Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ Hash-table(1)**

Дисциплина: «Основы алгоритмизации и программирования»

Тема : "Hash-таблицы"

Выполнил работу:

Студент группы РИС-22-1Б

Батин В.В.

Проверил:

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**

Реализовать хэш-таблицу

Вариант 5:

ФИО, адрес, паспорт

H(k) = kmodM, где M - ФИО

Метод цепочек

**Код**

#include <iostream>

#include <string>

using namespace std;

const int M = 10;

struct Node {

string fio;

int passport;

string address;

Node\* next;

Node(string \_fio, int \_passport, string \_address) : fio(\_fio), passport(\_passport), address(\_address), next(nullptr) {}

};

class HashTable {

private:

Node\* table[M];

int hashFunc(string fio) {

int sum = 0;

for (char c : fio) {

sum += c;

}

return sum % M;

}

public:

HashTable() {

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

table[i] = nullptr;

}

}

void insert(string fio, int passport, string address) {

int index = hashFunc(fio);

Node\* newNode = new Node(fio, passport, address);

if (table[index] == nullptr) {

table[index] = newNode;

}

else {

Node\* currentNode = table[index];

while (currentNode->next != nullptr) {

currentNode = currentNode->next;

}

currentNode->next = newNode;

}

}

Node\* search(string fio) {

int index = hashFunc(fio);

Node\* currentNode = table[index];

while (currentNode != nullptr && currentNode->fio != fio) {

currentNode = currentNode->next;

}

return currentNode;

}

void remove(string fio) {

int index = hashFunc(fio);

Node\* currentNode = table[index];

if (currentNode == nullptr) {

return;

}

if (currentNode->fio == fio) {

table[index] = currentNode->next;

delete currentNode;

return;

}

while (currentNode->next != nullptr && currentNode->next->fio != fio) {

currentNode = currentNode->next;

}

if (currentNode->next == nullptr) {

return;

}

Node\* nodeToDelete = currentNode->next;

currentNode->next = nodeToDelete->next;

delete nodeToDelete;

}

void print() {

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

Node\* currentNode = table[i];

cout << "Bucket " << i << ": ";

while (currentNode != nullptr) {

cout << "(" << currentNode->fio << ", " << currentNode->passport << ", " << currentNode->address << ")";

if (currentNode->next != nullptr) {

cout << " -> ";

}

currentNode = currentNode->next;

}

cout << endl;

}

}

};

int main() {

HashTable ht;

ht.insert("Ivanov Ivan Ivanovich", 1234, "Moscow");

ht.insert("Ivanov Ivan Ivanovich", 1234, "Perm");

ht.insert("Petrov Petr Petrovich", 5678, "St. Petersburg");

ht.insert("Sidorov Ivan Ivanovich", 9012, "Novosibirsk");

ht.insert("Kuznetsov Sergey Ivanovich", 3456, "Yekaterinburg");

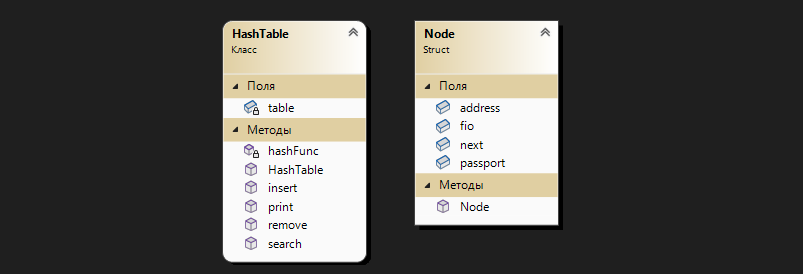
ht.insert("Smirnova Anna Petrovna", 7890, "Kazan");

cout << "Initial table:" << endl;

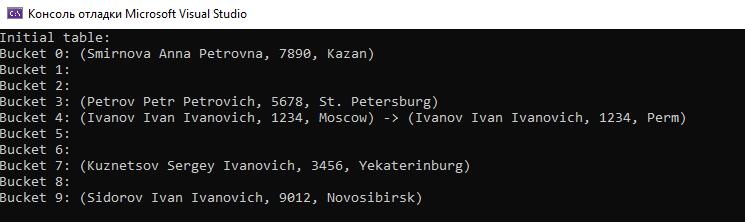
ht.print();

}

**UML диаграмма**

****

**Результат выполнения**

****