

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

|  |
| --- |
|  |

**ОТЧЕТ**

**ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2**

Хеширование: прямой доступ к данным

**по дисциплине**

«Структуры и алгоритмы обработки данных»

Выполнил студент группы ИКБО-01-21 Луковников Д.Р.

Принял преподаватель Туманова М.Б.

Практическая «\_\_»\_\_\_\_\_\_\_2022 г. \_\_\_\_\_\_\_\_\_\_

работа выполнена

«Зачтено» «\_\_»\_\_\_\_\_\_\_2022 г. \_\_\_\_\_\_\_\_\_\_

Москва 2022

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc114518106)

[ЦЕЛЬ РАБОТЫ 3](#_Toc114518107)

[ХОД РАБОТЫ 4](#_Toc114518108)

[1.1 Задание 1 4](#_Toc114518109)

[ВЫВОДЫ 11](#_Toc114518110)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 12](#_Toc114518111)

[ПРИЛОЖЕНИЯ 13](#_Toc114518112)

# ЦЕЛЬ РАБОТЫ

Освоить приёмы хеширования и эффективного поиска элементов множества. Реализовать программу используя полученные знания.

# ХОД РАБОТЫ

## **Задание 1**

**Формулировка задачи:** Разработайте приложение, которое использует хеш-таблицу 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.

Индивидуальный вариант: Открытая адресация (квадратичное пробирование. Страховой полис: номер, компания, фамилия владельца.

**Математическая модель решения:** Хеш-функция будет реализована на основе деления. Берётся номер полиса и нацело делиться на текущий размер таблицы, если возникает коллизия, тогда применяется метод двойного пробирования. Хэш каждой новой попытки в котором высчитывается по следующей формуле:

Адрес=h(x)+ci+di2

Удаление: если нам необходимо удалить элемент, то мы не просто удаляем элемент, мы помечаем его специальным флагом, в данном случае ссылкой на объект дочернего класса, который программа воспринимает как удалённый.

**Код программы:** начнём с генератора случайных записей, по заданию мы имеем номер полиса, название компании и фамилия владельца. Поскольку номер состоит из 16-и цифр для его хранения используем unsigned long long. Длинна компании и фамилия будет до 7 до 13 символов и начинаться с большой буквы листинг 1.1

Листинг 1.1 – Генератор рандомных полисов

void generateList(HashTable &table, int size = 1) {

/\*\*

\* The function generates a certain number of random records.

\*/

// Random utilities

default\_random\_engine u{};

uniform\_int\_distribution<> d{};

u.seed(random\_device()()); // Analog srand

InsurancePolicy \*insurancePolicy;

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

// Capital first letter generation

vector<char> company\_t({char(toupper(char(rand() % 26 + 0x61)))});

vector<char> surname\_t({char(toupper(char(rand() % 26 + 0x61)))});

// Generate random number of letters

for (int j = 0; j < d(u, uniform\_int\_distribution<>::param\_type{6, 12}); ++j) {

company\_t.push\_back(rand() % 26 + 0x61);

}

for (int j = 0; j < d(u, uniform\_int\_distribution<>::param\_type{6, 12}); ++j) {

surname\_t.push\_back(rand() % 26 + 0x61);

}

// Convert char vector to string

string company(company\_t.begin(), company\_t.end());

string surname(surname\_t.begin(), surname\_t.end());

insurancePolicy = new InsurancePolicy(

BORDER\_BOTTOM +

(((unsigned long long) d(u, uniform\_int\_distribution<>::param\_type{1000000, RAND\_MAX}) \*

(unsigned long long) d(u, uniform\_int\_distribution<>::param\_type{1000000, RAND\_MAX})) %

(BORDER\_TOP - BORDER\_BOTTOM)), company, surname

);

// Add policy to table

if (!table.add(insurancePolicy)) {

i--;

}

}

}

После генерации мы добавляем запись в хэш таблицу. В классе таблицы поле, отвечающее за хранение данных, будет является контейнером, а именно вектором, в связи с его удобством. Так же поля для хранения, текущего размер, текущего заполнения, количества удалений, а также указатель на объект, который используется как флаг удаления, листинг 1.2.

Листинг 1.2 – Свойства класса таблицы

class HashTable {

int size;

int filled;

int deleted;

vector<InsurancePolicy \*> data;

PoliceDelete \*deleteFlag;

...

Хэш функция – в данный реализации было решено использовать самый простой подход, а именно подход деления, благодаря этому хэш функция выглядит следующим образом, листинг 1.3

Листинг 1.3 – Хэш функция

int hashFunction(unsigned long long policy) {

/\*\*

\* Hash function based division

\*/

return (int) (policy % size);

}

Процедура добавления нового полиса. Для начала идёт проверка на эффективность использования текущего размера таблицы (как только заполнено более 75% таблица увеличивается в 2 раза и рехешируеться). После этого проверяется уникальность текущего элемента и только после этого запись ставиться в таблицу. Если возникает коллизия мы исправляем её с помощью двоёного пробирования, Листинг 1.4.

Листинг 1.4 – Добавление элемента

bool add(InsurancePolicy \*insurancePolicy) {

/\*\*

\* Adding a new policy to the table.

\*/

// Checking for efficiency using the current size

if (float(filled + deleted) / float(size) > MAX\_FILLED) rehash();

// Code uniqueness check

for (auto el: data)

if (el && el->number == insurancePolicy->number)

return false;

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

// We calculate the hash function until we hit an empty cell

int code =

(hashFunction(insurancePolicy->number) + CONST\_C \* i + CONST\_D \* i \* i) % size; // Quadratic probing

if (data[code] == NULL) {

filled++;

data[code] = insurancePolicy;

return true;

}

}

return false;

}

Рехеширование: оно происходит автоматически, когда размер текущей таблицы становиться не эффективным. Данные копируются, основная таблица очищается и все элементы добавляются в новую таблицу, Листинг 1.5.

Листинг 1.5 – Рехеширование

void rehash() {

/\*\*

\* Rehashing a table when it overflows.

\*/

vector<InsurancePolicy \*> data\_t = data;

data.clear();

filled = 0;

deleted = 0;

size \*= 2;

data.resize(size);

for (auto el: data\_t)

if (el && el->isAlive())

add(el);

}

Получение полиса по номеру осуществляется по той же логике, что и добавление, Листинг 1.6.

Листинг 1.6 – Получение элемента

int getCodeByPoliceNumber(unsigned long long policyNumber) {

/\*\*

\* Getting the hash of the policy by its number.

\*/

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

int code = (hashFunction(policyNumber) + CONST\_C \* i + CONST\_D \* i \* i) % size;

if (data[code] && data[code]->number == policyNumber)

return code;

}

return -1;

}

InsurancePolicy \*getPolicyByNumber(unsigned long long policyNumber) {

/\*\*

\* Returns a link to the policy object by its number.

\*/

// Getting code

int code = getCodeByPoliceNumber(policyNumber);

return (code != -1 ? data[code] : nullptr);

}

Удаление элемента. Удаление происходит путём замены элемента на дочерний с переопределённым методом isAlive. Так же изменяются свойства таблицы (заполнено/удалено) Листинг 1.7.

Листинг 1.7 – Удаление элемента

bool deleteByPolicyNumber(unsigned long long policyNumber) {

/\*\*

\* Deleting an entry in the table by policy number.

\*/

int code = getCodeByPoliceNumber(policyNumber);

if (code != -1) {

cout << code << endl;

InsurancePolicy \*policy\_t = data[code];

data[code] = deleteFlag;

deleted++;

filled--;

return true;

} else {

return false;

}

}

Интерфейс взаимодействия реализован через бесконечный цикл.

**Тестирование:** с помощью специальной команды сгенерируем 4 записи и выведем таблицу результат на рисунке 1.

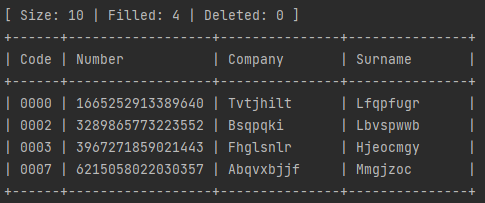


Рисунок 1 – Случайная генерация

Попробуем получить запись, которая имеет хэш 0 по номеру полиса, рисунок 2.

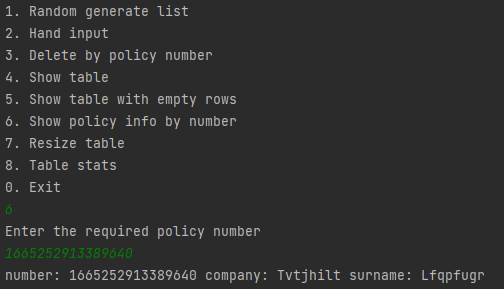


Рисунок 2 – Получение полиса по номеру

Затем удалим его и повторим поиск, рисунок 3.

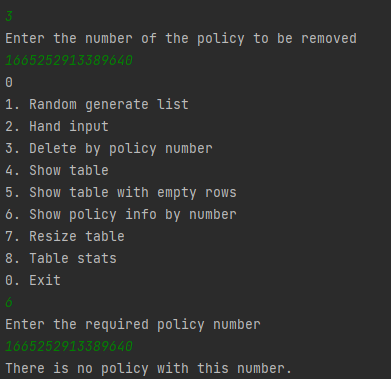


Рисунок 3 – Удаление

Как видим, программа более не может найти элемент. Проверим работу коллизии, для этого возьмём несколько полисов с номерами заканчивающихся на 0 (т.к. изначальный размер таблицы 10, а хэш таблица построена на делении), например 100, 200, 300, результат на рисунке 4.

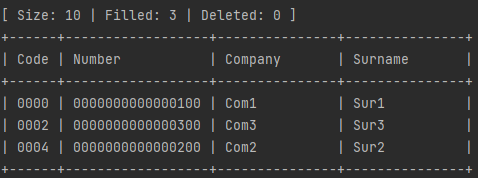


Рисунок 4 – Разрешение коллизии

Видим, что записи распределились по таблице. Так же присутствует функционал ручного изменения размера таблицы, рисунок 5.

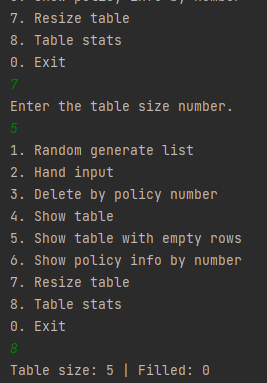


Рисунок 5 – Изменение размера

# ВЫВОДЫ

При выполнении работы были получены навыки реализации хэш таблиц, в частности хэш таблицы с открытой адресацией и квадратичным пробированием.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
2. Документация по языку С++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 01.09.2021).
3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 01.09.2021).

# ПРИЛОЖЕНИЯ

Приложение А – Исходный код программы хэш таблицы

**Приложение А**

Исходный код программы хэш таблицы

Листинг 2.1 – main.cpp

//

// Created by Lukov on 17.09.2022.

//

#include <iostream>

#include <string>

#include <utility>

#include <vector>

#include <random>

#include <ctime>

#include <algorithm>

#define BORDER\_BOTTOM 1000000000000000

#define BORDER\_TOP 9999999999999999

#define MAX\_FILLED 0.75

#define CONST\_C 2

#define CONST\_D 2

using namespace std;

class InsurancePolicy {

/\*\*

\* Medical policy class with number, company and last name.

\*/

public:

unsigned long long number;

string company;

string surname;

InsurancePolicy(unsigned long long number, string company, string surname) : number(number),

company(move(company)),

surname(move(surname)) {}

friend ostream &operator<<(ostream &os, const InsurancePolicy &policy) {

/\*\*

\* Overloading the cast-to-string operator.

\*/

os << "number: " << policy.number << " company: " << policy.company << " surname: " << policy.surname;

return os;

}

virtual bool isAlive() { return true; }

};

class PoliceDelete : public InsurancePolicy {

/\*\*

\* A class for tracking deleted entry in a table.

\*/

public:

PoliceDelete() : InsurancePolicy(0, "", "") {}

Продолжение Листинг 2.1

bool isAlive() override { return false; }

};

class HashTable {

int size;

int filled;

int deleted;

vector<InsurancePolicy \*> data;

PoliceDelete \*deleteFlag;

public:

int getSize() const {

return size;

}

int getFilled() const {

return filled;

}

explicit HashTable(int size = 10) : size(size), filled(0), deleted() {

/\*\*

\* Constructor with one param (explicit)

\* Default size = 10

\*/

data.resize(size);

deleteFlag = new PoliceDelete();

}

bool add(InsurancePolicy \*insurancePolicy) {

/\*\*

\* Adding a new policy to the table.

\*/

// Checking for efficiency using the current size

if (float(filled + deleted) / float(size) > MAX\_FILLED) rehash();

// Code uniqueness check

for (auto el: data)

if (el && el->number == insurancePolicy->number)

return false;

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

// We calculate the hash function until we hit an empty cell

int code =

(hashFunction(insurancePolicy->number) + CONST\_C \* i + CONST\_D \* i \* i) % size; // Quadratic probing

if (data[code] == NULL) {

filled++;

data[code] = insurancePolicy;

return true;

}

}

return false;

}

void showTableSeparator(int numSize) {

/\*\*

\* Table Separator

\*/

Продолжение Листинг 2.1

cout << '+' << string(numSize + 2, '-') << '+' << string(18, '-') << '+' << string(15, '-') <<

'+' << string(15, '-') << "+\n";

}

void showTable(bool withEmpty = false) {

/\*\*

\* Show table with all records (optional, show empty entry in table)

\*/

int numSize = to\_string(size).size();

numSize = (numSize < 4 ? 4 : numSize); // Len of index (minimum 4)

cout << "[ Size: " << size << " | Filled: " << filled << " | Deleted: " << deleted << " ] \n";

showTableSeparator(numSize);

cout << "| Code" << string(numSize - 3, ' ') << "| Number | Company | Surname | \n";

showTableSeparator(numSize);

// Table data output

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

if (data[i] && data[i]->isAlive())

printf("| %0\*d | %016lld | %-13s | %-13s | \n", numSize, i, data[i]->number, data[i]->company.c\_str(),

data[i]->surname.c\_str());

else if (withEmpty)

printf("| %0\*d | %16s | %13s | %13s | \n", numSize, i, "", "", "");

showTableSeparator(numSize);

}

int hashFunction(unsigned long long policy) {

/\*\*

\* Hash function based division

\*/

return (int) (policy % size);

}

void rehash() {

/\*\*

\* Rehashing a table when it overflows.

\*/

vector<InsurancePolicy \*> data\_t = data;

data.clear();

filled = 0;

deleted = 0;

size \*= 2;

data.resize(size);

for (auto el: data\_t)

if (el && el->isAlive())

add(el);

}

int getCodeByPoliceNumber(unsigned long long policyNumber) {

/\*\*

\* Getting the hash of the policy by its number.

\*/

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

int code = (hashFunction(policyNumber) + CONST\_C \* i + CONST\_D \* i \* i) % size;

if (data[code] && data[code]->number == policyNumber)

Продолжение Листинг 2.1

return code;

}

return -1;

}

bool deleteByPolicyNumber(unsigned long long policyNumber) {

/\*\*

\* Deleting an entry in the table by policy number.

\*/

int code = getCodeByPoliceNumber(policyNumber);

if (code != -1) {

cout << code << endl;

InsurancePolicy \*policy\_t = data[code];

data[code] = deleteFlag;

deleted++;

filled--;

return true;

} else {

return false;

}

}

InsurancePolicy \*getPolicyByNumber(unsigned long long policyNumber) {

/\*\*

\* Returns a link to the policy object by its number.

\*/

// Getting code

int code = getCodeByPoliceNumber(policyNumber);

return (code != -1 ? data[code] : nullptr);

}

bool resize(int newSize) {

/\*\*

\* Manual resizing of the table with a check for the effectiveness of the resulting table.

\*/

if (float(filled) / float(newSize) > MAX\_FILLED) return false; // Efficiency test

vector<InsurancePolicy \*> data\_t = data; // Copy data

data.clear(); // Clear current vector

size = newSize;

deleted = 0;

data.resize(size);

// Add entry to a new table

for (auto el: data\_t)

if (el && el->isAlive())

add(el);

return true;

}

// virtual ~HashTable() {

// for (auto el: data) {

// delete[]el;

// }

// }

};

Продолжение Листинг 2.1

void generateList(HashTable &table, int size = 1) {

/\*\*

\* The function generates a certain number of random records.

\*/

// Random utilities

default\_random\_engine u{};

uniform\_int\_distribution<> d{};

u.seed(random\_device()()); // Analog srand

InsurancePolicy \*insurancePolicy;

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

// Capital first letter generation

vector<char> company\_t({char(toupper(char(rand() % 26 + 0x61)))});

vector<char> surname\_t({char(toupper(char(rand() % 26 + 0x61)))});

// Generate random number of letters

for (int j = 0; j < d(u, uniform\_int\_distribution<>::param\_type{6, 12}); ++j) {

company\_t.push\_back(rand() % 26 + 0x61);

}

for (int j = 0; j < d(u, uniform\_int\_distribution<>::param\_type{6, 12}); ++j) {

surname\_t.push\_back(rand() % 26 + 0x61);

}

// Convert char vector to string

string company(company\_t.begin(), company\_t.end());

string surname(surname\_t.begin(), surname\_t.end());

insurancePolicy = new InsurancePolicy(

BORDER\_BOTTOM +

(((unsigned long long) d(u, uniform\_int\_distribution<>::param\_type{1000000, RAND\_MAX}) \*

(unsigned long long) d(u, uniform\_int\_distribution<>::param\_type{1000000, RAND\_MAX})) %

(BORDER\_TOP - BORDER\_BOTTOM)), company, surname

);

// Add policy to table

if (!table.add(insurancePolicy)) {

i--;

}

}

}

int main() {

srand(time(NULL));

setlocale(LC\_ALL, "rus");

HashTable table;

int ex = -1, size\_t;

unsigned long long number\_t;

string company\_t, surname\_t;

while (ex) {

cout << "1. Random generate list\n"

"2. Hand input \n"

"3. Delete by policy number\n"

"4. Show table\n"

"5. Show table with empty rows\n"

"6. Show policy info by number\n"

Продолжение Листинг 2.1

"7. Resize table\n"

"8. Table stats\n"

"0. Exit\n";

cin >> ex;

switch (ex) {

case 1:

cout << "New entries:";

cin >> size\_t;

generateList(table, size\_t);

break;

case 2:

cout << "Enter Police number, Company and Surname\n";

cin >> number\_t >> company\_t >> surname\_t;

if (!table.add(new InsurancePolicy(number\_t, company\_t, surname\_t)))

cout << "A policy with this number already exists.\n";

break;

case 3:

cout << "Enter the number of the policy to be removed\n";

cin >> number\_t;

if (!table.deleteByPolicyNumber(number\_t))

cout << "There is no policy with this number.\n";

break;

case 4:

table.showTable();

break;

case 5:

table.showTable(true);

break;

case 6:

cout << "Enter the required policy number\n";

cin >> number\_t;

InsurancePolicy \*policy\_t;

policy\_t = table.getPolicyByNumber(number\_t);

if (policy\_t) {

cout << \*policy\_t << endl;

} else {

cout << "There is no policy with this number.\n";

}

break;

case 7:

cout << "Enter the table size number. \n";

cin >> size\_t;

if (!table.resize(size\_t)) {

cout << "Using a table of this size is inefficient. \n";

}

break;

case 8:

cout << "Table size: " << table.getSize() << " | Filled: " << table.getFilled() << endl;

break;

case 0:

cout << "Bye, bye!\n";

break;

default:

cout << "Oops, Something went wrong! Command not found. \n";

break;

}

}

return 0;

}