МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики»

Кафедра телекоммуникационных систем и вычислительных средств (ТС и ВС)

Расчетно-графическая работа

по дисциплине:

Программирование

Студент: В.А.Лысенко

Группы: ИКС-433

Предподаватель: А.И.Вейлер

Новосибирск 2025 г.

**Задание:**

Разработать программу Cezar, выполняющую шифрование в заданном тексте и DeCezar – дешифровку текста. Текст до шифрования, после шифрования и после дешифровки должен выводиться на экран. Оценить криптостойкость шифра. Обязательно динамическое выделение памяти под входные данные. Функции записаны в динамическую библиотеку. Создать многофайловый проект. Сборка проекта осуществляется с использованием CMake. Покрыть UNIT-тестами код.

**Анализ задачи:**

Задача:

Требуется разработать программу, состоящую из двух основных функций:

* Cezar – шифрование текста с использованием шифра Цезаря.
* DeCezar – дешифрование текста, зашифрованного тем же методом.

Требования:

* Поддержка русского алфавита (строчные и заглавные буквы, включая "ё").
* Динамическое выделение памяти под входные данные.
* Вывод исходного, зашифрованного и дешифрованного текста.
* Реализация в виде динамической библиотеки.

Алгоритм шифрования Цезаря:

* Входные данные: текст, ключ (сдвиг shift).
* Выходные данные: зашифрованный текст.
* Принцип работы:
  + Каждая буква алфавита заменяется на букву, стоящую на shift позиций дальше.
  + Если достигнут конец алфавита, счёт продолжается с начала.
  + Неалфавитные символы (пробелы, цифры, знаки препинания) остаются без изменений.

Псевдокод:

Функция cezar\_char(символ c, сдвиг shift):

Если c ∈ русский алфавит (строчные буквы):

Найти позицию c в строке "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"

Сдвинуть позицию на shift (по модулю 33)

Вернуть новый символ

Иначе если c ∈ русский алфавит (заглавные буквы):

Аналогично для "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"

Иначе:

Вернуть c без изменений

Функция Cezar(файл\_вход, файл\_выход, shift):

Открыть файл\_вход для чтения

Открыть файл\_выход для записи

Для каждого символа в файле\_вход:

Зашифровать его через cezar\_char

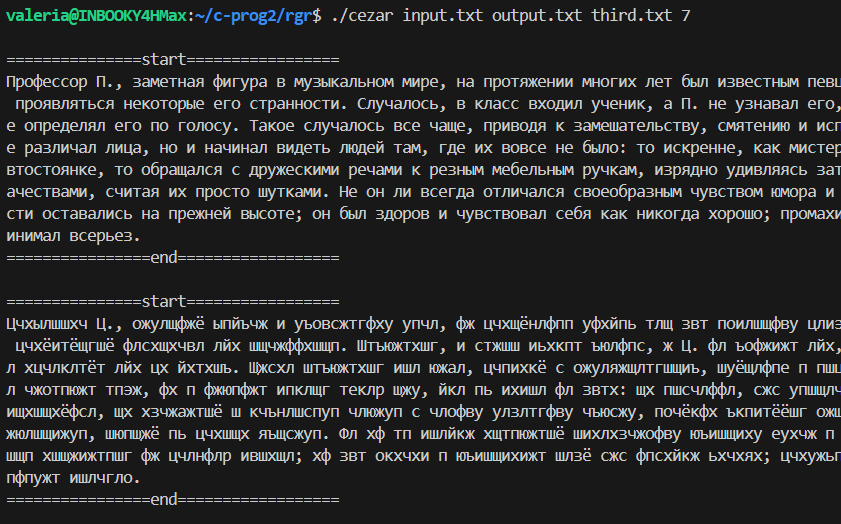
Записать результат в файл\_выход

Закрыть файлы

##### **Алгоритм дешифрования**

* Аналогичен шифрованию, но сдвиг выполняется в обратную сторону (-shift).

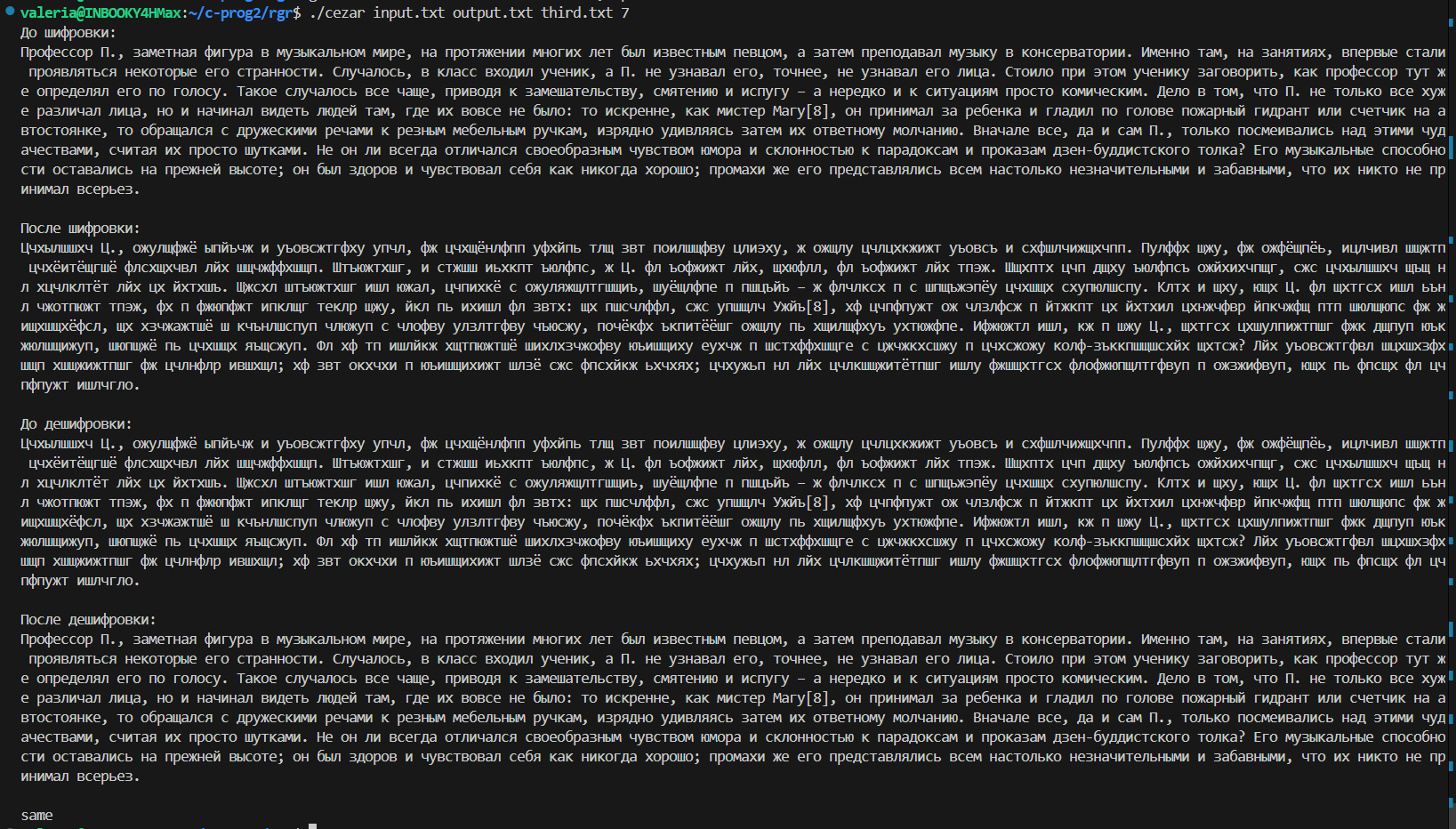
Корректный набор данных:

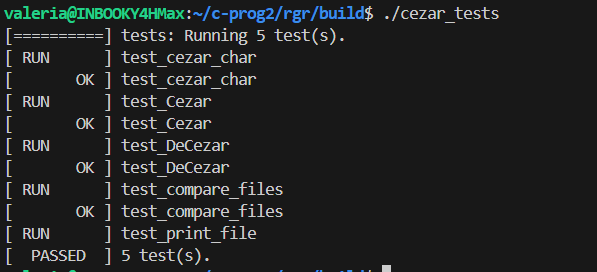


Не корректный набор данных (не хватает одного аргумента):



Скриншоты с результатами:





Листинг:

cezar.h:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h> //Корректное отображение русских символов

#include <wchar.h> //Обработка многобайтовых символов (Unicode, UTF-8/16/32)

#include <wctype.h> //добавляет логику для анализа широких символов.

wchar\_t cezar\_char(wchar\_t c, int shift);

void print\_file(const char\* filename);

int Cezar(char\* input\_filename, char\* cezar\_filename, int shift);

int DeCezar(char\* cezar\_filename, char\* decezar\_filename, int shift);

int compare\_files(char \*input\_filename, char \*decezar\_filename);

cezar.c:

#include"cezar.h"

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

    if (argc != 5) {

        printf("Неправильные аргументы командной строки\n");

        return 1;

    }

    char\* input\_file\_name = malloc(strlen(argv[1]) + 1);

    char\* cezar\_file\_name = malloc(strlen(argv[2]) + 1);

    char\* decezar\_file\_name = malloc(strlen(argv[3]) + 1);

    if (!input\_file\_name || !cezar\_file\_name || !decezar\_file\_name) {

        printf("Ошибка выделения памяти\n");

        free(input\_file\_name);

        free(cezar\_file\_name);

        free(decezar\_file\_name);

        return 1;

    }

    strcpy(input\_file\_name, argv[1]);

    strcpy(cezar\_file\_name, argv[2]);

    strcpy(decezar\_file\_name, argv[3]);

    int shift = atoi(argv[4]);

    Cezar(input\_file\_name, cezar\_file\_name, shift);

    DeCezar(cezar\_file\_name, decezar\_file\_name, shift);

    compare\_files(argv[1], argv[3]);

    free(input\_file\_name);

    free(cezar\_file\_name);

    free(decezar\_file\_name);

}

cezar\_f.c:

#include "cezar.h"

wchar\_t cezar\_char(wchar\_t c, int shift) {

    //wchar\_t - char для хранения "широких" символов

    wchar\_t \*lower\_rus = L"абвгдеёжзийклмнопрстуфхцчшщъыьэюя";

    wchar\_t \*upper\_rus = L"АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";

    wchar\_t \*pos;

    int index;

    if ((pos = wcschr(lower\_rus, c)) != NULL) {

        index = (pos - lower\_rus + shift) % 33;

        if (index < 0) index += 33;

        return lower\_rus[index];

    }

    else if ((pos = wcschr(upper\_rus, c)) != NULL) {

        index = (pos - upper\_rus + shift) % 33;

        if (index < 0) index += 33;

        return upper\_rus[index];

    }

    return c;

}

void print\_file(const char\* filename) {

    FILE \*file = fopen(filename, "r, ccs=UTF-8");

    if (!file) {

        wprintf(L"Ошибка открытия файла %s для чтения\n", filename);

        return;

    }

    wint\_t wc;

    while ((wc = fgetwc(file)) != WEOF) { //WEOF — "конец файла"

        putwchar(wc);

    }

    wprintf(L"\n\n");

    fclose(file);

}

int Cezar(char\* input\_filename, char\* cezar\_filename, int shift){

    setlocale(LC\_ALL, ""); // Установка локали из окружения

    FILE \*input\_file, \*cezar\_file;

    wint\_t wc;

    wprintf(L"До шифровки:\n");

    print\_file(input\_filename);

    input\_file = fopen(input\_filename, "r, ccs=UTF-8");

    if (!input\_file) {

        perror("Ошибка открытия входного файла");

        return 1;

    }

    cezar\_file = fopen(cezar\_filename, "w, ccs=UTF-8");

    if (!cezar\_file) {

        perror("Ошибка создания выходных файлов");

        fclose(input\_file);

        return 1;

    }

    while ((wc = fgetwc(input\_file)) != WEOF) {

        // Шифрование и запись в cezar\_file

        wchar\_t cezared\_char = cezar\_char(wc, shift);

        fputwc(cezared\_char, cezar\_file);

    }

    fclose(input\_file);

    fclose(cezar\_file);

    wprintf(L"После шифровки:\n");

    print\_file(cezar\_filename);

}

int DeCezar(char\* cezar\_filename, char\* decezar\_filename, int shift){

    setlocale(LC\_ALL, ""); // Установка локали из окружения

    FILE \*cezar\_file, \*decezar\_file;

    wint\_t cezared\_char;

    wprintf(L"До дешифровки:\n");

    print\_file(cezar\_filename);

    cezar\_file = fopen(cezar\_filename, "r, ccs=UTF-8");

    decezar\_file = fopen(decezar\_filename, "w, ccs=UTF-8");

    if (!cezar\_file || !decezar\_file) {

        perror("Ошибка создания выходного файлов");

        return 1;

    }

    while ((cezared\_char = fgetwc(cezar\_file)) != WEOF) {

        wchar\_t decezared\_char = cezar\_char(cezared\_char, -shift);

        fputwc(decezared\_char, decezar\_file);

    }

    fclose(cezar\_file);

    fclose(decezar\_file);

    wprintf(L"После дешифровки:\n");

    print\_file(decezar\_filename);

}

int compare\_files(char \*input\_filename, char \*decezar\_filename) {

    FILE \*input\_file = fopen(input\_filename, "r, ccs=UTF-8");

    FILE \*decezar\_file = fopen(decezar\_filename, "r, ccs=UTF-8");

    if (!input\_file || !decezar\_file) {

        if (input\_file) fclose(input\_file);

        if (decezar\_file) fclose(decezar\_file);

        return -1;

    }

    wint\_t c1, c2;

    int result = 0; // 0 - файлы одинаковые, 1 - нет

    while (1) {

        c1 = fgetwc(input\_file);

        c2 = fgetwc(decezar\_file);

        // Проверка на конец файла или ошибку чтения

        if (c1 == WEOF || c2 == WEOF) {

            if (c1 != c2) result = 1; // один файл закончился раньше

            break;

        }

        if (c1 != c2) {

            result = 1;

            break;

        }

    }

    fclose(input\_file);

    fclose(decezar\_file);

    if (result == 0) wprintf(L"same");

    else wprintf(L"not the same");

    wprintf(L"\n");

    return result;

}

CMakeLists.txt:

cmake\_minimum\_required(VERSION 3.10)

project(cezar\_project LANGUAGES C)

add\_library(cezar\_lib SHARED cezar\_f.c cezar.c)

target\_include\_directories(cezar\_lib PUBLIC include)

add\_executable(cezar cezar.c)

target\_link\_libraries(cezar cezar\_lib)

find\_package(cmocka REQUIRED)

enable\_testing()

add\_executable(cezar\_tests main\_test.c)

target\_include\_directories(cezar\_tests PRIVATE include)

target\_link\_libraries(cezar\_tests cezar\_lib cmocka)

add\_test(NAME cezar\_tests COMMAND cezar\_tests)

main\_tests.c:

#include "cezar.h"

#include <stdarg.h>

#include <stddef.h>

#include <setjmp.h>

#include <cmocka.h>

static void test\_cezar\_char(void\*\* state) {

    (void)state;

    assert\_int\_equal(cezar\_char(L'а', 3), L'г');

    assert\_int\_equal(cezar\_char(L'я', 1), L'а');

    assert\_int\_equal(cezar\_char(L'Г', 3), L'Ё');

    assert\_int\_equal(cezar\_char(L'А', -1), L'Я');

    assert\_int\_equal(cezar\_char(L' ', 10), L' ');

    assert\_int\_equal(cezar\_char(L'.', 3), L'.');

}

static void test\_Cezar(void\*\* state) {

    (void)state;

    char\* input\_file = "test\_input.txt";

    char\* encrypted\_file = "test\_encrypted.txt";

    wchar\_t\* text = L"Привет, мир!";

    int shift = 3;

    FILE\* file0= fopen(input\_file, "w, ccs=UTF-8");

    fputws(text, file0);

    int result = Cezar(input\_file, encrypted\_file, shift);

    assert\_int\_equal(result, 0);

    FILE\* file1 = fopen(encrypted\_file, "r, ccs=UTF-8");

    assert\_non\_null(file1);

    if (file1) fclose(file1);

    remove(input\_file);

    remove(encrypted\_file);

}

static void test\_DeCezar(void\*\* state) {

    (void)state;

    char\* encrypted\_file = "test\_encrypted.txt";

    char\* decrypted\_file = "test\_decrypted.txt";

    wchar\_t\* text = L"Привет, мир!";

    int shift = 5;

    FILE\* file0= fopen(encrypted\_file, "w, ccs=UTF-8");

    fputws(text, file0);

    int result = DeCezar(encrypted\_file, decrypted\_file, shift);

    assert\_int\_equal(result, 0);

    FILE\* file1 = fopen(decrypted\_file, "r, ccs=UTF-8");

    assert\_non\_null(file1);

    if (file1) fclose(file1);

    remove(encrypted\_file);

    remove(decrypted\_file);

}

static void test\_compare\_files(void\*\* state) {

    (void)state;

    char\* file\_name\_1 = "test\_file1.txt";

    char\* file\_name\_2 = "test\_file2.txt";

    char\* file\_name\_3 = "test\_file3.txt";

    FILE\* file1= fopen(file\_name\_1, "w, ccs=UTF-8");

    fputws(L"Один", file1);

    fclose(file1);

    FILE\* file2= fopen(file\_name\_2, "w, ccs=UTF-8");

    fputws(L"Один", file2);

    fclose(file2);

    FILE\* file3= fopen(file\_name\_3, "w, ccs=UTF-8");

    fputws(L"Два", file3);

    fclose(file3);

    int result = compare\_files(file\_name\_1, file\_name\_2);

    assert\_int\_equal(result, 0);

    result = compare\_files(file\_name\_1, file\_name\_3);

    assert\_int\_equal(result, 1);

    remove(file\_name\_1);

    remove(file\_name\_2);

    remove(file\_name\_3);

}

static void test\_print\_file(void\*\* state) {

    (void)state;

    char\* filename = "test\_print.txt";

    wchar\_t\* text = L"Тестовый текст для печати";

    FILE\* file0= fopen(filename, "w, ccs=UTF-8");

    fputws(text, file0);

    // Перенаправляем stdout в файл для проверки вывода

    freopen("test\_output.txt", "w", stdout);

    print\_file(filename);

    fclose(stdout);

    // Проверяем, что файл с выводом создан

    FILE\* file = fopen("test\_output.txt", "r");

    assert\_non\_null(file);

    if (file) fclose(file);

    remove(filename);

    remove("test\_output.txt");

}

int main(void) {

    const struct CMUnitTest tests[] = {

        cmocka\_unit\_test(test\_cezar\_char),

        cmocka\_unit\_test(test\_Cezar),

        cmocka\_unit\_test(test\_DeCezar),

        cmocka\_unit\_test(test\_compare\_files),

        cmocka\_unit\_test(test\_print\_file),

    };

    return cmocka\_run\_group\_tests(tests, NULL, NULL);

}

**Шифр Цезаря имеет низкий уровень криптостойкости по причинам**:

* **Уязвимость к частотному анализу**. Каждый символ заменяется однозначно, что делает возможным дешифровку при знании языка исходного текста.
* **Ограниченное количество ключей**. Например, для русского алфавита (33 буквы) существует всего 32 возможных сдвига (не считая нулевого).
* **Подверженность взлому при помощи перебора всех возможных сдвигов** (атака методом полного перебора).
* **Не подходит для шифрования длинных текстов**, так как их легко взломать.
* **Не обеспечивает конфиденциальность, целостность и аутентичность сообщения**.