод Остатков от деления:**

Данный метод считается одним из самых простых. Оно представляет собой деление по модулю числового значения ключа на некоторое число. Результат такой хеш-функции – число, индекс в хеш-таблице.

Однако такой метод имеет серьёзный недостаток: данный метод в процессе работы порождает большое количество коллизий, когда «хорошая» хеш-функция стремится свести их количество к нулю.

**Разрешение коллизий:**

Коллизии необходимо разрешать, потому что они осложняют использование хеш-таблицы, нарушая однозначность соответствия между данными и их хешированными значениями.

Одним из методов разрешение коллизий является **метод открытой адресации**. При открытой адресации если ячейка с вычисленным индексом занята, то просматриваются следующие записи таблицы по порядку до тех пор, пока не будет найден ключ или пуская позиция в таблице.

При любом методе разрешения коллизий необходимо ограничить длину поиска элемента. Если для поиска элемента необходимо более 3-4 сравнений, то эффективность такой хеш-таблицы пропадает и её следует изменить или поменять на другую.

**Реализация программы:**

Для использования операторов ввода-вывода необходимо подключить два заголовочных файла:

1. iostream, содержащий объекта потоков ввода-вывода: std::cin и std::cout;
2. fstream, содержащий классы для работы с файлами std::ifstream и std::ofstream
3. string, позволяющий работать со строками;

Для работа со списками необходимо подключить:

1. list, для работы со списками;
2. iterator, для работы с отдельными элементами списка;

Также, для удобства было подключено пространство имён std, который в себе содержит описанные выше потоки ввода-вывода. Заголовочные файлы подключаются при помощи директивы #include.

#include <iostream>

#include <fstream>

#include <string>

#include <list>

#include <iterator>

using namespace std;

Сама структура, описанная в постановке задача объявляется следующим образом:

struct Person

{

string FullName;

string PassportNumber;

string PhoneNumber;

};

Условимся, что пустой элементы структуры будет равен значению {empty}, который объявлен с помощью директивы #define.

#define EMPTY "{empty}

Для удобной реализации функций ввода с клавиатуры, было решено реализовать отдельные функции ввода натуральных чисел и строки (в том числе содержащей пробелы).

string ReadLine()

{

string x;

getline(cin, x);

return x;

}

int ReadNaturalNum()

{

int x;

do cin >> x; while (x <= 0);

return x;

}

**Реализация меню:**

Функция, отвечающая за работу меню, будет работать в цикле while, а её результатом работы служит флажок, отвечающий на вопрос: «Продолжать вы выполнение программы далее?». true – продолжать, false – завершить.

Главная функция выглядит следующим образом:

int main()

{

while (AcceptCommand());

return 0;

}

Само же меню (функция AcceptCommand()) работает по следующему правилу:

1. Принимается на ввод с клавиатуры строка – команда;
2. Через if-else введённое значение сравнивается с названиями команд;
3. Если строки совпали, то соответствующая команда выполняется;
4. Если же совпадений не нашлось, то выводится сообщение об ошибке и просьба нового ввода команды;

Меню содержит следующие команды:

1. Help – выводит все команды меню;
2. 0, exit, quit – завершение выполнения программы;
3. Clear – очистить список людей и хеш-таблицу;
4. Input – ввод элементов списка с людьми;
5. Print – вывести на экран всех людей;
6. PrintHash – вывести на экран все элемента хеш-таблицы;
7. Hash – создать хеш-таблицу;
8. Search – произвести поиск по хеш-таблице;

Для работы с самим списком с людьми и с хеш-таблицами, был создан класс PersonCluster. Из названия следует, что это кластер, содержащий как список с людьми и хеш-таблицы, так и методы, отвечающие за ввод-вывод данных, хеширование и поиск в хеш-таблице.

Посредством инкапсуляции, были объявлены служебные поля и методы, к которым не сможет обратиться пользователь напрямую:

1. Количество людей;
2. Количество элементов хеш-таблицы;
3. Количество коллизий;
4. Список с людьми;
5. Динамический массив – хеш-таблица;
6. Методы вывода на экран данных одного элемента списка;
7. Функция хеширования;

private:

int \_count = 0;

int \_collisions = 0;

int \_hashCount = 0;

list<Person> \_persons;

Person\* \_hashTable;

void PrintPerson(Person person, int index)

{

//Вывод на экран элемента хеш-таблицы

}

void PrintByIndex(int index)

{

//Вывод на экран элемента списка

}

int HashValue(string word)

{

//Функция хеширования

}

Для того, чтобы всё же пользователь мог узнать количество коллизий и длину списка, были реализованы публичные свойства, возвращающие значение инкапсулированных полей.

int Collisions()

{

return \_collisions;

}

int Count()

{

return \_count;

}

Для команды clear, реализован метод Clear, который «обнуляет» данные кластера:

* Обнуление счётчиков количества элементов списка и хеш-таблицы;
* Удаление данных списка и хеш-таблицы;

void Clear()

{

\_count = 0;

\_hashCount = 0;

\_persons.clear();

delete[] \_hashTable;

}

Для команды input, реализован перегруженный метод AddPersons, который добавляет новых людей в список. Перегрузка необходима для двух типов ввода данных:

1. Ввод с клавиатуры;
2. Ввод с файла;

void AddPersons(int personsCount)

{

Person person;

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

{

cout << " - Full name: ";

getline(cin, person.FullName);

cout << " - Passport name: ";

getline(cin, person.PassportNumber);

cout << " - Phone number: ";

getline(cin, person.PhoneNumber);

\_persons.push\_back(person);

\_count++;

}

}

void AddPersons(string filePath)

{

Person person;

ifstream reader(filePath);

while (!reader.eof())

{

getline(reader, person.FullName);

getline(reader, person.PassportNumber);

getline(reader, person.PhoneNumber);

\_persons.push\_back(person);

\_count++;

}

}

Для команды print реализован метод PrintEveryone, который выводит на экран каждого человека, используя приватный метод PrintByIndex. Доступ к конкретному элементу списка проводится через итератор – указатель на n-й элемента списка.

void PrintByIndex(int index)

{

list<Person>::iterator pointer = \_persons.begin();

advance(pointer, index);

cout << "Person #" << index + 1 << ":\n";

cout << " - Full name: " << pointer->FullName << ";\n";

cout << " - Passport number: " << pointer->PassportNumber

<< ";\n";

cout << " - Phone number: " << pointer->PhoneNumber

<< ";\n";

cout << endl;

}

void PrintEveryone()

{

for (int i = 0; i < \_count; i++)

PrintByIndex(i);

}

**Алгоритм хеширования:**

Как было описано выше, будет использоваться метод остатка от деления. Поскольку входные данные – это строка, соответственно необходимо эту строку превратить в число, которое будет делиться на некое число в функции.

Перевод слова в число будет работать по следующей формуле.

Итоговое значение будет суммой значений чисел каждого символа вводимой строки. Такое усложнение необходимо для того, чтобы уменьшить количество коллизий. Также, для уменьшения количества коллизий, остаток от деления будет браться от простого числа – у простых чисел всего два делителя: 1 и само число. В данном случае было выбрано число 2047 – одно из чисел Мерсена = .

int HashValue(string word)

{

int value = 0;

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

value += ((int)word[i] + 64) \* (i + 1);

return value % 2047;

}

Сама же функция, формирующая хеш-таблицу, работает по следующим шагам:

1. Формируется пустая хеш-таблица, количество элементов равно количеству элементов в списке;
2. Извлекается значение хеш-функции из строки в текущем элементе списка;
3. Идёт проверка индекса с количеством элементов, если индекс больше, то хеш-таблица удваивается;
4. Идёт проверка на коллизии, в случае нахождения коллизии идёт разрешение коллизии;

void CreateHashTable()

{

**//Создаётся новая хеш-таблица**

\_hashCount = \_count;

\_hashTable = new Person[\_hashCount];

\_collisions = 0;

for (int i = 0; i < \_hashCount; i++)

\_hashTable[i] = Person {EMPTY, EMPTY, EMPTY};

**//Заполнение хеш-таблицы**

for (list<Person>::iterator i = \_persons.begin();

i != \_persons.end(); i++)

{

**//Значение хеш-функции**

int index = HashValue(i->PassportNumber);

**//Проверка индекса на количество элементов**

while (index >= \_hashCount ||

\_hashTable[index].PassportNumber != EMPTY)

{

**//Удвоение хеш-таблицы**

Person\* newHashTable = new Person[\_hashCount \* 2];

for (int i = 0; i < \_hashCount; i++)

newHashTable[i] = \_hashTable[i];

for (int i = \_hashCount; i < \_hashCount \* 2; i++)

newHashTable[i] = Person {EMPTY, EMPTY, EMPTY};

delete[] \_hashTable;

\_hashTable = newHashTable;

\_hashCount \*= 2;

}

**//Проверка на коллизию**

if (\_hashTable[index].PassportNumber != EMPTY)

{

**//Разрешение коллизии**

\_collisions++;

while (\_hashTable[index].PassportNumber != EMPTY)

index++;

}

**//Запись элемента в хеш-таблицу**

\_hashTable[index] = \*i;

}

}

Вывод элементов происходит следующим образом:

1. Идёт проверка на пустоту элемента;
2. Если элемент не пустой, идёт вывод на экран;

void PrintPerson(Person person, int index)

{

if (person.FullName != EMPTY)

{

cout << "Person #" << index + 1 << ":\n";

cout << " - Full name: " << person.FullName << ";\n";

cout << " - Passport number: " << person.PassportNumber

<< ";\n";

cout << " - Phone number: " << person.PhoneNumber << ";\n";

cout << endl;

}

}

Поскольку в программе будет происходит работа только с одним кластером, то он будет объявлен в качестве глобальной переменной, чтобы функция с меню имела доступ к этому экземпляру класса.

PersonCluster \_cluster;

Реализация функции меню:

bool AcceptCommand()

{

**//принятие на ввод команды**

cout << ">> ";

string command = ReadLine();

**//обработка введённой команды – проверка на значение**

if (command == "0" || command == "quit" || command == "exit")

**//Завершение выполнения программы**

return false;

else

{

if (command == "clear")

**//Очистка списка и хеш-таблицы**

\_cluster.Clear();

else if (command == "input")

{

**//принятие на ввод способа заполнения списка**

cout << "Choose the type in entering the data:"

<< endl << " - file;" << endl << " - keyboard;"

<< endl << endl << "type> ";

string type = ReadLine();

if (type == "file")

{

**//Чтение из файла**

cout << "Enter the path to the file: ";

string path = ReadLine();

\_cluster.AddPersons(path);

}

else if (type == "keyboard")

{  
 **//Ввод с клавиатуры**

cout << "Enter the count of persons: ";

int count = ReadNaturalNum();

cin.ignore();

\_cluster.AddPersons(count);

}

else

**//Вывод ошибки в случае неверно указанного способа**

cout << "Error: the argument does not exist!" << endl;

}

else if (command == "print")

{

**//Вывод всех элементов списка**

cout << endl << "Persons count = " << \_cluster.Count()

<< ":\n\n";

\_cluster.PrintEveryone();

}

else if (command == "hash")

{

**//Создание хеш-таблицы и вывод количества коллизий**

\_cluster.CreateHashTable();

cout << "The hash table has been created! Collisions: " << \_cluster.Collisions() << endl;

}

else if (command == "printHash")

{

**//Вывод полученной хеш-таблицы**

\_cluster.PrintHashTable();

}

else if (command == "search")

{

**//Поиск в хеш-таблице по ключу -> номер паспорта**

cout << "Enter the seach key: ";

string key = ReadLine();

\_cluster.Search(key);

}

else if (command == "help")

{

**//Помощь -> вывод всех вышеописанных команд**

cout << "Command list:" << endl;

cout << " - clear -> clears person list" << endl;

cout << " - input -> to enter the data into person list" << endl;

cout << " types of input:" << endl;

cout << " \* keyboard -> input by typing on the keyboard" << endl;

cout << " \* file -> getting data from file (type the file path)" << endl;

cout << " - print -> prints to console the person list" << endl;

cout << " - hash -> hashes up the person list" << endl;

cout << " - printHash -> prints the hash-table" << endl;

cout << " - search -> making search from key by the hash-table" << endl;

cout << " - 0, quit, exit -> finish the program executing" << endl;

}

else

**//Вывод сообщения об ошибки в случае неверного ввода**

cout << "Error: the command does not exist!!!" << endl;

cout << endl;

**//Продолжается выполнение программы**

return true;

}

}

**Полный исходный код программы на языке программирования C++:**

#include <iostream>

#include <fstream>

#include <list>

#include <iterator>

#include <string>

#define EMPTY "{empty}"

using namespace std;

struct Person

{

string FullName;

string PassportNumber;

string PhoneNumber;

};

class PersonCluster

{

private:

int \_count = 0;

int \_collisions = 0;

int \_hashCount = 0;

list<Person> \_persons;

Person\* \_hashTable;

void PrintPerson(Person person, int index)

{

if (person.FullName != EMPTY)

{

cout << "Person #" << index + 1 << ":\n";

cout << " - Full name: " << person.FullName << ";\n";

cout << " - Passport number: " << person.PassportNumber << ";\n";

cout << " - Phone number: " << person.PhoneNumber << ";\n";

cout << endl;

}

}

void PrintByIndex(int index)

{

list<Person>::iterator pointer = \_persons.begin();

advance(pointer, index);

cout << "Person #" << index + 1 << ":\n";

cout << " - Full name: " << pointer->FullName << ";\n";

cout << " - Passport number: " << pointer->PassportNumber << ";\n";

cout << " - Phone number: " << pointer->PhoneNumber << ";\n";

cout << endl;

}

int HashValue(string word)

{

int value = 0;

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

value += ((int)word[i] + 64) \* (i + 1);

return value % 2047;

}

public:

int Collisions()

{

return \_collisions;

}

int Count()

{

return \_count;

}

void Clear()

{

\_count = 0;

\_hashCount = 0;

\_persons.clear();

delete[] \_hashTable;

}

void Search(string key)

{

int index = HashValue(key);

bool isElementFound = false;

while (index < \_hashCount && !isElementFound)

{

if (\_hashTable[index].PassportNumber == key)

{

isElementFound = true;

PrintPerson(\_hashTable[index], index);

}

else if (\_hashTable[index].PassportNumber == EMPTY)

index = \_hashCount;

else

index++;

}

if (!isElementFound)

cout << "Error: the list does not contain element with the key." << endl;

}

void AddPersons(int personsCount)

{

Person person;

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

{

cout << " - Full name: ";

getline(cin, person.FullName);

cout << " - Passport name: ";

getline(cin, person.PassportNumber);

cout << " - Phone number: ";

getline(cin, person.PhoneNumber);

\_persons.push\_back(person);

\_count++;

}

}

void AddPersons(string filePath)

{

Person person;

ifstream reader(filePath);

while (!reader.eof())

{

getline(reader, person.FullName);

getline(reader, person.PassportNumber);

getline(reader, person.PhoneNumber);

\_persons.push\_back(person);

\_count++;

}

}

void Initialize(int personsCount)

{

Clear();

AddPersons(personsCount);

}

void CreateHashTable()

{

\_hashCount = \_count;

\_hashTable = new Person[\_hashCount];

\_collisions = 0;

for (int i = 0; i < \_hashCount; i++)

\_hashTable[i] = Person {EMPTY, EMPTY, EMPTY};

for (list<Person>::iterator i = \_persons.begin(); i != \_persons.end(); i++)

{

int index = HashValue(i->PassportNumber);

while (index >= \_hashCount || \_hashTable[index].PassportNumber != EMPTY)

{

Person\* newHashTable = new Person[\_hashCount \* 2];

for (int i = 0; i < \_hashCount; i++)

newHashTable[i] = \_hashTable[i];

for (int i = \_hashCount; i < \_hashCount \* 2; i++)

newHashTable[i] = Person {EMPTY, EMPTY, EMPTY};

delete[] \_hashTable;

\_hashTable = newHashTable;

\_hashCount \*= 2;

}

if (\_hashTable[index].PassportNumber != EMPTY)

{

\_collisions++;

while (\_hashTable[index].PassportNumber != EMPTY)

index++;

}

\_hashTable[index] = \*i;

}

}

void PrintHashTable()

{

for (int i = 0; i < \_hashCount; i++)

PrintPerson(\_hashTable[i], i);

}

void PrintEveryone()

{

for (int i = 0; i < \_count; i++)

PrintByIndex(i);

}

~PersonCluster()

{

delete[] \_hashTable;

}

};

PersonCluster \_cluster;

string ReadLine()

{

string x;

getline(cin, x);

return x;

}

int ReadNaturalNum()

{

int x;

do cin >> x; while (x <= 0);

return x;

}

bool AcceptCommand()

{

cout << ">> ";

string command = ReadLine();

if (command == "0" || command == "quit" || command == "exit")

return false;

else

{

if (command == "clear")

\_cluster.Clear();

else if (command == "input")

{

cout << "Choose the type in entering the data:"

<< endl << " - file;" << endl << " - keyboard;"

<< endl << endl << "type> ";

string type = ReadLine();

if (type == "file")

{

cout << "Enter the path to the file: ";

string path = ReadLine();

\_cluster.AddPersons(path);

}

else if (type == "keyboard")

{

cout << "Enter the count of persons: ";

int count = ReadNaturalNum();

cin.ignore();

\_cluster.AddPersons(count);

}

else

cout << "Error: the argument does not exist!" << endl;

}

else if (command == "print")

{

cout << endl << "Persons count = " << \_cluster.Count() << ":\n\n";

\_cluster.PrintEveryone();

}

else if (command == "hash")

{

\_cluster.CreateHashTable();

cout << "The hash table has been created! Collisions: " << \_cluster.Collisions() << endl;

}

else if (command == "printHash")

{

\_cluster.PrintHashTable();

}

else if (command == "search")

{

cout << "Enter the seach key: ";

string key = ReadLine();

\_cluster.Search(key);

}

else if (command == "help")

{

cout << "Command list:" << endl;

cout << " - clear -> clears person list" << endl;

cout << " - input -> to enter the data into person list" << endl;

cout << " types of input:" << endl;

cout << " \* keyboard -> input by typing on the keyboard" << endl;

cout << " \* file -> getting data from file (type the file path)" << endl;

cout << " - print -> prints to console the person list" << endl;

cout << " - hash -> hashes up the person list" << endl;

cout << " - printHash -> prints the hash-table" << endl;

cout << " - search -> making search from key by the hash-table" << endl;

cout << " - 0, quit, exit -> finish the program executing" << endl;

}

else

cout << "Error: the command does not exist!!!" << endl;

cout << endl;

return true;

}

}

int main()

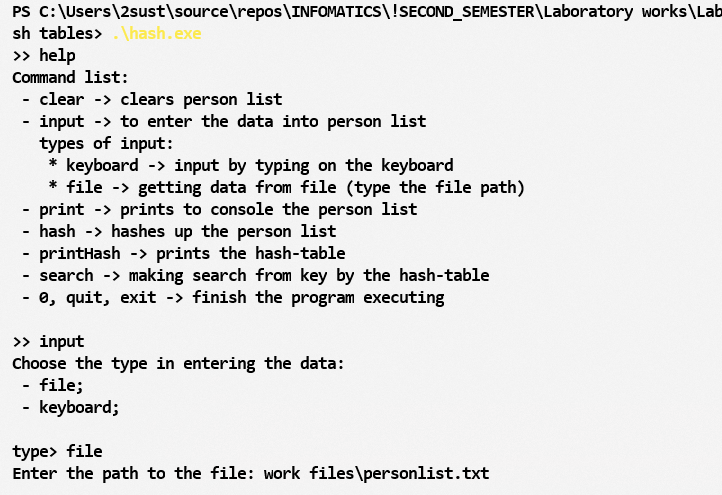
{

while (AcceptCommand());

return 0;

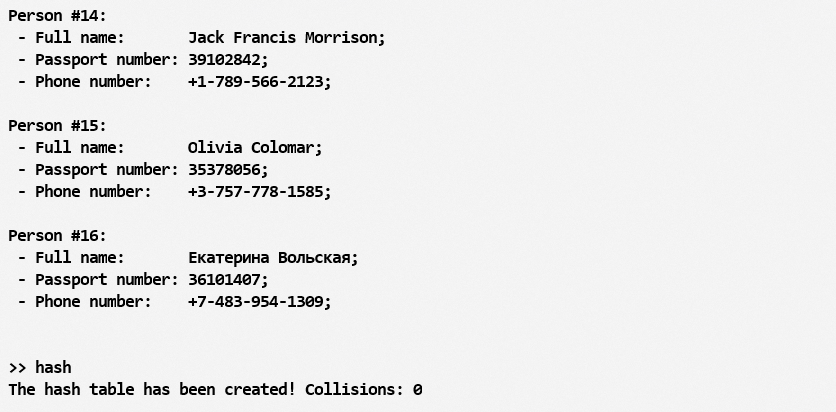
}

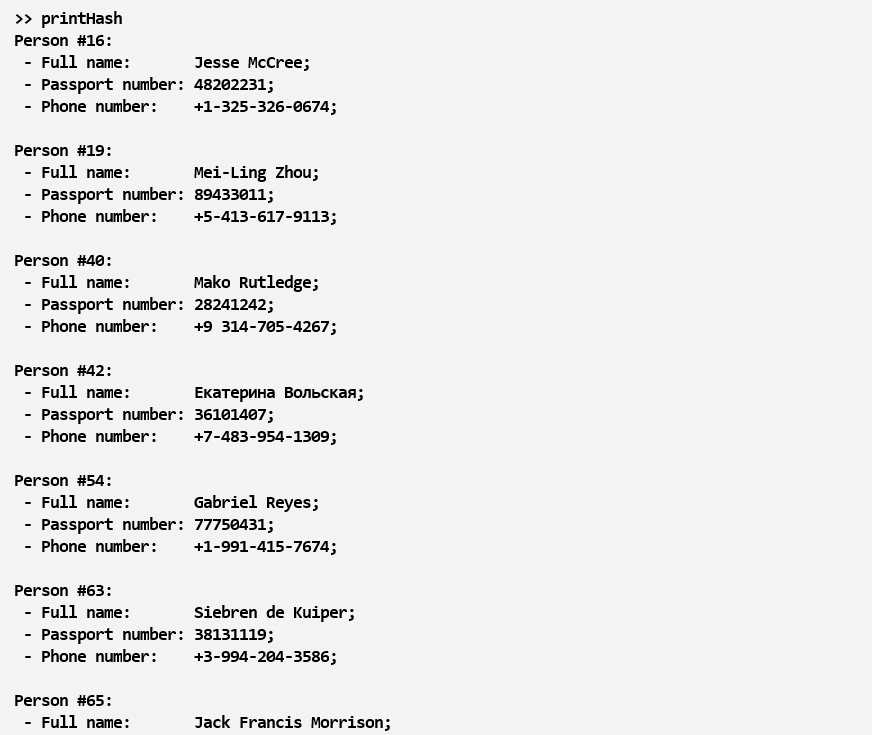
Aoei



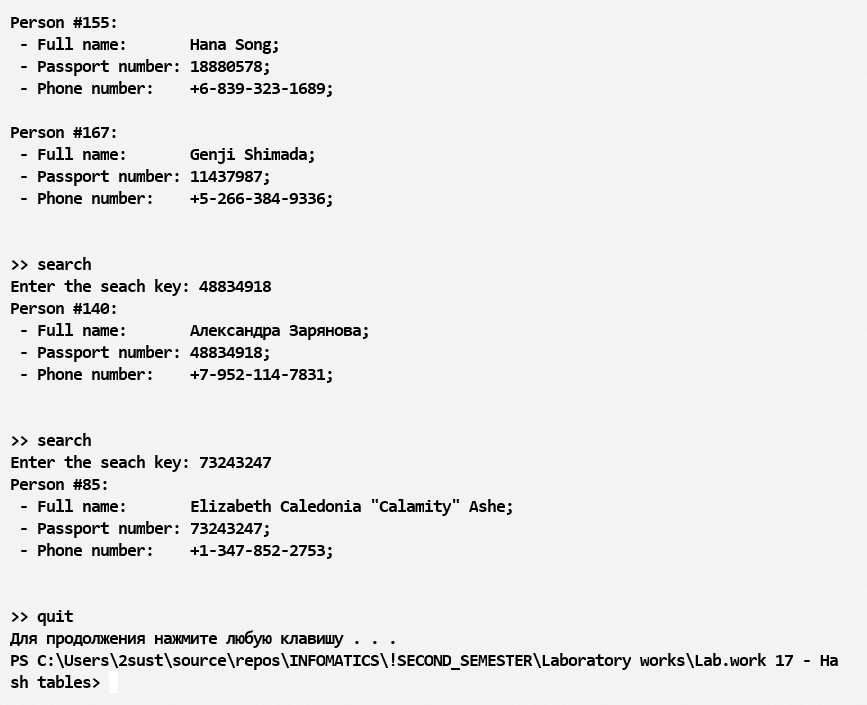












**Блок-схема** алгоритма программы:

