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 (проекта): Хеш-таблицы с цепочками — вставка и исключение

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: 08.12.2020

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 структуры данных, операции вставки, удаления и поиска элементов в ней, а также синтаксис языка программирования C++.

1. ЗАДАНИЕ

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

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Описание алгоритма

Был реализован класс HashTable, таблица в котором представляет собой вектор списков (цепочек), хеш элемента высчитывается по следующей формуле: 37 * hash + value[i], где hash — хеш элемента, value[i] — текущий символ, i — счётчик итерации по элементу. Для поиска элемента высчитываем хеш, берём нужный список и делаем по нему обход. Для добавления высчитываем хеш, после чего вставляем в нужный список элемент.

2.2. 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.

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 hash(T value) — возвращает хеш элемента. T value — элемент, хеш которого необходимо посчитать. Считается на основе длины элемента и размера таблицы.

4.friend std::ostream& operator<<(std::ostream &out, const HashTable<T> &table) — оператор вывода в поток. std::ostream &out -поток

вывода, HashTable<T> &table - таблица для вывода.Возвращает поток, в который требуется вывести таблицу.

5.int remove(T value) — удаление элемента из хеш-таблицы. T value — элемент, который необходимо удалить. Возвращает 1, если удаление успешно и 0 иначе.

2.3. 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.

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

Входные данные	Исходные данные
Your action: 2 Input: ok this is just a simple test adding elements to the table	Your string: ok this is just a simple test adding elements to the table Working with element (ok) Calculated hash = 4 Pushing to [4] Element (ok) successfully added to the hashmap Working with element (this) Calculated hash = 4 Pushing to [4] Element (this) successfully added to the hashmap Working with element (is) Calculated hash = 0 Pushing to [0] Element (is) successfully added to the hashmap Working with element (just) Calculated hash = 2 Pushing to [2] Element (just) successfully added to the hashmap Working with element (a) Calculated hash = 7 Pushing to [7] Element (a) successfully added to the hashmap Working with element (simple) Calculated hash = 0 Pushing to [0] Element (simple) successfully added to the hashmap Working with element (test) Calculated hash = 8 Pushing to [8] Element (test) successfully added to the hashmap Working with element (adding) Calculated hash = 5 Pushing to [5] Element (adding) successfully added to the hashmap Working with element (elements) Calculated hash = 7 Pushing to [7] Element (elements) successfully added to the hashmap Working with element (to)

	Calculated hash = 3 Pushing to [3] Element (to) successfully added to the hashmap Working with element (the) Calculated hash = 3 Pushing to [3] Element (the) successfully added to the hashmap Working with element (table) Calculated hash = 6 Pushing to [6] Element (table) successfully added to the hashmap The table is: Table[0] = is->simple->\ Table[1] = Table[2] = just->\ Table[3] = to->the->\ Table[4] = ok->this->\ Table[5] = adding->\ Table[6] = table->\ Table[7] = a->elements->\ Table[8] = test->\ Table[9] =
Your action: 1 Input: very interesting , right ? don ' t think so ? me neither bro	Your string: very interesting , right ? don ' t think so ? me neither bro Counting element (very) Hash = 2 Counting the element: Got (just) Element (very) contains 0 times in the map Counting element (interesting) Hash = 8 Counting the element: Got (test) Element (interesting) contains 0 times in the map Counting element (,) Hash = 4 Counting the element: Got (ok) Got (this) Element (,) contains 0 times in the map Counting element (right) Hash = 0 Counting the element:

	Got (is) Got (simple) Element (right) contains 0 times in the map Counting element (?) Hash = 3 Counting the element: Got (to) Got (the) Element (?) contains 0 times in the map Counting element (don) Hash = 7 Counting the element: Got (a) Got (elements) Element (don) contains 0 times in the map Counting element (') Hash = 9 Counting the element: Element (') contains 0 times in the map Counting element (t) Hash = 6 Counting the element: Got (table) Element (t) contains 0 times in the map Counting element (think) Hash = 0 Counting the element: Got (is) Got (simple) Element (think) contains 0 times in the map Counting element (so) Hash = 6 Counting the element: Got (table) Element (so) contains 0 times in the map Counting element (?) Hash = 3 Counting the element: Got (to) Got (the) Element (?) contains 0 times in the map Counting element (me) Hash = 4 Counting the element:

	Got (ok) Got (this) Element (me) contains 0 times in the map Counting element (neither) Hash = 5 Counting the element: Got (adding) Element (neither) contains 0 times in the map Counting element (bro) Hash = 1 Counting the element: Element (bro) contains 0 times in the map The table is: Table[0] = is->simple-> Table[1] = Table[2] = just-> Table[3] = to->the-> Table[4] = ok->this-> Table[5] = adding-> Table[6] = table-> Table[7] = a->elements-> Table[8] = test-> Table[9] =
Your action: 5 Input: adding	Your string: adding Working with element (adding) Hash = 5 Found the element, deleting.. The table is: Table[0] = is->simple-> Table[1] = Table[2] = just-> Table[3] = to->the-> Table[4] = ok->this-> Table[5] = Table[6] = table-> Table[7] = a->elements-> Table[8] = test-> Table[9] =
Your action: 2 Input: writing reports is interesting and	The table is: Table[0] = is->simple->is-> Table[1] = reports-> Table[2] = just->writing->

cost-effective	Table[3] = to->the->and-> Table[4] = ok->this-> Table[5] = Table[6] = table-> Table[7] = a->elements-> Table[8] = test->interesting-> Table[9] = cost-effective->
Your action: 5 Input: and	Your string: and Working with element (and) Hash = 3 Got (to) Got (the) Found the element, deleting.. The table is: Table[0] = is->simple->is-> Table[1] = reports-> Table[2] = just->writing-> Table[3] = to->the-> Table[4] = ok->this-> Table[5] = Table[6] = table-> Table[7] = a->elements-> Table[8] = test->interesting-> Table[9] = cost-effective->

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: hashtable.h

```
#include <list>
#include <vector>
#include <iostream>

/*
 * 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 HashMap {
    std::vector < std::list<T> > table_;
    int size_;

public:
    explicit HashMap(int size) : size_(size) {
        table_.resize(size_);
    }

    void add(T value) {
        auto hash = getHash(value); // get the hash of a value
        std::cout << "Calculated hash = " << hash << '\n';
        std::cout << "Pushing to [" << hash << "]" << '\n';
        table_[hash].push_back(value); // add the element
    }

    int count(T value) {
        auto key = getHash(value), count = 0; // get the hash
        std::cout << "Hash = " << key << '\n';
        std::cout << "Counting the element: " << '\n';

        for (const auto &elem : table_[key]) {
            if (value == elem) {
```

```
                    std::cout << "Found the element, counting.." << '\
n';
                    count++;
                }
                else
                    std::cout << "Got (" << elem << ")\n";
            }

            return count;
        }

        int remove(T value) {
            auto key = getHash(value); // get the hash
            std::cout << "Hash = " << key << '\n';

            for (const auto &elem : table_[key]) { // remove all
repeated elements in foreach
                if (value == elem) {
                    std::cout << "Found the element, deleting.." << '\
n';
                    table_[key].remove(value);
                    return 1;
                } else
                    std::cout << "Got (" << elem << ")\n";
            }

            std::cout << "The element is not in the map" << '\n';
            return 0;

        }

        friend std::ostream& operator<<(std::ostream &out, const
HashMap<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 getHash(T value) { // hash function
            auto hash = 0;

            for (auto i = 0; i < value.size(); i++)
                hash = 37 * hash + value[i]; // calculated based on the
length

            hash %= size_; // correct the hash

            return hash >= 0 ? hash : hash + size_;
        }
    };
```

Название файла: main.cpp

```
    // 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. 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() {
        HashMap<std::string> table(10);
        int action;

        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::istream *input = &std::cin; // input stream

        while ((action = getAction(std::cin)) != 6) {

            switch (action) {
                case 1:
                    readString(*input, string); // read input
```

```
                    elements = split(string, ' '); // split input

                    for (auto &i : elements) { // count the element
                        std::cout << "Counting element (" << i << ")\
n";
                        auto count = table.count(i);
                        std::cout << "Element (" << i << ") contains "
<< count << " times in the map" << '\n';
                    }

                    break;
                case 2:
                    readString(*input, string); // read string
                    elements = split(string, ' '); // split input

                    for (auto &i : elements)  { // add elements
                        std::cout << "Working with element (" << i <<
")" << '\n';
                        table.add(i);
                        std::cout << "Element (" << i << ")
successfully added to the hashmap" << '\n';
                    }

                    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

                    for (auto &i : elements) { // delete the element
                        std::cout << "Working with element (" << i <<
")" << '\n';
                        table.remove(i);
                    }
                    break;
                case 6:
                default:
                    std::cout << "Exiting the program" << '\n';
                    return 0;
            }

            std::cout << '\n' << "The table is: " << '\n';
            std::cout << table << '\n'; // output table with operator
        }

        return 0;
    }
```