Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

о лабораторной работе

Тема: «Хэш»

Выполнил

Студент группы РИС-22-2б

Зубов Р. А.

Проверил доцент кафедры

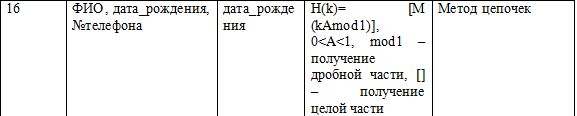
ИТАС

Полякова О. А.

Пермь 2023

**Постановка задачи**

Реализовать Хэш-таблицу с использованием языка программирования C++.



**Исходный код программы**

#include <iostream>

#include <fstream>

#include <string>

#include <ctime>

const int ARRAY\_SIZE = 100;

const int HASH\_TABLE\_SIZES[] = { 40, 75, 90 };

struct Record {

std::string name;

std::string birthdate;

std::string phoneNumber;

};

struct Node {

Record record;

Node\* next;

};

Record generateRandomRecord() {

Record record;

record.name = "Name" + std::to\_string(rand() % 100);

record.birthdate = "01/01/" + std::to\_string(1920 + rand() % 100);

record.phoneNumber = "+7-988-" + std::to\_string(rand() % 10000);

return record;

}

int hashFunction(const std::string& key, int hashTableSize) {

int hash = 0;

for (char c : key) {

hash = (hash \* 31 + c) % hashTableSize;

}

return hash;

}

void saveArrayToFile(Record\* array, const std::string& fileName) {

std::ofstream outputFile(fileName);

for (int i = 0; i < ARRAY\_SIZE; ++i) {

outputFile << array[i].name << " " << array[i].birthdate << " " << array[i].phoneNumber << std::endl;

}

outputFile.close();

}

void loadArrayFromFile(Record\* array, const std::string& fileName) {

std::ifstream inputFile(fileName);

for (int i = 0; i < ARRAY\_SIZE; ++i) {

inputFile >> array[i].name >> array[i].birthdate >> array[i].phoneNumber;

}

inputFile.close();

}

void addRecord(Record\* array, int& size, const Record& record) {

if (size >= ARRAY\_SIZE) {

std::cerr << "Array is full" << std::endl;

return;

}

array[size] = record;

++size;

}

void removeRecord(Record\* array, int& size, const std::string& birthdate) {

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

if (array[i].birthdate == birthdate) {

for (int j = i; j < size - 1; ++j) {

array[j] = array[j + 1];

}

--size;

return;

}

}

}

Node\*\* createHashTable(Record\* array, int size, int hashTableSize) {

Node\*\* hashTable = new Node \* [hashTableSize];

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

hashTable[i] = nullptr;

}

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

int index = hashFunction(array[i].birthdate, hashTableSize);

Node\* newNode = new Node{ array[i], hashTable[index] };

hashTable[index] = newNode;

}

return hashTable;

}

void deleteHashTable(Node\*\* hashTable, int hashTableSize) {

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

Node\* current = hashTable[i];

while (current != nullptr) {

Node\* next = current->next;

delete current;

current = next;

}

}

delete[] hashTable;

}

Record\* findRecordInHashTable(Node\*\* hashTable, int hashTableSize, const std::string& birthdate) {

int index = hashFunction(birthdate, hashTableSize);

Node\* current = hashTable[index];

while (current != nullptr) {

if (current->record.birthdate == birthdate) {

return &current->record;

}

current = current->next;

}

return nullptr;

}

int countCollisions(Node\*\* hashTable, int hashTableSize) {

int collisionCount = 0;

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

if (hashTable[i] != nullptr && hashTable[i]->next != nullptr) {

++collisionCount;

}

}

return collisionCount;

}

int main() {

srand(time(0));

int size = ARRAY\_SIZE;

Record\* array = new Record[ARRAY\_SIZE];

// Заполняем массив значениями

for (int i = 0; i < ARRAY\_SIZE; ++i) {

array[i] = generateRandomRecord();

}

saveArrayToFile(array, "records.txt");

loadArrayFromFile(array, "records.txt");

Record newRecord = { "Ivan Ivan", "01/01/2000", "+7-988-1234" };

addRecord(array, size, newRecord);

removeRecord(array, size, "01/01/2000");

// Создаем хеш таблицу и ищем данные

for (int hashTableSize : HASH\_TABLE\_SIZES) {

Node\*\* hashTable = createHashTable(array, size, hashTableSize);

Record\* record = findRecordInHashTable(hashTable, hashTableSize, "01/01/2002");

if (record != nullptr) {

std::cout << "Found record: " << record->name << " " << record->birthdate << " " << record->phoneNumber << std::endl;

}

else {

std::cout << "Record not found" << std::endl;

}

int collisionCount = countCollisions(hashTable, hashTableSize);

std::cout << "Collision count for hash table size " << hashTableSize << ": " << collisionCount << std::endl;

deleteHashTable(hashTable, hashTableSize);

}

delete[] array;

return 0;

}

**Блок схема**

**Результаты выполнения программы**

