МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Хеш-таблицы с цепочками — вставка и исключение

Студент гр. 9381 _____ Камакин Д.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Камакин Д.В.

Группа 9381

Тема работы:

Вариант 23. Хеш-таблицы с цепочками – вставка и исключение.

Демонстрация.

Исходные данные:

строка, слова из которой необходимо добавить в хеш-таблицу

Содержание пояснительной записки:

«Содержание», «Введение», «Ход выполнения работы», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 08.12.2020

Дата защиты реферата: 10.12.2020

Студент _____ Камакин Д.В.

Преподаватель _____ Фирсов М.А.

АННОТАЦИЯ

На языке программирования C++ был реализован класс хеш-таблицы с цепочками. Продемонстирорована вставка, удаление и подсчёт введённых элементов. Был написан интерфейс для работы с программой с консоли.

SUMMARY

A class of a hash table with chains was implemented in the C++ programming language. Inserting, deleting, and counting the entered elements is demonstrated. An interface was written for working with the program from the console.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является написание программы, реализующей хеш-таблицу с цепочками. Для этого необходимо изучить соответствующую структуру данных, операции вставки, удаления и поиска элементов в ней, а также синтаксис языка программирования C++.

Хеш-таблица (hash table) — это специальная структура данных для хранения пар ключей и их значений. По сути это ассоциативный массив, в котором ключ представлен в виде хеш-функции.

Пожалуй, главное свойство hash-таблиц — все три операции вставка, поиск и удаление в среднем выполняются за время O(1), среднее время поиска по ней также равно O(1) и O(n) в худшем случае.

Метод цепочек часто называют открытым хешированием. Его суть проста — элементы с одинаковым хешем попадают в одну ячейку в виде связного списка.

То есть, если ячейка с хешем уже занята, но новый ключ отличается от уже имеющегося, то новый элемент вставляется в список в виде пары ключ-значение.

Если выбран метод цепочек, то вставка нового элемента происходит за O(1), а время поиска зависит от длины списка и в худшем случае равно O(n)

1. ЗАДАНИЕ

Требуется на языке программирования C++ реализовать хеш-таблицу, разрешив коллизию методом цепочек. Кроме того, необходимо спроектировать интерфейс для работы с программой, продемонстрировав основные операции с хеш-таблицей.

Программа должна предоставлять основной функционал для работы с хеш-таблицей (вставка/удаление/поиск), а также выводить справочную информацию пользователю, меню для работы и промежуточные выводы.

2. ОПИСАНИЕ АЛГОРИТМА

2.1.Хеш-таблица

Согласно условию задачи, требуется реализовать хеш-таблицу. Для этого на языке программирования C++ был написан соответствующий класс, содержащий API для основных операций (вставка/удаление/поиск). Для хранения данных использованы контейнеры STL (std::vector и std::list).

2.2.Разрешение коллизии

Как было сказано ранее, хеш-таблица — контейнер для хранения пар ключей и их значений, где ключи — хешы элементов, которые могут пересекаться. Для разрешения этой проблемы и используется std::list (метод цепочек, в котором каждый элемент таблицы — список, в каждом хранятся элементы с одинаковым хешем). Если происходит коллизия, то новый элемент добавляется в конец списка, а сами списки хранятся в векторе. Таким образом, при обращении к индексу вектора мы получаем список элементов с одинаковым хешем.

2.3.Вычисление хеша

Вычисление хеша происходит в цикле (с 0 до длины элемента), на выбор пользователю предоставлены следующие формулы:

1. hash = 37 * hash + element[i],

2. hash += element[i],

hash изначально равняется 0, i — счётчик цикла, element — элемент, хеш которого требуется вычислить.

2.4.Основные операции

Вставка реализована следующим образом: вычисляем хеш элемента, затем обращаемся по индексу хеша в векторе, получаем тем самым список, в конец которого добавляем элемент.

Удаление похоже на вставку: вычисляем хеш элемента, обращаемся по индексу хеша в векторе, получаем список, производим обход по нему, сравнивая элементы с тем, который требуется удалить. Если произошло совпадение — удаляем его из списка.

Поиск элемента реализован по следующему алгоритму: заводим счётчик повторений элемента, изначально приравниваем его к 0, вычисляем хеш элемента, обращаемся по индексу хеша в векторе, получаем список, производим его обход, сравнивая элементы с тем, который необходимо посчитать, при совпадении увеличиваем счётчик.

3. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ФУНКЦИЙ

3.1.Класс хеш-таблицы HashTable

Для работы с хеш-таблицей был реализован шаблонный класс HashTable, приватными полями которого являются: int size_ - размер таблицы, std::vector < std::list<T> > table_ - хеш-таблица, std::shared_ptr< HashTableState<T> > state_ - умный указатель состояния таблицы, в зависимости от которого хеш высчитывается по формуле (1) или (2).

Рассмотрим публичные методы класса HashTable:

1.void add(T value) — добавление элемента в хеш-таблицу. T value — элемент, который необходимо добавить. При помощи вызова функции hash() высчитываем хеш, после чего добавляем по нему наш элемент.

2.int count(T value) — поиск элемента в хеш-таблице. T value — элемент, который необходимо посчитать. Получаем хеш функцией hash(), после чего считаем и возвращаем количество элементов по данному хешу. Используется цикл foreach(). Возвращает количество повторений элемента в таблице.

3.int getHashSimple(T value) — возвращает хеш элемента. T value — элемент, хеш которого необходимо посчитать. Считается на основе формулы (2).

3.int getHashAdvanced(T value) — возвращает хеш элемента. T value — элемент, хеш которого необходимо посчитать. Считается на основе формулы (1).

4.void remove(T value) — удаление элемента из хеш-таблицы. T value — элемент, который необходимо удалить.

5.void setState(std::shared_ptr< HashTableState<T> > state) — принимает умный указатель на состояние, после чего устанавливет его в поле state_.

6.void resize(int newSize) — изменение размера таблицы, принимает int newSize — новый размер. Переситывает все хеши старой таблицы относительно нового размера в новой таблице и меняет таблицы местами.

3.2.Класс промежуточных выводов HashTableOutput

Данный класс предоставляет методы для промежуточных выводов. Приватными полями являются: HashTable<T> &table_ - ссылка на хеш-таблицу, с которой необходимо работать, pOStream output_ - указатель на поток вывода.

Рассмотрим публичные методы класса:

1.void remove(T value) — принимает T value — элемент, который требуется удалить. Выводит все элементы таблицы, выделяя цветом тот, который требутся удалить и вызывает метод remove() хеш-таблицы.

2.void add(std::vector<T> &value) — принимает ссылку на вектор значений, которые необходимо добавить в таблицу, после чего заносит их в список и выводит промежуточную информацию.

3.std::map<T, int> count(std::vector<T> &value) - принимает ссылку на вектор значений, которые необходимо посчитать. Возвращает std::map<T, int> - словарь, в котором хранятся значения с количеством их повторений.

4.void printTable(std::ostream &out) — принимает std::ostream &out — ссылку на поток вывода, куда заносятся все элементы таблицы.

5.bool isIn(std::vector<T> &vector, T value) — проверяет, содержится ли элемент T value в векторе std::vector<T> &vector, возвращает bool в зависимости от результата.

6.std::map<int, std::vector<T>> getHashMap(std::vector<T> &value) — возвращает std::map<int, std::vector<T>> - словарь, полученный в результате обработки вектора std::vector<T> &value.

3.3.Класс состояния таблицы HashTableState

Используется для реализации паттерна Состояния. Это абстрактный класс состояния хеш-таблицы, который содержится в её приватном поле. В зависимости от его значения вызываются разные методы для подсчёта хеша элемента.

Методы:

virtual int hash(HashTable<T> &map, T value, pOStream output) = 0 — виртуальная функций подсчёта хеша, принимает HashTable<T> &map — ссылка на хеш-таблицу, T value — элемент, pOStream — поток вывода информации при необходимости

3.4.Класс состояния AdvancedState

Наследник абстрактного класса HashTableState, реализующий метод hash().

Методы:

int hash(HashTable<T> &map, T value, pOStream output) override — высчитывает хеш элемента по формуле (1).

3.5.Класс состояния SimpleState

Наследник абстрактного класса HashTableState, реализующий метод hash().

Методы:

int hash(HashTable<T> &map, T value, pOStream output) override — высчитывает хеш элемента по формуле (2).

3.6.Описание функций

1.void outputHelp(std::ostream &output) — выводит в output справку по использованию программы. std::ostream &output — ссылка на поток вывода.

2.int getAction(std::istream &input) — считывает из input выбранное пользователем действие и возвращает его. std::istream &input — ссылка на потко ввода.

3.std::vector<std::string> split(const std::string &str, char delim) — разбиение строки str по разделителю delim, возвращает вектор строк. const std::string &str — строка, которую необходимо разбить, delim — символ-разделитель.

4.void readString(std::istream &input, std::string &string) — считывает строку из input в string, разделитель — символ переноса строки. std::istream &input — ссылка на поток ввода, std::string &string — ссылка на строку, в которую будет произведено считывание.

4. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Для консоли был реализован интерфейс пользователя для работы с программой. Он состоит из 10 пунктов, которые необходимо выбирать путём ввода цифры/числа с клавиатуры. Рассмотрим варианты, доступные пользователю:

1. Count the elements — подсчёт элементов в хеш-таблице

2. Add the elements — добавление элементов в хеш-таблицу

3. Open a file — открытие файла для считывания

4. Close the file and read from std::cin — закрытие файла и считывание с клавиатуры

5. Delete an element — удаление элемента

6. Resize the hashmap — изменение размера таблицы и перерасчёт хешей

7. Output the hashmap — вывод таблицы

8. Set advanced hash function — высчитывание хеша по формуле (1)

9. Set simple hash function — высчитывание хеша по формуле (2)

10. Exit — выход из программы

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была написана программа на языке программирования C++, реализующая хеш-таблицу с цепочками. Были продемонстрированы операции вставки, удаления и подсчёта элементов. Кроме того, был написан интерфейс для удобной работы с программой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б. И Ритчи Д. Язык программирования Си М.: Вильямс, 1978 288 с.

ПРИЛОЖЕНИЕ А

ТЕСТИРОВАНИЕ

Номер	Входные данные	Выходные данные	Комментарий
1	2 test add bs report	Your string: test add bs report Calculating hash for (test) Hash = 37 * 0 + 116 = 116 Hash = 37 * 116 + 101 = 4393 Hash = 37 * 4393 + 115 = 162656 Hash = 37 * 162656 + 116 = 6018388 Correcting hash = hash % size_ = 6018388 % 10 = 8 Hash for (test) = 8 Calculating hash for (add) Hash = 37 * 0 + 97 = 97 Hash = 37 * 97 + 100 = 3689 Hash = 37 * 3689 + 100 = 136593 Correcting hash = hash % size_ = 136593 % 10 = 3 Hash for (add) = 3 Calculating hash for (bs) Hash = 37 * 0 + 98 = 98 Hash = 37 * 98 + 115 = 3741 Correcting hash = hash % size_ = 3741 % 10 = 1 Hash for (bs) = 1 Calculating hash for (report) Hash = 37 * 0 + 114 = 114 Hash = 37 * 114 + 101 = 4319 Hash = 37 * 4319 + 112 = 159915 Hash = 37 * 159915 +	Элементы были успешно добавлены в таблицу, показан процесс высчитывания хеша

		111 = 5916966 Hash = 37 * 5916966 + 114 = 218927856 Hash = 37 * 218927856 + 116 = 8100330788 Correcting hash = hash % size_ = 8100330788 % 10 = 8 Hash for (report) = 8 Table[0] = Table[1] = bs->\ Table[2] = Table[3] = add->\ Table[4] = Table[5] = Table[6] = Table[7] = Table[8] = test->report->\ Table[9] =	
2	6 2	Input new size: 2 Calculating hash for (bs) Hash = 37 * 0 + 98 = 98 Hash = 37 * 98 + 115 = 3741 Correcting hash = hash % size_ = 3741 % 2 = 1 Calculating hash for (add) Hash = 37 * 0 + 97 = 97 Hash = 37 * 97 + 100 = 3689 Hash = 37 * 3689 + 100 = 136593 Correcting hash = hash % size_ = 136593 % 2 = 1 Calculating hash for (test) Hash = 37 * 0 + 116 = 116 Hash = 37 * 116 + 101 = 4393 Hash = 37 * 4393 + 115 = 162656 Hash = 37 * 162656 + 116 = 6018388 Correcting hash = hash %	Размер таблицы был успешно изменён, при этом все хеши пересчитаны и показан процесс подсчёта

		size_ = 6018388 % 2 = 0 Calculating hash for (report) Hash = 37 * 0 + 114 = 114 Hash = 37 * 114 + 101 = 4319 Hash = 37 * 4319 + 112 = 159915 Hash = 37 * 159915 + 111 = 5916966 Hash = 37 * 5916966 + 114 = 218927856 Hash = 37 * 218927856 + 116 = 8100330788 Correcting hash = hash % size_ = 8100330788 % 2 = 0 Table[0] = test->report->Table[1] = bs->add->	
3	5 test	Your string: test Calculating hash for (test) Hash = 37 * 0 + 116 = 116 Hash = 37 * 116 + 101 = 4393 Hash = 37 * 4393 + 115 = 162656 Hash = 37 * 162656 + 116 = 6018388 Correcting hash = hash % size_ = 6018388 % 2 = 0 Table[0] = test->report->Table[1] = bs->add->\n\nAfter deleting: Table[0] = report->Table[1] = bs->add->	Удаление элемента из начала списка.
4	5 add	Your string: add Calculating hash for (add) Hash = 37 * 0 + 97 = 97 Hash = 37 * 97 + 100 = 3689 Hash = 37 * 3689 + 100 =	Удаление элемента из конца списка.

		136593 Correcting hash = hash % size_ = 136593 % 2 = 1 Table[0] = report-> Table[1] = bs->add-> After deleting: Table[0] = report->	
5	2 this is actually so funny	Input: this is actually so funny Your string: this is actually so funny Calculating hash for (this) Hash = 37 * 0 + 116 = 116 Hash = 37 * 116 + 104 = 4396 Hash = 37 * 4396 + 105 = 162757 Hash = 37 * 162757 + 115 = 6022124 Correcting hash = hash % size_ = 6022124 % 2 = 0 Hash for (this) = 0 Calculating hash for (is) Hash = 37 * 0 + 105 = 105 Hash = 37 * 105 + 115 = 4000 Correcting hash = hash % size_ = 4000 % 2 = 0 Hash for (is) = 0 Calculating hash for (actually) Hash = 37 * 0 + 97 = 97 Hash = 37 * 97 + 99 = 3688 Hash = 37 * 3688 + 116 = 136572 Hash = 37 * 136572 + 117 = 5053281 Hash = 37 * 5053281 + 97 = 186971494 Hash = 37 * 186971494 +	Добавление новых элементов в таблицу

		108 = 6917945386 Hash = 37 * 6917945386 + 108 = 255963979390 Hash = 37 * 255963979390 + 121 = 9470667237551 Correcting hash = hash % size_ = 9470667237551 % 2 = 1 Hash for (actually) = 1 Calculating hash for (so) Hash = 37 * 0 + 115 = 115 Hash = 37 * 115 + 111 = 4366 Correcting hash = hash % size_ = 4366 % 2 = 0 Hash for (so) = 0 Calculating hash for (funny) Hash = 37 * 0 + 102 = 102 Hash = 37 * 102 + 117 = 3891 Hash = 37 * 3891 + 110 = 144077 Hash = 37 * 144077 + 110 = 5330959 Hash = 37 * 5330959 + 121 = 197245604 Correcting hash = hash % size_ = 197245604 % 2 = 0 Hash for (funny) = 0 Table[0] = report->this->is->so->funny-> Table[1] = bs->actually->	
6	5 is	Your string: is Calculating hash for (is) Hash = 37 * 0 + 105 = 105 Hash = 37 * 105 + 115 = 4000 Correcting hash = hash %	Успешное удаление из середины списка

		size_ = 4000 % 2 = 0 Table[0] = report->this->is->so->funny->Table[1] = bs->actually->\n\nAfter deleting: Table[0] = report->this->so->funny->Table[1] = bs->actually->	
7	1 WHY	Your string: WHY Calculating hash for (WHY) Hash = 37 * 0 + 87 = 87 Hash = 37 * 87 + 72 = 3291 Hash = 37 * 3291 + 89 = 121856 Correcting hash = hash % size_ = 121856 % 2 = 0 Hash for (WHY) = 0 Table[0] = report->this->so->funny->Table[1] = bs->actually->	Поиск элемента, которого нет в таблице, ничего не дал
8	1 the point of it	Input: the point of it Your string: the point of it Calculating hash for (the) Hash = 37 * 0 + 116 = 116 Hash = 37 * 116 + 104 = 4396 Hash = 37 * 4396 + 101 = 162753 Correcting hash = hash % size_ = 162753 % 2 = 1 Hash for (the) = 1 Calculating hash for (point) Hash = 37 * 0 + 112 = 112 Hash = 37 * 112 + 111 = 4255 Hash = 37 * 4255 + 105 = 157540 Hash = 37 * 157540 +	Поиск того, чего нет.

		110 = 5829090 Hash = 37 * 5829090 + 116 = 215676446 Correcting hash = hash % size_ = 215676446 % 2 = 0 Hash for (point) = 0 Calculating hash for (of) Hash = 37 * 0 + 111 = 111 Hash = 37 * 111 + 102 = 4209 Correcting hash = hash % size_ = 4209 % 2 = 1 Hash for (of) = 1 Calculating hash for (it) Hash = 37 * 0 + 105 = 105 Hash = 37 * 105 + 116 = 4001 Correcting hash = hash % size_ = 4001 % 2 = 1 Hash for (it) = 1 Table[0] = report->this->so->funny-> Table[1] = bs->actually->	

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: hashtable.h

```
#ifndef COURSEWORK_HASHTABLE_H
#define COURSEWORK_HASHTABLE_H

#include <list>
#include <vector>
#include <map>
#include <iostream>
#include "hashtablestate.h"
#include "advancedstate.h"
#include "simplestate.h"

#define RED_COLOR "\033[31m"
#define BLUE_COLOR "\033[34m"
#define GREEN_COLOR "\033[32m"
#define RESET "\033[0m"

/*
 * This is class template of a hash table. Hash function calculated
 * based on the table's size and the length of a value.
 * Table based on std::list.
 */

template <typename T>
class HashTable {
    std::vector < std::list<T> > table_;
    std::shared_ptr< HashTableState<T> > state_; // smart pointer
for an abstract class
    int size_;

public:
    explicit HashTable(int size, std::shared_ptr<HashTableState<T>>
state) : size_(size), state_(state) {
        table_.resize(size_);
    }
```

```
        void setState(std::shared_ptr<HashTableState<T>> state) {
            state_ = state;
        }

        // resize the hashmap and recalculate all hashes
        void resize(int newSize) {
            HashTable<T> newMap(newSize, state_);

            for (auto i = 0; i < size_; i++)
                for (auto elem : table_[i])
                    newMap.add(elem);

            *this = newMap;
        }

        // add a single element to the map
        void add(T value) {
            auto key = state_->hash(*this, value);
            table_[key].push_back(value);
        }

        // add an array of the elements to the map
        void add(std::vector<T> &value) {
            std::map<int, std::vector<T>> hash = getHashMap(value); //
get a map of the hashes

            for (auto i = 0; i < hash.size(); i++) { // iterating
through the map
                std::cout << "Table[" << i << "] = ";
                for (const auto &elem : table_[i]) // output old
elements
                    std::cout << elem << "->";
                for (const auto &elem : hash[i]) { // insert new
elements and output them
                    table_[i].push_back(elem);
                    std::cout << GREEN_COLOR << elem << RESET << "->";
                }

                std::cout << '\n';
```

```
            }
        }

        int count(T value) {
            auto hash = getHashAdvanced(value), count = 0;

            for (const auto &elem : table_[hash])
                if (value == elem)
                    count++;

            return count;
        }

        std::map<T, int> count(std::vector<T> &value) {
            std::map<T, int> count;

            for (auto i = 0; i < table_.size(); i++) { // iterating
through the map
                std::cout << "Table[" << i << "] = ";
                for (const auto &elem : table_[i]) {
                    if (isIn(value, elem)) { // if elem is in the map
then count and output
                        std::cout << BLUE_COLOR << elem << RESET << "-
>";
                        count[elem]++;
                    }
                    else // just output the elem
                        std::cout << elem << "->";
                }

                std::cout << '\n';
            }

            return count;
        }

        void remove(T value) {
            for (auto i = 0; i < table_.size(); i++) { // iterating
through the map
```

```
                std::cout << "Table[" << i << "] = ";

                for (auto elem = begin(table_[i]); elem !=
end(table_[i]); elem++) {
                    if (*elem == value){ // if elem is for deleting
then erase and return
                        std::cout << RED_COLOR << *elem << RESET << "-
>";
                        table_[i].erase(elem);
                        std::cout << '\n';
                        return;
                    } else // just output
                        std::cout << *elem << "->";
                }

                std::cout << '\n';
            }
        }

        friend std::ostream& operator<<(std::ostream &out, const
HashTable<T> &table) { // output operator
            for (auto i = 0; i < table.size_; i++) {
                out << "Table[" << i << "] = ";

                for (const auto &elem : table.table_[i])
                    out << elem << "->";

                out << "\n";
            }

            return out;
        }

        int getHashAdvanced(T value) { // hash function
            std::cout << "Calculating hash for (" << value << ")\n";
            size_t hash = 0;

            for (auto i = 0; i < value.size(); i++) {
```

```
                std::cout << "Hash = 37 * " << hash << " + " <<
(int)value[i];
                hash = 37 * hash + value[i]; // calculated based on the
length
                std::cout << " = " << hash << '\n';
            }

            std::cout << "Correcting hash = hash % size_ = " << hash <<
" % " << size_;
            hash %= size_; // correct the hash
            std::cout << " = " << hash << '\n';
            return hash;
        }

        int getHashSimple(T value) {
            std::cout << "Calculating hash for (" << value << ")\n";
            size_t hash = 0;

            for (auto i = 0; i < value.size(); i++) {
                std::cout << "Hash += " << (int)value[i];
                hash += value[i]; // calculated based on the length
                std::cout << " = " << hash << '\n';
            }

            std::cout << "Correcting hash = hash % size_ = " << hash <<
" % " << size_;
            hash %= size_; // correct the hash
            std::cout << " = " << hash << '\n';
            return hash;
        }

    private:
        bool isIn(std::vector<T> &vector, T value) {
            for (const auto &elem : vector)
                if (value == elem)
                    return true;

            return false;
        }
```

```
        std::map<int, std::vector<T>> getHashMap(std::vector<T> &value)
{
            std::map<int, std::vector<T>> hash;

            for (const auto &elem : value) {
                auto key = state_->hash(*this, elem);
                hash[key].push_back(elem);
                std::cout << "Hash for (" << elem << ") = " << key <<
'\n';
            }

            return hash;
        }
    };

    #endif //COURSEWORK_HASHTABLE_H
```

Название файла: main.cpp

```
    #include <string>
    #include <fstream>
    #include "hashtable.h"

    // help for the user
    void outputHelp(std::ostream &output) {
        output << "Choose one of the following actions: " << '\n';
        output << "1. Count the elements" << '\n';
        output << "2. Add the elements" << '\n';
        output << "3. Open a file" << '\n';
        output << "4. Close the file and read from std::cin" << '\n';
        output << "5. Delete an element" << '\n';
        output << "6. Resize the hashmap" << '\n';
        output << "7. Output the hashmap" << '\n';
        output << "8. Set advanced hash function" << '\n';
        output << "9. Set simple hash function" << '\n';
        output << "10. Exit" << '\n';
        output << "Your action: ";
    }
```

```
    // get an action from the user
    int getAction(std::istream &input) {
        int action;
        outputHelp(std::cout);
        input >> action;
        input.ignore();
        return action;
    }

    // splits str on delimiter delim
    std::vector<std::string> split(const std::string &str, char delim)
{
        std::vector<std::string> strings; // result
        size_t start;
        size_t end = 0;

        while ((start = str.find_first_not_of(delim, end)) !=
std::string::npos) { // while can find delimiters
            end = str.find(delim, start);
            strings.push_back(str.substr(start, end - start)); // get a
substr and add to the result
        }

        return strings;
    }

    // get a string from the stream
    void readString(std::istream &stream, std::string &string) {
        std::cout << "Input: ";
        getline(stream, string, '\n');
        std::cout << "Your string: " << string << '\n';
    }

    int main() {
        HashTable<std::string> table(10,
std::make_shared<AdvancedState<std::string>>());
        int action;
        int size;
```

```
            std::ifstream file; // file to read from
            std::string filePath; // path to the file
            std::string string; // input string
            std::vector<std::string> elements; // split input
            std::map<std::string, int> count; // count elements
            std::istream *input = &std::cin; // input stream

            while ((action = getAction(std::cin)) != 10) {

                switch (action) {
                    case 1:
                        readString(*input, string); // read input
                        elements = split(string, ' '); // split input
                        count = table.count(elements);

                        for (const auto &elem : count) {
                            std::cout << "Elem (" << elem.first << ")
contains " << elem.second << " times " << '\n';
                        }

                        break;
                    case 2:
                        readString(*input, string); // read string
                        elements = split(string, ' '); // split input
                        table.add(elements);
                        break;
                    case 3:
                        std::cout << "Path to the file: ";
                        std::cin >> filePath; // read the file path
                        file.open(filePath); // open file

                        if (!file.is_open()) { // check if it opens
                            std::cout << "Couldn't open the file, please
try again" << '\n';
                            continue;
                        }

                        input = &file; // change stream
                        break;
```

```
                case 4:
                    if (file.is_open()) // close file if it was open
                        file.close();

                    input = &std::cin; // change stream
                    break;
                case 5:
                    readString(*input, string); // read input
                    elements = split(string, ' '); // split string
                    table.remove(elements[0]);
                    break;
                case 6:
                    std::cout << "Input new size: ";
                    std::cin >> size;
                    table.resize(size);
                    break;
                case 7:
                    std::cout << "The table is: " << '\n';
                    std::cout << table << '\n';
                    break;
                case 8:

table.setState(std::make_shared<AdvancedState<std::string>>());
                    std::cout << "The hash function has been changed to
advanced" << '\n';
                    break;
                case 9:

table.setState(std::make_shared<SimpleState<std::string>>());
                    std::cout << "The hash function has been changed to
simple" << '\n';
                    break;
                case 10:
                default:
                    std::cout << "Exiting the program" << '\n';
                    return 0;
            }
```

```
        std::cout << '\n';
    }

    return 0;
}
```

Название файла: hashtablestate.h

```
#ifndef COURSEWORK_HASHTABLESTATE_H
#define COURSEWORK_HASHTABLESTATE_H

#include <memory>
#include <vector>

template <typename T> class HashTable;

/*
 * This is an abstract class for pattern "state".
 * Hash table has different states for different hash functions;
 */

template <typename T>
class HashTableState {
public:
    virtual int hash(HashTable<T> &map, T value) = 0;
    ~HashTableState() = default;
};

#endif //COURSEWORK_HASHTABLESTATE_H
```

Название файла: advancedstate.h

```
#ifndef COURSEWORK_ADVANCEDSTATE_H
#define COURSEWORK_ADVANCEDSTATE_H

#include "hashtablestate.h"
#include "hashtable.h"

template <typename T>
class AdvancedState : public HashTableState<T> {
public:
    int hash(HashTable<T> &map, T value) override {
```

```
        return map.getHashAdvanced(value);
    }
};

#endif //COURSEWORK_ADVANCEDSTATE_H
```

Название файла: simplestate.h

```
#ifndef COURSEWORK_SIMPLESTATE_H
#define COURSEWORK_SIMPLESTATE_H

#include "hashtablestate.h"
#include "hashtable.h"

template <typename T>
class SimpleState : public HashTableState<T> {
public:
    int hash(HashTable<T> &map, T value) override {
        return map.getHashSimple(value);
    }
};

#endif //COURSEWORK_SIMPLESTATE_H
```