

КАФЕДРА №

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

должность, уч. степень, звание

подпись, дата

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

ХЕШИРОВАНИЕ ДАННЫХ

по курсу: Структуры и алгоритмы обработки данных

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

подпись, дата

инициалы, фамилия

Санкт-Петербург 2022

1.1 Цель работы Целью работы является изучение методов хеширования данных и получение практических навыков реализации хеш-таблиц.

1.2 Задание на лабораторную работу Составить хеш-функцию в соответствии с заданным вариантом и проанализировать ее. При необходимости доработать хеш-функцию. Используя полученную хеш-функцию разработать на языке программирования высокого уровня программу, которая должна выполнять следующие функции: – создавать хеш-таблицу; – добавлять элементы в хеш-таблицу; – просматривать хеш-таблицу; – искать элементы в хеш-таблице по номеру сегмента/по ключу; – выгружать содержимое хеш-таблицы в файл для построения гистограммы в MS Excel, или в аналогичном подходящем ПО; – удалять элементы из хеш-таблицы; – в программе должна быть реализована проверка формата вводимого ключа; – при удалении элементов из хеш-таблицы, в программе должен быть реализован алгоритм, позволяющий искать элементы, вызвавшие коллизию с удаленным; – в программе должен быть реализован алгоритм, обрабатывающий ситуации с переполнением хеш-таблицы.

Вариант 4

4	цццБцц	2500	Двойное хеширование
---	--------	------	---------------------

Листинг

Проверка эффективности хеш функции

Функция хеширования:

```
1. // хеширование
2. int My_hash::hash(char* key) {
3.     int value = 1;
4.     for (int i = 0; i < size_key; i++) {
5.         value += (int)key[i] * (int)key[i];
6.     }
7.     return (value % count_sigments);
8. }
```

Программа берет код каждого символа в строке и складывает их квадраты, деля на количество сегментов (В моем случае 2000)

main.cpp

```
/*
```

```
4 вариант
```

```
Линейное опрабирование
```

```
2500 сигментов
```

```
цццБцц
```

```
*/
```

```
#include <iostream>
```

```
using namespace std;
```

```
#include <cmath>
```

```
#include <time.h>
```

```

#include <iomanip>

// функции для работы со строками
#include "simple_char.h"

// методы для хеширования
#include "hash.h"

// генерирует случайное число в диапазоне от A до B
int random_int(int a, int b) {
    return a + (rand() % ( b - a + 1 ));
}

// (говно) Ввод вещественного числа с проверкой
double read_double(){
    double x;
    while ( (scanf("%lf",&x) ) != 1 ) {
        printf("Неверное введенное значение, попробуйте еще: ");
        while(getchar() != '\n');
    }
    fflush(stdin);
    return x;
}

int menu() {
    int id;
    while (true) {
        cout << endl;
        cout << "1) Сгенерировать ключ" << endl;
        cout << "2) Ввести ключ вручную" << endl;
        cout << "3) Сгенерировать список ключей" << endl;
        cout << "4) Вывести список ключей" << endl;
        cout << "5) Очистить список ключей" << endl;
        cout << "6) Экспортировать в файл" << endl;
        cout << "7) Поиск по номеру сегмента" << endl;
        cout << "8) Поиск по ключу" << endl;
        cout << "0) Выход" << endl;
        cout << " >>> ";
        id = read_double();
        if (id >= 0 && id <= 8) {
            return id;
        } else {
            cout << "Этого нет в меню" << endl;
        }
    }
}

int main() {
    // смена кодировки

```

```

system("chcp 65001");

srand(time(NULL));

/*
i (int) = число
c (char) = буква
*/

My_hash my_hash("iiicii", 6);

int menu_i;
while (true) {
    menu_i = menu();

    switch (menu_i) {
        case (0):
            return 0;
            break;

        case (1):
            my_hash.generate();
            break;

        case (2):
            my_hash.generate(false);
            break;

        case (3):
            int count;
            while (true) {
                cout << "Количество ключей: ";
                count = read_double();
                if (count > 0) {
                    for (int i = 0; i < count; i++)
                        my_hash.generate();
                    break;
                } else
                    cout << "Число должно быть больше 0." << endl;
            }
            break;

        case (4):
            my_hash.draw_hash_list();
            break;

        case (5):
            my_hash.clear_hash_list();
            break;

        case (7):{
            cout << "Номер сегмента: ";

```

```

    int id = read_double();
    my_hash.get_find_by_id(id);
    break;
}

case (8):{
    char* key = my_hash.read_key();
    my_hash.get_find_by_key(key);
    break;
}

case (6):
    int file_name_length;
    cout << "Название файла: ";
    char* file_name = get_string(&file_name_length);
    my_hash.export_to_file(file_name);
    break;

}

}

cout << my_hash.generate() << endl;

return 0;
}

```

hash.h

```

#include <iostream>
using namespace std;

#include <cstdio>

struct hash_struct {
    int hash;
    char* key;
};

class My_hash {
public:
    My_hash(const char* Key_example, int Size_key, int Count_sigments);
    ~My_hash();

    int hash(char*);
    char* generate(bool);
    char* read_key();

    void draw_hash_list();
    void clear_hash_list();

    bool chek_key(char*);

```

```

bool find_by_hash(int);
bool find_by_key(char*);
void get_find_by_key(char*);
void get_find_by_id(int);

void export_to_file(char*);

private:
void append_list(char*);

const char* key_example;
int size_key;

int count_sgments;

int size_list = 0;
hash_struct* hash_list;
};

// конструктор
My_hash::My_hash(const char* Key_example, int Size_key, int Count_sgments =
2500) {
    key_example = Key_example;

    size_key = Size_key;
    hash_list = (hash_struct*)malloc(sizeof(hash_struct));

    count_sgments = Count_sgments;
}

// деструктор
My_hash::~My_hash() {
    free(hash_list);
}

// хеширование
int My_hash::hash(char* key) {
    int value = 1;
    for (int i = 0; i < size_key; i++) {
        value += (int)key[i] * (int)key[i];
    }
    return (value % count_sgments);
}

char* My_hash::read_key() {
    char* key;
    int length;
    while (true) {
        cout << "Введите ключ формата " << key_example << ": ";
        key = get_string(&length);
    }
}

```

```

    if (length != size_key || !chek_key(key)) {
        cout << "ключ не соответствует формату." << endl;
    } else break;
}
return key;
}

// генератор ключей
char* My_hash::generate(bool random = true) {
    char* key = (char*) malloc(size_key * sizeof(char));
    int length;
    while (true) {
        if (random) {
            for (int i = 0; i < size_key; i++) {
                if (key_example[i] == 'i') key[i] = 48 + rand() % 10;
                if (key_example[i] == 'c') key[i] = 65 + rand() % 26;
            }
        } else {
            while (true) {
                cout << "Введите ключ формата " << key_example << ": ";
                key = get_string(&length);

                if (length != size_key || !chek_key(key)) {
                    cout << "ключ не соответствует формату." << endl;
                } else break;
            }
        }
        if (!find_by_key(key)) break;
    }

    append_list(key);

    return key;
}

// проверка на правильность ключа (возвращает bool)
bool My_hash::chek_key(char* key) {
    for (int i = 0; i < size_key; i++) {
        if (key_example[i] == 'i' && !(key[i] >= 48 && key[i] <= 57)) {return false;}
        if (key_example[i] == 'c' && !(toupper(key[i]) >= 65 && toupper(key[i]) <= 90)) {return false;}
    }
    return true;
}

// добавление ключа в список
void My_hash::append_list(char* key) {
    hash_list = (hash_struct*)realloc(hash_list, (++size_list) * sizeof(hash_struct));
    hash_list[size_list - 1].key = (char*) malloc(size_key * sizeof(char));
    hash_list[size_list - 1].key = key;
}

```

```

int hash_num = hash(key);
int hash_buf;
int i = 0;
while (true) {
    hash_buf = hash_num + (i * hash_num);
    if (!find_by_hash(hash_buf)) {
        hash_list[size_list - 1].hash = hash_buf;
        break;
    }
    i++;
}

}

// нахождение хеша в списке (возвращает bool)
bool My_hash::find_by_hash(int hash) {
    for (int i = 0; i < size_list; i++) {
        if (hash == hash_list[i].hash) {
            // cout << "1" << endl;
            return true;
        }
    }
    return false;
}

// нахождение ключа в списке (возвращает bool)
bool My_hash::find_by_key(char* key) {
    for (int i = 0; i < size_list; i++) {
        if (key == hash_list[i].key) {
            // cout << "2" << endl;
            return true;
        }
    }
    return false;
}

// нахождение ключа в списке (возвращает ключ)
void My_hash::get_find_by_id(int id) {
    bool ok = false;
    for (int i = 0; i < size_list; i++) {
        if (i == id) {
            cout << "Найденый ключ: ";
            cout << hash_list[i].hash << " ";
            draw_char_array(hash_list[i].key, size_key);
            cout << endl;
            ok = true;
            break;
        }
    }
    if (!ok) cout << "Такого ключа не существует." << endl;
}

```



```

bool check_enterd_key(char* key1, char*key2, int size_key) {
    for (int i = 0; i < size_key; i++) {
        if (toupper(key1[i]) != toupper(key2[i])) return false;
    }
    return true;
}

```

// нахождение ключа в списке (возвращает ключ)

```

void My_hash::get_find_by_key(char* key) {
    bool ok = false;
    for (int i = 0; i < size_list; i++) {
        // if (key == hash_list[i].key) {
        if (check_enterd_key(key, hash_list[i].key, size_key)) {
            cout << "Найденый ключ: ";
            cout << i << " " << hash_list[i].hash << " ";
            draw_char_array(hash_list[i].key, size_key);
            cout << endl;
            ok = true;
            break;
        }
    }
    if (!ok) cout << "Такого ключа не существует." << endl;
}

```

// очистка списка

```

void My_hash::clear_hash_list() {
    hash_list = (hash_struct*)malloc(sizeof(hash_struct));
    size_list = 0;
}

```

// вывод ключей, хеша и id

```

void My_hash::draw_hash_list() {
    for (int i = 0; i < size_list; i++) {
        cout << i << " " << hash_list[i].hash << " ";
        draw_char_array(hash_list[i].key, size_key);
        cout << endl;
    }
}

```

// экспорт списка в файл

```

void My_hash::export_to_file(char* file_name) {
    FILE *output_file;

    output_file = fopen(file_name, "w");

    for (int i = 0; i < size_list; i++) {
        fprintf(output_file, "%d %d ", i, hash_list[i].hash);
        for (int j = 0; j < size_key; j++)
            fprintf(output_file, "%c", hash_list[i].key[j]);
        fprintf(output_file, "\n");
    }
}

```

```
fclose(output_file);  
}
```

График идет на всей области определения равномерно.
Диаграмма основана на 1000 ключей



Результат работы

```
C:\Windows\system32\cmd.exe

-----
1) Сгенерировать ключ
2) Ввести ключ вручную
3) Сгенерировать список ключей
4) Вывести список ключей
5) Очистить список ключей
6) Экспортировать в файл

0) Выход
>>> 3
Количество ключей: 10
-----
1) Сгенерировать ключ
2) Ввести ключ вручную
3) Сгенерировать список ключей
4) Вывести список ключей
5) Очистить список ключей
6) Экспортировать в файл

0) Выход
>>> 4
0 1887 903R04
1 1544 656J71
2 280 735D60
3 1665 660J86
4 1917 329K76
5 911 262G49
6 1200 071X65
7 1992 904W69
8 1742 983I71
9 53 550U30
-----
1) Сгенерировать ключ
2) Ввести ключ вручную
3) Сгенерировать список ключей
4) Вывести список ключей
5) Очистить список ключей
6) Экспортировать в файл

0) Выход
>>>
```

Вывод

Я освоил методы хеширования данных и получил практические навыки реализации хеш-таблиц.