МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ образовательное учреждениевысшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра вычислительной техники

**ОТЧЁТ по ПРАКТИЧЕСКОЙ работе №1**

**по дисциплине «ПРОГРАММИРОВАНИНЕ»**

**«Разработка системы управления базой данных»**

|  |  |
| --- | --- |
| Факультет: АВТ  Группа: АБ-220  Студент(ы): Лазарев Кирилл | Преподаватель: Исаев Г.А. |

Новосибирск, 2023 г.

**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)

Таблица операций

Тип Контейнера Добавление Удаление Чтение

Множество SADD SREM SISMEMBER

Стек SPUSH SPOP SPOP

Очередь QPUSH QPOP QPOP

Хеш-таблица HSET HDEL HGET

\*Для множества операция SISMEMBER выполняет проверку является ли элемент частью множества

\*Для стека и очереди операции SPOP и QPOP выполняют удаление и чтение одновременно.

1. **Постановка задачи:**

1. Реализовать заданные структуры данных заданным набором операций;

2. Реализовать запись и чтение из файла.

1. **Алгоритм решения задачи**

Реализовать и написать заданные структуры данных с заданных набором операций с асимптотикой O(1).

Распарсить строку входных аргументов и добавить обязательную проверку для флагов --file и –query.

Открыть файл базы данных на чтение и найти в нем нужную структуру с заданным именем при ее наличии.

Если такой структуры нет, но операция на добавление элемента выполнена корректно, то добавить в файл новую структуру.

Открыть файл на запись и изменить файл соответствуя запросу

**Текст программы**

**stack.h:**

**typedef** **struct** {

**char** data[**64**][**128**];

**unsigned** **int** head;

} Stack;

**void** **spush**(Stack\* stack, **char** value[]) { // Заносим значение в стек

snprintf(stack->data[stack->head], **sizeof**(stack->data[stack->head]), "%s", value);

stack->head++;

}

**char**\* **spop**(Stack\* stack) { // Достаем значение из стека

**return** stack->head != **0** ? stack->data[--stack->head] : NULL;

}

**void** **stack\_save**(**char**\* name, Stack\* stack, **FILE**\* file) { // Функция для сохранения стека в файл

**char** data[**1024**];

snprintf(data, **sizeof**(data), "stack %s", name);

**for** (**int** i = **0**; i != stack->head; i++)

snprintf(data, **sizeof**(data), "%s %s", data, stack->data[i]);

snprintf(data, **sizeof**(data), "%s**\n**", data);

printf("**\n**STACK DATA:%s", data);

fputs(data, file);

}

**queue.h:**

**typedef** **struct** {

**char** data[**64**][**128**];

**unsigned** **int** start, end;

} Queue;

**void** **qpush**(Queue\* queue, **char** value[]) { // Добавляем значение в очередь

snprintf(queue->data[queue->end], **sizeof**(queue->data[queue->end]), "%s", value);

queue->end++;

}

**char**\* **qpop**(Queue\* queue) { // Удаляем элемент из очерди

**return** (queue->end != queue->start) ? queue->data[queue->start++] : NULL;

}

**void** **queue\_save**(**char**\* name, Queue\* queue, **FILE**\* file) { // Функция для сохранения очереди в файл

**char** data[**1024**];

snprintf(data, **sizeof**(data), "queue %s", name);

**for** (**int** i = queue->start; i != queue->end; i++)

snprintf(data, **sizeof**(data), "%s %s", data, queue->data[i]);

snprintf(data, **sizeof**(data), "%s**\n**", data);

printf("**\n**QUEUE DATA:%s", data);

fputs(data, file);

}

**set.h:**

#define set\_lenght 256

**typedef** **struct** {

**char** data[set\_lenght][**128**];

**unsigned** **int** len;

**int** (\*hash)(**char**\*);

} Set;

**void** **sadd**(Set\* set, **char** value[]) { // Добавляем значение в множество

**int** hash = set->hash(value);

printf("HASH **\"**%s**\"** = %d**\n**", value, hash);

snprintf(set->data[hash], **sizeof**(set->data[hash]), "%s", value);

set->len++;

}

**int** **srem**(Set\* set, **char** value[]) { // Удаляем значение из множества

**if** (set->len == **0**) **return** -**1**;

**int** hash = set->hash(value);

**if** (strcmp(set->data[hash], "") != **0**) {

snprintf(set->data[hash], **sizeof**(set->data[hash]), "");

set->len--;

**return** hash;

}

**return** -**1**;

}

**int** **sismember**(Set\* set, **char** value[]) { // Проверяем есть ли значение в множестве

**return** strcmp(set->data[set->hash(value)], "") != **0** ? **1** : **0**;

}

**void** **set\_save**(**char**\* name, Set\* set, **FILE**\* file) { // Функция для сохранения множества в файл

printf("len:%d**\n**", set->len);

**char** data[**1024**];

snprintf(data, **sizeof**(data), "set %s", name);

**for** (**int** i = **0**, c = **0**; i < set\_lenght && c < set->len; i++) {

**if** (strcmp(set->data[i], "") != **0**) {

snprintf(data, **sizeof**(data), "%s %s", data, set->data[i]);

c++;

}

}

snprintf(data, **sizeof**(data), "%s**\n**", data);

printf("**\n**SET DATA:%s", data);

fputs(data, file);

}

**hash\_table.h:**

#define hash\_table\_lenght 256

**typedef** **struct** {

**char** keys[hash\_table\_lenght][**128**];

**char** values[hash\_table\_lenght][**128**];

**unsigned** **int** len;

**int** (\*hash)(**char**\*);

} Hash\_table;

**void** **hset**(Hash\_table\* hash\_table, **char**\* key, **char**\* value) { // Добавляем ключ и значение в таблицу

**int** hash = hash\_table->hash(key);

**if** (strcmp(hash\_table->keys[hash], key) == **0**) {

snprintf(hash\_table->values[hash], **sizeof**(hash\_table->values[hash]), "%s", value);

hash = hash\_table\_lenght;

}

**for** (; hash < hash\_table\_lenght; hash++) {

**if** (strcmp(hash\_table->keys[hash], "") == **0**) {

snprintf(hash\_table->keys[hash], **sizeof**(hash\_table->keys[hash]), "%s", key);

snprintf(hash\_table->values[hash], **sizeof**(hash\_table->values[hash]), "%s", value);

**break**;

}

}

printf("HSET:%s | %s**\n**", hash\_table->keys[hash], hash\_table->values[hash]);

hash\_table->len++;

}

**void** **hdel**(Hash\_table\* hash\_table, **char**\* key) { // Удаляем ключ и значение из таблицу

**if** (hash\_table->len != **0**) {

**int** hash = hash\_table->hash(key);

**for** (; hash < hash\_table\_lenght; hash++) {

**if** (strcmp(hash\_table->keys[hash], key) == **0**) {

snprintf(hash\_table->keys[hash], **sizeof**(hash\_table->keys[hash]), "");

snprintf(hash\_table->values[hash], **sizeof**(hash\_table->values[hash]), "");

hash\_table->len--;

**break**;

}

}

}

}

**char**\* **hget**(Hash\_table\* hash\_table, **char**\* key) { // Получаем значение по ключу

**if** (hash\_table->len != **0**) {

**int** hash = hash\_table->hash(key);

**for** (; hash < hash\_table\_lenght; hash++) {

**if** (strcmp(hash\_table->keys[hash], key) == **0**) {

**return** hash\_table->values[hash];

}

}

}

**return** NULL;

}

**void** **hash\_table\_save** (**char**\* name, Hash\_table\* hash\_table, **FILE**\* file) { // Функция для сохранения таблицы в файл

printf("len:%d**\n**", hash\_table->len);

**char** data[**1024**];

snprintf(data, **sizeof**(data), "hash %s", name);

**for** (**int** i = **0**, c = **0**; i < hash\_table\_lenght && c < hash\_table->len; i++) {

**if** (strcmp(hash\_table->keys[i], "") != **0**) {

snprintf(data, **sizeof**(data), "%s %s:%s", data, hash\_table->keys[i], hash\_table->values[i]);

c++;

}

}

snprintf(data, **sizeof**(data), "%s**\n**", data);

printf("HASH DATA:%s", data);

fputs(data, file);

}

**main.c:**

**Результат работы программы**



Вызов программы для работы с множеством 1



Результат программы после работы с множеством 1



Вызов программы для работы с множеством 2



Результат программы после работы с множеством 2



Вызов программы для работы с множеством 3



Результат программы после работы с множеством 3



Вызов программы для работы с множеством 4



Результат программы после работы с множеством 1



Вызов программы для работы с множеством 5



Результат программы после работы с множеством 5



Вызов программы для работы с множеством 6



Результат программы после работы с множеством 6



Вызов программы для работы со стеком 1



Результат программы после работы со стеком 1



Вызов программы для работы со стеком 2



Результат программы после работы со стеком 2



Вызов программы для работы со стеком 3



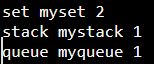
Результат программы после работы со стеком 3



Результат программы после работы со стеком 3.1



Вызов программы для работы с очередью 1



Результат программы после работы с очередью 1



Вызов программы для работы с очередью 2



Результат программы после работы с очередью 2



Вызов программы для работы с очередью 3



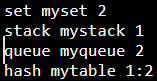
Результат программы после работы с очередью 3



Результат программы после работы с очередью 3.1



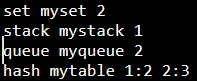
Вызов программы для работы с хэш-таблицей 1



Результат программы после работы с хэш-таблицей 1



Вызов программы для работы с хэш-таблицей 2



Результат программы после работы с хэш-таблицей 2



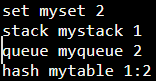
Вызов программы для работы с хэш-таблицей 3



Результат программы после работы с хэш-таблицей 3



Вызов программы для работы с хэш-таблицей 4



Результат программы после работы с хэш-таблицей 4

**Вывод**: Результат работы программы совпадает с ожидаемыми результатами, что говорит о корректной работе программы. Мы изучили новые структуры данных и научились полностью их реализовывать на C что дало незабываемый опыт.

Спасибо что не на ассемблере