# ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**Факультет безопасности информационных технологий**

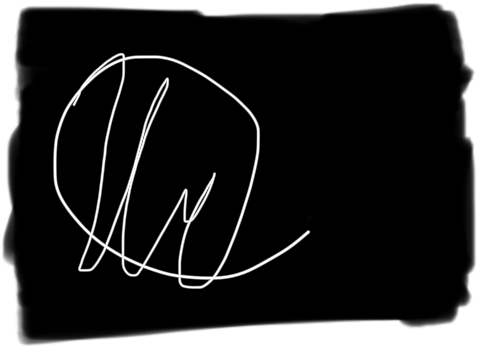
**Дисциплина:**

# «Программирование»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

**Выполнил:**

# Студент группы N3149

Ильчук Денис.



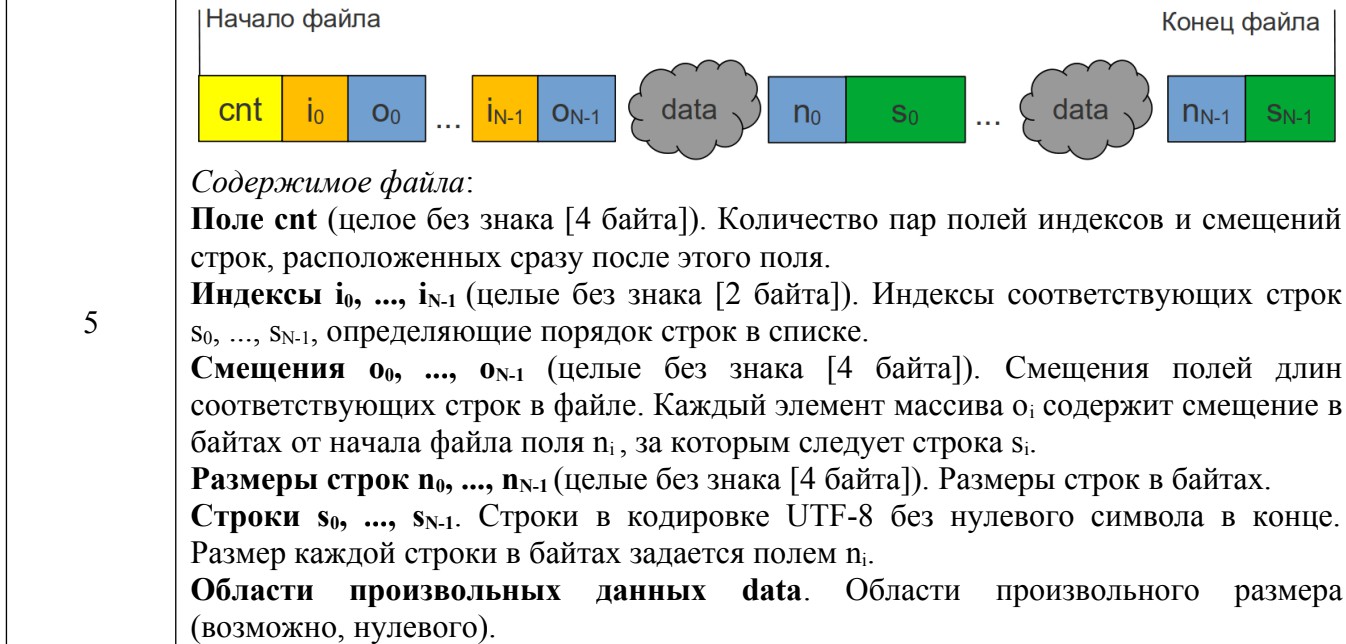
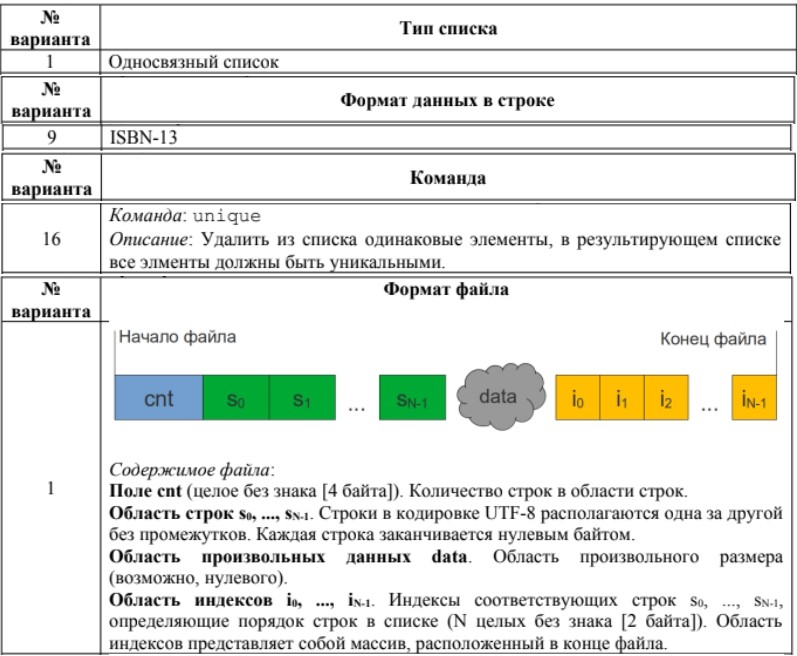
**Проверила:**

|

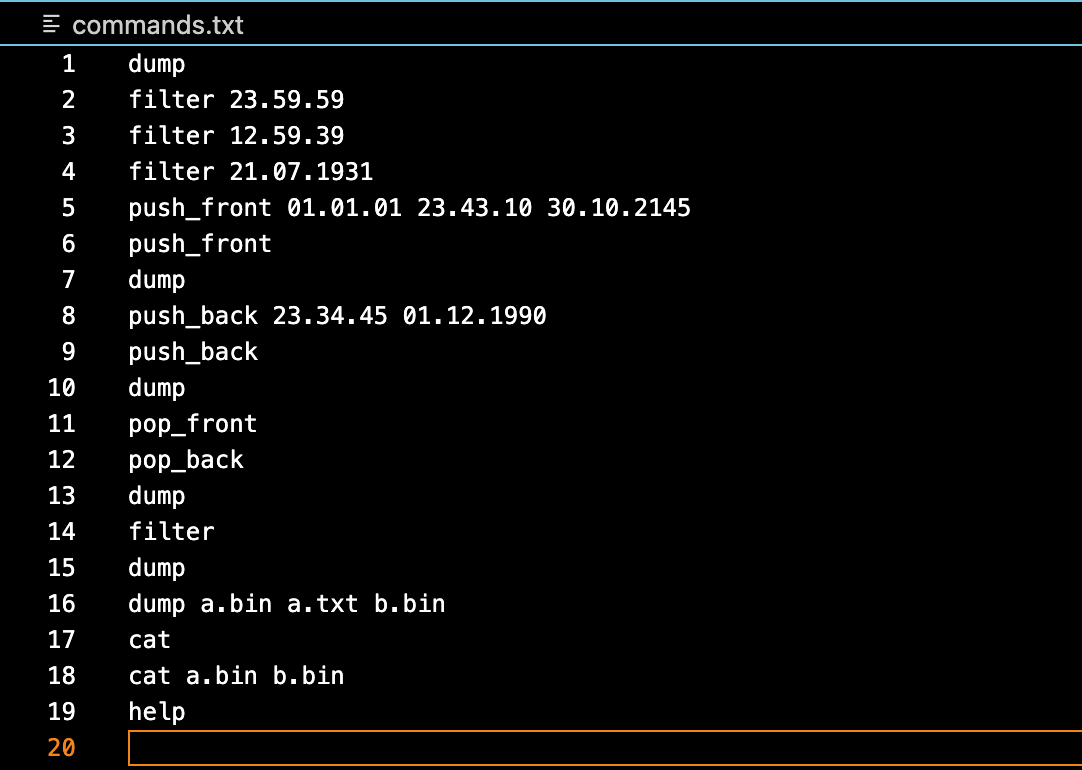
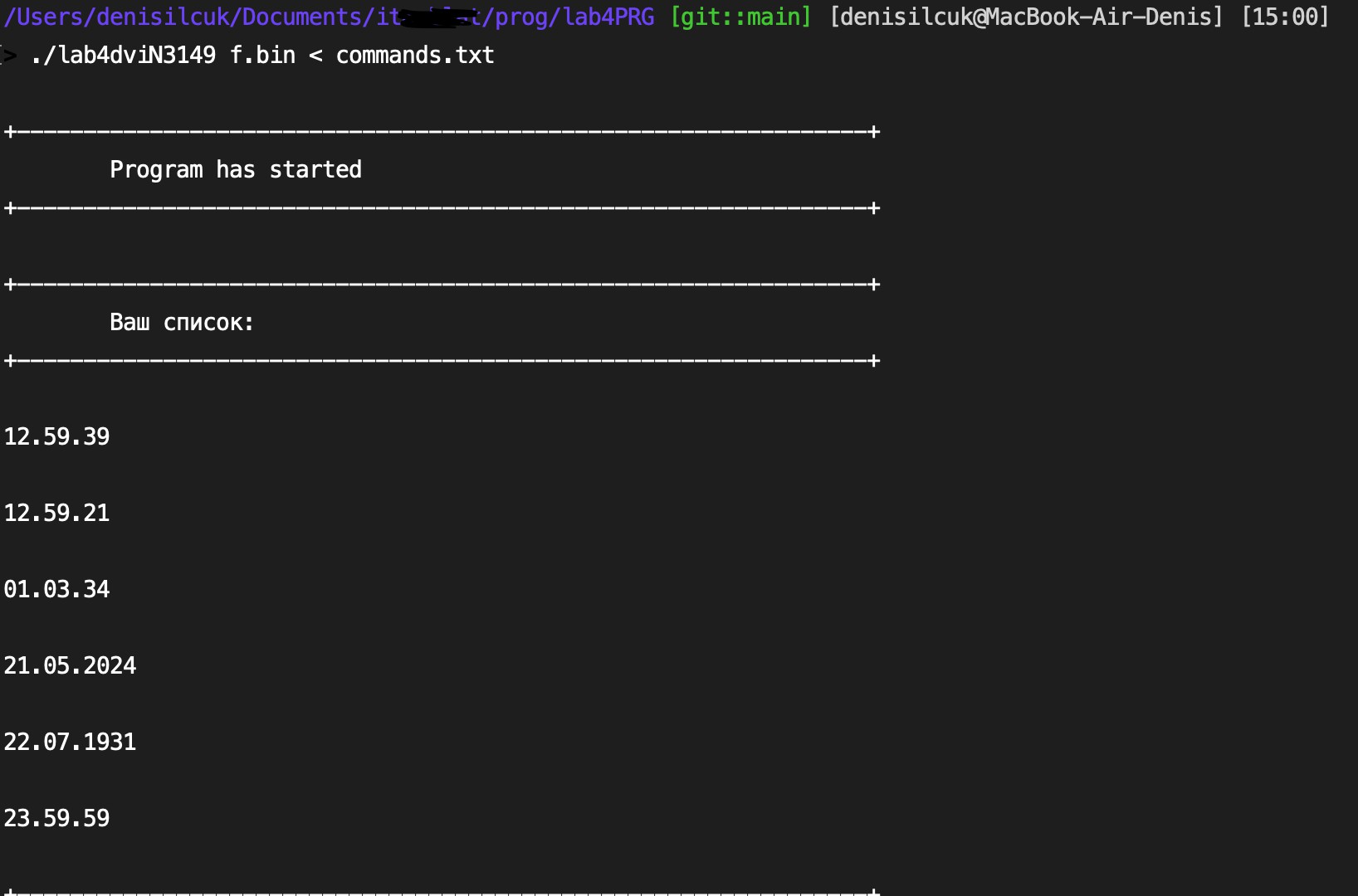
|

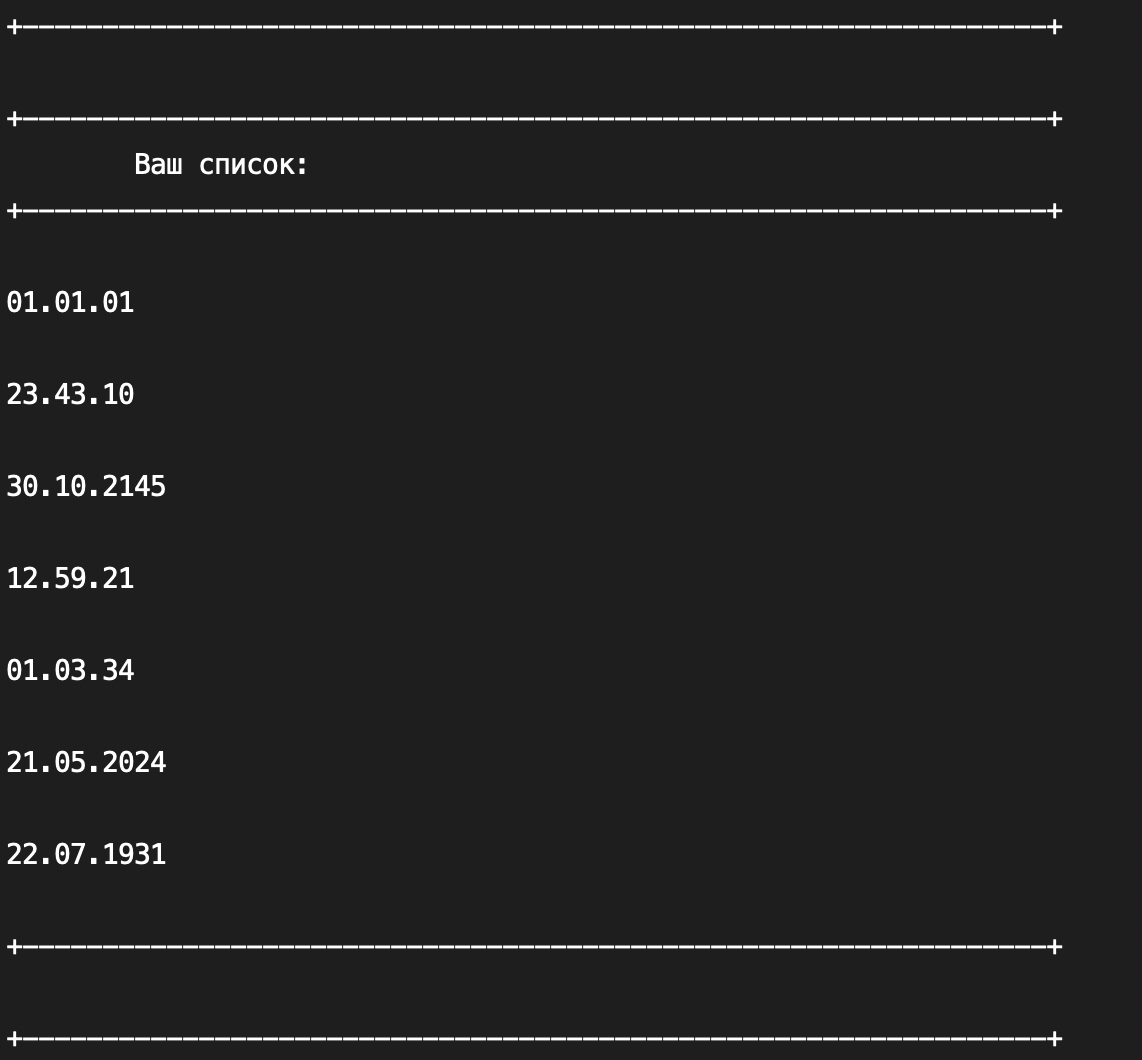
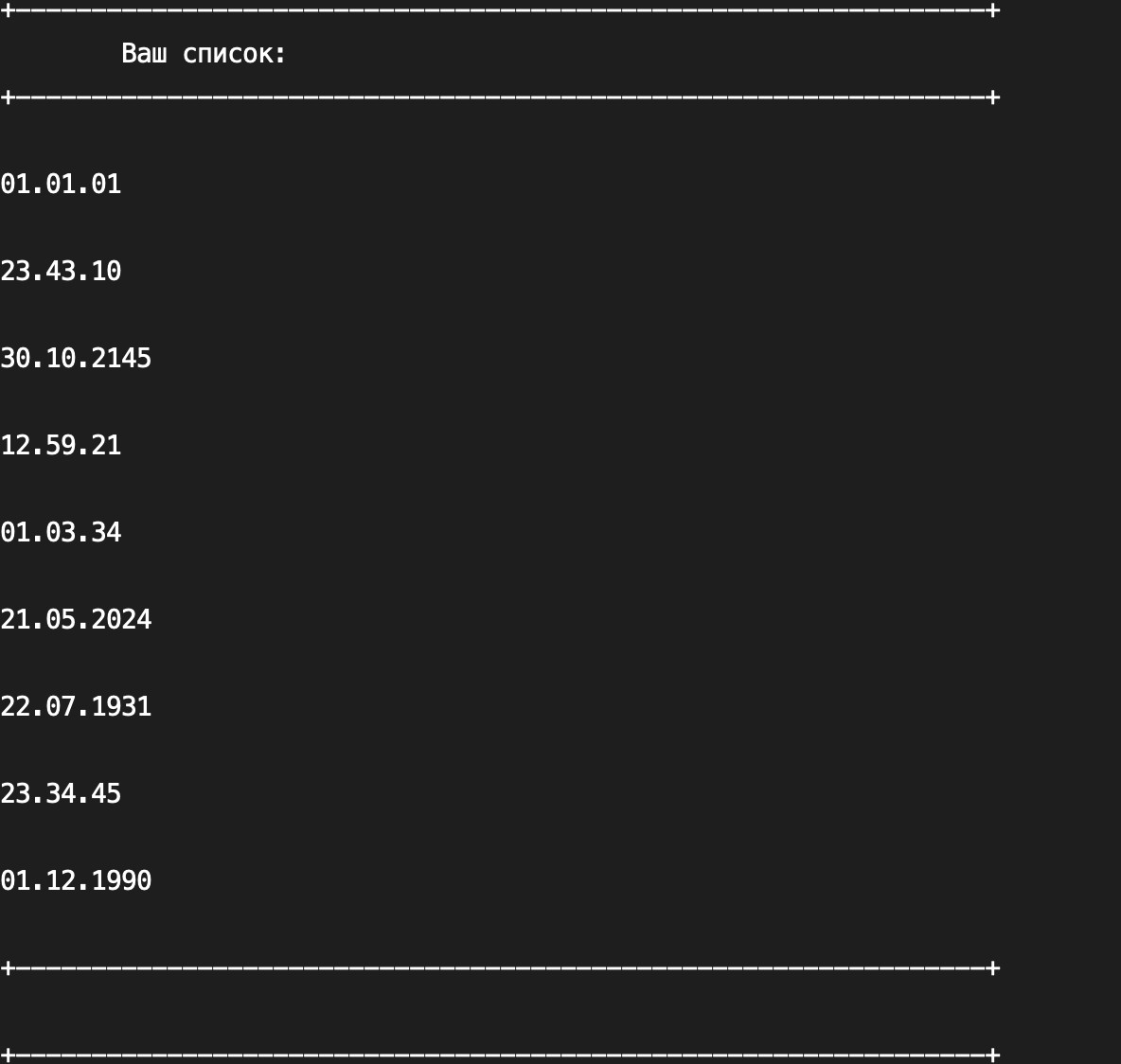
# Горлина А.В.

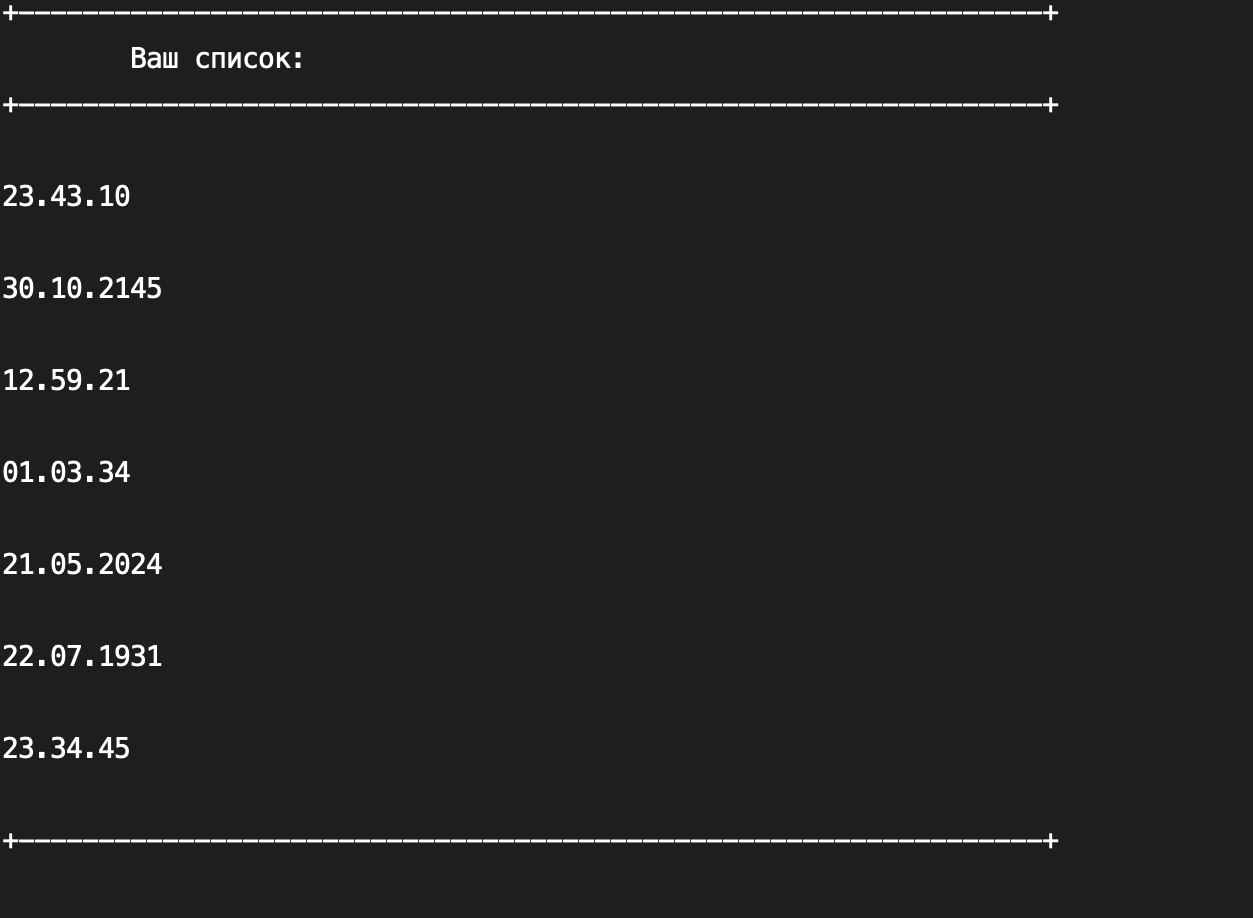
Задание 2-8-2-5

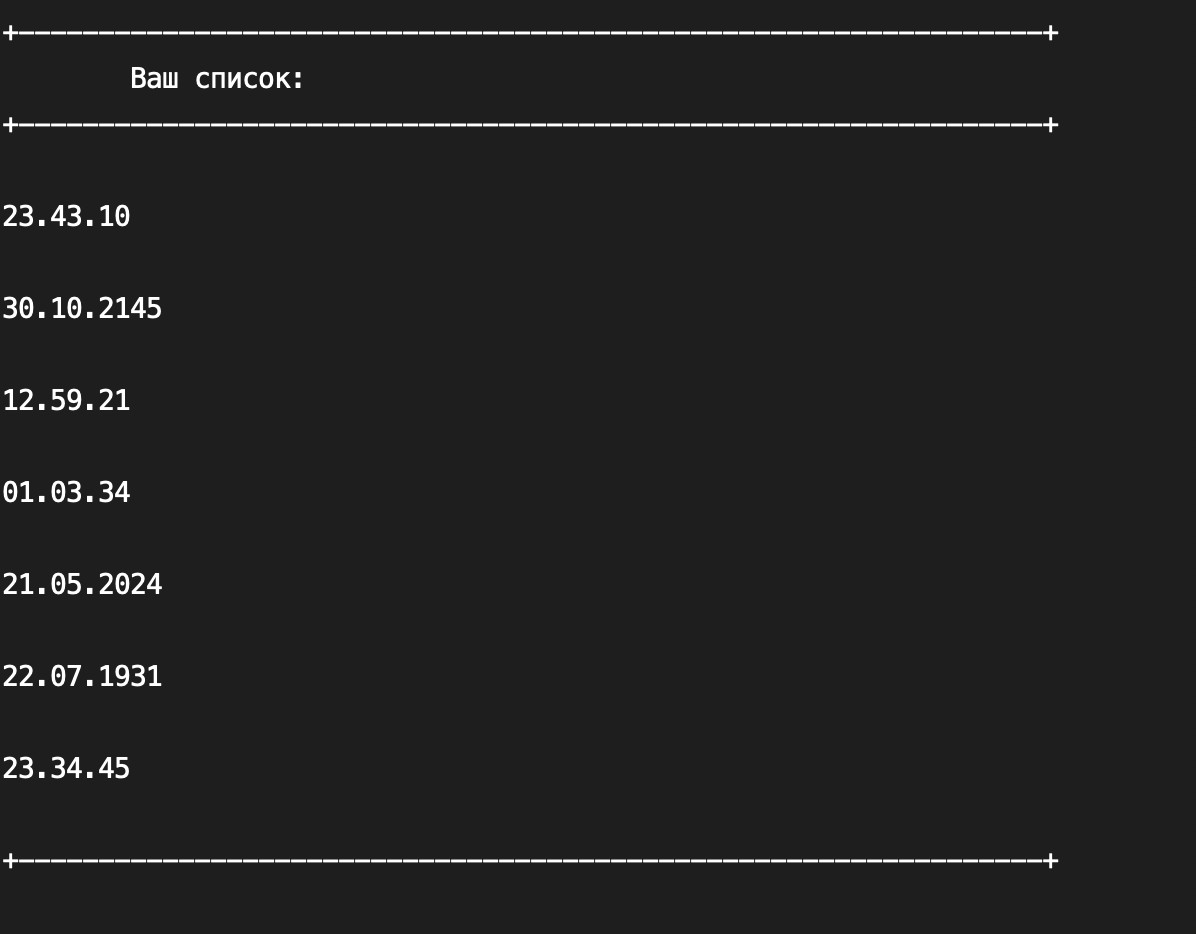


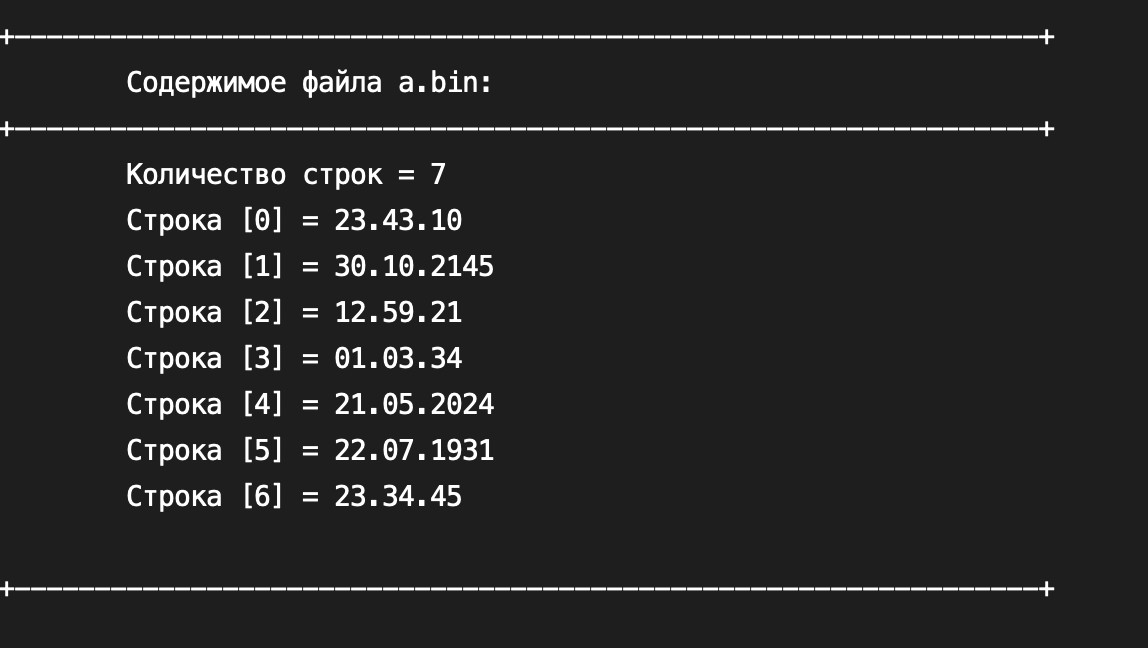
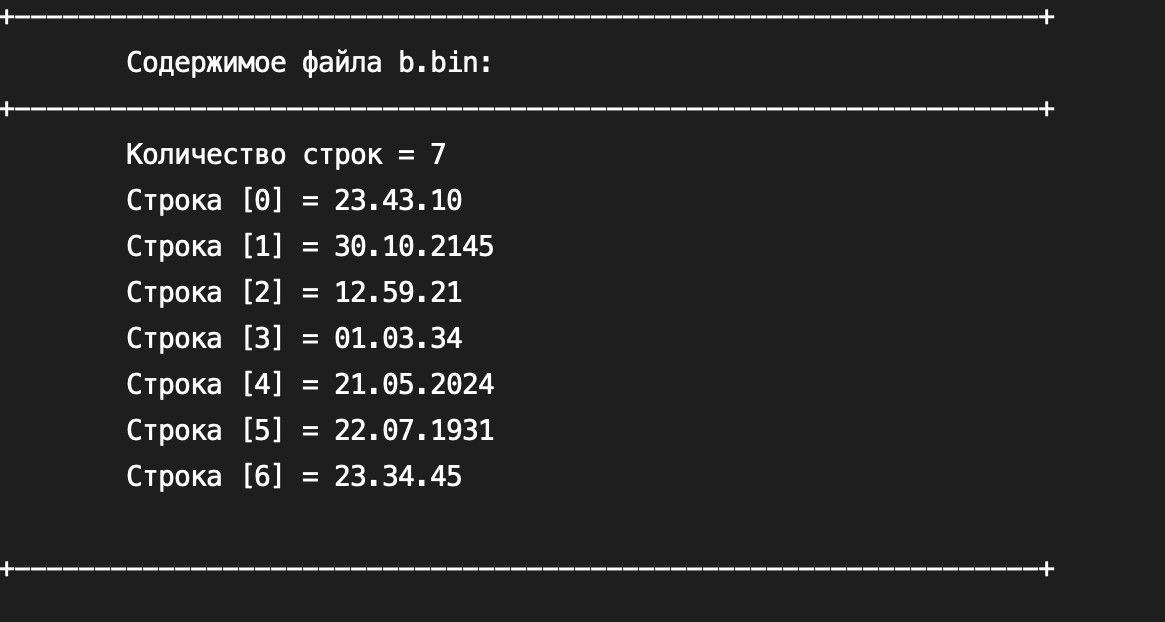
# Примеры работы программы на различных исходных данных (скриншоты)

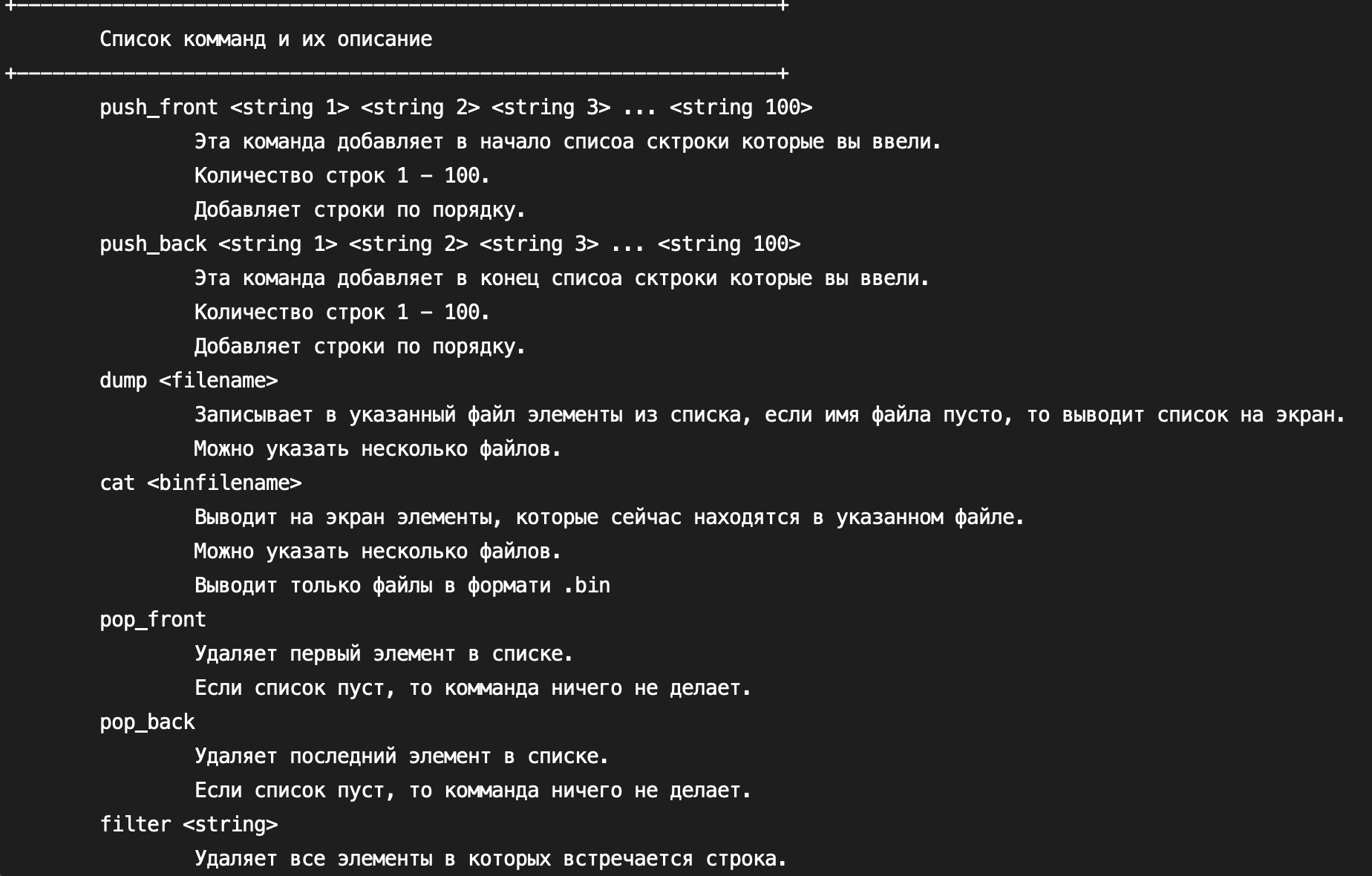
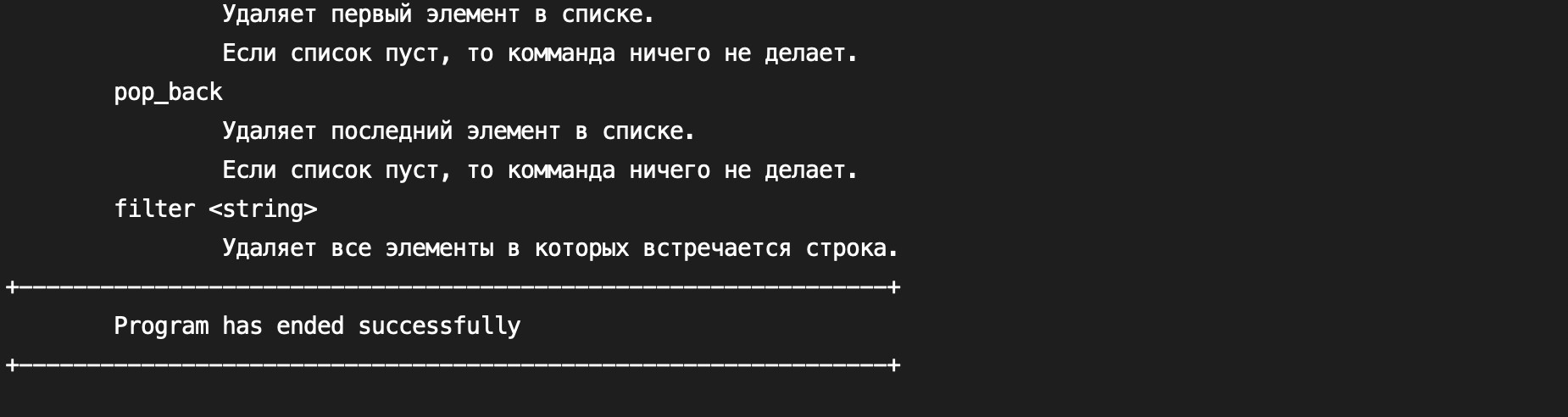












Исходный код программы

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <regex.h>

#include "My\_Lib.h"

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

{

char \*binfilename;

char command[256];

Node \*head = NULL;

if (argc == 2 && strcmp(argv[1], "-v") == 0) //ну тут понятно

{

printf("%s", "\tДенис Ильчук Витальевич, гр. N3149\n\tВариант: 2-8-2-5\n");

return 0;

}

else if (argc == 2 && (Is\_Valid\_File\_Name(argv[1]) == 0))

{

Start();

binfilename = argv[1];

}

else

{

Errors(1);

}

Read\_And\_Append\_To\_List(binfilename, &head); // сразу и список создается и в него из файла читается

while (1)//цикл для того чтобы можно было сколько хочешь команд дать

{

if(fgets(command, sizeof(command), stdin) == NULL) // чтобы control+D работал и с EOF тоже подружить

{

End();

}

remove\_last\_character(command); //удаляет последний \n из комманды

char \*token = strtok(command, " "); // считываем программу

if (strcmp(token, "cat") == 0)//сам добавил для дебага, но решил оставить

{

while (1) // сколько хочешь файлов можно посмотреть

{

token = strtok(NULL, " ");

if(token == NULL)

{

break;

}

Print\_File(token);

}

}

else if (strcmp(token, "push\_front") == 0)// заполняется массив и в обратном порядке добавляется в список

{

char \*array[100] = {NULL};

char \*arg;

int i = 0;

while(1)

{

token = strtok(NULL, " ");

if(token == NULL)

{

break;

}

arg = token;

if (Is\_Valid\_Date\_Time\_Format(arg) == 0)

{

Errors(9);

}

Insert\_String(array, arg, i);

i += 1;

}

for (int j = i - 1; j >= 0; j--)

{

Push\_Front(array[j], &head);

}

Prepare\_List(&head);

}

else if(strcmp(token, "dump") == 0) // я конечно на искосок читал, но посмотреть список и записать в файл можно

{

Prepare\_List(&head);

token = strtok(NULL, " ");

if (token == NULL)

{

Print\_List(head);

}

else

{

while(1)

{

Txt\_Or\_Bin(token, &head);

token = strtok(NULL, " ");

if(token == NULL)

{

break;

}

}

}

}

else if (strcmp(token, "push\_back") == 0)// как push\_front только добавляется последовательно

{

char \*arg;

while(1)

{

token = strtok(NULL, " ");

if(token == NULL)

{

break;

}

arg = token;

if (Is\_Valid\_Date\_Time\_Format(arg) == 0)

{

Errors(9);

}

Push\_Back(arg, &head);

}

Prepare\_List(&head);

}

else if (strcmp(token, "pop\_front") == 0)

{

Remove\_First(&head);

Prepare\_List(&head);

}

else if (strcmp(token, "pop\_back") == 0)

{

Remove\_Last(&head);

Prepare\_List(&head);

}

else if (strcmp(token, "help") == 0)

{

Print\_Help\_Message();

}

else if (strcmp(token, "filter") == 0)

{

token = strtok(NULL, " ");

if(token == NULL)

{

}

else

{

Delete\_Node(token, &head);

}

}

else

{

printf("\n+----------------------------------------------------------------+\n\tНеизвестная комманда. Список комманд доступен при вводе help\n+----------------------------------------------------------------+\n");

}

}

Free\_List(head);

return 0;

}

Файл My\_Lib.h

void Errors(int err)

{

if (err == 0)

{

return;

}

else if (err == 1)

{

printf("Ошибка!:\n\tНеправильные использование, используйте:\n\t ./lab4dviN3149 -v\n\t./lab4dviN3149 fileName.bin\n");

exit(EXIT\_FAILURE);

}

else if(err == 2)

{

printf("Ошибка!:\n\tНе удалось открыть файл\n");

exit(EXIT\_FAILURE);

}

else if(err == 3)

{

printf("Ошибка!:\n\tНе удалось выделить память для нового узла списка\n");

exit(EXIT\_FAILURE);

}

else if(err == 4)

{

printf("Ошибка!:\n\tПустой файл\n");

exit(EXIT\_FAILURE);

}

else if(err == 5)

{

printf("Ошибка!:\n\tФайл поврежден\n");

exit(EXIT\_FAILURE);

}

else if(err == 6)

{

printf("Ошибка!:\n\tНе удалось выделить память\n");

exit(EXIT\_FAILURE);

}

else if(err == 7)

{

printf("Ошибка!:\n\tСлишком много строк\n");

exit(EXIT\_FAILURE);

}

else if(err == 8)

{

printf("Ошибка!:\n\tОшибка компиляции регулярного выражения\n");

exit(EXIT\_FAILURE);

}

else if(err == 9)

{

printf("Ошибка!:\n\tСтроки не соответствуют формату\n");

exit(EXIT\_FAILURE);

}

}

int Is\_Valid\_Date\_Time\_Format(const char \*str)

{

// Регулярные выражения для времени и даты

const char \*time\_regex = "^(0[0-9]|1[0-9]|2[0-3])\\.(0[0-9]|[1-5][0-9])\\.(0[0-9]|[1-5][0-9])$";

const char \*date\_regex = "^(0[1-9]|[12][0-9]|3[01])\\.(0[1-9]|1[0-2])\\.(19[0-9]{2}|20[0-9]{2}|[0-9]{4})$";

regex\_t regex\_time, regex\_date;

int reti\_time, reti\_date;

reti\_time = regcomp(&regex\_time, time\_regex, REG\_EXTENDED);

reti\_date = regcomp(&regex\_date, date\_regex, REG\_EXTENDED);

if (reti\_time || reti\_date)

{

Errors(8);

}

// Проверяем строку на соответствие регулярным выражениям

reti\_time = regexec(&regex\_time, str, 0, NULL, 0);

reti\_date = regexec(&regex\_date, str, 0, NULL, 0);

regfree(&regex\_time);

regfree(&regex\_date);

if (!reti\_time || !reti\_date)

{

return 1;

} else

{

return 0;

}

}

void remove\_last\_character(char \*str)

{

int len = strlen(str);

if (len > 0)

{

str[len - 1] = '\0';

}

}

// Структура для узла двусвязного списка

typedef struct Node

{

int i;

int o;

char \*str;

struct Node \*prev;

struct Node \*next;

} Node;

// функция для создания нового узла

Node\* Create\_Node(int i, int o, const char \*str)

{

Node \*newNode = (Node\*)malloc(sizeof(Node));

if (newNode == NULL)

{

Errors(3);

}

newNode->i = i;

newNode->o = o;

newNode->str = strdup(str); // пока не разобрался как strtok работает 5 лет прошло

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

// Функция для добавления узла в конец списка

void Append\_To\_End(Node \*\*head, int i, int o, const char \*str)

{

Node \*newNode = Create\_Node(i, o, str);

if (\*head == NULL)

{

\*head = newNode;

}

else

{

Node \*temp = \*head;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

}

// такая же для добавления в начало

void Append\_To\_Front(Node \*\*head, int i, int o, const char \*str)

{

Node \*newNode = Create\_Node(i, o, str);

if (\*head == NULL)

{

\*head = newNode;

}

else

{

newNode ->prev = NULL;

newNode ->next = \*head;

\*head = newNode;

}

}

// Функция для считывания данных из бинарного файла и добавления их в список

void Read\_And\_Append\_To\_List(const char \*filename, Node \*\*head)

{

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

if (file == NULL)

{

Errors(2);

}

int n, i, o;

fread(&n, sizeof(int), 1, file); // считываем значение n

if(n <= 0)

{

Errors(4);

}

fseek(file, (n) \* (sizeof(int) + sizeof(int)), SEEK\_CUR);

fseek(file, 100, SEEK\_CUR);

for (int j = 0; j < n; j++)

{

i = j;

o = ftell(file);

int k;

fread(&k, sizeof(int), 1, file); // считываем размер строки

char \*str = (char\*)malloc((k + 1) \* sizeof(char)); // выделяем память для строки

fread(str, sizeof(char), k, file); // считываем саму строку

str[k] = '\0'; // добавляем конец\_строки-символ

if (Is\_Valid\_Date\_Time\_Format(str) == 0)

{

Errors(9);

}

Append\_To\_End(head, i, o, str); // логичнее добавлять в конец

free(str); // освобождаем память, выделенную для строки

}

fclose(file);

}

// Функция для освобождения памяти, выделенной под список

void Free\_List(Node \*head)

{

Node \*temp;

while (head != NULL)

{

temp = head;

head = head->next;

free(temp->str);

free(temp);

}

}

void Print\_List(Node \*head)

{

Node \*current = head;

printf("%s", "\n+----------------------------------------------------------------+\n\tВаш список:\n+----------------------------------------------------------------+\n");

while (current != NULL)

{

printf("\n%s\n", current->str);

current = current->next;

}

printf("%s", "\n+----------------------------------------------------------------+\n");

}

int Is\_Valid\_File\_Name(char \*str)// 1 это да 0 это нет

{

int len = strlen(str);

// Проверяем длину строки

if (len <= 4)

{

return 1;

}

// Проверяем расширение файла

if (strcmp(str + len - 4, ".bin") != 0)

{

return 1;

}

return 0;

}

// скопировал алгоритм из программы которая делает бинарные файлы

void Print\_File(const char \*filename)

{

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

if (file == NULL)

{

Errors(2);

}

int n, k;

fread(&n, sizeof(int), 1, file);

fseek(file, (n) \* (sizeof(int) + sizeof(int)), SEEK\_CUR);

fseek(file, 100, SEEK\_CUR);

printf("\n+----------------------------------------------------------------+\n\tСодержимое файла %s:\n+----------------------------------------------------------------+\n", filename);

printf("\tКоличество строк = %d\n", n);

for (int i = 0; i < n; i++)

{

fread(&k, sizeof(int), 1, file);

char \*str = (char\*)malloc((k + 1) \* sizeof(char));

if (str == NULL)

{

Errors(6);

}

str[k] = '\0';

fread(str, sizeof(char), k, file);

printf("\tСтрока [%d] = %s\n", i, str);

free(str);

}

printf("%s", "\n+----------------------------------------------------------------+\n");

fclose(file);

}

void Start()

{

printf("%s", "\n+----------------------------------------------------------------+\n\tProgram has started\n+----------------------------------------------------------------+\n");

}

void End()

{

printf("%s", "\tProgram has ended successfully\n+----------------------------------------------------------------+\n");

exit(EXIT\_SUCCESS);

}

// веселые функции, которые занимают стэк

void Push\_Front(char \*str, Node \*\*head)

{

Append\_To\_Front(head, -1, -1, str);

}

void Push\_Back(char \*str, Node \*\*head)

{

Append\_To\_End(head, -1, -1, str);

}

void Insert\_String(char \*arr[], const char \*str, int i) // опять же для того, чтобы strtok не поламалась

{

int max\_size = 100;

if (i < 0 || i >= max\_size)

{

Errors(7);

}

if (arr[i] != NULL)

{

free(arr[i]); // Освободить память, если в этом месте уже есть строка

}

arr[i] = (char \*)malloc(strlen(str) + 1); // Выделить память для новой строки

if (arr[i] == NULL)

{

Errors(6);

}

strcpy(arr[i], str); // Копировать строку в массив

}

void Remove\_First(Node \*\*head)

{

if (head == NULL || \*head == NULL)

{

return; // Список пуст или head является NULL

}

Node \*temp = \*head;

\*head = (\*head)->next;

if (\*head != NULL)

{

(\*head)->prev = NULL;

}

free(temp->str);

free(temp);

}

void Remove\_Last(Node \*\*head)

{

if (head == NULL || \*head == NULL)

{

return; // Список пуст или head является NULL

}

Node \*current = \*head;

while (current->next != NULL)

{

current = current->next;

}

if (current->prev != NULL)

{

current->prev->next = NULL;

} else

{

\*head = NULL; // Удаляемый элемент был единственным элементом списка

}

free(current->str);

free(current);

}

void Print\_Help\_Message()

{

printf("\n+----------------------------------------------------------------+\n\tСписок комманд и их описание\n+----------------------------------------------------------------+\n");

printf("\tpush\_front <string 1> <string 2> <string 3> ... <string 100>\n\t\tЭта команда добавляет в начало списоа сктроки которые вы ввели.\n\t\tКоличество строк 1 - 100.\n\t\tДобавляет строки по порядку.\n");

printf("\tpush\_back <string 1> <string 2> <string 3> ... <string 100>\n\t\tЭта команда добавляет в конец списоа сктроки которые вы ввели.\n\t\tКоличество строк 1 - 100.\n\t\tДобавляет строки по порядку.\n");

printf("\tdump <filename>\n\t\tЗаписывает в указанный файл элементы из списка, если имя файла пусто, то выводит список на экран.\n\t\tМожно указать несколько файлов.\n");

printf("\tcat <binfilename>\n\t\tВыводит на экран элементы, которые сейчас находятся в указанном файле.\n\t\tМожно указать несколько файлов.\n\t\tВыводит только файлы в формати .bin\n");

printf("\tpop\_front\n\t\tУдаляет первый элемент в списке.\n\t\tЕсли список пуст, то комманда ничего не делает.\n");

printf("\tpop\_back\n\t\tУдаляет последний элемент в списке.\n\t\tЕсли список пуст, то комманда ничего не делает.\n");

printf("\tfilter <string>\n\t\tУдаляет все элементы в которых встречается строка.");

printf("\n+----------------------------------------------------------------+\n");

}

void Prepare\_List(Node \*\*head)// из-за особенностей того как у меня записывается файл, я просто заного вычисляю номера и положение строк, тем более, что когда новая строка добавляется то мы не знаем ее положение, а если в начало то вообще ужас

{

int n = 0;

Node \*last = \*head;

while (last->next != NULL)

{

last = last->next;

n += 1;

}

n += 1;

int o = 1 + n \* (2 \* sizeof(int)) + 100;

Node \*current = \*head;

for (int i = 0; i < n; i++)

{

current->i = i;

current->o = o;

o += 1 + strlen(current->str);

current = current->next;

}

}

void Write\_To\_File(const char\* filename, Node \*\*head)

{

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

if (!file)

{

perror("Failed to open file for writing");

exit(EXIT\_FAILURE);

}

int n = 0;

Node \*last = \*head;

while (last->next != NULL)

{

last = last->next;

n += 1;

}

n += 1;

// Запись n

fwrite(&n, sizeof(int), 1, file);

Node \*current = \*head;

// Запись i, o облясти

for(int i = 0; i < n; i++)

{

fwrite(&current->i, sizeof(int), 1, file);

fwrite(&current->o, sizeof(int), 1, file);

current = current->next;

}

// Запись мусорной области в 100 байт

char emptySpace[100] = {0};

fwrite(emptySpace, sizeof(char), 100, file);

Node \*current1 = \*head;

// Запись области k, s

for(int i = 0; i < n; i++)

{

int len = strlen(current1->str);

fwrite(&len, sizeof(int), 1, file);

fwrite(current1->str, sizeof(char), len, file);

current1 = current1->next;

}

fclose(file);

}

void Write\_To\_file\_txt(const char \*filename, Node \*\*head)

{

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

if (file == NULL)

{

Errors(2);

}

Node \*current = \*head;

while(current != NULL )

{

fprintf(file, "0x%lx %p %p %s\n", (uintptr\_t)current->o, (void \*)current->prev, (void \*)current->next, current->str);

current = current->next;

}

fclose(file);

}

void Txt\_Or\_Bin(const char \*filename, Node \*\*head) // для корректной работы dump с разными типами файлов

{

const char \*txt = ".txt";

const char \*bin = ".bin";

size\_t name\_len = strlen(filename);

size\_t format\_len = strlen(txt);

if (format\_len != strlen(bin)) //не возможная ошибка, но вдруг

{

exit(EXIT\_FAILURE);

}

if (name\_len < format\_len)

{

Errors(2);

}

if (strcmp(filename + name\_len - format\_len, txt) == 0)

{

Write\_To\_file\_txt(filename, head);

}

else if (strcmp(filename + name\_len - format\_len, bin) == 0)

{

Write\_To\_File(filename, head);

}

else

{

Errors(2);

}

}

void Delete\_Node(const char \*str, Node \*\*head)

{

Node \*current = \*head;

while (current != NULL)

{

if (strcmp(current->str, str) == 0)

{

Node \*to\_delete = current;

if (current->prev == NULL && current->next != NULL) // Удаляем первый элемент, есть следующий

{

current = current->next;

current->prev = NULL;

\*head = current;

}

else if (current->next == NULL && current->prev != NULL) // Удаляем последний элемент

{

current = current->prev;

current->next = NULL;

}

else if (current->next != NULL && current->prev != NULL) // Удаляем средний элемент

{

current->prev->next = current->next;

current->next->prev = current->prev;

current = current->next;

}

else // Удаляем единственный элемент

{

\*head = NULL;

current = NULL;

}

free(to\_delete->str);// освобождаем чтобы не болтались

free(to\_delete);

}

else

{

current = current->next;

}

}

}