Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

Лабораторная работа  
«Хэш-таблицы»

Выполнил:   
студент группы РИС-23-1б   
Мокрушин Никита Дмитриевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

Пермь 2024 г.

**Постановка задачи:**

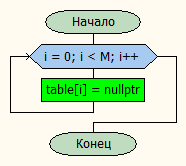
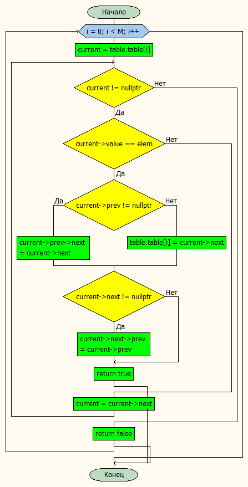
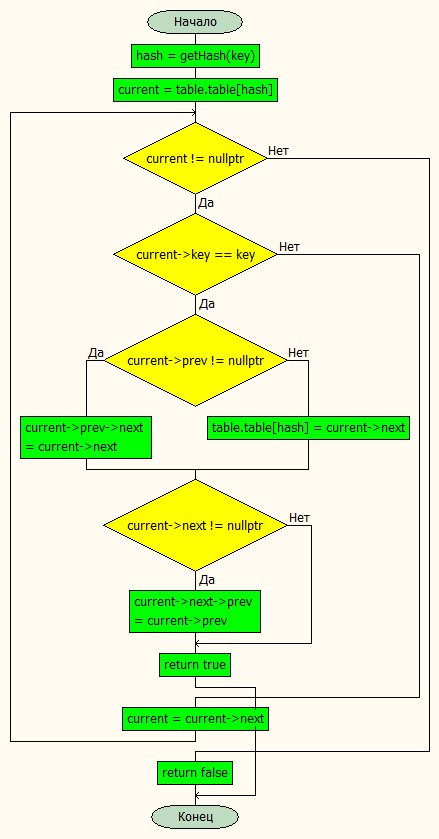
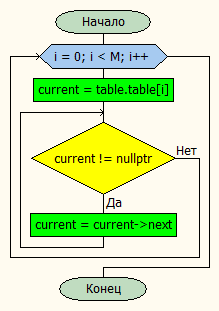
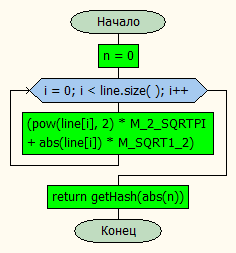
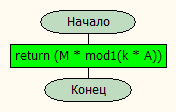
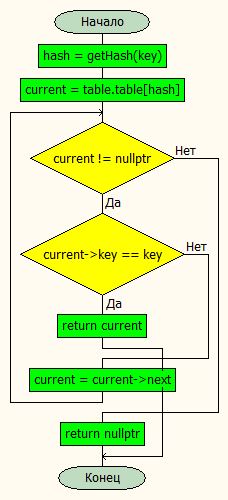
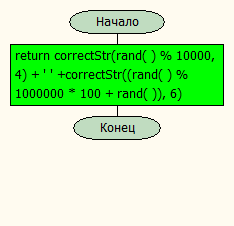
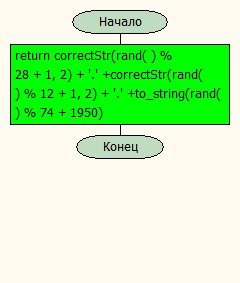
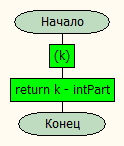
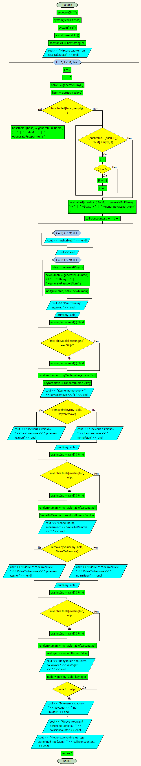
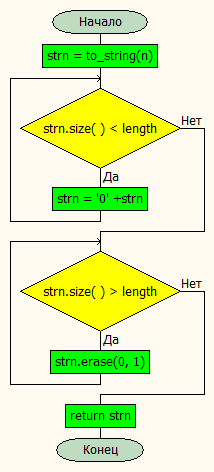
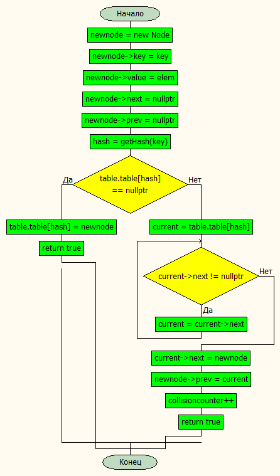
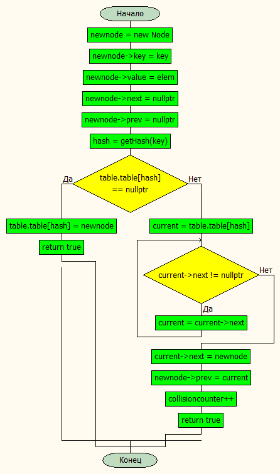
Вариант 11:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | ФИО, №паспорта, №телефона | №паспорта | H(k)=k mod M | Метод открытой адресации |

**Анализ задачи:**

* Создать хэш-таблицу, используя два метода рехэширования
* Реализовать хэш таблицу в соответствии с вариантом
* Посчитать количество коллизий для двух вариантов

**Блок схемы:**

****

**Код на языке с++:**

#include <iostream>

#include <string>

#include <ctime>

#define \_USE\_MATH\_DEFINES

#include <math.h>

using namespace std;

const int M = 5;

const double A = M\_PI\_4;

int collisioncounter = 0;

int collisioncounter1 = 0;

struct Node {

string key = "";

string value = "";

Node\* next = nullptr;

Node\* prev = nullptr;

};

struct Hashtable

{

Node\* table[M];

Hashtable()

{

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

{

table[i] = nullptr;

}

}

};

double mod1(double k)

{

int intPart = static\_cast<int>(k);

return k - intPart;

}

int getHash(double k)

{

return static\_cast<int>(M \* mod1(k \* A));

}

int getHash(string line)

{

int n = 0;

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

{

n += static\_cast<int>(pow(line[i], 2) \* M\_2\_SQRTPI + abs(line[i]) \* M\_SQRT1\_2);

}

return getHash(abs(n));

}

bool add(Hashtable& table, string key, string elem)

{

Node\* newnode = new Node;

newnode->key = key;

newnode->value = elem;

newnode->next = nullptr;

newnode->prev = nullptr;

int hash = getHash(key);

if (table.table[hash] == nullptr)

{

table.table[hash] = newnode;

return true;

}

else

{

Node\* current = table.table[hash];

while (current->next != nullptr)

{

current = current->next;

}

current->next = newnode;

newnode->prev = current;

collisioncounter++;

return true;

}

}

bool removebykey(Hashtable& table, string key)

{

int hash = getHash(key);

Node\* current = table.table[hash];

while (current != nullptr)

{

if (current->key == key)

{

if (current->prev != nullptr)

{

current->prev->next = current->next;

}

else

{

table.table[hash] = current->next;

}

if (current->next != nullptr)

{

current->next->prev = current->prev;

}

delete current;

return true;

}

current = current->next;

}

return false;

}

bool removebyvalue(Hashtable& table, string elem)

{

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

{

Node\* current = table.table[i];

while (current != nullptr)

{

if (current->value == elem)

{

if (current->prev != nullptr)

{

current->prev->next = current->next;

}

else

{

table.table[i] = current->next;

}

if (current->next != nullptr)

{

current->next->prev = current->prev;

}

delete current;

return true;

}

current = current->next;

}

return false;

}

}

Node\* get(Hashtable& table, string key)

{

int hash = getHash(key);

Node\* current = table.table[hash];

while (current != nullptr)

{

if (current->key == key)

{

return current;

}

current = current->next;

}

return nullptr;

}

void print(Hashtable& table)

{

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

{

Node\* current = table.table[i];

while (current != nullptr)

{

cout << "[" << current->key << ": " << current->value << "]\n";

current = current->next;

}

}

}

string surnames[] = {

"Мудя", "Кривошерти", "Тихий", "Валинат", "Сингурочка",

"Молодой", "Легенда" , "Чумаков", "Корсачок", "Талант",

"Федоро", "Гжелькин", "Кудияшечка", "Стовторой", "Громкий"

};

string names[] = {

"Никита", "Алехандро", "Тимоха", "Ринуров", "Иван",

"Саня", "Егорчик" , "Дмитрий", "Максик", "Серега",

"Степан", "Хриплый", "Антон", "Енокентий", "Вальдемар"

};

string patronymics[] = {

"СВОкин", "Александрович", "Тимофеевич", "Ринурович", "Иванович",

"Алексевич", "Егоров" , "Дмитриевич", "Максимович", "Сергеевич",

"Степанович", "Хриплыч", "Антонович", "Енокентьевич", "Вальдемарович"

};

string generateFullName() {

return surnames[rand() % 15] + ' ' + names[rand() % 15] + ' ' + patronymics[rand() % 15];

}

string correctStr(int n, int length)

{

string strn = to\_string(n);

while (strn.size() < length)

{

strn = '0' + strn;

}

while (strn.size() > length)

{

strn.erase(0, 1);

}

return strn;

}

string generateBDay()

{

return correctStr(rand() % 28 + 1, 2) + '.' + correctStr(rand() % 12 + 1, 2) + '.' + to\_string(rand() % 74 + 1950);

}

string generatePassportNum()

{

return correctStr(rand() % 10000, 4) + ' ' + correctStr((rand() % 1000000 \* 100 + rand()), 6);

}

int main()

{

setlocale(0, "");

system("chcp 1251");

system("cls");

srand(time(NULL));

Hashtable myTable;

string\* hashtable1 = new string[M];

cout << "Массив элементов хэш таблицы:" << endl;

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

{

int j = 0;

int j1 = 0;

string bday1 = generateBDay();

int hash = getHash(bday1);

if (hashtable1[hash].empty())

{

hashtable1[hash] = generateFullName() + " | " + bday1 + " | " + generatePassportNum();

}

else

{

while (!(hashtable1[(hash + j) % M].empty()))

{

j++;

if (j >= 5)

{

j1 += j;

j -= 5;

}

}

hashtable1[(hash + j) % M] = generateFullName() + " | " + bday1 + " | " + generatePassportNum();

collisioncounter1 += j1 + j;

}

}

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

{

cout << hashtable1[i] << endl;

}

cout << endl;

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

{

string bday = generateBDay();

string newhuman = generateFullName() + " | " + bday + " | " + generatePassportNum();

add(myTable, bday, newhuman);

}

cout << "Хэш-таблица задана: " << endl;

print(myTable);

int existingInd = rand() % M;

while (myTable.table[existingInd] == nullptr)

{

existingInd = rand() % M;

}

Node\* randomHuman = myTable.table[existingInd];

string keytoremove = randomHuman->key;

cout << "Удаление по ключу " << keytoremove << ": " << endl;

if (removebykey(myTable, keytoremove))

{

cout << "Элемент с ключом " << keytoremove << " успешно удален" << endl;

}

else

{

cout << "Элемент с ключом " << keytoremove << " не найден" << endl;

}

print(myTable);

existingInd = rand() % M;

while (myTable.table[existingInd] == nullptr)

{

existingInd = rand() % M;

}

randomHuman = myTable.table[existingInd];

string ValueToRemove = randomHuman->value;

cout << "Удаление по значению " << ValueToRemove << ":" << endl;

if (removebyvalue(myTable, ValueToRemove))

{

cout << "Элемент со значением " << ValueToRemove << " успешно удален" << endl;

}

else

{

cout << "Элемент со значением " << ValueToRemove << " не найден" << endl;

}

print(myTable);

existingInd = rand() % M;

while (myTable.table[existingInd] == nullptr)

{

existingInd = rand() % M;

}

randomHuman = myTable.table[existingInd];

string keytoget = randomHuman->key;

cout << "Получение элемента по ключу " << keytoget << ":" << endl;

Node\* node = get(myTable, keytoget);

if (node != nullptr)

{

cout << "Найден элемент: " << node->value << endl;

}

else

{

cout << "Элемент с ключом: " << keytoget << " не найден" << endl;

}

cout << "Число коллизий в методе цепочки: " << collisioncounter << endl;

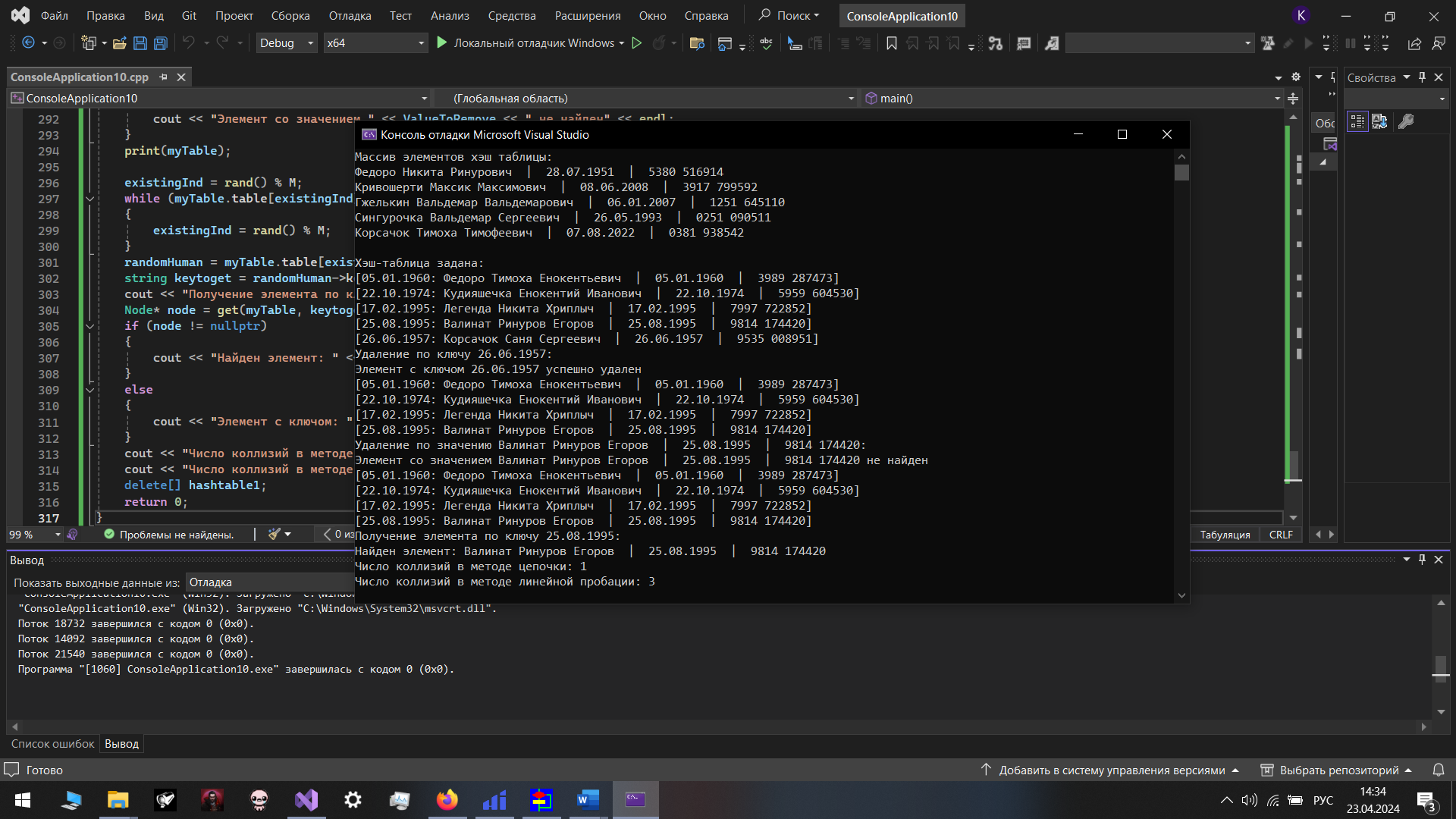
cout << "Число коллизий в методе линейной пробации: " << collisioncounter1 << endl;

delete[] hashtable1;

return 0;

}

**Вывод:**



Программа отработала алгоритм как надо