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).

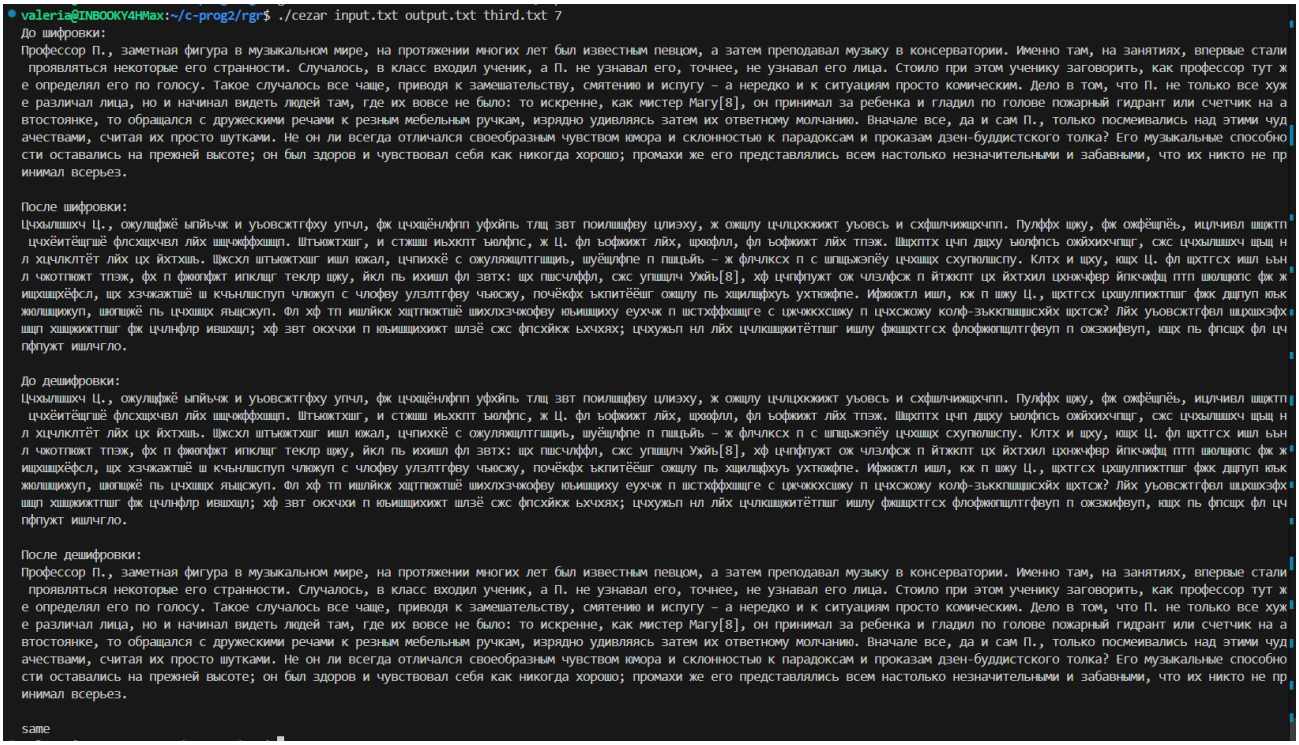
Корректный набор данных:

```
valeria@INBOOKY4HMax:~/c-prog2/rgr$ ./cezar input.txt output.txt third.txt 7

==========start==========
Профессор П., заметная фигура в музыкальном мире, на протяжении многих лет был известным певц
 проявляться некоторые его странности. Случалось, в класс входил ученик, а П. не узнавал его,
е определял его по голосу. Такое случалось все чаще, приводя к замешательству, смятению и исп
е различал лица, но и начинал видеть людей там, где их вовсе не было: то искренне, как мистер
втостоянке, то обращался с дружескими речами к резным мебельным ручкам, изрядно удивляясь зат
ачествами, считая их просто шутками. Не он ли всегда отличался своеобразным чувством юмора и
сти оставались на прежней высоте; он был здоров и чувствовал себя как никогда хорошо; промахи
инимал всерьез.
==========end==========

==========start==========
Цчхылшшхч Ц., ожулщфжё ыпйьчж и уъовсжтгфху упчл, фж цчхщёнлфпп уфхйпь тлщ звт поилшщфву цлиэ
 цчхёитёщгшё флсхщхчвл лйх шщчжффхшщп. Штъюжтхшг, и стжшш иьхкпт ъюлфпс, ж Ц. фл ъофжижт лйх,
л хцчлклтёт лйх цх йхтхшь. Щжсхл штъюжтхшг ишл южал, цчпихкё с ожуляжщлтгшщиь, шуёщлфпе п пшш
л чжотпюжт тпэж, фх п фжюпфжт ипклщг теклр щжу, йкл пь ихишл фл звтх: щх пшсчлффл, сжс упшщлч
ищхшщхёфсл, щх хэчжажтшё ш кчьнлшспуп члюжуп с члофву улэлтгфву чъюсжу, почёкфх ъкпитёёшг ожш
жюлшщижуп, шюпщжё пь цчхшщх яъщсжуп. Фл хф тп ишлйкж хщтпюжтшё шихлхзчжофву юъишщиху еухчж п
шщп хшщжижтпшг фж цчлнфлр ившхщл; хф звт окхчхи п юъишщихижт шлзё сжс фпсхйкж ьхчхях; цчхужьп
пфпужт ишлчгло.
==========end==========
```

Не корректный набор данных (не хватает одного аргумента):

```
valeria@INBOOKY4HMax:~/c-prog2/rgr$ ./cezar input.txt output.txt 7
Неправильные аргументы командной строки
```

Скриншоты с результатами:



```
valeria@INBOOKY4HMax:~/c-prog2/rgr/build$ ./cezar_tests
[==========] tests: Running 5 test(s).
[ RUN      ] test_cezar_char
[       OK ] test_cezar_char
[ RUN      ] test_Cezar
[       OK ] test_Cezar
[ RUN      ] test_DeCezar
[       OK ] test_DeCezar
[ RUN      ] test_compare_files
[       OK ] test_compare_files
[ RUN      ] test_print_file
[  PASSED  ] 5 test(s).
```

Листинг:

cezar.h:

```c
#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:

```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**.