****

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МИРЭА – Российский технологический университет»**

# РТУ МИРЭА

Отчет по выполнению практического задания №2.2

**Тема:** Хеширование: прямой доступ к данным

Дисциплина: **«**СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ**»**

Выполнил студент Кузнецов А. А.

группа ИКБО-01-21

**Москва 2022**

**Цель работы:** освоить приёмы хеширования и эффективного поиска элементов множества.

**Задание**

Разработайте приложение, которое использует **хеш-таблицу** (пары «ключ – хеш») для организации прямого доступа к элементам **динамического множества** полезных данных. Множество реализуйте на массиве, структура элементов (перечень полей) которого приведена в индивидуальном варианте.

Приложение должно содержать **класс** с **базовыми операциями**: вставки, удаления, поиска по ключу, вывода. Включите в класс массив полезных данных и хеш-таблицу. Хеш-функцию подберите самостоятельно, используя правила выбора функции.

Реализуйте расширение размера таблицы и **рехеширование**, когда это требуется, в соответствии с типом разрешения **коллизий**.

Предусмотрите автоматическое заполнение таблицы 5-7 записями.

Реализуйте текстовый **командный интерфейс** пользователя для возможности вызова методов в любой произвольной последовательности, сопроводите вывод достаточными для понимания происходящего сторонним пользователем подсказками.

Проведите полное тестирование программы (все базовые операции, изменение размера и рехеширование), тест-примеры определите самостоятельно. Результаты тестирования включите в отчет по выполненной работе.

Оформите отчёт с подробным описанием созданного массива и хеш-таблицы, подходов к программной реализации базовых операций, описанием текста исходного кода и проведенного тестирования программы.

В отчёте сделайте вывод о проделанной работе, основанный на полученных результатах.

Содержание отчёта:

1. Титульный лист.

2. Цель работы.

3. Ход работы (по каждому заданию):

a. Формулировка задачи.

b. Математическая модель решения (описание алгоритма).

c. Код программы с комментариями.

d. Результаты тестирования.

4. Вывод (решены ли задачи, достигнута ли цель).

Для сдачи практической работы потребуется:

- отчёт – оформляется в виде электронного документа в форматах Word или PDF, прикрепляется к соответствующему заданию в СДО;

- программные проекты, реализованные по заданиям;

- доклад по результатам выполнения практической работы (по отчёту).

**Индивидуальный вариант задания**

|  |  |  |
| --- | --- | --- |
| 22 | Открытая адресация (квадратичное пробирование) | Счет в банке: номер счета целое семизначное число, ФИО, адрес |

**Математическая модель решения**

Поскольку в качестве ключа выступает номер счета в банке (последовательность 7 целочисленных символов), и, согласно индивидуальному варианту, в качестве механизма разрешения коллизий необходимо использовать квадратичное пробирование, были выбраны хеш-функции, основанные на методе деления. Таким образом, в процессе выполнения работы была использована следующая хеш-функция:

адрес=h(x)+ci+d

где i – номер попытки разрешить коллизию, c и d – константы.

**Код программы**

#include <iostream>

#include <string>

#include <map>

#include <ctime>

#include <Windows.h>

#include <vector>

using namespace std;

// structure of storing information about users (full name and address of the user)

struct DataPerson {

string address;

vector<string>\* FIO = new vector<string>(); // the vector of storing the Surname, First name and Patronymic

DataPerson(string F, string I, string O, string address) :

address(address) {

FIO->push\_back(F);

FIO->push\_back(I);

FIO->push\_back(O);

};

};

class DataInfo {

private:

string check; // bank account number (key)

DataPerson\* info;

bool empty = true; // cell occupancy indicator

public:

// creating an empty record (empty = true)

DataInfo() {

info = new DataPerson("", "", "", "");

}

// creating a record initialized with values (empty = false)

DataInfo(string check, string F, string I, string O, string address) {

info = new DataPerson(F, I, O, address);

this->check = check;

empty = false;

}

string getCheck() {

return check;

}

vector<string>\* getFIO() {

return info->FIO;

}

string getAddress() {

return info->address;

}

bool isEmpty() {

return empty;

}

};

class DataManager {

private:

map<string, int> hashTable; // hash table value-key

DataInfo\* dataArray; // array of user information records

int count; // current number of records

// Hash enumeration function

int hashFun(string& str) {

int result = 0;

// the sum of the ascii-codes of the string characters for the subsequent hash calculation

for (int i = 0; i < 7; i++) {

result += str[i];

}

return result % count;

}

// A method for doubling the array and recalculating the hashes of already added records

void recalculateHashes() {

cout << "Recalculation of hashes and array expansion" << endl;

count \*= 2; // Doubling the size of the array

DataInfo temp;

DataInfo\* arrayExpanded = new DataInfo[count]; // Creating a new empty array twice the previous size

DataInfo\* copy = dataArray;

dataArray = arrayExpanded;

// a loop passing through all key-value pairs in a hash table

for (pair<string, int> keyValuePair : hashTable) {

temp = copy[keyValuePair.second]; // saving an entry from the old array for re-adding to the new one

addEntry(temp); // adding an entry to a new empty array (includes hash calculation)

}

cout << "Recalculation of hashes is finished." << endl;

}

public:

DataManager(int count) {

this->count = count;

dataArray = new DataInfo[count];

}

DataInfo get(string& check) {

return dataArray[hashTable[check]];

}

int gcd(int x, int y) // mutually prime numbers (== 1 -> yes, else -> no)

{

return y ? gcd(y, x % y) : x;

}

void addEntry(DataInfo& entry) {

string key = entry.getCheck();

//int hash1 = hashOne(key), hash2 = hashTwo(key); // рассчёт двух хэшей

int hash;

int c = 3; // the first constant for hashing

int d = 7; // the second constant for hashing

int i = 0; // number of offset steps with open addressing

// recalculation of constants (c and d) for correct hashing

// (making them mutually simple with the maximum number of records)

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

for (int j = 0; j < count; j++)

if (gcd(count,gcd(i,j)) == 1)

{

c = i;

d = j;

break;

}

while (true) {

// consider the hash using quadratic probing

// (according to the assignment of option #22)

hash = (hashFun(key) + c \* i + d \* i \* i) % count;

// adding an element to a cell if it is not occupied

if (dataArray[hash].isEmpty()) {

hashTable[key] = hash;

dataArray[hash] = entry;

break;

}

cout << "A collision has occurred: " << key << " and "

<< dataArray[hash].getCheck() << " have the same hash." << endl;

i++;

}

// switch to recalculation of hashes if the coefficient of array filling >= 0.75

if ((float)hashTable.size() / (float)count >= 0.75) {

recalculateHashes();

}

}

// deleting an entry by key

void removeEntry(string& check) {

DataInfo empty;

dataArray[hashTable[check]] = empty;

hashTable.erase(check);

}

// output of all current records in the array

void printData() {

cout << "Account number:\t\tFCs:\t\t\t\t\tAddress:\n";

for (pair<string, int> keyValuePair : hashTable) {

cout << keyValuePair.first << "\t\t\t"

<< dataArray[keyValuePair.second].getFIO()->at(0) << " "

<< dataArray[keyValuePair.second].getFIO()->at(1) << " "

<< dataArray[keyValuePair.second].getFIO()->at(2) << " "

<< "\t\t"

<< dataArray[keyValuePair.second].getAddress() << endl;

}

}

};

// filling an array with random entries

void addRandom(DataManager& manager, int count) {

string cities[5]{ "Moscow", "Piter", "Izhevsk", "Yakutsk", "Krasnodar" };

string Fs[5]{ "Kuznetsov", "Vasiliev", "Galikhanov", "Marov", "Alekperov" };

string Is[5]{ "Andrey", "Victor", "Emil", "Herman", "Allan" };

string Os[5]{ "Aleksandrovich", "Vladimirovich", "Rinatov", "Andreev", "Raufov" };

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

string address = cities[rand() % 5];

string F = Fs[rand() % 5];

string I = Is[rand() % 5];

string O = Os[rand() % 5];

string check;

for (int j = 0; j < 7; j++) {

check += (char)(rand() % 7 + 48);

}

DataInfo entry(check, F, I, O, address);

manager.addEntry(entry);

}

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

srand(time(NULL));

DataManager manager(10);

addRandom(manager, 6);

while (true) {

cout << "\nList of available commands:" << endl << endl

<< "1. Output of the current state of the array" << endl

<< "2. Adding an entry to an array" << endl

<< "3. Getting an entry from an array by key" << endl

<< "4. Deleting an entry by key." << endl

<< "5. Finish the program." << endl

<< "Enter the number of the selected operation: ";

int option;

cin >> option;

string check;

switch (option) {

case 1: {

cout << endl;

manager.printData();

break;

}

case 2: {

string address;

cout << "Enter a bank account (7 characters): ";

cin >> check;

check = check.substr(0, 7);

string F;

cout << "Enter Last Name: ";

cin >> F;

string I;

cout << "Enter First Name: ";

cin >> I;

string O;

cout << "Enter Patronymic: ";

cin >> O;

cout << "Enter the address: ";

cin >> address;

DataInfo entry(check, F, I, O, address);

manager.addEntry(entry);

cout << "Entry added." << endl;

break;

}

case 3: {

cout << "Enter the phone number of the desired entry: ";

cin >> check;

check = check.substr(0, 7);

DataInfo entry = manager.get(check);

cout << entry.getFIO()->at(0) << " "

<< entry.getFIO()->at(1) << " "

<< entry.getFIO()->at(2) << '\t'

<< entry.getAddress() << endl;

break;

}

case 4: {

cout << "Enter the phone number of the deleted entry: ";

cin >> check;

check = check.substr(0, 7);

manager.removeEntry(check);

cout << "Entry complitly deleted." << endl;

break;

}

case 5: {

return 0;

}

}

}

}

**Результаты тестирования**

Поскольку в качестве хешируемого числа считается сумма ascii-кодов символов номера счета в банке, очевидно, что можно намеренно получить одинаковые суммы, например: строки 22 и 13 будут иметь одинаковые суммы (50 + 50 = 49 + 51 = 100). Когда эти суммы совпадают, результат хеш-функции для этих сумм одинаков, и возникает коллизия. Этим можно воспользоваться, чтобы протестировать механизмы разрешения коллизии. На скриншоте ниже представлен пример такого случая: приложение оповестило пользователя о коллизии и успешно разрешило данную ситуацию даже для нескольких хешей, случайно попавших в ситуацию коллизии. (Рис. 1)

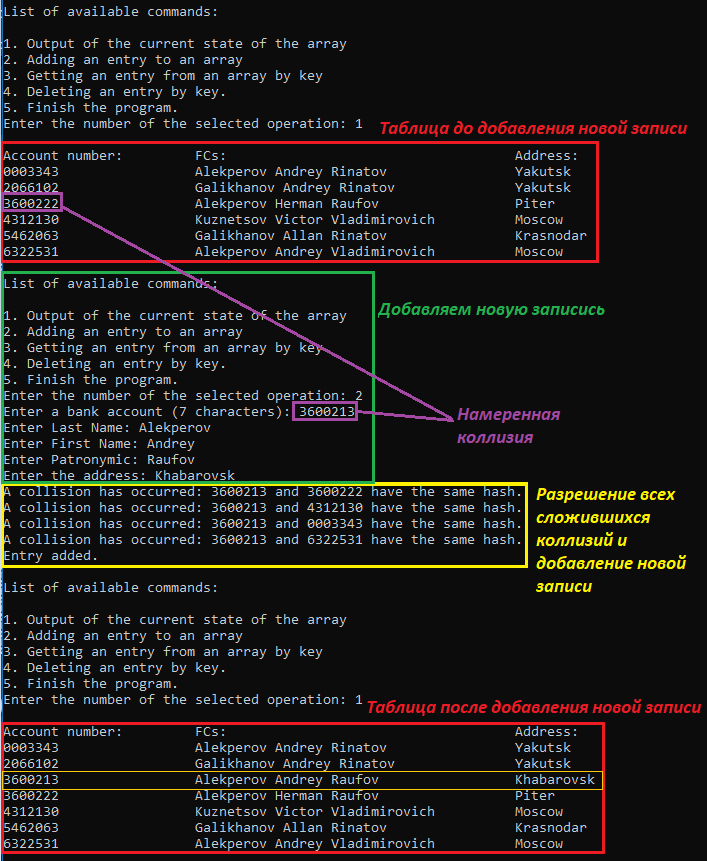
****

Рис. 1 – Результат тестирования программы во время коллизии

Протестируем также получение записей с ключами, ранее создавшими коллизию. (Рис. 2)

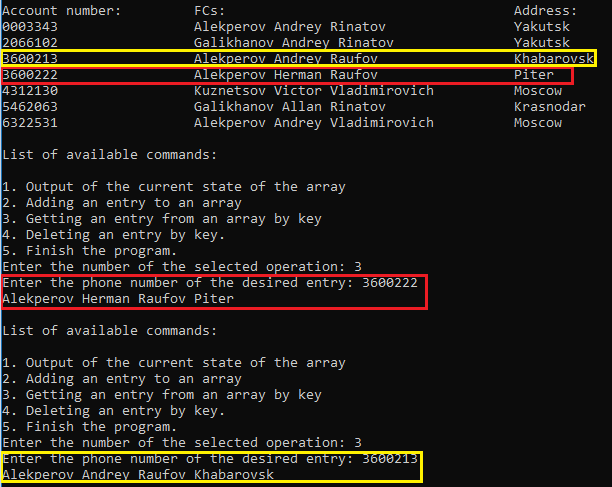
****

Рис. 2 – Результат тестирования получения ранее добавленных записей

В программе задан изначальный размер массива хешей равный 10. По условию задачи размер должен увеличиваться вдвое при заполнении таблицы на более чем 0.75 её части. Значит, чтобы вызвать её увеличение с рехешированием, мы должны добавить новую восьмую запись. (Рис. 3)

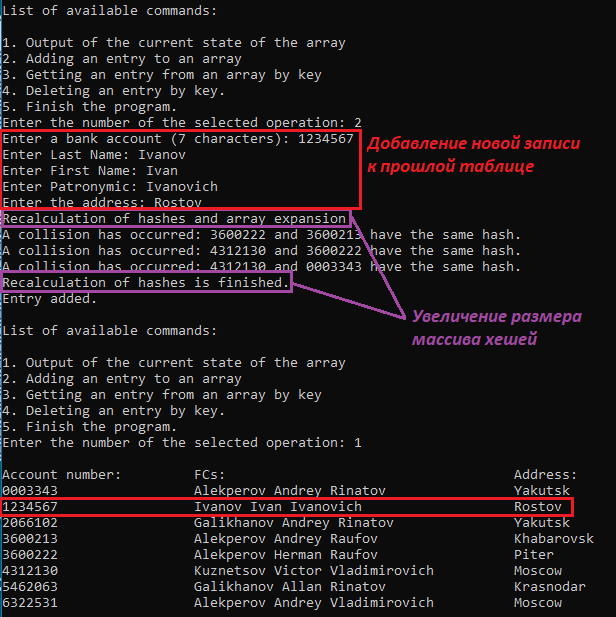


Рис. 3 – Расширение массива вдвое с рехешированием записей.

Теперь проверим работу программы на удаление записей. Удалим недавно добавленную запись и любую, изначально созданную рандомным образом, выведем получившуюся таблицу и закончим работу программы. (Рис. 4)

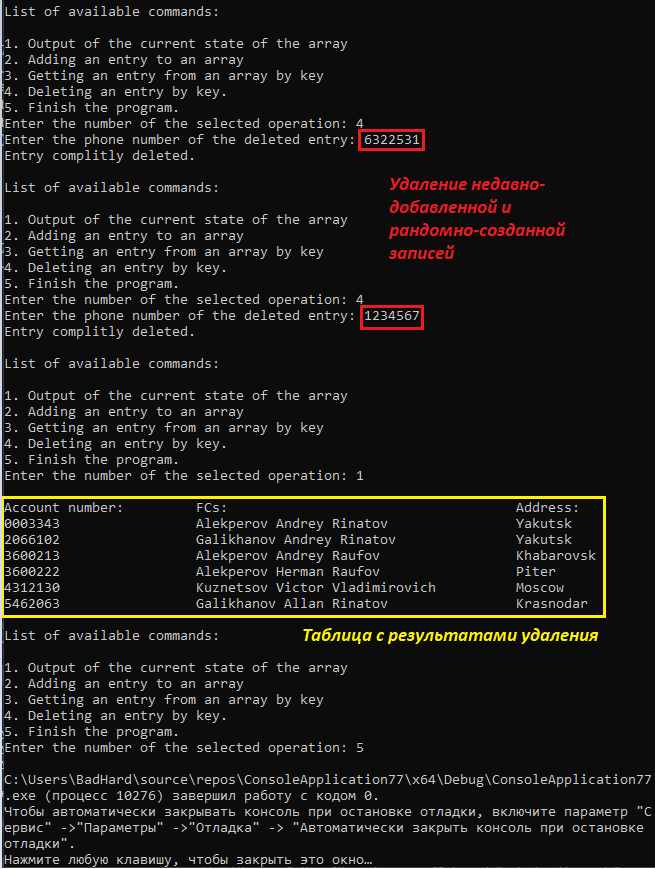


Рис. 4 – Тестирование программы при удалении пары записей

**ВЫВОДЫ**

По результатам тестирования видно, что все базовые операции, а также механизмы разрешения коллизий, расширения массива и рехеширования всех добавленных записей после расширения работают корректно.

В результате выполнения данной работы мной были освоены навыки хеширования и эффективного поиска элементов в множестве. В результате практического применения хеш-функций в реализации собственного динамического множества с хеш-таблицей мной был получен опыт использования хеширования на практике в целях повышения эффективности работы с элементами динамического множества, их поиска, добавления, а также удаления.