

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и
информатики»
(СибГУТИ)

Кафедра телекоммуникационных систем и вычислительных средств
(ТС и ВС)

РГР по теме: «Шифрование текста квадратом Полибия»
по дисциплине
"Программирование»

Студент:
Группа ИКС-433

Штейнбрехер С.В.

Преподаватель:
Преподаватель

А.И. Вейлер

Новосибирск 2025

1 ВВЕДЕНИЕ

1.1 Задание:

Разработать программу Polibi выполняющую шифрование в заданном тексте и DePolibi – дешифровку текста. Текст до шифрования, после шифрования и после дешифровки должен выводиться на экран.

1.2 Критерии оценки:

1.2.1 Оценка «удовлетворительно»:

реализована проверка того, что исходный текст и полученные после дешифровки совпадают. Не предусмотрено динамическое выделение памяти под входные данные. Функции записаны в статическую библиотеку.

1.2.2 Оценка «хорошо»:

на вход программы подается 2 файла. Первый файл содержит текст на русском языке. Вторым файлом будет содержать зашифрованный текст. Обязательно динамическое выделение памяти под входные данные. Функции записаны в статическую библиотеку.

1.2.3 Оценка «отлично»:

оценить криптостойкость шифра. Обязательно динамическое выделение памяти под входные данные. Функции записаны в динамическую библиотеку.

1.3 Анализ задачи:

Одной модификацией одноалфавитной замены является квадрат Полибия, в котором символ алфавита заменяется парой чисел или символов по определенному правилу. В прямоугольник записывается алфавит.

В процессе шифрования каждая буква открытого текста представляется в шифротексте парой букв, указывающих строку и столбец, в которых расположена данная буква. Так представлениями букв В, Г, П, У будут АВ, АГ, ВВ, ГА соответственно. Если использовать приведенный выше квадрат в качестве ключа шифрования, то фраза «ПРИМЕР» будет зашифрована в «ВВВГВВБЕАЕВГ».

1.4 Алгоритм:

1. Поиск позиции буквы в таблице Полибия
2. Шифрование текста
3. Дешифрование текста
4. Чтение файла
5. Запись текста в файл
6. Проверка совпадения
7. Оценка криптостойкости

2 РЕАЛИЗАЦИЯ

2.1 Архитектура проекта

/rgr

CMakeLists.txt - сборка проекта

polybius.c - файл с функциями

polybius.h - файл с заголовками функций

test.c - файл с тестами

main.c - главный файл

2.2 Сборка проекта

2.2.1 Статическая библиотека

- gcc -c polybius.c -c
- ar rcs libpolybius.a polybius.o
- gcc main.c -L. -lpolybius -o a.out

2.2.2 Динамическая библиотека

- gcc -shared -fPIC -o libpolybius.so polybius.c
- gcc -o a.out main.c -ldl -L. -lpolybius

2.3 Примеры работы

```
Исходный текст: ПРИМЕР
Зашифрованный текст: BVVG6VB6EAEVG
Дешифрованный текст: ПРИМЕР
Исходный и дешифрованный тексты совпадают
Оценка криптостойкости:
- Шифр Полибия уязвим к частотному анализу, так как каждая буква заменяется фиксированной парой.
- Рекомендации: использовать случайный порядок строк и столбцов в таблице для каждого сеанса или добавлять шумовые символы.
```

Рисунок 1 — Вывод главного файла с содержимым input.txt "ПРИМЕР"

```
Исходный текст: ШИФРОВКА
Зашифрованный текст: ГЕБVG6VG6BAVB6ГАА
Дешифрованный текст: ШИФРОВКА
Исходный и дешифрованный тексты совпадают
Оценка криптостойкости:
- Шифр Полибия уязвим к частотному анализу, так как каждая буква заменяется фиксированной парой.
- Рекомендации: использовать случайный порядок строк и столбцов в таблице для каждого сеанса или добавлять шумовые символы.
```

Рисунок 2 — Вывод главного файла с содержимым input.txt "ШИФРОВКА"

3 ИСХОДНЫЙ КОД

3.1 main.c

```
#include <stdio.h>
#include <wchar.h>
#include <stdlib.h>
#include <string.h>
#include <dlfcn.h>
#include <locale.h>
#include "polybius.h"
// Чтение файла с динамическим выделением памяти
wchar_t* read_file(const char* filename) {
    FILE* file = fopen(filename, "r");
    if (!file) {
        printf("Ошибка открытия файла %s\n", filename);
        return NULL;
    }

    int capacity = 1024;
    wchar_t* buffer = (wchar_t*)malloc(capacity);
    if (!buffer) {
        fclose(file);
        return NULL;
    }

    int pos = 0;
    wchar_t c;
    while ((c = fgetwc(file)) != EOF) {
        if (pos >= capacity - 1) {
            capacity *= 2;
            wchar_t* new_buffer = (wchar_t*)realloc(buffer, capacity);
            if (!new_buffer) {
                free(buffer);
            }
        }
        buffer[pos] = c;
        pos++;
    }
    buffer[pos] = L'\0';
    return buffer;
}
```

```

        fclose(file);
        return NULL;
    }
    buffer = new_buffer;
}
buffer[pos++] = c;
}
buffer[pos] = '\0';
fclose(file);
return buffer;
}

// Запись текста в файл
void write_file(const char* filename, const wchar_t* text) {
    FILE* file = fopen(filename, "w");
    if (!file) {
        printf("Ошибка открытия файла %s для записи\n", filename);
        return;
    }
    fputws(text, file);
    fclose(file);
}

int main() {
    setlocale(LC_ALL, "");
    // Загрузка динамической библиотеки
    void* handle = dlopen("./libpolybius.so", RTLD_LAZY);
    if (!handle) {
        printf("Ошибка загрузки библиотеки: %s\n", dlerror());
        return 1;
    }

    // Чтение исходного текста из файла input.txt
    wchar_t* input_text = read_file("input.txt");
    if (!input_text) {

```

```

        printf("Не удалось прочитать входной файл\n");
        dlclose(handle);
        return 1;
    }

    printf("Исходный текст: %ls\n", input_text);

    // Шифрование
    wchar_t* encrypted = Polibi(input_text);
    if (!encrypted) {
        printf("Ошибка шифрования\n");
        free(input_text);
        dlclose(handle);
        return 1;
    }

    //
    printf("Зашифрованный текст: %ls\n", encrypted);

    // Запись зашифрованного текста в файл output.txt
    write_file("output.txt", encrypted);

    // Дешифрование
    wchar_t* decrypted = DePolibi(encrypted);
    if (!decrypted) {
        printf("Ошибка дешифрования\n");
        free(input_text);
        free(encrypted);
        dlclose(handle);
        return 1;
    }

    printf("Дешифрованный текст: %ls\n", decrypted);

    // Проверка совпадения

```



```

    if (wcscmp(input_text, decrypted) == 0) {
        printf("Исходный и дешифрованный тексты совпадают\n");
    } else {
        printf("Исходный и дешифрованный тексты НЕ совпадают\n");
    }

    // Оценка криптостойкости
    printf("Оценка криптостойкости:\n");
    printf("- Шифр Полибия уязвим к частотному анализу, так как каждая
    printf("- Рекомендации: использовать случайный порядок строк и столбцов

    // Освобождение памяти
    free(input_text);
    free(encrypted);
    free(decrypted);
    dlclose(handle);

    return 0;
}

```

3.2 polybius.c

```

#include "polybius.h"
#include <stdlib.h>
#include <string.h>

// Таблица Полибия (5x6, как в задании)
const wchar_t polybius[5][6] = {
    {L'A', L'B', L'В', L'Г', L'Д', L'E'},
    {L'Ж', L'З', L'И', L'К', L'Л', L'M'},
    {L'Н', L'O', L'П', L'Р', L'С', L'T'},
    {L'У', L'Ф', L'Х', L'Ц', L'Ч', L'Ш'},
    {L'Щ', L'Ъ', L'Ы', L'Ь', L'Э', L'Ю'}
};

```

```
// Символы строки и столбца (заголовки таблицы)
const wchar_t rows[5] = {L'A', L'B', L'В', L'Г', L'Д'};
const wchar_t cols[6] = {L'A', L'B', L'В', L'Г', L'Д', L'E'};
```

```
// Поиск позиции буквы в таблице Полибия
void find_position(wchar_t c, int* row, int* col) {
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 6; j++) {
            if (polybius[i][j] == c) {
                *row = i;
                *col = j;
                return;
            }
        }
    }
    *row = -1; // Если символ не найден
    *col = -1;
}
```

```
// Поиск индекса символа в массиве rows или cols
int find_index(wchar_t c, const wchar_t* array, int size) {
    for (int i = 0; i < size; i++) {
        if (array[i] == c) return i;
    }
    return -1;
}
```

```
// Шифрование текста
wchar_t* Polibi(const wchar_t* input) {
    int len = wcslen(input);
    //printf("len %d\n", len);
    // Зашифрованный текст: каждая буква → пара букв, длина удваивается.
    wchar_t* output = (wchar_t*)malloc(2 * len + 1);
    if (!output) return NULL;
```

```

    int k = 0;
    for (int i = 0; i < len; i++) {
        int row, col;
        find_position(input[i], &row, &col);
        if (row != -1 && col != -1) {
            output[k++] = rows[row]; // Буква строки
            output[k++] = cols[col]; // Буква столбца
        }
    }
    output[k] = L'\0';
    return output;
}

```

// Дешифрование текста

```

wchar_t* DePolibi(const wchar_t* input) {
    int len = wcslen(input);
    // Дешифрованный текст: пара букв → одна буква, длина уменьшается
    wchar_t* output = (wchar_t*)malloc(len / 2 + 1);
    if (!output) return NULL;

    int k = 0;
    for (int i = 0; i < len; i += 2) {
        int row = find_index(input[i], rows, 5);
        int col = find_index(input[i + 1], cols, 6);
        if (row != -1 && col != -1) {
            output[k++] = polybius[row][col];
        }
    }
    output[k] = L'\0';
    return output;
}

```

3.3 polybius.h

```
#ifndef POLYBIUS_H
#define POLYBIUS_H

#include "wchar.h"
#include "stdio.h"

// Функция шифрования
wchar_t* Polibi(const wchar_t* input);

// Функция дешифрования
wchar_t* DePolibi(const wchar_t* input);

#endif
```

3.4 test.c

```
#include <stdarg.h>
#include <stddef.h>
#include <setjmp.h>
#include <cmocka.h>
#include <stdlib.h>
#include <string.h>
#include <wchar.h>
#include <polybius.h>

// Тест шифрования
void test_polibi_encrypt(void **state) {
    (void)state;
    const wchar_t *input = L"ПРИВЕТ";
    const wchar_t *expected = L"ВГВБГДВВ"; // Ожидаемый результат

    wchar_t *encrypted = Polibi(input);
```

```

    assert_non_null(encrypted);
    assert_wstring_equal(encrypted, expected);
    free(encrypted);
}

int main(void) {
    const struct CMUnitTest tests[] = {
        cmocka_unit_test(test_polibi_encrypt),
    };

    return cmocka_run_group_tests(tests, NULL, NULL);
}

```

3.5 CMakeLists.txt

```

cmake_minimum_required(VERSION 3.10)

project(rgr)

add_executable(main main.c)

if (BUILD_STATIC)
    message("Build static")
    add_library(polybius STATIC polybius.c)
    target_link_libraries(main polybius)
else()
    message("Build shared")
    add_library(polybius SHARED polybius.c)
    target_link_libraries(main polybius)
endif()

if(EXISTS "${CMAKE_CURRENT_SOURCE_DIR}test.c")
    enable_testing()
    find_package(cmocka REQUIRED)

```

```
    add_executable(test_polybius test.c)
    target_link_libraries(test_polybius polybius cmocka)
    add_test(NAME polybius_test COMMAND test_polybius)
endif()
```