МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации



**Практика**

**«**Разработка СУБД**»**

**по дисциплине: «*Программирование*»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБ-221», «АВТФ»  Стеклов Данил Евгеньевич  «11» октября 2023г | Проверил:  *Ассистент ЗИ*  *Исаев Глеб Андреевич*  «\_\_\_» \_\_\_\_\_\_ 2023г |

Новосибирск 2023

**Цели и задачи работы:** разобраться в разработке СУБД на С, Rust или Go

**Задание к работе:**

Разработать no-SQL СУБД, отвечающую следующим требованиям:

1. База данных умеет хранить данные в следующих контейнерах:

1.1 Множество

1.2 Стэк

1.3 Очередь

1.4 Хэш-таблица.

2. Для простоты ЛР необходимо поддерживать только строковый

тип данных.

3. Каждый тип данных поддерживает операцию вставки, доступа

к данным и удаления.

4. Все данные СУБД сохраняет в указанный файл.

5. СУБД имеет консольный интерфейс удовлетворяющий

следующему требованию.

./<имя вашей программы> --file <путь до файла с данными> --

query <запрос к СУБД>

6. Все операции выполняются за O(1)

**Задание на языке C:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

**enum** Command {

SADD,

SREM,

SISMEMBER,

SPUSH,

SPOP,

QPUSH,

QPOP,

HSET,

HDEL,

HGET

};

**typedef** **struct** Set {

char value[30];

} Set;

**typedef** **struct** List {

char value[30];

**struct** List\* next;

} List;

**typedef** **struct** HashTable { *//Реализация с помощью метода открытых адресаций*

char key[15];

char value[30];

char is\_exist;

} HashTable;

FILE\* fp;

void PushToHeadList(List\*\* node, List\*\* tail, char\* value);

void PushToTailList(List\*\* node, List\*\* tail, char\* value);

char\* PopFromHead(List\*\* node);

void WriteListToFile(char\* name, char\* value);

void DeleteStrFromFile(char\* str, char\* path);

void ReplaceStrInFile(char\* str, char\* new\_str, char\* path);

void СoncatenationStr(char\* name, char\* value, char\* str);

void СoncatenationStrForHash(char\* key, char\* value, char is\_exist, char\* str);

void SetAdd(Set\* array, char\* name, char\* value);

void SetRem(Set\* array, char\* name, char\* value, char\* path);

void SetIsMember(Set\* array, char\* value);

void SetReadFromFile(Set\*\* array, char\* main\_name);

void StackPush(List\* node, List\* tail, char\* name, char\* value);

void StackPop(List\* node, char\* name, char\* path);

void StackReadFromFile(List\*\* node, List\*\* tail, char\* name);

void QueuePush(List\* node, List\* tail, char\* name, char\* value);

void QueuePop(List\* node, char\* name, char\* path);

void QueueReadFromFile(List\*\* node, List\*\* tail, char\* name);

void HashSet(HashTable\* row, char\* key, char\* value, char\* path);

void HashDelete(HashTable\* row, char\* key, char\* path);

void HashGet(HashTable\* row, char\* key, char\* path);

void HashReadFromFile(HashTable\*\* row);

int FuncHash(char\* key);

int FuncHashSec(char ch);

int main(int argc, char\* argv[]) {

setlocale(0, "Ru");

Set\* array = NULL;

List\* node = NULL;

List\* tail = NULL;

HashTable\* row = NULL;

char order[12];

char name[20];

char key[15];

char value[30];

**if** (argc < 5) {

puts("Ошибка аргументов");

exit(EXIT\_FAILURE);

}

**if** (strcmp(argv[1], "--file") != 0) {

perror("Ошибка флагов");

exit(EXIT\_FAILURE);

}

**if** ((fopen\_s(&fp, argv[2], "a+")) != NULL) {

perror("Ошибка открытия файла");

exit(EXIT\_FAILURE);

}

**if** (strcmp(argv[3], "--query") != 0) {

perror("Ошибка флагов");

exit(EXIT\_FAILURE);

}

**for** (int i = 0; i < strlen(argv[4]); ++i) {

order[i] = \*(argv[4] + i + 1);

}

**if** (argc == 6) {

**for** (int i = 0; i <= strlen(argv[5]) - 1; ++i) {

name[i] = \*(argv[5] + i);

}

name[strlen(argv[5]) - 1] = '\0';

}

**else** **if** (argc == 7) {

**for** (int i = 0; i <= strlen(argv[5]); ++i) {

name[i] = \*(argv[5] + i);

}

**for** (int i = 0; i < strlen(argv[6]) - 1; ++i) {

value[i] = \*(argv[6] + i);

}

value[strlen(argv[6]) - 1] = '\0';

}

**else** **if** (argc == 8) {

**for** (int i = 0; i <= strlen(argv[5]); ++i) {

name[i] = \*(argv[5] + i);

}

**for** (int i = 0; i <= strlen(argv[6]); ++i) {

key[i] = \*(argv[6] + i);

}

**for** (int i = 0; i < strlen(argv[7]) - 1; ++i) {

value[i] = \*(argv[7] + i);

}

value[strlen(argv[7]) - 1] = '\0';

}

**enum** Command command;

**if** (strcmp(order, "SADD") == 0) command = SADD;

**else** **if** (strcmp(order, "SREM") == 0) command = SREM;

**else** **if** (strcmp(order, "SISMEMBER") == 0) command = SISMEMBER;

**else** **if** (strcmp(order, "QPUSH") == 0) command = QPUSH;

**else** **if** (strcmp(order, "QPOP") == 0) command = QPOP;

**else** **if** (strcmp(order, "SPUSH") == 0) command = SPUSH;

**else** **if** (strcmp(order, "SPOP") == 0) command = SPOP;

**else** **if** (strcmp(order, "HSET") == 0) command = HSET;

**else** **if** (strcmp(order, "HDEL") == 0) command = HDEL;

**else** **if** (strcmp(order, "HGET") == 0) command = HGET;

**switch** (command)

{

**case** SADD:

SetReadFromFile(&array, name);

SetAdd(array, name, value);

**break**;

**case** SREM:

SetReadFromFile(&array, name);

SetRem(array, name, value, argv[2]);

**break**;

**case** SISMEMBER:

SetReadFromFile(&array, name);

SetIsMember(array, value);

**break**;

**case** SPUSH:

StackReadFromFile(&node, &tail, name);

StackPush(node, tail, name, value);

**break**;

**case** SPOP:

StackReadFromFile(&node, &tail, name);

StackPop(node, name, argv[2]);

**break**;

**case** QPUSH:

QueueReadFromFile(&node, &tail, name);

QueuePush(node, tail, name, value);

**break**;

**case** QPOP:

QueueReadFromFile(&node, &tail, name);

QueuePop(node, name, argv[2]);

**break**;

**case** HSET:

HashReadFromFile(&row);

HashSet(row, key, value, argv[2]);

**break**;

**case** HDEL:

HashReadFromFile(&row);

HashDelete(row, value, argv[2]);

**break**;

**case** HGET:

HashReadFromFile(&row);

HashGet(row, value, argv[2]);

**break**;

default:

**break**;

}

fclose(fp);

exit(EXIT\_SUCCESS);

}

void PushToHeadList(List\*\* node, List\*\* tail, char\* value) {

List\* tmp = (List\*)malloc(**sizeof**(List));

strcpy\_s(tmp->value, 30 ,value);

**if** (\*node == NULL) {

\*tail = tmp;

\*node = tmp;

tmp->next = NULL;

}

**else** {

tmp->next = \*node;

\*node = tmp;

}

}

void PushToTailList(List\*\* node, List\*\* tail, char\* value) {

List\* tmp = (List\*)malloc(**sizeof**(List));

strcpy\_s(tmp->value, 30 ,value);

tmp->next = NULL;

**if** (\*node == NULL) {

\*tail = tmp;

\*node = tmp;

}

**else** {

(\*tail)->next = tmp;

(\*tail) = tmp;

}

}

char\* PopFromHead(List\*\* node) {

List\* tmp = \*node;

char value[30];

strcpy\_s(value, 30, tmp->value);

\*node = tmp->next;

free(tmp);

**return** value;

}

void WriteListToFile(char\* name, char\* value) {

char str[50];

strcat\_s(value, 30, "**\n**");

СoncatenationStr(name, value, str);

**if** (fwrite(str, 1, strlen(str), fp) != strlen(str)) {

perror("Ошибка записи в файл");

exit(EXIT\_FAILURE);

}

}

void WriteHashToFile(char\* key, char\* value, char is\_exist) {

char str[50];

СoncatenationStr(key, value, str);

str[strlen(str) + 3] = '\0';

str[strlen(str) + 2] = '\n';

str[strlen(str) + 1] = is\_exist;

str[strlen(str)] = ' ';

**if** (fwrite(str, 1, strlen(str), fp) != strlen(str)) {

perror("Ошибка записи в файл");

exit(EXIT\_FAILURE);

}

}

void DeleteStrFromFile(char\* str, char\* path) {

char buff[50];

char change\_buff[50];

char is\_del = '0';

fseek(fp, 0, SEEK\_END);

int size = ftell(fp);

char\* data = (char\*)calloc(size, **sizeof**(char));

fseek(fp, 0, SEEK\_SET);

**while** (fgets(buff, 50, fp)) {

strcpy\_s(change\_buff, 50, buff);

change\_buff[strlen(change\_buff) - 1] = '\0';

**if** ((strcmp(change\_buff, str) != 0) || (is\_del == '1')) {

strcat\_s(data, size, buff);

}

**else** is\_del = '1';

}

fclose(fp);

**if** ((fopen\_s(&fp, path, "w")) != NULL) {

perror("Ошибка открытия файла");

exit(EXIT\_FAILURE);

}

fputs (data, fp);

free(data);

}

void ReplaceStrInFile(char\* str, char\* new\_str, char\* path) {

char buff[50];

fseek(fp, 0, SEEK\_END);

int size = ftell(fp) + 50;

char\* data = (char\*)calloc(size, **sizeof**(char));

fseek(fp, 0, SEEK\_SET);

**while** (fgets(buff, 50, fp)) {

**if** (strcmp(buff, str) != 0) {

strcat\_s(data, size, buff);

}

**else** {

strcat\_s(data, size, new\_str);

}

}

fclose(fp);

**if** ((fopen\_s(&fp, path, "w")) != NULL) {

perror("Ошибка открытия файла");

exit(EXIT\_FAILURE);

}

fputs (data, fp);

free(data);

}

void СoncatenationStr(char\* name, char\* value, char\* str) {

strcpy\_s(str, 50, name);

str[strlen(str) + 1] = '\0';

str[strlen(str)] = ' ';

strcat\_s(str, 50, value);

}

void СoncatenationStrForHash(char\* key, char\* value, char is\_exist, char\* str) {

СoncatenationStr(key, value, str);

str[strlen(str) + 3] = '\0';

str[strlen(str) + 2] = '\n';

str[strlen(str) + 1] = is\_exist;

str[strlen(str)] = ' ';

}

void SetAdd(Set\* array, char\* name, char\* value) {

int index = FuncHash(value);

char is\_coincidence = '0';

**while** (array[index].value[0] != '\0') {

**if** (strcmp(array[index].value, value) == 0) {

is\_coincidence = '1';

**break**;

}

index = (index + FuncHashSec(value[0])) % 100;

}

**if** (is\_coincidence == '0') WriteListToFile(name, value);

**else** puts("Данный элемент уже есть в множестве");

}

void SetRem(Set\* array, char\* name, char\* value, char\* path) {

char str[50];

int index = FuncHash(value);

char is\_coincidence = '0';

**while** (array[index].value[0] != '\0') {

**if** (strcmp(array[index].value, value) == 0) {

is\_coincidence = '1';

СoncatenationStr(name, value, str);

**break**;

}

index = (index + FuncHashSec(value[0])) % 100;

}

**if** (is\_coincidence == '1') DeleteStrFromFile(str, path);

**else** puts("Такого элемента нет в множестве");

}

void SetIsMember(Set\* array, char\* value) {

int index = FuncHash(value);

char is\_coincidence = '0';

**while** (array[index].value[0] != '\0') {

**if** (strcmp(array[index].value, value) == 0) {

is\_coincidence = '1';

**break**;

}

index = (index + FuncHashSec(value[0])) % 100;

}

**if** (is\_coincidence == '1') puts("TRUE");

**else** puts("FALSE");

}

void SetReadFromFile(Set\*\* array, char\* main\_name) {

\*array = (Set\*)calloc(100, **sizeof**(Set));

char name[20];

char value[30];

**while** (!feof(fp)) {

fscanf\_s(fp, "%s %s**\n**", name, (unsigned)**sizeof**(name), value, (unsigned)**sizeof**(value));

**if** (strcmp(main\_name, name) == 0) {

int index = FuncHash(value);

**while** ((\*array)[index].value[0] != '\0') {

index = (index + FuncHashSec(value[0])) % 100;

}

strcpy\_s((\*array)[index].value, 30, value);

}

}

}

void StackPush(List\* node, List\* tail, char\* name, char\* value) {

PushToHeadList(&node, &tail, value); *//Добавляю в голову - становится первым элементом*

WriteListToFile(name, value);

}

void StackPop(List\* node, char\* name, char\* path) {

char value[30];

char str[50];

strcpy\_s(value, 30, PopFromHead(&node)); *//Удаляю голову - первый элемент*

printf("%s**\n**", value);

СoncatenationStr(name, value, str);

DeleteStrFromFile(str, path);

}

void StackReadFromFile(List\*\* node, List\*\* tail, char\* name) {

char text\_name[20];

char text\_value[30];

**while** (!feof(fp)) {

fscanf\_s(fp, "%s %s**\n**", text\_name, (unsigned)**sizeof**(text\_name), text\_value, (unsigned)**sizeof**(text\_value));

**if** (strcmp(text\_name, name) == 0) {

PushToHeadList(node, tail, text\_value); *//Первый элмент в файле - хвост*

}

}

**if** ((\*node) == NULL) {

puts("Стек пуст");

fclose(fp);

exit(EXIT\_SUCCESS);

}

}

void QueuePush(List\* node, List\* tail, char\* name, char\* value) {

PushToTailList(&node, &tail, value); *//Ложу в хвост - становится последним элементом*

WriteListToFile(name, value);

}

void QueuePop(List\* node, char\* name, char\* path) {

char value[30];

char str[50];

strcpy\_s(value, 30, PopFromHead(&node)); *//Удаляю голову - первый элемент*

printf("%s**\n**", value);

СoncatenationStr(name, value, str);

DeleteStrFromFile(str, path);

}

void QueueReadFromFile(List\*\* node, List\*\* tail, char\* name) {

char text\_name[20];

char text\_value[30];

**while** (!feof(fp)) {

fscanf\_s(fp, "%s %s**\n**", text\_name, (unsigned)**sizeof**(text\_name), text\_value, (unsigned)**sizeof**(text\_value));

**if** (strcmp(text\_name, name) == 0) {

PushToTailList(node, tail, text\_value); *//Первый элемент в файле - голова*

}

}

**if** ((\*node) == NULL) {

puts("Очередь пуста");

fclose(fp);

exit(EXIT\_SUCCESS);

}

}

void HashSet(HashTable\* row, char\* key, char\* value, char\* path) {

int index = FuncHash(key);

char is\_coincidence = '0';

char is\_del = '0';

int index\_del;

char str[50];

char new\_str[50];

**while** (row[index].value[0] != '\0') {

**if** (row[index].is\_exist == '0') {

**if** (is\_del == '0') {

index\_del = index;

is\_del = '1';

}

**else** **break**;

}

**if** (strcmp(row[index].key, key) == 0) {

is\_coincidence = '1';

**break**;

}

index = (index + FuncHashSec(key[0])) % 100;

}

**if** (is\_coincidence == '1') {

СoncatenationStrForHash(row[index].key, row[index].value, row[index].is\_exist, str);

strcpy\_s(row[index].key, 15, key);

strcpy\_s(row[index].value, 30, value);

row[index].is\_exist = '1';

СoncatenationStrForHash(row[index].key, row[index].value, row[index].is\_exist, new\_str);

ReplaceStrInFile(str, new\_str, path);

}

**else** **if** (is\_del == '1') {

СoncatenationStrForHash(row[index\_del].key, row[index\_del].value, row[index\_del].is\_exist, str);

strcpy\_s(row[index\_del].key, 15, key);

strcpy\_s(row[index\_del].value, 30, value);

row[index\_del].is\_exist = '1';

СoncatenationStrForHash(row[index\_del].key, row[index\_del].value, row[index\_del].is\_exist, new\_str);

ReplaceStrInFile(str, new\_str, path);

}

**else** {

СoncatenationStrForHash(row[index].key, row[index].value, row[index].is\_exist, str);

strcpy\_s(row[index].key, 15, key);

strcpy\_s(row[index].value, 30, value);

row[index].is\_exist = '1';

СoncatenationStrForHash(row[index].key, row[index].value, row[index].is\_exist, new\_str);

WriteHashToFile(row[index].key, row[index].value, row[index].is\_exist);

}

}

void HashDelete(HashTable\* row, char\* key, char\* path) {

int index = FuncHash(key);

char str[50];

char new\_str[50];

char is\_coincidence = '0';

**while** (row[index].value[0] != '\0') {

**if** (strcmp(row[index].key, key) == 0) {

is\_coincidence = '1';

**break**;

}

index = (index + FuncHashSec(key[0])) % 100;

}

**if** (is\_coincidence == '1') {

СoncatenationStrForHash(row[index].key, row[index].value, '1', str);

СoncatenationStrForHash(row[index].key, row[index].value, '0', new\_str);

ReplaceStrInFile(str, new\_str, path);

}

}

void HashGet(HashTable\* row, char\* key, char\* path) {

int index = FuncHash(key);

char is\_coincidence = '0';

**while** (row[index].value[0] != '\0') {

**if** ((strcmp(row[index].key, key) == 0) && (row[index].is\_exist == '1')) {

is\_coincidence = '1';

**break**;

}

index = (index + FuncHashSec(key[0])) % 100;

}

**if** (is\_coincidence == '1') printf("%s", row[index].value);

**else** printf("Значение по такому ключу не найдено**\n**");

}

void HashReadFromFile(HashTable\*\* row) {

\*row = (HashTable\*)calloc(100, **sizeof**(HashTable));

char key[15];

char value[30];

char is\_exist = '\0';

**while** (!feof(fp)) {

fscanf\_s(fp, "%s %s %c**\n**", key, (unsigned)**sizeof**(key), value, (unsigned)**sizeof**(value), &is\_exist, (unsigned)**sizeof**(is\_exist));

int index = FuncHash(key);

**while** ((\*row)[index].value[0] != '\0') {

index = (index + FuncHashSec(key[0])) % 100;

}

strcpy\_s((\*row)[index].key, 15, key);

strcpy\_s((\*row)[index].value, 30, value);

(\*row)[index].is\_exist = is\_exist;

}

}

int FuncHash(char\* key) {

int sum = 0;

**for** (int i = 0; key[i]; ++i)

sum += key[i];

**return** sum % 100;

}

int FuncHashSec(char ch) {

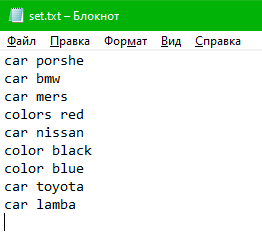
int sum = ch \* 37;

**return** sum % 100;

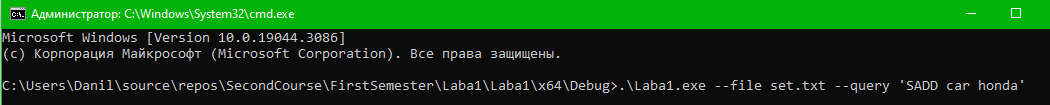
}

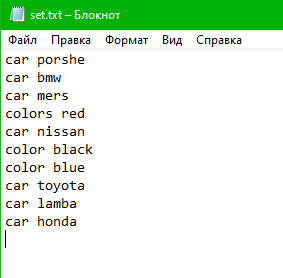
**Пример работы:**

1. Множество

Файл set.txt до работы с множеством, все команды выполняются по порядку, каждая последующая включает изменения файла от предыдущей команды

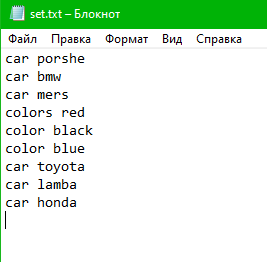
1.1 SADD



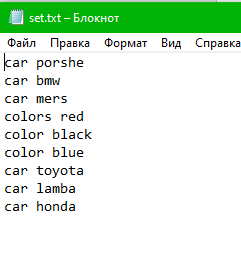


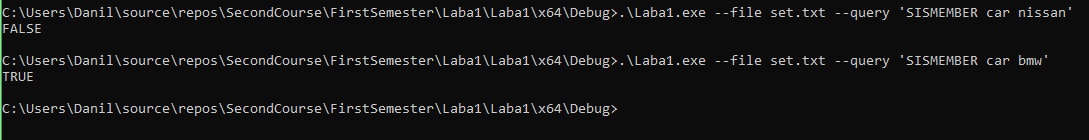
1.2 SREM





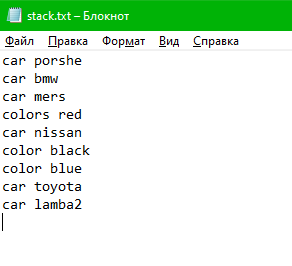
1.3 SISMEMBER





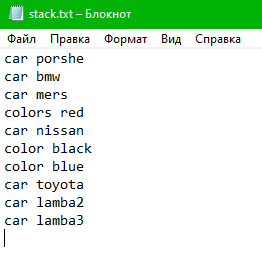
2. Стек

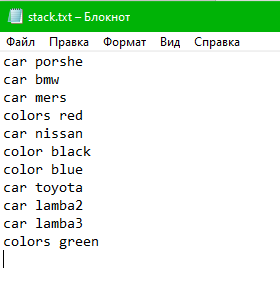
Файл stack.txt



2.1 SPUSH

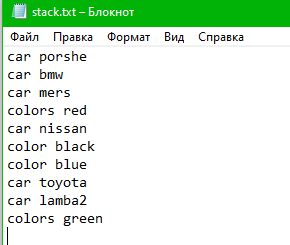




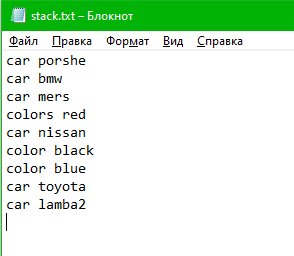


2.2 SPOP



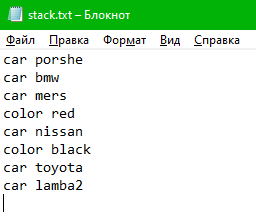






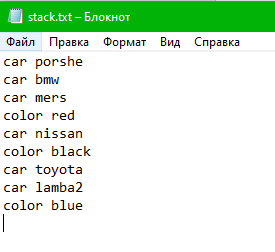
3. Queue

Файл stack.txt

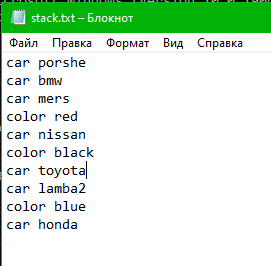


3.1 QPUSH



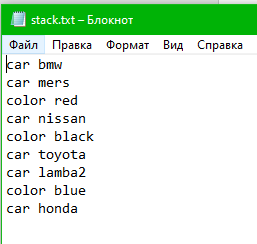




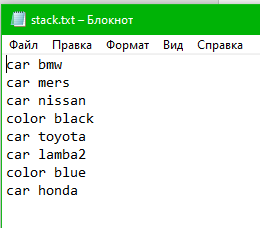


3.2 QPOP



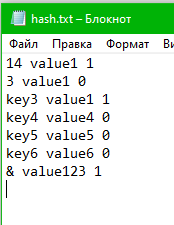




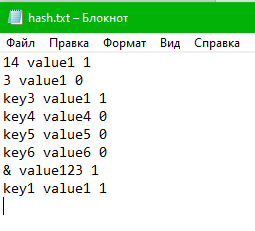


4. Hash

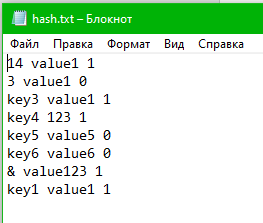
Файл hash.txt



4.1 HSET

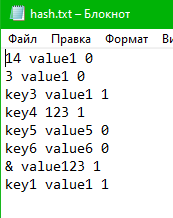






4.2 HDEL





4.3 HGET

