Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет

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

направление подготовки: 09.03.04 - «Программная инженерия»

Лабораторная работа

По теме **«Hash-таблицы»**

Вариант №20

Выполнял:

студент группы РИС-24-1б

Морозова Н.С.

Проверял:

Доцент кафедры ИТАС

Полякова О.А.

Постанока задачи:

1. Создать динамический массив из записей (в соответствии с вариантом), содержащий не менее 100 элементов. Для заполнения элементов массива использовать ДСЧ.
2. Предусмотреть сохранение массива в файл и загрузку массива из файла.
3. Предусмотреть возможность добавления и удаления элементов из массива (файла).
4. Выполнить поиск элемента в массиве по ключу в соответствии с вариантом. Для поиска использовать hash-таблицу.
5. Подсчитать количество коллизий при размере hash-таблицы 40, 75 и 90 элементов.

|  |  |  |  |
| --- | --- | --- | --- |
| **Данные** | **Ключ (string)** | **Хэш-функция** | **Метод рехеширования** |
| ФИО, дата\_рождения, №паспорта | дата\_рождения | H(k)= [M (kAmod1)], 0<A<1, mod1 – получение дробной части, [] – получение целой части | Метод коллизий  Метод цепочек |

Анализ задачи:

1. Создать массивы из различных данных и функции к ним для создания рандомного человека.
2. Описать структуру создания человека.
3. Описать структуру hash-таблицы, где будет использоваться hash-функция.
4. Заполняем файл данными hash-таблицы.

Код

#include <iostream>

#include <Windows.h>

#include <string>

#include <cstdlib>

#include <ctime;

using namespace std;

// Массивы для генерации случайных данных

string names[] = {

"Александр", "Максим", "Дмитрий", "Артем", "Иван", "Сергей", "Никита", "Егор", "Денис", "Андрей", "Михаил", "Роман", "Владимир", "Илья", "Станислав",

"Павел", "Виктор", "Анатолий", "Григорий", "Василий", "Кирилл", "Даниил", "Тимур", "Станислав", "Юрий", "Константин", "Федор", "Арсений", "Валентин", "Леонид",

"Савелий", "Ярослав", "Алексей", "Роман", "Виктор", "Геннадий", "Станислав", "Тихон", "Николай", "Валерий", "Анатолий", "Денис", "Сергей", "Александр", "Игорь",

"Дмитрий", "Артем", "Иван", "Никита", "Егор", "Михаил", "Роман", "Владимир", "Илья", "Станислав", "Павел", "Виктор", "Анатолий", "Григорий", "Василий",

"Кирилл", "Даниил", "Тимур", "Станислав", "Юрий", "Константин", "Федор", "Арсений", "Валентин", "Леонид", "Савелий", "Ярослав", "Алексей", "Роман", "Виктор",

"Геннадий", "Станислав", "Тихон", "Николай", "Валерий", "Анатолий", "Денис", "Сергей", "Александр", "Игорь", "Дмитрий", "Артем", "Иван", "Никита", "Егор",

"Михаил", "Роман", "Владимир", "Илья", "Станислав", "Павел", "Виктор", "Анатолий", "Григорий", "Василий"

};

string patronymic[] = {

"Александрович", "Максимович", "Дмитриевич", "Артемович", "Иванович", "Сергеевич", "Никитич", "Егорович", "Денисович", "Андреевич",

"Михайлович", "Романович", "Владимирович", "Ильич", "Станиславович", "Павлович", "Викторович", "Анатольевич", "Григорьевич", "Васильевич",

"Кириллович", "Даниилович", "Тимурович", "Станиславович", "Юрьевич", "Константинович", "Федорович", "Арсеньевич", "Валентинович", "Леонидович",

"Савельевич", "Ярославович", "Алексеевич", "Романович", "Викторович", "Геннадиевич", "Станиславович", "Тихонович", "Николаевич", "Валерьевич",

"Анатольевич", "Денисович", "Сергеевич", "Александрович", "Игоревич", "Дмитриевич", "Артемович", "Иванович", "Никитич", "Егорович", "Денисович", "Андреевич",

"Михайлович", "Романович", "Владимирович", "Ильич", "Станиславович", "Павлович", "Викторович", "Анатольевич", "Григорьевич", "Васильевич",

"Кириллович", "Даниилович", "Тимурович", "Станиславович", "Юрьевич", "Константинович", "Федорович", "Арсеньевич", "Валентинович", "Леонидович",

"Савельевич", "Ярославович", "Алексеевич", "Романович", "Викторович", "Геннадиевич", "Станиславович", "Тихонович", "Николаевич", "Валерьевич"

};

string surnames[] = {

"Иванов", "Петров", "Сидоров", "Кузнецов", "Смирнов", "Попов", "Лебедев", "Ковалев", "Новиков", "Морозов", "Соловьев", "Волков", "Алексеев", "Григорьев", "Степанов",

"Борисов", "Семенов", "Тихонов", "Федоров", "Киселев", "Зайцев", "Савельев", "Михайлов", "Кузьмин", "Сергеев", "Ковалев", "Сидоров", "Иванов", "Петров", "Смирнов",

"Половников", "Лебедев", "Ковалев", "Новиков", "Морозов", "Соловьев", "Волков", "Алексеев", "Григорьев", "Степанов", "Борисов", "Семенов", "Тихонов", "Федоров", "Киселев",

"Зайцев", "Савельев", "Михайлов", "Кузьмин", "Сергеев", "Ковалев", "Сидоров", "Иванов", "Петров", "Смирнов"

};

string phones[] = {

"+7 900 000 0000", "+7 901 111 1111", "+7 902 222 2222", "+7 903 333 3333", "+7 904 444 4444", "+7 905 555 5555", "+7 906 666 6666", "+7 907 777 7777", "+7 908 888 8888", "+7 909 999 9999",

"+7 910 000 0001", "+7 911 111 1112", "+7 912 222 2223", "+7 913 333 3334", "+7 914 444 4445", "+7 915 555 5556", "+7 916 666 6667", "+7 917 777 7778", "+7 918 888 8889", "+7 919 999 9990",

"+7 920 000 0002", "+7 921 111 1113", "+7 922 222 2224", "+7 923 333 3335", "+7 924 444 4446", "+7 925 555 5557", "+7 926 666 6668", "+7 927 777 7779", "+7 928 888 8880", "+7 929 999 9991",

"+7 930 000 0003", "+7 931 111 1114", "+7 932 222 2225", "+7 933 333 3336", "+7 934 444 4447", "+7 935 555 5558", "+7 936 666 6669", "+7 937 777 7770", "+7 938 888 8881", "+7 939 999 9992",

"+7 940 000 0004", "+7 941 111 1115", "+7 942 222 2226", "+7 943 333 3337", "+7 944 444 4448", "+7 945 555 5559", "+7 946 666 6670", "+7 947 777 7771", "+7 948 888 8882", "+7 949 999 9993",

"+7 950 000 0005", "+7 951 111 1116", "+7 952 222 2227", "+7 953 333 3338", "+7 954 444 4449", "+7 955 555 5560", "+7 956 666 6671", "+7 957 777 7772", "+7 958 888 8883", "+7 959 999 9994"

};

string ID[] = {

"1234 567890", "2345 678901", "3456 789012", "4567 890123", "5678 901234", "6789 012345", "7890 123456", "8901 234567", "9012 345678", "456789",

"1234 567891", "2345 678902", "3456 789013", "4567 890124", "5678 901235", "6789 012346", "7890 123457", "8901 234568", "9012 345679", "0123 465738",

"0123 456790", "1234 567892", "2345 678903", "3456 789014", "4567 890125", "5678 901236", "6789 012347", "7890 123458", "8901 234569", "9012 345680",

"0123 456791", "1234 567893", "2345 678904", "3456 789015", "4567 890126", "5678 901237", "6789 012348", "7890 123459", "8901 234570", "9012 345681",

"0123 456792", "1234 567894", "2345 678905", "3456 789016", "4567 890127", "5678 901238", "6789 012349", "7890 123460", "8901 234571", "9012 345682",

"0123 456793", "1234 567895", "2345 678906", "3456 789017", "4567 890128", "5678 901239", "6789 012350", "7890 123461", "8901 234572", "9012 345683",

"0123 456794", "1234 567896", "2345 678907", "3456 789018", "4567 890129", "5678 901240", "6789 012351", "7890 123462", "8901 234573", "9012 345684"

};

string dates[] = {

"01.01.1990", "02.02.1991", "03.03.1992", "04.04.1993", "05.05.1994", "06.06.1995", "07.07.1996", "08.08.1997", "09.09.1998", "10.10.1999",

"11.11.2000", "12.12.2001", "13.01.2002", "14.02.2003", "15.03.2004", "16.04.2005", "17.05.2006", "18.06.2007", "19.07.2008", "20.08.2009",

"21.09.2010", "22.10.2011", "23.11.2012", "24.12.2013", "25.01.2014", "26.02.2015", "27.03.2016", "28.04.2017", "29.05.2018", "30.06.2019",

"31.07.2020", "01.08.2021", "02.09.2022", "03.10.2023", "04.11.2024", "05.12.2025", "06.01.2026", "07.02.2027", "08.03.2028", "09.04.2029",

"10.05.2030", "11.06.2031", "12.07.2032", "13.08.2033", "14.09.2034", "15.10.2035", "16.11.2036", "17.12.2037", "18.01.2038", "19.02.2039",

"20.03.2040", "21.04.2041", "22.05.2042", "23.06.2043", "24.07.2044", "25.08.2045", "26.09.2046", "27.10.2047", "28.11.2048", "29.12.2049",

"30.01.2050", "31.02.2051", "01.03.2052", "02.04.2053", "03.05.2054", "04.06.2055", "06.08.2056", "07.09.2057", "08.10.2058", "09.11.2059",

"11.01.2061", "12.02.2062", "13.03.2063", "14.04.2064", "15.05.2065", "16.06.2066", "17.07.2067", "18.08.2068", "19.09.2069", "20.10.2070",

"21.11.2071", "22.12.2072", "23.01.2073", "24.02.2074", "25.03.2075", "26.04.2076", "27.05.2077", "28.06.2078", "29.07.2079", "30.08.2080",

"31.09.2081", "01.10.2082", "02.11.2083", "03.12.2084", "04.01.2085", "05.02.2086", "06.03.2087", "07.04.2088", "08.05.2089", "10.12.2060"

};

// Структура для хранения информации о человеке

struct Human

{

Human() {

full\_name = "NULL";

ID\_number = "NULL";

phone\_number = "NULL";

date\_of\_birth = "NULL";

}

string full\_name;

string ID\_number;

string phone\_number;

string date\_of\_birth;

// Перегрузка оператора << для вывода

friend ostream& operator<<(ostream& out, const Human& object) {

out << "ФИО: " << object.full\_name << "\nДата рождения: " << object.date\_of\_birth

<< "\nТелефон: " << object.phone\_number << "\nНомер паспорта: " << object.ID\_number << endl << endl;

return out;

}

// Перегрузка оператора >> для ввода

friend istream& operator>>(istream& in, Human& object) {

cout << "Введите ФИО: ";

getline(in, object.full\_name);

cout << "Введите дату рождения (дд.мм.гггг): ";

getline(in, object.date\_of\_birth);

cout << "Введите номер телефона: ";

getline(in, object.phone\_number);

cout << "Введите номер паспорта: ";

getline(in, object.ID\_number);

return in;

}

};

// Функция для генерации случайного числа

int rand\_num() {

return rand() % 50;

}

string rand\_data() {

return dates[rand\_num()];

}

string rand\_name() {

return surnames[rand\_num()] + " " + names[rand\_num()] + " " + patronymic[rand\_num()];

}

string rand\_ID() {

return ID[rand\_num()];

}

string rand\_phon() {

return phones[rand\_num()];

}

// Функция для создания случайного человека

Human create\_random\_human()

{

Human temp;

temp.full\_name = rand\_name();

temp.ID\_number = rand\_ID();

temp.phone\_number = rand\_phon();

temp.date\_of\_birth = rand\_data();

return temp;

}

// Функция для вывода массива людей

void print\_human(const Human\* const array, const int size) {

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

cout << array[i];

}

// Функция для заполнения массива случайными людьми

void Fill\_rand(Human\* array, const int count) {

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

array[i] = create\_random\_human();

}

// Хэш-функция

int hash\_function(string str, const int size) {

double k = (stoi(str.substr(0, 2)) + stoi(str.substr(3, 2)) \* 31 + stoi(str.substr(6, 4))) \* 365;

double A = (sqrt(5.0) - 1) / 2.0; // ≈ 0.6180339887

double c = fmod(k \* A, 1.0);

return static\_cast<int>(c \* size);

}

// Структура для хэш-таблицы

struct hash\_table

{

Human\* array;

int collisions\_count = 0;

// Конструктор

hash\_table(int size) {

array = new Human[size];

}

// Деструктор

~hash\_table() {

delete[] array;

}

void find\_index(string date\_of\_birth, const int size);

void add(Human temp, const int size);

void pop(string date\_to\_delete, const int size);

// Функция для отображения информации о человеке

void show\_info(const Human temp) {

cout << "ФИО: " << temp.full\_name

<< "\nДата рождения: " << temp.date\_of\_birth

<< "\nТелефон: " << temp.phone\_number

<< "\nНомер паспорта: " << temp.ID\_number << endl << endl;

}

void show\_hash\_table(const int size) {

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

show\_info(array[i]); // Используем this->array

}

};

void hash\_table::find\_index(string date\_of\_birth, const int size)

{

int hash = hash\_function(date\_of\_birth, size); // Вычисляем хэш

int index = hash; // Начальное значение индекса

// Линейный поиск до конца массива

while (array[index].date\_of\_birth != date\_of\_birth && index < size) {

index++;

}

// Если достигнут конец массива, начинаем поиск с начала

if (index == size) {

index = 0; // Возвращаемся к началу массива

while (array[index].date\_of\_birth != date\_of\_birth && index != hash)

index++;

}

// Проверка, найден ли элемент

if (index == hash && array[index].date\_of\_birth != date\_of\_birth)

cout << "Человека с датой рождения \"" << date\_of\_birth << "\" нет.\n\n";

else

cout << "Человек с датой рождения \"" << date\_of\_birth << "\" находится по индексу " << index << endl << endl;

}

void hash\_table::add(Human temp, const int size)

{

int index = hash\_function(temp.date\_of\_birth, size); // Вычисляем хэш

int original\_index = index; // Запоминаем начальный индекс для проверки закольцовывания

// Если место свободно, добавляем объект

if (array[index].full\_name == "NULL") {

array[index] = temp;

return;

}

else

{

// Линейное зондирование до конца массива

while (index < size) {

if (array[index].full\_name == "NULL") { // Если место свободно

array[index] = temp;

return;

}

index++; // Переходим к следующему индексу

collisions\_count++; // Увеличиваем счётчик коллизий

}

// Если индекс вышел за пределы массива, начинаем с начала

if (index >= size) {

index = 0;

// Поиск свободного места до исходного индекса

while (index < original\_index) {

if (array[index].full\_name == "NULL") { // Если место свободно

array[index] = temp;

return;

}

index++; // Переходим к следующему индексу

collisions\_count++; // Увеличиваем счётчик коллизий

}

}

cout << "Хэш-таблица заполнена. Невозможно добавить объект.\n";

}

}

void hash\_table::pop(string date\_to\_delete, const int size)

{

int index = hash\_function(date\_to\_delete, size);

int original\_index = index; // Запоминаем начальный индекс для проверки закольцовывания

// Если объект найден по начальному индексу

if (array[index].date\_of\_birth == date\_to\_delete) {

array[index] = Human(); // Удаляем объект, заменяя его на пустой

return;

}

else

{

// Линейная проверка до конца массива

while (index < size) {

if (array[index].date\_of\_birth == date\_to\_delete) { // Если объект найден

array[index] = Human(); // Удаляем объект

return;

}

index++; // Переходим к следующему индексу

}

// Если индекс вышел за пределы массива, начинаем с начала

if (index >= size)

{

index = 0;

// Поиск объекта до исходного индекса

while (index < original\_index) {

if (array[index].date\_of\_birth == date\_to\_delete) {

array[index] = Human();

return;

}

index++;

}

}

}

// Если объект не найден

cout << "Человека с датой рождения \"" << date\_to\_delete << "\" нет." << endl;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(0));

// Ввод размера массива

int size;

do {

cout << "Введите количество элементов массива: ";

cin >> size;

} while (size > 100);

cout << endl;

// Создание массива людей

Human\* array = new Human[size];

hash\_table table(size);

Fill\_rand(array, size);

// Добавление элементов в хэш-таблицу

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

table.add(array[i], size);

}

table.show\_hash\_table(size);

table.find\_index("02.09.2022", size);

table.pop("02.09.2022", size);

cout << endl << endl << "Обновлённая хэш-таблица:" << endl << endl;

table.show\_hash\_table(size);

cout << "Количество коллизий: " << table.collisions\_count << endl << endl;

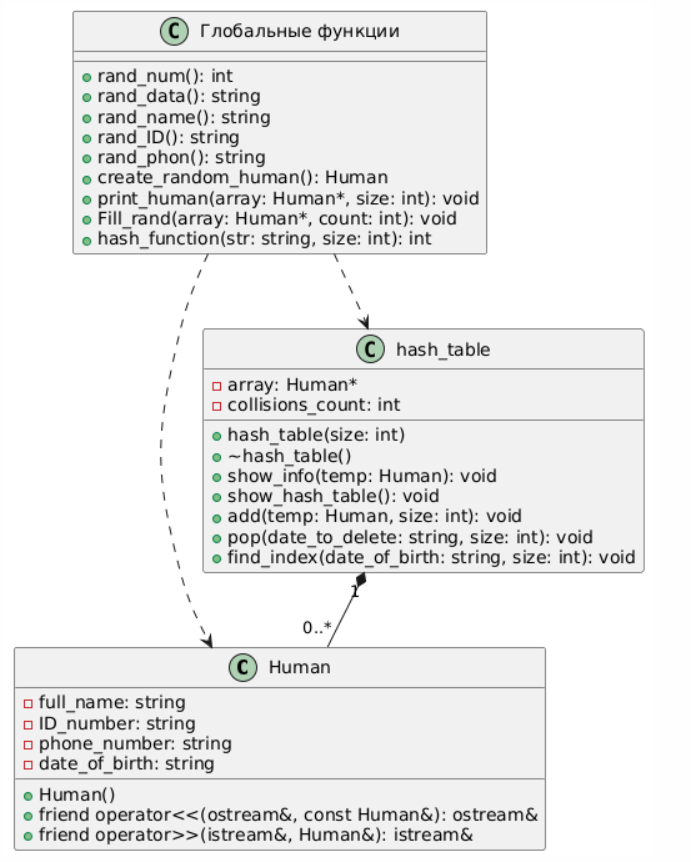
// Освобождение памяти

delete[] array;

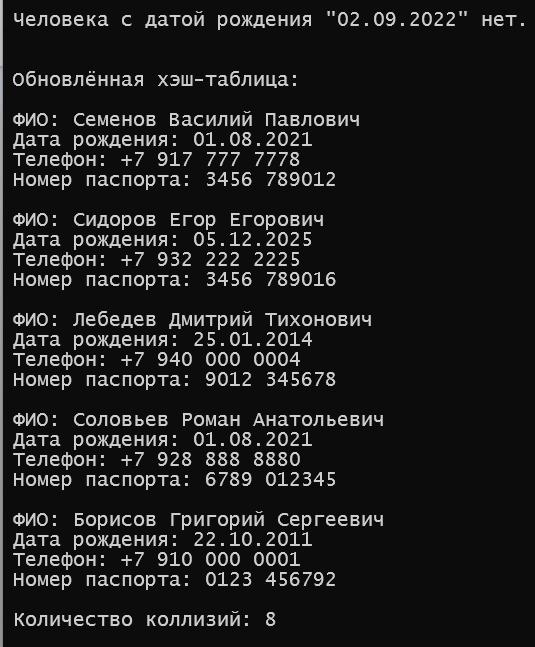
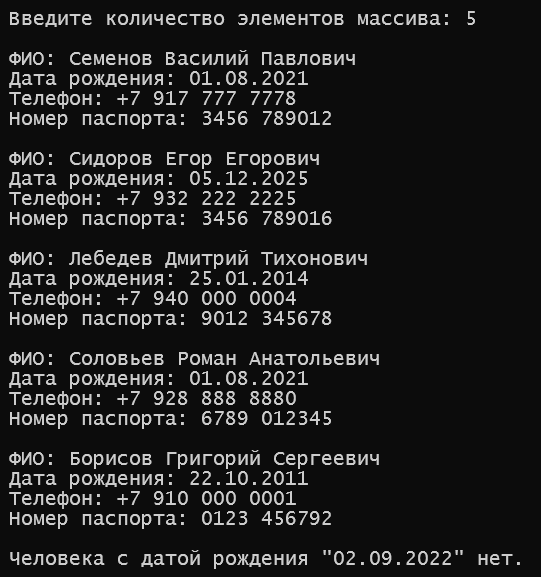
return 0;

}

UML-диаграмма



Пример работы программы:



Коллизии при размере 40: от 124 до 182

Коллизии при размере 75: от 293 до 619

Коллизии при размере 90: от 485 до 1069

1. **й вариант заполнения таблицы:**

Изменения в структуре hash-таблицы

struct hash\_table

{

vector<list<Human>> table; // Массив списков для хранения цепочек

int collisions\_count = 0;

hash\_table(int size) {

table.resize(size); // Инициализируем массив пустыми списками

}

void add(Human temp, const int size);

void pop(string date\_to\_delete, const int size);

void find\_index(string date\_of\_birth, const int size);

void show\_hash\_table(const int size);

};

void hash\_table::find\_index(string date\_of\_birth, const int size)

{

int index = hash\_function(date\_of\_birth, size);

// Ищем элемент в списке

for (const auto& human : table[index]) {

if (human.date\_of\_birth == date\_of\_birth) {

cout << "Человек найден в цепочке с индексом " << index << endl;

return;

}

}

cout << "Человека с датой рождения \"" << date\_of\_birth << "\" нет." << endl;

}

void hash\_table::add(Human temp, const int size)

{

int index = hash\_function(temp.date\_of\_birth, size);

// Если список не пустой, увеличиваем счётчик коллизий

if (!table[index].empty()) {

collisions\_count++;

}

// Добавляем элемент в список

table[index].push\_back(temp);

}

void hash\_table::pop(string date\_to\_delete, const int size)

{

int index = hash\_function(date\_to\_delete, size);

// Ищем элемент в списке

for (auto it = table[index].begin(); it != table[index].end(); ++it) {

if (it->date\_of\_birth == date\_to\_delete) {

table[index].erase(it); // Удаляем элемент

return;

}

}

cout << "Человека с датой рождения \"" << date\_to\_delete << "\" нет." << endl;

}

void hash\_table::show\_hash\_table(const int size) {

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

if (!table[i].empty()) {

cout << "--- Индекс " << i << " ---" << endl;

for (const auto& human : table[i]) {

cout << human << endl;

}

}

}

}

void hash\_table::show\_hash\_table(const int size)

{

ofstream file("C:\\Users\\user\\Desktop\\C++\\Х-табл2\\table.txt");

if (!file.is\_open())

cout << "Не удалось открыть файл!" << endl;

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

if (!table[i].empty()) {

cout << endl;

file << "--- Индекс " << i << " ---" << endl;

for (const auto& human : table[i]) {

cout << human << endl;

file << human << endl;

}

}

}

}

//изменённая главная функция

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(0));

// Ввод размера массива

int size;

do {

cout << "Введите количество заполняемых людей: ";

cin >> size;

} while (size > 100);

cout << endl;

Human\* array = new Human[size];

hash\_table table(size);

Fill\_rand(array, size);

// Добавление элементов в хэш-таблицу

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

table.add(array[i], size);

}

table.show\_hash\_table(size);

cout << "Количество коллизий: " << table.collisions\_count << endl << endl;

string dat;

cout << "Введите дату рождения человека, которого нужно удалить ДД.ММ.ГГГГ: ";

cin >> dat;

table.find\_index(dat, size);

table.pop(dat, size);

// Повторное отображение хэш-таблицы

cout << endl << endl << "Обновлённая хэш-таблица:" << endl << endl;

table.show\_hash\_table(size);

cout << "Количество коллизий: " << table.collisions\_count << endl << endl;

delete[] array;

return 0;

}

**Результат - меньше коллизий**

При размере 40: от 16 до 21

При размере 75: от 41 до 45

При размере 90: от 47 до 54

Пример работы программы:

