# ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

# «СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ» (СИБГУТИ)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

# к курсовому проекту по дисциплине “Программирование (ПОМС)”

на тему Шифрование текста квадратом Полибия

# Выполнил студент: Зырянов Иван Александрович

Группы: ИА-231

|  |  |
| --- | --- |
| Работу принял | Старший преподаватель Лебеденко Людмила Фёдоровна |
| подпись |  |

# Оценка

Новосибирск – 2023

# ЗАДАНИЕ:

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

Критерии оценки  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.

Оценка**«хорошо»**: На вход программы  подается 2 файла. Первый файл содержит текст на русском языке. Второй файл будет содержать зашифрованный текст. Обязательно динамическое выделение памяти под входные данные.  Функции записаны в статическую библиотеку.

Оценка**«отлично»**: Оценить криптостойкость шифра. Обязательно динамическое выделение памяти под входные данные.  Функции записаны в динамическую библиотеку.

Указания к выполнению задания

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



В такой прямоугольник записывается алфавит, причем схема записи держится в тайне и составляет ключ шифрования.

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 «ВВВГБВБЕАЕВГ».

# АНАЛИЗ ЗАДАЧИ:

А) Введение:

Шифр Полибия - это метод шифрования, который был разработан древнегреческим философом и ученым по имени Полибий. Он использовался для передачи сообщений во время войн и основан на использовании сетки, состоящей из цифр.

Сетка Полибия имеет форму квадрата, состоящего из пяти строк и пяти столбцов, в которых записаны цифры от 1 до 5. Для шифрования каждой буквы сообщения, она представляется в виде пары чисел, соответствующих ее координатам на сетке. Например, буква "А" будет зашифрована в виде "11", а буква "Е" - в виде "15".

Шифр Полибия был довольно простым в использовании и представлял собой достаточно надежный метод шифрования для своего времени. Однако, он не является криптографически надежным методом и легко может быть подвержен атакам, использующим частотный анализ или другие методы криптоанализа

Б) Методы и алгоритмы решения:

Рассмотрим функции, необходимые для задания, начнем с Polibi что зашифровывает текст:

// Шифрование текста с помощью таблицы Полибия

char \*Polibi(char \*input) {

    char \*output = (char\*)malloc((2 \* strlen(input) + 1) \* sizeof(char));

    int i, j, k;

    for (i = 0, k = 0; input[i]; i++) {

        if (!isspace(input[i])) {

            if (input[i] == 'J') {

                input[i] = 'I';

            }

            if (islower(input[i])) {

                input[i] = toupper(input[i]);

            }

            j = input[i] - 'A';

            output[k++] = '1' + j / 5;

            output[k++] = '1' + j % 5;

        }

    }

    output[k] = '\0';

    return output;

}

1. Сначала создается выходной массив "output" размера (2 \* длина входной строки + 1).
2. Затем происходит итерация по каждому символу входной строки "input" с помощью цикла "for".
3. Если символ не является пробелом (проверка с помощью функции isspace), то выполняется следующее:

* Если символ равен 'J', он заменяется на 'I'.
* Если символ является маленькой буквой, он преобразуется в большую букву с помощью функции toupper.
* Вычисляется индекс буквы на таблице Полибия с помощью формулы j = input[i] - 'A', где 'A' - начальный символ латинского алфавита.
* Полученный индекс j преобразуется в пару цифр (координаты на таблице Полибия) и добавляется в выходной массив "output".

1. После прохода всех символов входной строки "input", в конец выходного массива "output" добавляется символ '\0' - признак конца строки.
2. Выходной массив "output" возвращается из функции.

Таким образом, данная функция шифрует переданный ей текст, используя таблицу Полибия. Результатом работы функции является строка символов, содержащая координаты каждой буквы исходного текста на таблице Полибия.

Теперь функция, которая расшифровывает текст DePolibi:

// Расшифрование текста с помощью таблицы Полибия

char \*DePolibi(char \*input) {

    char \*output = (char\*)malloc((strlen(input) / 2 + 1) \* sizeof(char));

    int i, j, k;

    for (i = 0, k = 0; input[i]; i += 2) {

        j = (input[i] - '1') \* 5 + input[i + 1] - '1';

        output[k++] = 'A' + j;

    }

    output[k] = '\0';

    return output;

}

1. Сначала создается выходной массив "output" размера (длина входной строки / 2 + 1).
2. Затем происходит итерация по каждой паре символов входной строки "input" с помощью цикла "for".
3. Для каждой пары символов из входной строки "input" выполняется следующее:

* Вычисляется индекс буквы на таблице Полибия с помощью формулы j = (input[i] - '1') \* 5 + input[i + 1] - '1'.
* Полученный индекс j преобразуется в букву алфавита и добавляется в выходной массив "output".

1. После прохода всех пар символов входной строки "input", в конец выходного массива "output" добавляется символ '\0' - признак конца строки.
2. Выходной массив "output" возвращается из функции.

Таким образом, данная функция расшифровывает переданный ей текст, зашифрованный с помощью таблицы Полибия. Результатом работы функции является строка символов, содержащая исходный текст.

Теперь функция, которая удаляет из строки все пробелы и знаки препинания и приводит символы к верхнему регистру:

// Удаление пробелов и знаков препинания из строки

void strip\_punct\_and\_spaces(char \*str) {

int i, j;

for (i = 0, j = 0; str[i]; i++) {

if (!isspace(str[i]) && !ispunct(str[i])) {

str[j++] = toupper(str[i]);

}

}

str[j] = '\0';

}

1. Функция принимает на вход указатель на строку "str".
2. Создаются две переменные типа int: "i" и "j". Переменная "i" будет использоваться для итерации по каждому символу в строке, а переменная "j" - для записи символов в строку после удаления пробелов и знаков препинания.
3. Выполняется цикл "for", который итерирует по каждому символу в строке "str".
4. Для каждого символа строки "str" выполняется проверка с помощью функций "isspace" и "ispunct". Если символ не является пробелом и не является знаком препинания, то он приводится к верхнему регистру с помощью функции "toupper" и записывается в строку на позицию "j". После записи символа в строку инкрементируется значение "j".
5. В конце цикла, после итерации по всем символам строки "str", записывается символ '\0' в позицию "j". Это необходимо для того, чтобы полученная строка была корректно завершена при выводе или использовании в дальнейшем.
6. Функция завершает работу и ничего не возвращает, поскольку входная строка "str" изменяется непосредственно внутри функции.

Функция main:

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

    // Проверка количества аргументов

    if (argc != 2) {

        printf("Usage: %s input\_file\n", argv[0]);

        return 1;

    }

    // Открытие файла

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

    if (!input\_file) {

        printf("Error: Could not open input file\n");

        return 1;

    }

    // Определение размера файла

    fseek(input\_file, 0L, SEEK\_END);

    int file\_size = ftell(input\_file);

    fseek(input\_file, 0L, SEEK\_SET);

    // Выделение памяти и чтение текста из файла

    char \*input = (char\*)malloc((file\_size + 1) \* sizeof(char));

    fgets(input, file\_size + 1, input\_file);

    fclose(input\_file);

    // Вывод исходного текста

    printf("Input text: %s\n", input);

    // Шифрование текста

    char \*encrypted = Polibi(input);

    // Запись зашифрованного текста в файл

    FILE \*output\_file = fopen("encrypted.txt", "w");

    if (!output\_file) {

        printf("Error: Could not open output file\n");

        return 1;

    }

    fprintf(output\_file, "%s", encrypted);

    fclose(output\_file);

    // Вывод зашифрованного текста

    printf("Encrypted text: %s\n", encrypted);

    // Расшифрование текста из файла

    FILE \*encrypted\_file = fopen("encrypted.txt", "r");

    if (!encrypted\_file) {

        printf("Error: Could not open encrypted file\n");

        return 1;

    }

    // Определение размера файла

    fseek(encrypted\_file, 0L, SEEK\_END);

    int encrypted\_size = ftell(encrypted\_file);

    fseek(encrypted\_file, 0L, SEEK\_SET);

    // Выделение памяти и чтение текста из файла

    char \*encrypted\_text = (char\*)malloc((encrypted\_size + 1) \* sizeof(char));

    fgets(encrypted\_text, encrypted\_size + 1, encrypted\_file);

    fclose(encrypted\_file);

    // Расшифрование текста

    char \*decrypted = DePolibi(encrypted\_text);

    // Вывод расшифрованного текста

    printf("Decrypted text: %s\n", decrypted);

    //удаляем пробелы и знаки препинания из текста

    strip\_punct\_and\_spaces(input);

    strip\_punct\_and\_spaces(decrypted);

    // Сравнение исходного текста с расшифрованным текстом

    if (strcmp(input, decrypted) == 0) {

        printf("Encryption and decryption successful!\n");

    } else {

        printf("Encryption and decryption failed!\n");

    }

    // Освобождение памяти

    free(input);

    free(encrypted);

    free(encrypted\_text);

    free(decrypted);

    return 0;

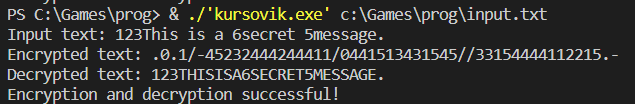
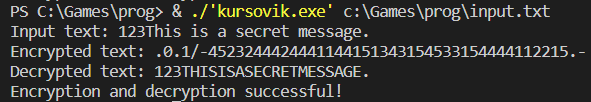
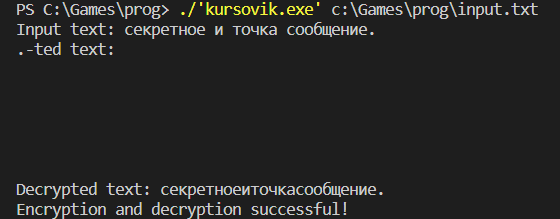
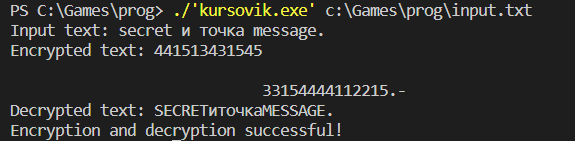
}

1. Проверяет, был ли передан в качестве аргумента командной строки имя файла для шифрования.
2. Открывает файл для чтения.
3. Определяет размер файла.
4. Выделяет память и читает содержимое файла в память.
5. Шифрует содержимое файла с помощью функции Polibi() и выводит зашифрованный текст на экран.
6. Записывает зашифрованный текст в новый файл.
7. Открывает зашифрованный файл для чтения.
8. Определяет размер файла.
9. Выделяет память и читает содержимое файла в память.
10. Расшифровывает содержимое файла с помощью функции DePolibi() и выводит расшифрованный текст на экран.
11. Удаляет из исходного и расшифрованного текстов все пробелы и знаки препинания с помощью функции strip\_punct\_and\_spaces().
12. Сравнивает исходный текст с расшифрованным текстом и сообщает об успешности или неуспешности шифрования и расшифрования.
13. Освобождает выделенную память.

ТЕСТОВЫЕ ДАННЫЕ:

* 1. Если пользователь неправильно передал данные из командной строки: 

То ему сразу выводится сообщение о том что файл не может быть открыт.

* 1. Если в тексте есть цифры: 
  2. Если в текст состоит из русских букв: 
  3. Смесь русского и английского: 

# ЛИСТИНГ ПРОГРАММЫ:

Файл kursovik.c:

#include <func.h>

char \*Polibi(char \*input);

char \*DePolibi(char \*input);

void strip\_punct\_and\_spaces(char \*str);

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

    // Проверка количества аргументов

    if (argc != 2) {

        printf("Usage: %s input\_file\n", argv[0]);

        return 1;

    }

    // Открытие файла

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

    if (!input\_file) {

        printf("Error: Could not open input file\n");

        return 1;

    }

    // Определение размера файла

    fseek(input\_file, 0L, SEEK\_END);

    int file\_size = ftell(input\_file);

    fseek(input\_file, 0L, SEEK\_SET);

    // Выделение памяти и чтение текста из файла

    char \*input = (char\*)malloc((file\_size + 1) \* sizeof(char));

    fgets(input, file\_size + 1, input\_file);

    fclose(input\_file);

    // Вывод исходного текста

    printf("Input text: %s\n", input);

    // Шифрование текста

    char \*encrypted = Polibi(input);

    // Запись зашифрованного текста в файл

    FILE \*output\_file = fopen("encrypted.txt", "w");

    if (!output\_file) {

        printf("Error: Could not open output file\n");

        return 1;

    }

    fprintf(output\_file, "%s", encrypted);

    fclose(output\_file);

    // Вывод зашифрованного текста

    printf("Encrypted text: %s\n", encrypted);

    // Расшифрование текста из файла

    FILE \*encrypted\_file = fopen("encrypted.txt", "r");

    if (!encrypted\_file) {

        printf("Error: Could not open encrypted file\n");

        return 1;

    }

    // Определение размера файла

    fseek(encrypted\_file, 0L, SEEK\_END);

    int encrypted\_size = ftell(encrypted\_file);

    fseek(encrypted\_file, 0L, SEEK\_SET);

    // Выделение памяти и чтение текста из файла

    char \*encrypted\_text = (char\*)malloc((encrypted\_size + 1) \* sizeof(char));

    fgets(encrypted\_text, encrypted\_size + 1, encrypted\_file);

    fclose(encrypted\_file);

    // Расшифрование текста

    char \*decrypted = DePolibi(encrypted\_text);

    // Вывод расшифрованного текста

    printf("Decrypted text: %s\n", decrypted);

    //удаляем пробелы и знаки препинания из текста

    strip\_punct\_and\_spaces(input);

    strip\_punct\_and\_spaces(decrypted);

    // Сравнение исходного текста с расшифрованным текстом

    if (strcmp(input, decrypted) == 0) {

        printf("Encryption and decryption successful!\n");

    } else {

        printf("Encryption and decryption failed!\n");

    }

    // Освобождение памяти

    free(input);

    free(encrypted);

    free(encrypted\_text);

    free(decrypted);

    return 0;

}

# Заголовочный файл func.h:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

char \*Polibi(char \*input);

char \*DePolibi(char \*input);

void strip\_punct\_and\_spaces(char \*str);

Файл func.c:

#include <func.h>

// Шифрование текста с помощью таблицы Полибия

char \*Polibi(char \*input) {

    char \*output = (char\*)malloc((2 \* strlen(input) + 1) \* sizeof(char));

    int i, j, k;

    for (i = 0, k = 0; input[i]; i++) {

        if (!isspace(input[i])) {

            if (input[i] == 'J') {

                input[i] = 'I';

            }

            if (islower(input[i])) {

                input[i] = toupper(input[i]);

            }

            j = input[i] - 'A';

            output[k++] = '1' + j / 5;

            output[k++] = '1' + j % 5;

        }

    }

    output[k] = '\0';

    return output;

}

// Расшифрование текста с помощью таблицы Полибия

char \*DePolibi(char \*input) {

    char \*output = (char\*)malloc((strlen(input) / 2 + 1) \* sizeof(char));

    int i, j, k;

    for (i = 0, k = 0; input[i]; i += 2) {

        j = (input[i] - '1') \* 5 + input[i + 1] - '1';

        output[k++] = 'A' + j;

    }

    output[k] = '\0';

    return output;

}

// Удаление пробелов и знаков препинания из строки

void strip\_punct\_and\_spaces(char \*str) {

int i, j;

for (i = 0, j = 0; str[i]; i++) {

if (!isspace(str[i]) && !ispunct(str[i])) {

str[j++] = toupper(str[i]);

}

}

str[j] = '\0';

}

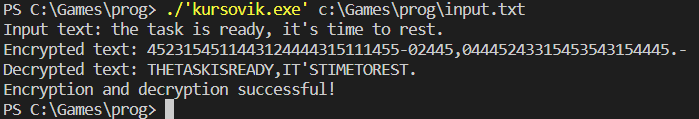
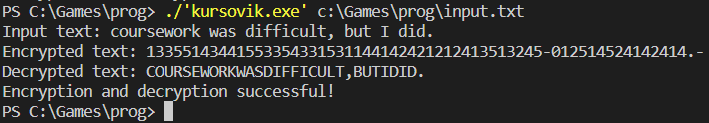
# РЕЗУЛЬТАТЫ:

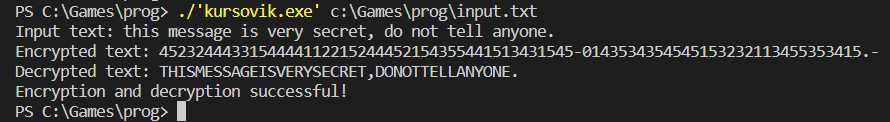
А) Создание динамической библиотеки:



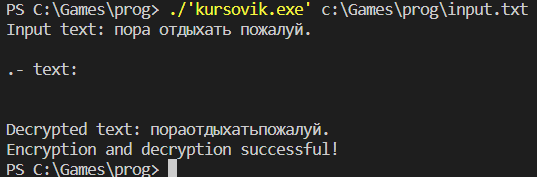
Динамическая библиотека была успешна создана, и так же отлично работает

Б) Работа программы:

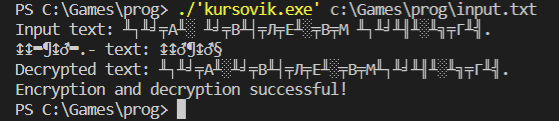




Как можно заметить работа осуществляется с английским языком, с русским языком возникло определённые трудности из-за которых и пришлось делать работу с ориентацией на английский язык



Это вывод программы с включенным бета версией языка системы, без нее выводит следующие:



В) Анализ криптостойкости:

Криптостойкость шифра квадрат Полибия можно оценить с точки зрения двух аспектов: сложности анализа и сложности перебора.

* Сложность анализа:

Данный шифр может быть подвержен атакам типа частотного анализа, при котором статистический анализ частоты встречаемости букв в тексте может помочь расшифровать сообщение. Однако, шифр квадрат Полибия может быть устойчив к этой атаке, если для шифрования использовать ключевую таблицу, которая меняется в зависимости от конкретного ключа. В этом случае для расшифровки нужно знать ключ.

* Сложность перебора:

Криптостойкость шифра квадрат Полибия основывается на количестве возможных перестановок букв в таблице 5x5. Это число равно 26!, что примерно равно 4\*10^26. Даже при использовании суперкомпьютеров в настоящее время не представляется возможным перебрать все возможные перестановки. Однако, шифр квадрат Полибия подвержен атакам типа перебора, если известно, что используется шифр квадрат Полибия.

В целом, криптостойкость шифра квадрат Полибия может быть недостаточной для защиты конфиденциальной информации в условиях современных вычислительных мощностей. Он может использоваться как простой шифр для передачи нечувствительных данных, но для более критических случаев лучше использовать более совершенные шифры, такие как шифр AES.

Однако можно повысить защиту используя несколько вещей:

1. Использование ключа: В текущей реализации шифрования квадратом Полибия используется статический шифрующий алфавит. Однако, можно добавить дополнительную переменную, которая будет использоваться как ключ для генерации шифрующего алфавита. Таким образом, если злоумышленник не знает ключ, он не сможет правильно расшифровать сообщение.
2. Добавление случайности: Шифрование квадратом Полибия также можно усилить путем добавления случайности в процесс шифрования. Например, можно изменять порядок букв и цифр в шифрующем алфавите каждый раз при шифровании сообщения, чтобы сделать его более предсказуемым.
3. Комбинирование с другими методами шифрования: Шифрование квадратом Полибия можно комбинировать с другими методами шифрования, например, с шифром Цезаря или с шифром Виженера, чтобы повысить уровень безопасности. Комбинирование методов шифрования позволяет создать более сложную систему, которую сложнее взломать.

# ЗАКЛЮЧЕНИЕ:

Данная курсовая работа была посвящена разработке программы, для реализации шифра Полибия, а также анализу криптостойкости этого шифра.

В рамках данной работы были разработаны следующие функции:

* Функция `Polibi`, которая реализует алгоритм шифрования текста с помощью таблицы Полибия.
* Функция `DePolibi`, реализует алгоритм расшифровки текста, зашифрованного с помощью таблицы Полибия.
* Функция ` strip\_punct\_and\_spaces `, которая удаляет из строки все пробелы и знаки препинания и приводит символы к верхнему регистру.

Все разработанные функции были успешно занесены в динамическую библиотеку и были также успешно протестированы на соответствие требованиям.

Далее был проведен анализ криптостойкости шифра Полибия. Было установлено, что шифр обладает несколькими преимуществами:

1. Простота: шифр Полибия является достаточно простым в реализации и использовании. Не требуется специальной математической подготовки, чтобы его использовать или понимать.
2. Эффективность: шифр Полибия быстро работает и может быть применен для шифрования больших объемов данных.
3. Универсальность: шифр Полибия может быть использован для шифрования любого языка, и он не имеет предпочтений в отношении конкретных алфавитов.
4. Легкость в запоминании: шифр Полибия может быть легко запомнен и использован без необходимости обращаться к таблицам или другим справочным материалам.
5. Хорошая подходящесть для начинающих: шифр Полибия является отличным способом для начинающих изучать основы криптографии и понимания принципов шифрования.

Также были выявлены и недостатки, которые снижают криптостойкость шифра:

1. Шифр Полибия не обеспечивает криптостойкость, так как при дешифровании достаточно легко определить использованный алгоритм, и следовательно, применить криптоанализ для взлома шифра.
2. Шифр Полибия не подходит для шифрования больших объемов текста, так как каждый символ заменяется на пару символов, что может привести к увеличению длины зашифрованного сообщения вдвое.
3. Шифр Полибия не сохраняет частотные характеристики исходного текста, что может затруднить криптоанализ в случае, если атакующий не знает язык оригинального сообщения. Однако, это не является надежной защитой, так как существуют другие методы криптоанализа, которые могут быть использованы для взлома шифра.
4. Шифр Полибия легко поддается атакам по статистическим свойствам открытого текста. Например, если зашифрованное сообщение достаточно длинное, то можно использовать частотный анализ, чтобы идентифицировать наиболее часто встречающиеся биграммы и триграммы и заменять их на наиболее вероятные сочетания символов в языке.
5. Шифр Полибия не обеспечивает защиты от атак по подмене символов. Если злоумышленник знает, какой символ заменен на какую пару символов, то он может изменить зашифрованное сообщение, подменяя один символ на другой.

В целом, шифр Полибия может быть полезным инструментом для учебных и игровых целей, но не рекомендуется использовать его для защиты конфиденциальной информации в реальных ситуациях.

# СПИСОК ЛИТЕРАТУРЫ:

* <https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82_%D0%9F%D0%BE%D0%BB%D0%B8%D0%B1%D0%B8%D1%8F>
* <https://questhint.ru/kvadrat-polibiya/>
* https://fib0.ru/metod-kvadrata-polibija-chto-jeto-takoe-i-kak-ispolzovat.html