ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИСCЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий

**Дисциплина:**

«Программирование»

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4-5**

**Варианты  
2-1-17-1**

**1-1**

**Выполнил:**

Студент гр. N3151, Мочеков Семён Сергеевич



**Проверил:**

Грозов Владимир Андреевич

Санкт-Петербург

2024

1. **Задание.**

Лабораторная 4:

Составить программу для работы с двусвязным списком и файлами. Элементами списка является IPv4-адрес. Дополнительно поддерживается команда swap.

Лабораторная 5:

Написать генератор файлов для lab4

Файл имеет структуру:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

1. **Makefile**

CXX = gcc

WORKFLAGS = -Wall -Wextra -Werror -O

SRCS = lab4

NAME = lab4

all: $(SRCS)

$(SRCS): $(SRCS).c

$(CXX) $(WORKFLAGS) -o $(NAME) $(SRCS).c

clean:

rm -f $(NAME).exe

rm -f \*~

**III. Исходный код алгоритма**

#include "libs/libb.h"

//--------------------------------------------//

int compare(const void \*a, const void \*b);

void mallocCheck(void\* ptr);

bool isFileCorrect(char\* name);

//--------------------------------------------//

node\_pair\* readWordsFromFile(FILE\* file, ull cnt);

void readIndexesFromFile(FILE\* file, ull cnt, ull file\_size, node\_pair\* words);

//--------------------------------------------//

Node\* createNode(char \*pair);

void push\_back(Node \*\*head, Node \*\*tail, char \*pair, ull \*cnt);

void push\_front(Node \*\*head, char \*pair, ull \*cnt);

void pop\_front(Node \*\*head, Node \*\*tail, ull \*cnt);

void pop\_back(Node \*\*head, Node \*\*tail, ull \*cnt);

void swap(Node \*\*head, ll a, ll b, ull cnt);

void dump(Node \*\*head, ull cnt, char\* name);

//--------------------------------------------//

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

    //~~~~~~~~~~~~~Init~~~~~~~~~~~~~~~~//

    node\_pair\* words\_buffer = NULL;

    Node \*head = createNode("");

    Node \*tail = head;

    ull \*cnt = malloc(sizeof(ull));

    \*cnt = 0;

    bool is\_empty\_list = false;

    //~~~~~~~~~~~~~Valid~~~~~~~~~~~~~~//

    if (argc != 2){

        fprintf(stderr, "Wrong number of arguments\n");

        exit(HOWMANY);

    } else if (strcmp(argv[1], "-v") == 0){

        printf("Mochekov Semyon Sergeevich, N3151\nTask: 2-1-17-1\n");

        return 0;

    }

    //~~~~~~~~~~~~~File~~~~~~~~~~~~~~//

    FILE \*source = fopen(argv[1], "r");

    if (!source){

        is\_empty\_list = true;

    } else {

        head->previous = head->next = NULL;

        fseek(source, 0, SEEK\_END);

        ull file\_size = ftell(source);

        rewind(source);

        // работа с файлом

        if (file\_size != 0){

            char\* cnt\_word = malloc(32);

            mallocCheck(cnt\_word);

            fscanf(source, "%s ", cnt\_word);

            \*cnt = strtoull(cnt\_word, NULL, 10);

            free(cnt\_word);

            words\_buffer = readWordsFromFile(source, \*cnt);

            fseek(source, 0, SEEK\_END);

            readIndexesFromFile(source, \*cnt, file\_size, words\_buffer);

            fclose(source);

            qsort(words\_buffer, \*cnt, sizeof(node\_pair), compare);

        } else {

            is\_empty\_list = true;

        }

    }

    if(!is\_empty\_list) {

        strcpy(tail->txt, (words\_buffer[0].IP));

        for (ull i = 1; i < \*cnt; i++) {

            push\_back(&head, &tail, (char\*)(words\_buffer[i].IP), cnt);

            --(\*cnt);

        }

    }

    if (words\_buffer){

        free(words\_buffer);

    }

    //~~~~~~~~~~Online-ans~~~~~~~~~~~//

    int c;

    bool is\_changed = false;

    char\* line = malloc(256 \* sizeof(char));

    while((c = getchar()) != EOF) {

        int i = 0;

        line = realloc(NULL, 256 \* sizeof(char));

        mallocCheck(line);

        line[i++] = (char)c;

        while ((c = getchar()) != '\n'){

            line[i++] = (char)c;

        }

        line[i] = '\0';

        char\* tok = strtok\_r(line, " ", &line);

        while (tok != NULL) {

            if (strcmp(tok, "dump") == 0) {

                tok = strtok\_r(NULL, " ", &line);

                dump(&head, \*cnt, tok);

            } else if (strcmp(tok, "push\_back") == 0) {

                while ((tok = strtok\_r(NULL, " ", &line)) != NULL) {

                    char\* tok2 = malloc(strlen(tok) + 1);

                    strcpy(tok2, tok);

                    if (isValidIpAddress(tok2)){

                        push\_back(&head, &tail, tok, cnt);

                        is\_changed = true;

                    } else {

                        fprintf(stderr, "Incorrect IP address\n");

                        goto contin;

                    }

                    free(tok2);

                }

            } else if (strcmp(tok, "push\_front") == 0) {

                while ((tok = strtok\_r(NULL, " ", &line)) != NULL) {

                    char\* tok2 = malloc(strlen(tok) + 1);

                    strcpy(tok2, tok);

                    if (isValidIpAddress(tok2)){

                        push\_front(&head, tok, cnt);

                        is\_changed = true;

                    } else {

                        fprintf(stderr, "Incorrect IP address\n");

                        goto contin;

                    }

                    free(tok2);

                }

            } else if (strcmp(tok, "pop\_back") == 0) {

                pop\_back(&head, &tail, cnt);

                is\_changed = true;

            } else if (strcmp(tok, "pop\_front") == 0) {

                pop\_front(&head, &tail, cnt);

                is\_changed = true;

            } else if (strcmp(tok, "swap") == 0) {

                ll k = 0, l = 0;

                char\* endpt\_k;

                char\* endpt\_l;

                tok = strtok\_r(NULL, " ", &line);

                if (tok != NULL) {

                    k = strtoll(tok, &endpt\_k, 10);

                }

                tok = strtok\_r(NULL, " ", &line);

                if (tok != NULL) {

                    l = strtoll(tok, &endpt\_l, 10);

                }

                if (tok == NULL || \*endpt\_k != '\0' || \*endpt\_l != '\0' ||

                    k < 0 || l < 0 || (ull)k >= \*cnt || (ull)l >= \*cnt) {

                    fprintf(stderr, "Uncorrect index\n");

                    goto contin;

                }

                swap(&head, k, l, \*cnt);

                is\_changed = true;

            } else {

                fprintf(stderr, "Unapropriate command\n");

                goto contin;

            }

            tok = strtok\_r(NULL, " ", &line); // Move to the next token

        }

        contin:

    }

    //~~~~~~~~~~~~~End~~~~~~~~~~~~~//

    if(is\_changed) {

        dump(&head, \*cnt, "result.txt");

    }

    return 0;

}

import argparse

import random

import string

def generateRubbish(length):

    letters = string.ascii\_letters + string.digits

    return ''.join(random.choice(letters) for t in range(length))

def generateIPv4Address():

    return '.'.join(str(random.randint(0, 255)) for t in range(4))

def write\_to\_file\_txt(filename, num\_strings, debug\_lines):

    with open(filename, 'w') as file:

        # Число элементов

        file.write(f"{num\_strings} ")

        # Перестановка индексов

        indices = list(range(num\_strings))

        random.shuffle(indices)

        # Генерация и записывание

        for i in range(num\_strings):

            ipv4 = generateIPv4Address()

            file.write(f"{ipv4}\0")

            debug\_lines.append((ipv4, indices[i]))

        # Область мусора

        for i in range(num\_strings % 17):

            file.write(f"{generateRubbish(32 + i + num\_strings % 31)}\0")

        # Внесeние индексов

        file.write("\n")

        for index in indices:

            file.write(f"{index} ")

        return list(range(num\_strings))

def main():

    debug\_lines = list() # Список для хранения пары

    parser = argparse.ArgumentParser(description="Генератор файла для двусвязного списка, наполненного IPv4-адресами.")

    parser.add\_argument('filename', nargs='?', type=str, help="Название создаваемого файла")  # nargs='?' для опционального аргумента

    parser.add\_argument('-n', type=int, help="Число элементов, которе надо сгенерировать")

    parser.add\_argument('-v', action='store\_true', help="Информация о разработчике")

    # Спарсенные аргументы

    args = parser.parse\_args()

    # Проверка аргументов

    if args.v:

        print("Мочеков Семён Сергеевич, гр. N3151\nВариант: 1-1")

        return

    elif not args.filename:

        raise ValueError("Требуется выходной файл")

    # Число строк если не указываем -n

    num\_strings = args.n if args.n else random.randint(10, 1000)

    lines = write\_to\_file\_txt(args.filename, num\_strings, debug\_lines)

    # Дебаг информация

    print(f"Сгенерированые {num\_strings} IPv4-адреса и их индексы:")

    for ipv4, index in debug\_lines:

        print(f"{ipv4} -> {index}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()