МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ образовательное учреждениевысшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра защиты информации



**ОТЧЁТ по ПРАКТИЧЕСКОЙ работе №3**

**по дисциплине «ПРОГРАММИРОВАНИЕ»**

**«Разработка сервиса сокращения ссылок»**

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АБс-222  Студент: Гатауллин Д.Р. | Преподаватель: Архипова А.Б. |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

**Цели и задачи работы**: Изучение тем Сеть, ООП, Web.

**Задание к работе**: Требуется реализовать сервис для сокращения ссылок. Сервис должен принимать запросы по http протоколу.

Сервис должен иметь две конечные точки:

1. Сокращение ссылки

URL: /

Метод: POST

Запрос содержит ссылку которую нужно сократить.

Ответ содержит сокращенную ссылку.

2. Переход по сокращённой ссылке

URL: /<сокращение>

Метод: GET

Ответ должен перенаправлять пользователя на исходную

страницу (см. перенаправления)

Для хранения сокращенных ссылок используем СУБД из практики

2.

2. Реализовать клиента для СУБД на том языке, на которым вы

пишите сервис сокращения ссылок.

Пример аналогичного сервиса: <https://clck.ru>

**Текст программы**

==========================================================

С

==========================================================

main.c

==========================================================

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "stack.h" // Подключение заголовочного файла для работы со стеком

#include "set.h" // Подключение заголовочного файла для работы с множеством

#include "queue.h" // Подключение заголовочного файла для работы с очередью

#include "table.h" // Подключение заголовочного файла для работы с хеш-таблицей

#include <winsock2.h>

#include <ws2tcpip.h>

#include <windows.h>

#pragma comment(lib, "ws2\_32.lib")

#define PORT 6379 // Определение порта для сервера

#define BACKLOG 10 // Определение максимального количества ожидающих подключений

#define BUFFER\_SIZE 104857600 // Определение размера буфера

HANDLE mutex; // Объявление мьютекса

DWORD WINAPI handle\_client(LPVOID lpParam); // Объявление функции обработки клиента

int main() {

WSADATA wsa\_data;

int result = WSAStartup(MAKEWORD(2, 2), &wsa\_data); // Инициализация библиотеки сокетов

SOCKET listen\_socket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP); // Создание сокета для прослушивания

struct sockaddr\_in server\_address; // Структура для хранения адреса сервера

server\_address.sin\_family = AF\_INET;

server\_address.sin\_addr.s\_addr = htonl(INADDR\_ANY);

server\_address.sin\_port = htons(PORT);

result = bind(listen\_socket, (struct sockaddr\*)&server\_address, sizeof(server\_address)); // Привязка сокета к адресу

result = listen(listen\_socket, BACKLOG); // Начало прослушивания сокета

printf("The server is running and waiting for connections on port %d\n", PORT);

mutex = CreateMutex(NULL, FALSE, NULL); // Создание мьютекса

while (1) {

SOCKET client\_socket = accept(listen\_socket, NULL, NULL); // Принятие подключения от клиента

CreateThread(NULL, 0, handle\_client, (LPVOID)client\_socket, 0, NULL); // Создание потока для обработки клиента

}

closesocket(listen\_socket); // Закрытие сокета прослушивания

WSACleanup(); // Очистка библиотеки сокетов

return 0;

}

DWORD WINAPI handle\_client(LPVOID lpParam) {

WaitForSingleObject(mutex, INFINITE); // Получение мьютекса

SOCKET client\_socket = (SOCKET)lpParam; // Получение сокета клиента

char\* buffer = (char\*)malloc(BUFFER\_SIZE \* sizeof(char)); // Выделение памяти под буфер

int bytes\_received;

bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE, 0); // Получение данных от клиента

buffer[bytes\_received] = '\0';

char\*\* argv = NULL;

int argc = 0;

char\* token = strtok(buffer, " "); // Разбиение полученных данных на токены

while (token != NULL) {

char\*\* temp = realloc(argv, sizeof(char\*) \* (argc + 1));

argv = temp;

argv[argc++] = token;

token = strtok(NULL, " ");

}

// Обработка полученных данных и выполнение соответствующих операций

char\* filename = NULL;

char\* query = NULL;

char\* key = NULL;

char\* basename = NULL;

char\* item = NULL;

int temp;

char\* result = NULL;

if (argc < 4 || argc > 7) {

result = malloc(100);

sprintf(result, "Error.\n");

goto skip;

}

for (int i = 0; i < argc; i++) {

if (strcmp(argv[i], "--file") == 0 && i + 1 < argc) {

filename = argv[i + 1];

}

else if (strcmp(argv[i], "--query") == 0 && i + 1 < argc) {

query = argv[i + 1];

temp = i + 1;

basename = argv[i + 2];

if (i + 5 > argc) key = argv[i + 3];

else {

item = argv[i + 3];

key = argv[i + 4];

if (key == NULL || item == NULL) {

result = malloc(100);

sprintf(result, "Error.\n");

goto skip;

}

}

}

}

int pos1 = 0;;

int pos2 = 0;;

int status = 0;;

if (filename != NULL && query != NULL) {

FILE\* file = fopen(filename, "r");

if (!file) {

FILE\* file = fopen(filename, "w");

}

if (strcmp(argv[temp], "SPUSH") == 0) {

if (key == NULL) {

result = malloc(100);

sprintf(result, "Error.\n");

goto skip;

}

Stack\* stack = loadFromFileStack(filename, basename, &pos1, &pos2, &status);

if (stack == NULL) {

result = malloc(100);

sprintf(result, "Error when opening a file!\n");

fclose(file);

}

else {

if (pos1 + pos2 == 0) {

result = malloc(100);

sprintf(result, "Such a database, alas, does not exist!\n");

fclose(file);

}

else {

SPUSH(stack, key);

result = malloc(strlen(key) + 5);

sprintf(result, "-> %s\n", key);

if (status == 1) status = 0;

fclose(file);

saveToFileStack(stack, filename, basename, &pos1, &pos2, &status);

}

}

}

if (strcmp(argv[temp], "SPOP") == 0) {

Stack\* stack = loadFromFileStack(filename, basename, &pos1, &pos2, &status);

if (stack == NULL) {

result = malloc(100);

sprintf(result, "Error when opening a file!\n");

fclose(file);

}

else {

if (pos1 + pos2 == 0) {

result = malloc(40);

sprintf(result, "Such a database, alas, does not exist!\n");

fclose(file);

}

else {

char\* element = SPOP(stack);

result = malloc(strlen(element) + 5);

sprintf(result, "-> %s\n", element);

if (status == 2) status = 0;

fclose(file);

saveToFileStack(stack, filename, basename, &pos1, &pos2, &status);

}

}

}

if (strcmp(argv[temp], "SADD") == 0) {

if (key == NULL) {

result = malloc(100);

sprintf(result, "Error.\n");

goto skip;

}

Set\* set = loadFromFileSet(filename, basename, &pos1, &pos2, &status);

if (set == NULL) {

result = malloc(100);

sprintf(result, "Error when opening a file!\n");

fclose(file);

}

else {

if (pos1 + pos2 == 0) {

result = malloc(100);

sprintf(result, "Such a database, alas, does not exist!\n");

fclose(file);

}

else {

SADD(set, key);

result = malloc(strlen(key) + 5);

sprintf(result, "-> %s\n", key);

if (status == 1) status = 0;

fclose(file);

saveToFileSet(set, filename, basename, &pos1, &pos2, &status);

}

}

}

if (strcmp(argv[temp], "SREM") == 0) {

if (key == NULL) {

result = malloc(100);

sprintf(result, "Error.\n");

goto skip;

}

Set\* set = loadFromFileSet(filename, basename, &pos1, &pos2, &status);

if (set == NULL) {

result = malloc(100);

sprintf(result, "Error when opening a file!\n");

fclose(file);

}

else {

if (pos1 + pos2 == 0) {

result = malloc(100);

sprintf(result, "Such a database, alas, does not exist!\n");

fclose(file);

}

else {

SREM(set, key);

result = malloc(strlen(key) + 5);

sprintf(result, "-> %s\n", key);

if (status == 2) status = 0;

fclose(file);

saveToFileSet(set, filename, basename, &pos1, &pos2, &status);

}

}

}

if (strcmp(argv[temp], "SISMEMBER") == 0) {

if (key == NULL) {

result = malloc(100);

sprintf(result, "Error.\n");

goto skip;

}

Set\* set = loadFromFileSet(filename, basename, &pos1, &pos2, &status);

if (set == NULL) {

result = malloc(100);

sprintf(result, "Error when opening a file!\n");

fclose(file);

}

else {

if (pos1 + pos2 == 0) {

result = malloc(100);

sprintf(result, "Such a database, alas, does not exist!\n");

fclose(file);

}

else {

result = malloc(100);

if (SISMEMBER(set, key)) sprintf(result, "-> True\n");

else sprintf(result, "-> False\n");

fclose(file);

}

}

}

if (strcmp(argv[temp], "QPUSH") == 0) {

if (key == NULL) {

result = malloc(100);

sprintf(result, "Error.\n");

goto skip;

}

Queue\* queue = loadFromFileQueue(filename, basename, &pos1, &pos2, &status);

if (queue == NULL) {

result = malloc(100);

sprintf(result, "Error when opening a file!\n");

fclose(file);

}

else {

if (pos1 + pos2 == 0) {

result = malloc(100);

sprintf(result, "Such a database, alas, does not exist!\n");

fclose(file);

}

else {

QPUSH(queue, key);

result = malloc(strlen(key) + 5);

sprintf(result, "-> %s\n", key);

if (status == 1) status = 0;

fclose(file);

saveToFileQueue(queue, filename, basename, &pos1, &pos2, &status);

}

}

}

if (strcmp(argv[temp], "QPOP") == 0) {

Queue\* queue = loadFromFileQueue(filename, basename, &pos1, &pos2, &status);

if (queue == NULL) {

result = malloc(100);

sprintf(result, "Error when opening a file!\n");

fclose(file);

}

else {

if (pos1 + pos2 == 0) {

result = malloc(100);

sprintf(result, "Such a database, alas, does not exist!\n");

fclose(file);

}

else {

char\* element = QPOP(queue);

result = malloc(strlen(element) + 5);

sprintf(result, "-> %s\n", element);

if (status == 2) status = 0;

fclose(file);

saveToFileQueue(queue, filename, basename, &pos1, &pos2, &status);

}

}

}

if (strcmp(argv[temp], "HSET") == 0) {

if (key == NULL || item == NULL) {

result = malloc(100);

sprintf(result, "Error.\n");

goto skip;

}

HashTable\* hashtable = loadFromFileTable(filename, basename, &pos1, &pos2, &status);

if (hashtable == NULL) {

result = malloc(100);

sprintf(result, "Error when opening a file!\n");

fclose(file);

}

else {

if (pos1 + pos2 == 0) {

result = malloc(100);

sprintf(result, "Such a database, alas, does not exist!\n");

fclose(file);

}

else {

HSET(hashtable, key, item);

result = malloc(strlen(item) + strlen(key) + 20);

sprintf(result, "-> %s %s\n", item, key);

if (status == 1) status = 0;

fclose(file);

saveToFileTable(hashtable, filename, basename, &pos1, &pos2, &status);

}

}

}

if (strcmp(argv[temp], "HDEL") == 0) {

if (key == NULL) {

result = malloc(100);

sprintf(result, "Error.\n");

goto skip;

}

HashTable\* hashtable = loadFromFileTable(filename, basename, &pos1, &pos2, &status);

if (hashtable == NULL) {

result = malloc(100);

sprintf(result, "Error when opening a file!\n");

fclose(file);

}

else {

if (pos1 + pos2 == 0) {

result = malloc(100);

sprintf(result, "Such a database, alas, does not exist!\n");

fclose(file);

}

else {

HDEL(hashtable, key);

result = malloc(strlen(key) + 5);

sprintf(result, "-> %s\n", key);

if (status == 2) status = 0;

fclose(file);

saveToFileTable(hashtable, filename, basename, &pos1, &pos2, &status);

}

}

}

if (strcmp(argv[temp], "HGET") == 0) {

if (key == NULL) {

result = malloc(100);

sprintf(result, "Error.\n");

goto skip;

}

HashTable\* hashtable = loadFromFileTable(filename, basename, &pos1, &pos2, &status);

if (hashtable == NULL) {

result = malloc(100);

sprintf(result, "Error.\n");

fclose(file);

}

else {

if (pos1 + pos2 == 0) {

result = malloc(100);

sprintf(result, "Error.\n");

fclose(file);

}

else {

if (HGET(hashtable, key) != NULL) {

result = malloc(strlen(HGET(hashtable, key)) + 15);

sprintf(result, "%s\n", HGET(hashtable, key));

}

else {

result = malloc(100);

sprintf(result, "-> False\n");

}

fclose(file);

}

}

}

}

else {

result = malloc(100);

sprintf(result, "Error.\n");

}

skip: {

if (result == NULL) {

result = malloc(100);

sprintf(result, "Error.\n");

}

int bytes\_sent = send(client\_socket, result, strlen(result), 0); // Отправка результата клиенту

closesocket(client\_socket); // Закрытие сокета клиента

free(argv);

free(result);

free(buffer);

}

ReleaseMutex(mutex); // Освобождение мьютекса

}

==========================================================

queue.c

==========================================================

#define \_CRT\_SECURE\_NO\_WARNINGS // Отключает предупреждения о небезопасных функциях CRT

#include "queue.h" // Подключает заголовочный файл для работы с очередью

#include <stdio.h> // Подключает стандартную библиотеку ввода-вывода

#include <stdlib.h> // Подключает библиотеку, содержащую функции для работы с памятью, процессами и преобразованиями

#include <string.h> // Подключает библиотеку для работы со строками

int countWordsInFileQueue(const char\* filename) { // Функция для подсчета слов в файле

FILE\* file = fopen(filename, "r"); // Открывает файл для чтения

int count = 0; // Переменная для подсчета слов

char word[10000]; // Массив для хранения слов

while (fscanf(file, "%s", word) != EOF) { // Цикл для чтения слов из файла

count++; // Увеличивает счетчик слов

}

fclose(file); // Закрывает файл

return count; // Возвращает количество слов в файле

}

Queue\* initQueue() { // Функция для инициализации очереди

Queue\* queue = (Queue\*)malloc(sizeof(Queue)); // Выделяет память под структуру очереди

queue->front = NULL; // Устанавливает указатель на начало очереди в NULL

queue->rear = NULL; // Устанавливает указатель на конец очереди в NULL

return queue; // Возвращает указатель на очередь

}

void QPUSH(Queue\* queue, char\* element) { // Функция для добавления элемента в очередь

NodeQueue\* newNode = (NodeQueue\*)malloc(sizeof(NodeQueue)); // Выделяет память под новый узел очереди

newNode->element = \_strdup(element); // Копирует элемент в новый узел

newNode->next = NULL; // Устанавливает указатель на следующий узел в NULL

if (queue->front == NULL) { // Если очередь пуста

queue->front = newNode; // Устанавливает новый узел как начало очереди

queue->rear = newNode; // Устанавливает новый узел как конец очереди

}

else { // Если в очереди уже есть элементы

queue->rear->next = newNode; // Добавляет новый узел в конец очереди

queue->rear = newNode; // Устанавливает новый узел как конец очереди

}

}

char\* QPOP(Queue\* queue) { // Функция для удаления элемента из очереди

if (queue->front == NULL) { // Если очередь пуста

return ("The queue is empty"); // Возвращает сообщение об ошибке

}

NodeQueue\* poppedNode = queue->front; // Сохраняет указатель на узел, который нужно удалить

char\* element = poppedNode->element; // Сохраняет элемент, который нужно удалить

queue->front = poppedNode->next; // Перемещает указатель на начало очереди на следующий узел

if (queue->front == NULL) { // Если очередь стала пустой

queue->rear = NULL; // Устанавливает указатель на конец очереди в NULL

}

free(poppedNode); // Освобождает память, занятую узлом

return element; // Возвращает удаленный элемент

}

void saveToFileQueue(Queue\* queue, const char\* filename, const char\* basename, int\* pos1, int\* pos2, int\* status) {

FILE\* file = fopen(filename, "r");

FILE\* tempFile = fopen("temp.data", "w");

int ch;

fseek(file, 0, SEEK\_SET);

fseek(tempFile, 0, SEEK\_SET);

while ((ch = fgetc(file)) != EOF) {

fputc(ch, tempFile);

if (ftell(tempFile) == \*pos1 - 2 && \*status == 2)

fprintf(tempFile, "\t%s", queue->front->element);

else if (ftell(tempFile) == \*pos1) {

NodeQueue\* currentNode = queue->front;

while (currentNode != NULL) {

if (currentNode->next == NULL)

fprintf(tempFile, "%s\n", currentNode->element);

else

fprintf(tempFile, "%s\t", currentNode->element);

currentNode = currentNode->next;

}

if (\*status == 1) {

fseek(tempFile, \*pos1 - 1, SEEK\_SET);

fprintf(tempFile, "\n");

}

fseek(file, \*pos2, SEEK\_SET);

}

}

free(queue->rear);

free(queue->front);

free(queue);

fclose(file);

fclose(tempFile);

remove(filename);

rename("temp.data", filename);

}

Queue\* loadFromFileQueue(const char\* filename, const char\* basename, int\* pos1, int\* pos2, int\* status) {

FILE\* file = fopen(filename, "r");

if (file == NULL) {

return NULL;

}

int num\_lines = countWordsInFileQueue(filename);

char\*\* line = malloc(num\_lines \* sizeof(char\*));

for (int i = 0; i < num\_lines; i++) line[i] = malloc(10000 \* sizeof(char));

Queue\* queue = initQueue();

int tempory = 0;

int pos3 = 0;

int temp1 = 0;

int temp2 = 0;

char c = '1';

for (int i = 0; i < num\_lines; ++i) {

fscanf(file, "%s", line[i]);

c = getc(file);

pos3 = ftell(file);

if (!strcmp(line[i], basename)) {

fseek(file, -3 - strlen(line[i]), SEEK\_CUR);

if (getc(file) == '\n' || i == 0) {

fseek(file, pos3, SEEK\_SET);

tempory = 1;

\*pos1 = ftell(file);

temp1 = i + 1;

}

else fseek(file, pos3, SEEK\_SET);

}

if (c == '\n' && tempory == 1) {

temp2 = i;

\*pos2 = ftell(file);

tempory = 0;

}

if (feof(file))

break;

}

if (temp1 == temp2)

\*status = 1;

if (temp1 == temp2 + 1)

\*status = 2;

while (temp1 < temp2 + 1) {

QPUSH(queue, line[temp1]);

temp1++;

}

fclose(file);

for (int i = 0; i < num\_lines; i++) {

free(line[i]);

}

free(line);

return queue;

}

==========================================================

queue.h

==========================================================

#ifndef QUEUE\_H

#define QUEUE\_H

typedef struct NodeQueue {

char\* element;

struct NodeQueue\* next;

} NodeQueue;

typedef struct Queue {

NodeQueue\* front;

NodeQueue\* rear;

} Queue;

Queue\* initQueue();

void QPUSH(Queue\* queue, char\* element);

char\* QPOP(Queue\* queue);

void saveToFileQueue(Queue\* queue, const char\* filename, const char\* basename, int \*pos1, int \*pos2, int \*status);

Queue\* loadFromFileQueue(const char\* filename, const char\* basename, int \*pos1, int \*pos2, int \*status);

#endif

==========================================================

set.c

==========================================================

#define \_CRT\_SECURE\_NO\_WARNINGS // Отключает предупреждения о небезопасных функциях CRT

#include "set.h" // Подключает заголовочный файл для работы с множествами

#include <stdio.h> // Подключает стандартную библиотеку ввода-вывода

#include <stdlib.h> // Подключает библиотеку, содержащую функции для работы с памятью, процессами и преобразованиями

#include <string.h> // Подключает библиотеку для работы со строками

#define MAX\_SIZE 100000 // Определяет максимальный размер хеш-таблицы

int countWordsInFileSet(const char\* filename) { // Функция для подсчета слов в файле

FILE\* file = fopen(filename, "r"); // Открывает файл для чтения

int count = 0; // Переменная для подсчета слов

char word[10000]; // Массив для хранения слов

while (fscanf(file, "%s", word) != EOF) { // Цикл для чтения слов из файла

count++; // Увеличивает счетчик слов

}

fclose(file); // Закрывает файл

return count; // Возвращает количество слов в файле

}

Set\* initSet() { // Функция для инициализации множества

Set\* set = (Set\*)malloc(sizeof(Set)); // Выделяет память под структуру множества

set->head = NULL; // Устанавливает указатель на голову множества в NULL

set->size = 0; // Устанавливает размер множества в 0

set->tableSize = MAX\_SIZE; // Устанавливает размер хеш-таблицы

set->hashTable = (Node\*\*)malloc(MAX\_SIZE \* sizeof(Node\*)); // Выделяет память под хеш-таблицу

set->emptySlots = (int\*)malloc(MAX\_SIZE \* sizeof(int)); // Выделяет память под массив пустых слотов

for (int i = 0; i < MAX\_SIZE; i++) { // Цикл для инициализации хеш-таблицы и массива пустых слотов

set->hashTable[i] = NULL; // Устанавливает каждый элемент хеш-таблицы в NULL

set->emptySlots[i] = 1; // Устанавливает каждый элемент массива пустых слотов в 1

}

return set; // Возвращает указатель на множество

}

int calculateHashS(const char\* element) { // Функция для вычисления хеша элемента

int hash = 0; // Переменная для хранения хеша

for (int i = 0; element[i] != '\0'; i++) { // Цикл для обработки каждого символа элемента

hash = 31 \* hash + element[i]; // Вычисляет хеш

}

return abs(hash) % MAX\_SIZE; // Возвращает абсолютное значение хеша, ограниченное максимальным размером

}

void SADD(Set\* set, char\* element) { // Функция для добавления элемента в множество

int hash = calculateHashS(element); // Вычисляет хеш элемента

if (set->hashTable[hash] != NULL) { // Если в хеш-таблице уже есть элемент с таким хешем

return; // Возвращает управление

}

Node\* newNode = (Node\*)malloc(sizeof(Node)); // Выделяет память под новый узел

newNode->element = \_strdup(element); // Копирует элемент в новый узел

newNode->hash = hash; // Сохраняет хеш в новом узле

newNode->next = set->head; // Устанавливает указатель на следующий узел в голову множества

if (set->head != NULL) { // Если множество не пусто

set->head->prev = newNode; // Устанавливает указатель на предыдущий узел головы множества в новый узел

}

set->head = newNode; // Устанавливает новый узел как голову множества

set->hashTable[hash] = newNode; // Сохраняет новый узел в хеш-таблице

set->size++; // Увеличивает размер множества

}

void SREM(Set\* set, const char\* element) { // Функция для удаления элемента из множества

int hash = calculateHashS(element); // Вычисляет хеш элемента

if (set->hashTable[hash] != NULL && strcmp(set->hashTable[hash]->element, element) == 0) { // Если в хеш-таблице есть элемент с таким хешем и элемент совпадает

Node\* nodeToRemove = set->hashTable[hash]; // Сохраняет указатель на узел, который нужно удалить

if (nodeToRemove == set->head) { // Если узел является головой множества

set->head = nodeToRemove->next; // Перемещает указатель на голову множества на следующий узел

}

else { // Если узел не является головой множества

if (nodeToRemove->prev != NULL) { // Если у узла есть предыдущий узел

nodeToRemove->prev->next = nodeToRemove->next; // Перемещает указатель на следующий узел предыдущего узла на следующий узел удаляемого узла

}

}

if (nodeToRemove->next != NULL) { // Если у узла есть следующий узел

nodeToRemove->next->prev = nodeToRemove->prev; // Перемещает указатель на предыдущий узел следующего узла на предыдущий узел удаляемого узла

}

free(nodeToRemove->element); // Освобождает память, занятую элементом удаляемого узла

free(nodeToRemove); // Освобождает память, занятую узлом

set->hashTable[hash] = NULL; // Устанавливает элемент хеш-таблицы с хешем удаляемого элемента в NULL

set->size--; // Уменьшает размер множества

return; // Возвращает управление

}

}

int SISMEMBER(Set\* set, const char\* element) { // Функция для проверки, содержит ли множество элемент

int hash = calculateHashS(element); // Вычисляет хеш элемента

if (set->hashTable[hash] != NULL && strcmp(set->hashTable[hash]->element, element) == 0) { // Если в хеш-таблице есть элемент с таким хешем и элемент совпадает

return 1; // Возвращает 1 (истина)

}

return 0; // Возвращает 0 (ложь)

}

void saveToFileSet(Set\* set, const char\* filename, const char\* basename, int\* pos1, int\* pos2, int\* status) {

FILE\* file = fopen(filename, "r");

FILE\* tempFile = fopen("temp.data", "w");

int ch;

fseek(file, 0, SEEK\_SET);

fseek(tempFile, 0, SEEK\_SET);

char\*\* elements = (char\*\*)malloc(set->size \* sizeof(char\*));

Node\* current = set->head;

int i = 0;

while (current != NULL) {

elements[i] = current->element;

current = current->next;

i++;

}

while ((ch = fgetc(file)) != EOF) {

fputc(ch, tempFile);

if (ftell(tempFile) == \*pos1 - 2 && \*status == 2)

fprintf(tempFile, "\t%s", set->head->element);

else if (ftell(tempFile) == \*pos1) {

for (int j = set->size - 1; j >= 0; j--) {

fprintf(tempFile, "%s", elements[j]);

if (j > 0)

fprintf(tempFile, "\t");

else

fprintf(tempFile, "\n");

}

if (\*status == 1) {

fseek(tempFile, \*pos1 - 1, SEEK\_SET);

fprintf(tempFile, "\n");

}

fseek(file, \*pos2, SEEK\_SET);

}

}

free(elements);

free(set->hashTable);

free(set->emptySlots);

free(set);

fclose(file);

fclose(tempFile);

remove("2.data");

rename("temp.data", "2.data");

}

Set\* loadFromFileSet(const char\* filename, const char\* basename, int\* pos1, int\* pos2, int\* status) {

FILE\* file = fopen(filename, "r");

if (file == NULL) {

return NULL;

}

int num\_lines = countWordsInFileSet(filename);

char\*\* line = malloc(num\_lines \* sizeof(char\*));

for (int i = 0; i < num\_lines; i++) line[i] = malloc(10000 \* sizeof(char));

Set\* set = initSet();

int tempory = 0;

int pos3 = 0;

int temp1 = 0;

int temp2 = 0;

char c = '1';

for (int i = 0; i < num\_lines; ++i) {

fscanf(file, "%s", line[i]);

c = getc(file);

pos3 = ftell(file);

if (!strcmp(line[i], basename)) {

fseek(file, -3 - strlen(line[i]), SEEK\_CUR);

if (getc(file) == '\n' || i == 0) {

fseek(file, pos3, SEEK\_SET);

tempory = 1;

\*pos1 = ftell(file);

temp1 = i + 1;

}

else fseek(file, pos3, SEEK\_SET);

}

if (c == '\n' && tempory == 1) {

temp2 = i;

\*pos2 = ftell(file);

tempory = 0;

}

if (feof(file))

break;

}

if (temp1 == temp2)

\*status = 1;

if (temp1 == temp2 + 1)

\*status = 2;

while (temp1 < temp2 + 1) {

SADD(set, line[temp1]);

temp1++;

}

fclose(file);

for (int i = 0; i < num\_lines; i++) {

free(line[i]);

}

free(line);

return set;

}

==========================================================

set.h

==========================================================

#ifndef SET\_H

#define SET\_H

typedef struct Node {

char\* element;

int hash;

struct Node\* next;

struct Node\* prev;

} Node;

typedef struct Set {

Node\* head;

int size;

Node\*\* hashTable;

int tableSize;

int\* emptySlots;

} Set;

Set\* initSet();

void SADD(Set\* set, char\* element);

void SREM(Set\* set, char\* element);

int SISMEMBER(Set\* set, char\* element);

void saveToFileSet(Set\* set, const char\* filename, const char\* basename, int \*pos1, int \*pos2, int \*status);

Set\* loadFromFileSet(const char\* filename, const char\* basename, int \*pos1, int \*pos2, int \*status);

#endif

==========================================================

stack.c

==========================================================

#define \_CRT\_SECURE\_NO\_WARNINGS // Отключает предупреждения о небезопасных функциях CRT

#include "stack.h" // Подключает заголовочный файл для работы со стеком

#include <stdio.h> // Подключает стандартную библиотеку ввода-вывода

#include <stdlib.h> // Подключает библиотеку, содержащую функции для работы с памятью, процессами и преобразованиями

#include <string.h> // Подключает библиотеку для работы со строками

int countWordsInFileStack(const char\* filename) { // Функция для подсчета слов в файле

FILE\* file = fopen(filename, "r"); // Открывает файл для чтения

int count = 0; // Переменная для подсчета слов

char word[10000]; // Массив для хранения слов

while (fscanf(file, "%s", word) != EOF) { // Цикл для чтения слов из файла

count++; // Увеличивает счетчик слов

}

fclose(file); // Закрывает файл

return count; // Возвращает количество слов в файле

}

Stack\* initStack() { // Функция для инициализации стека

Stack\* stack = (Stack\*)malloc(sizeof(Stack)); // Выделяет память под структуру стека

stack->top = NULL; // Устанавливает указатель на вершину стека в NULL

return stack; // Возвращает указатель на стек

}

void SPUSH(Stack\* stack, char\* element) { // Функция для добавления элемента в стек

NodeStack\* newNode = (NodeStack\*)malloc(sizeof(NodeStack)); // Выделяет память под новый узел стека

newNode->element = \_strdup(element); // Копирует элемент в новый узел

newNode->next = stack->top; // Устанавливает указатель на следующий узел в вершину стека

stack->top = newNode; // Устанавливает новый узел как вершину стека

}

char\* SPOP(Stack\* stack) { // Функция для удаления элемента из стека

if (stack->top == NULL) { // Если стек пуст

return ("The stack is empty"); // Возвращает сообщение об ошибке

}

NodeStack\* poppedNode = stack->top; // Сохраняет указатель на узел, который нужно удалить

stack->top = poppedNode->next; // Перемещает указатель на вершину стека на следующий узел

char\* element = poppedNode->element; // Сохраняет элемент, который нужно удалить

free(poppedNode); // Освобождает память, занятую узлом

return element; // Возвращает удаленный элемент

}

void saveToFileStack(Stack\* stack, const char\* filename, const char\* basename, int\* pos1, int\* pos2, int\* status) {

FILE\* file = fopen(filename, "r");

FILE\* tempFile = fopen("temp.data", "w");

int ch;

fseek(file, 0, SEEK\_SET);

fseek(tempFile, 0, SEEK\_SET);

while ((ch = fgetc(file)) != EOF) {

fputc(ch, tempFile);

if (ftell(tempFile) == \*pos1 - 2 && \*status == 2)

fprintf(tempFile, "\t%s", stack->top->element);

else if (ftell(tempFile) == \*pos1) {

NodeStack\* currentNode = stack->top;

NodeStack\* prevNode = NULL;

while (currentNode != NULL) {

NodeStack\* nextNode = currentNode->next;

currentNode->next = prevNode;

prevNode = currentNode;

currentNode = nextNode;

}

currentNode = prevNode;

while (currentNode != NULL) {

if (currentNode->next == NULL)

fprintf(tempFile, "%s\n", currentNode->element);

else

fprintf(tempFile, "%s\t", currentNode->element);

currentNode = currentNode->next;

}

if (\*status == 1) {

fseek(tempFile, \*pos1 - 1, SEEK\_SET);

fprintf(tempFile, "\n");

}

fseek(file, \*pos2, SEEK\_SET);

}

}

free(stack->top);

free(stack);

fclose(file);

fclose(tempFile);

remove(filename);

rename("temp.data", filename);

}

Stack\* loadFromFileStack(const char\* filename, const char\* basename, int\* pos1, int\* pos2, int\* status) {

FILE\* file = fopen(filename, "r");

if (file == NULL) {

return NULL;

}

int num\_lines = countWordsInFileStack(filename);

char\*\* line = malloc(num\_lines \* sizeof(char\*));

for (int i = 0; i < num\_lines; i++) line[i] = malloc(10000 \* sizeof(char));

Stack\* stack = initStack();

int tempory = 0;

int pos3 = 0;

int temp1 = 0;

int temp2 = 0;

char c = '1';

for (int i = 0; i < num\_lines; ++i) {

fscanf(file, "%s", line[i]);

c = getc(file);

pos3 = ftell(file);

if (!strcmp(line[i], basename)) {

fseek(file, -3 - strlen(line[i]), SEEK\_CUR);

if (getc(file) == '\n' || i == 0) {

fseek(file, pos3, SEEK\_SET);

tempory = 1;

\*pos1 = ftell(file);

temp1 = i + 1;

}

else fseek(file, pos3, SEEK\_SET);

}

if (c == '\n' && tempory == 1) {

temp2 = i;

\*pos2 = ftell(file);

tempory = 0;

}

if (feof(file))

break;

}

if (temp1 == temp2)

\*status = 1;

if (temp1 == temp2 + 1)

\*status = 2;

while (temp1 < temp2 + 1) {

SPUSH(stack, line[temp1]);

temp1++;

}

fclose(file);

for (int i = 0; i < num\_lines; i++) {

free(line[i]);

}

free(line);

return stack;

}

==========================================================

stack.h

==========================================================

#ifndef STACK\_H

#define STACK\_H

typedef struct NodeStack {

char\* element;

struct NodeStack\* next;

} NodeStack;

typedef struct Stack {

NodeStack\* top;

} Stack;

Stack\* initStack();

void SPUSH(Stack\* stack, char\* element);

char\* SPOP(Stack\* stack);

void saveToFileStack(Stack\* stack, const char\* filename, const char\* basename, int \*pos1, int \*pos2, int \*status);

Stack\* loadFromFileStack(const char\* filename, const char\* basename, int \*pos1, int \*pos2, int \*status);

#endif

==========================================================

table.c

==========================================================

#define \_CRT\_SECURE\_NO\_WARNINGS // Отключает предупреждения о небезопасных функциях CRT

#include "table.h" // Подключает заголовочный файл для работы с хеш-таблицей

#include <stdio.h> // Подключает стандартную библиотеку ввода-вывода

#include <stdlib.h> // Подключает библиотеку, содержащую функции для работы с памятью, процессами и преобразованиями

#include <string.h> // Подключает библиотеку для работы со строками

#define MAX\_SIZE 100000 // Определяет максимальный размер хеш-таблицы

int countWordsInFileTable(const char\* filename) { // Функция для подсчета слов в файле

FILE\* file = fopen(filename, "r"); // Открывает файл для чтения

int count = 0; // Переменная для подсчета слов

char word[10000]; // Массив для хранения слов

while (fscanf(file, "%s", word) != EOF) { // Цикл для чтения слов из файла

count++; // Увеличивает счетчик слов

}

fclose(file); // Закрывает файл

return count; // Возвращает количество слов в файле

}

HashTable\* initHashTable() { // Функция для инициализации хеш-таблицы

HashTable\* ht = (HashTable\*)malloc(sizeof(HashTable)); // Выделяет память под структуру хеш-таблицы

ht->head = NULL; // Устанавливает указатель на голову хеш-таблицы в NULL

ht->size = 0; // Устанавливает размер хеш-таблицы в 0

ht->tableSize = MAX\_SIZE; // Устанавливает размер таблицы в MAX\_SIZE

ht->hashTable = (NodeHashTable\*\*)malloc(MAX\_SIZE \* sizeof(NodeHashTable\*)); // Выделяет память под хеш-таблицу

ht->keys = (char\*\*)malloc(MAX\_SIZE \* sizeof(char\*)); // Выделяет память под массив ключей

for (int i = 0; i < MAX\_SIZE; i++) { // Цикл для инициализации хеш-таблицы и массива ключей

ht->hashTable[i] = NULL; // Устанавливает каждый элемент хеш-таблицы в NULL

ht->keys[i] = NULL; // Устанавливает каждый элемент массива ключей в NULL

}

return ht; // Возвращает указатель на хеш-таблицу

}

int calculateHashT(const char\* element) { // Функция для вычисления хеша элемента

int hash = 0; // Переменная для хранения хеша

for (int i = 0; element[i] != '\0'; i++) { // Цикл для обработки каждого символа элемента

hash = 31 \* hash + element[i]; // Вычисляет хеш

}

return abs(hash) % MAX\_SIZE; // Возвращает абсолютное значение хеша, ограниченное максимальным размером

}

void HSET(HashTable\* ht, char\* key, char\* value) { // Функция для добавления пары ключ-значение в хеш-таблицу

int hash = calculateHashT(key); // Вычисляет хеш ключа

if (ht->hashTable[hash] != NULL) { // Если в хеш-таблице уже есть элемент с таким хешем

return; // Возвращает управление

}

NodeHashTable\* newNode = (NodeHashTable\*)malloc(sizeof(NodeHashTable)); // Выделяет память под новый узел хеш-таблицы

newNode->element = \_strdup(value); // Копирует значение в новый узел

newNode->hash = hash; // Сохраняет хеш в новом узле

newNode->next = ht->head; // Устанавливает указатель на следующий узел в голову хеш-таблицы

if (ht->head != NULL) { // Если хеш-таблица не пуста

ht->head->prev = newNode; // Устанавливает указатель на предыдущий узел головы хеш-таблицы в новый узел

}

ht->head = newNode; // Устанавливает новый узел как голову хеш-таблицы

ht->hashTable[hash] = newNode; // Сохраняет новый узел в хеш-таблице

ht->size++; // Увеличивает размер хеш-таблицы

ht->keys[ht->size - 1] = \_strdup(key); // Сохраняет ключ в массиве ключей

}

char\* HGET(HashTable\* ht, const char\* key) { // Функция для получения значения по ключу из хеш-таблицы

if (ht->hashTable[calculateHashT(key)] != NULL) { // Если в хеш-таблице есть элемент с таким хешем

return ht->hashTable[calculateHashT(key)]->element; // Возвращает значение этого элемента

}

return NULL; // Возвращает NULL, если элемента с таким хешем нет

}

void HDEL(HashTable\* ht, const char\* key) { // Функция для удаления элемента из хеш-таблицы

if (ht->hashTable[calculateHashT(key)] != NULL) { // Если в хеш-таблице есть элемент с таким хешем

NodeHashTable\* nodeToRemove = ht->hashTable[calculateHashT(key)]; // Сохраняет указатель на узел, который нужно удалить

if (nodeToRemove == ht->head) { // Если узел является головой хеш-таблицы

ht->head = nodeToRemove->next; // Перемещает указатель на голову хеш-таблицы на следующий узел

}

else { // Если узел не является головой хеш-таблицы

if (nodeToRemove->prev != NULL) { // Если у узла есть предыдущий узел

nodeToRemove->prev->next = nodeToRemove->next; // Перемещает указатель на следующий узел предыдущего узла на следующий узел удаляемого узла

}

}

if (nodeToRemove->next != NULL) { // Если у узла есть следующий узел

nodeToRemove->next->prev = nodeToRemove->prev; // Перемещает указатель на предыдущий узел следующего узла на предыдущий узел удаляемого узла

}

free(nodeToRemove->element); // Освобождает память, занятую элементом удаляемого узла

free(nodeToRemove); // Освобождает память, занятую узлом

ht->hashTable[calculateHashT(key)] = NULL; // Устанавливает элемент хеш-таблицы с хешем удаляемого элемента в NULL

ht->size--; // Уменьшает размер хеш-таблицы

return; // Возвращает управление

}

}

void saveToFileTable(HashTable\* hashtable, const char\* filename, const char\* basename, int\* pos1, int\* pos2, int\* status) {

FILE\* file = fopen(filename, "r");

FILE\* tempFile = fopen("temp.data", "w");

int ch;

fseek(file, 0, SEEK\_SET);

fseek(tempFile, 0, SEEK\_SET);

while ((ch = fgetc(file)) != EOF) {

fputc(ch, tempFile);

if (ftell(tempFile) == \*pos1 - 2 && \*status == 2) {

fprintf(tempFile, "\t%s\t%s", hashtable->hashTable[calculateHashT(hashtable->keys[0])]->element, hashtable->keys[0]);

}

else if (ftell(tempFile) == \*pos1) {

for (int i = 0; i < hashtable->size; i++) {

if (i == hashtable->size - 1) {

fprintf(tempFile, "%s\t%s\n", hashtable->hashTable[calculateHashT(hashtable->keys[i])]->element, hashtable->keys[i]);

}

else {

fprintf(tempFile, "%s\t%s\t", hashtable->hashTable[calculateHashT(hashtable->keys[i])]->element, hashtable->keys[i]);

}

}

if (\*status == 1) {

fseek(tempFile, \*pos1 - 1, SEEK\_SET);

fprintf(tempFile, "\n");

}

fseek(file, \*pos2, SEEK\_SET);

}

}

free(hashtable->hashTable);

free(hashtable->keys);

free(hashtable);

fclose(file);

fclose(tempFile);

remove(filename);

rename("temp.data", filename);

}

HashTable\* loadFromFileTable(const char\* filename, const char\* basename, int\* pos1, int\* pos2, int\* status) {

FILE\* file = fopen(filename, "r");

int num\_lines = countWordsInFileTable(filename);

char\*\* line = malloc(num\_lines \* sizeof(char\*));

for (int i = 0; i < num\_lines; i++) line[i] = malloc(10000 \* sizeof(char));

HashTable\* hashtable = initHashTable();

int tempory = 0;

int tempory2 = 0;

int count = 0;

int temp1 = 0;

int temp2 = 0;

char c = '1';

for (int i = 0; i < num\_lines; ++i) {

fscanf(file, "%s", line[i]);

c = getc(file);

if (c == '\n') {

tempory2 = ftell(file);

}

if (!strcmp(line[i], basename) && (tempory2 == ftell(file) || tempory2 == ftell(file) - strlen(line[i]) - 1 || i == 0)) {

tempory = 1;

\*pos1 = ftell(file);

\*pos2 = strlen(line[i]);

temp1 = i + 1;

}

if (c == '\n' && tempory == 1) {

temp2 = i;

\*pos2 = ftell(file);

tempory = 0;

count++;

}

if (feof(file))

break;

}

if (temp1 + 1 == temp2) \*status = 1;

if (temp1 == temp2 + 1) \*status = 2;

while (temp1 < temp2) {

char\* value = line[temp1];

char\* key = line[temp1 + 1];

HSET(hashtable, key, value);

temp1 += 2;

}

fclose(file);

for (int i = 0; i < num\_lines; i++) {

free(line[i]);

}

free(line);

return hashtable;

}

==========================================================

table.h

==========================================================

#ifndef TABLE\_H

#define TABLE\_H

typedef struct NodeHashTable {

char\* element;

int hash;

struct NodeHashTable\* next;

struct NodeHashTable\* prev;

} NodeHashTable;

typedef struct HashTable {

NodeHashTable\* head;

int size;

NodeHashTable\*\* hashTable;

int tableSize;

char\*\* keys;

} HashTable;

HashTable\* initHashTable();

void HSET(HashTable\* hashtable, char\* key, char\* value);

void HDEL(HashTable\* hashtable, char\* key);

char\* HGET(HashTable\* hashtable, char\* key);

void saveToFileTable(HashTable\* hashtable, const char\* filename, const char\* basename, int \*pos1, int \*pos2, int \*status);

HashTable\* loadFromFileTable(const char\* filename, const char\* basename, int \*pos1, int \*pos2, int \*status);

#endif

==========================================================

Python

==========================================================

# Импортируем необходимые библиотеки

import string

import random

import socket

from flask import Flask, request, redirect

from flask\_cors import CORS

import time

import requests

import datetime

# Создаем экземпляр приложения Flask

app = Flask(\_\_name\_\_)

# Разрешаем кросс-доменные запросы

CORS(app)

# Определяем маршрут для формы

@app.route('/form', methods=['GET'])

def form():

# Возвращаем HTML-форму для ввода URL

return '''

<!DOCTYPE html>

<html>

<body>

<h2>Сокращения URL</h2>

<form id="myForm">

<label for="url">Введите URL:</label><br>

<input type="text" id="url" name="url"><br>

<input type="submit" value="Сократить URL">

</form>

<p id="shortUrl"></p>

<script>

document.getElementById('myForm').addEventListener('submit', function(e) {

e.preventDefault();

var url = document.getElementById('url').value;

if(url.trim() === '') {

document.getElementById('shortUrl').innerText = 'Пожалуйста, введите URL';

return;

}

if(new Blob([url]).size > 50000) {

document.getElementById('shortUrl').innerText = 'URL не должен превышать 50000 байт';

return;

}

fetch('/', {

method: 'POST',

headers: {

'Content-Type': 'text/plain'

},

body: url

}).then(function(response) {

if (!response.ok) {

throw new Error('Ошибка сети при попытке сократить URL');

}

return response.text();

}).then(function(text) {

if (text === 'Error.') {

document.getElementById('shortUrl').innerText = 'Произошла ошибка при обработке вашего запроса';

} else {

document.getElementById('shortUrl').innerText = 'Ваш сокращенный URL: ' + text;

}

}).catch(function(error) {

document.getElementById('shortUrl').innerText = 'Произошла ошибка: ' + error.message;

});

});

</script>

</body>

</html>

'''

# Функция для генерации короткого URL

def generate\_short\_url():

return ''.join(random.choices(string.ascii\_letters + string.digits, k=8))

# Определяем маршрут для сокращения URL

@app.route('/', methods=['POST'])

def shorten\_url():

# Получаем URL из запроса

url = request.data.decode()

# Проверка на повтор и возврат имеющего значения

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

try:

s.connect((SERVER\_IP, SERVER\_PORT))

except Exception as e:

return f'Не удалось подключиться к серверу: {str(e)}', 500

s.sendall(f'--file urls.data --query HGET short\_urls {url}'.encode())

existing\_url = s.recv(1024).decode()

existing\_url = existing\_url.replace('\n', '').replace('\r', '')

if existing\_url == "Error.":

return 'Error.', 500

if existing\_url != "-> False":

return f'{request.url\_root}SU/{existing\_url}'

while True:

# Генерируем короткий URL

short\_url = generate\_short\_url()

# Создаем сокет для связи с сервером

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

try:

# Подключаемся к серверу

s.connect((SERVER\_IP, SERVER\_PORT))

except Exception as e:

# Если возникла ошибка, возвращаем ее

return f'Не удалось подключиться к серверу: {str(e)}', 500

# Отправляем запрос на сервер

s.sendall(f'--file urls.data --query HGET urls {short\_url}'.encode())

# Получаем ответ от сервера

existing\_url = s.recv(1024).decode()

existing\_url = existing\_url.replace('\n', '').replace('\r', '')

# Если возникла ошибка, возвращаем ее

if existing\_url == "Error.":

return 'Error.', 500

# Если URL не существует, создаем новый

if existing\_url == "-> False":

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

try:

# Подключаемся к серверу

s.connect((SERVER\_IP, SERVER\_PORT))

except Exception as e:

# Если возникла ошибка, возвращаем ее

return f'Не удалось подключиться к серверу: {str(e)}', 500

# Отправляем запрос на сервер

s.sendall(f'--file urls.data --query HSET urls {url} {short\_url}'.encode())

# Если возникла ошибка, возвращаем ее

if s.recv(1024).decode().replace('\n', '').replace('\r', '') == "Error.":

return 'Error.', 500

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

try:

s.connect((SERVER\_IP, SERVER\_PORT))

except Exception as e:

return f'Не удалось подключиться к серверу: {str(e)}', 500

s.sendall(f'--file urls.data --query HSET short\_urls {short\_url} {url}'.encode())

if s.recv(1024).decode().replace('\n', '').replace('\r', '') == "Error.":

return 'Error.', 500

# Возвращаем сокращенный URL

return f'{request.url\_root}SU/{short\_url}', 201

# Определяем маршрут для перенаправления по короткому URL

@app.route('/SU/<short\_url>', methods=['GET'])

def redirect\_url(short\_url):

# Создаем сокет для связи с сервером

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

try:

# Подключаемся к серверу

s.connect((SERVER\_IP, SERVER\_PORT))

except Exception as e:

# Если возникла ошибка, возвращаем ее

return f'Не удалось подключиться к серверу: {str(e)}', 500

# Отправляем запрос на сервер

s.sendall(f'--file urls.data --query HGET urls {short\_url}'.encode())

# Получаем ответ от сервера

url = s.recv(1024).decode()

url = url.replace('\n', '').replace('\r', '')

# Если возникла ошибка, возвращаем ее

if url == "Error.":

return 'Error.', 500

# Перенаправляем на исходный URL

return redirect(url, code=302)

# Запускаем приложение

if \_\_name\_\_ == '\_\_main\_\_':

# Получаем IP-адрес и порт сервера

SERVER\_IP, SERVER\_PORT = input("Введите IP-адрес и порт сервера (формат: myip:myport): ").split(':')

SERVER\_PORT = int(SERVER\_PORT)

# Запускаем приложение в режиме отладки

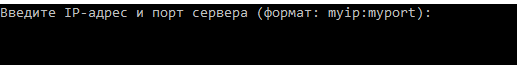
app.run(debug=True)

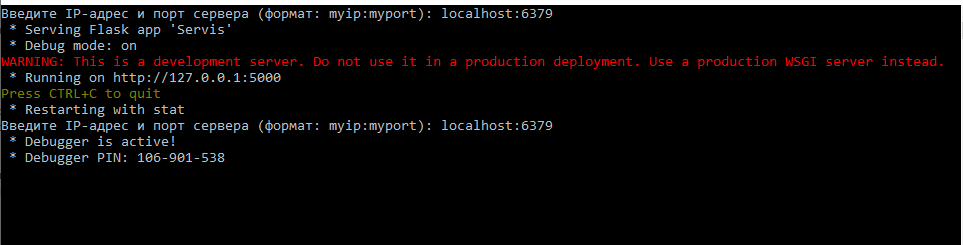
==========================================================**Результат работы программы**

**C**



**Python**





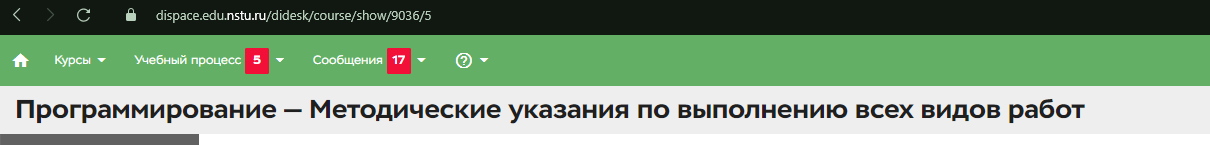
Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание





Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

**Вывод**: Сервис по сокращению ссылок и сам сервер базы данных были проверены на работоспособность посредством подключения, сначала, к локальному хосту, затем запуском двух программ на разных серверах и железе. Для проверки были вызваны добровольцы, которые пытались положить мои сервера в обычном использовании, однако они устояли и продолжили работу. Все условия задания были учтены. Таким образом, программы являются работоспособными, отвечающими всем требованиям заданию.