МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра информационной безопасности

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Частотный криптоанализ шифра Цезаря

Студент гр. 6361 _____ Нерсисян А.

Преподаватель _____ Халиуллин Р.А.

Санкт-Петербург

2017

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Нерсисян А.

Группа 6361

Тема работы: Частотный криптоанализ шифра Цезаря

Задание:

Разработать на языке C++ приложение для частотного криптоанализа шифра Цезаря. Приложение должно иметь консольный интерфейс. Приложение должно читать входные данные из файла. Выходные данные должны сохраняться в файл. Приложение должно корректно обрабатывать ошибки, в том числе ошибки ввода/вывода, выделения/освобождения памяти.

Содержание пояснительной записки:

Введение, теоретическая часть, шифрование и расшифровывание, атака, методы криптоанализа, основные методы криптоанализа, дополнительные методы криптоанализа, реализация программы, Используемое ПО, результаты тестирования программы, описание функций, заключение, список использованных источников, приложение 1 − руководство пользователя, приложение 2 − блок-схема, приложение 3 − исходный код.

Предполагаемый объем пояснительной записки:

Не менее 25 страниц

Дата выдачи задания: 08.02.2017

Дата сдачи реферата: 26.05.2017

Дата защиты реферата: 02.06.2017

Студент _____ Нерсисян А.

Преподаватель _____ Халиуллин Р.А.

АННОТАЦИЯ

Шифр Цезаря — это вид шифра подстановки, в котором каждый символ в открытом тексте заменяется символом, находящимся на некотором постоянном числе позиций левее или правее него в алфавите.

Задача: разработать на языке C++ консольное приложение для частотного криптоанализа шифра Цезаря. В проекте описывается теоретическая составляющая, реализация консольного приложения для частотного криптоанализа шифра Цезаря, а также функции и их предназначение в данном приложении.

В отчете присутствует исходный код программы.

SUMMARY

In cryptography, a Caesar cipher is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet.

The task. Develop a C ++ console application for frequency Caesar cipher cryptanalysis. The project describes the theoretical component, the implementation of the console application for frequency cryptanalysis Caesar cipher, as well as the functions and their purpose in this application.

The report contains the source code of the program.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Шифр Цезаря− это шифр простой подстановки, при шифровании каждый символ текста «сдвигается» на некоторое постоянное расстояние в алфавите. Это расстояние − ключ. Обратная процедура расшифровки выполняется аналогично.

Задача состоит в том, что нужно расшифровывать текст без ключа. Для этого в данном проекте используется частотный анализ букв. В двух словах суть можно описать следующим образом. Необходимо сравнить частоты появления символов в шифре с эталонными. Потом, на основе частотного анализа определить ключ и расшифровывает текст.

После расшифровки нужно полученный результат записать в новый файл.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Шифрование и расшифровывание

Шифр Цезаря, также известный как шифр сдвига, код Цезаря или сдвиг Цезаря − это шифр простой подстановки, при шифровании каждый символ текста заменяется символом, находящимся на некотором постоянном расстоянии левее или правее в алфавите (см. рис. 1). Это расстояние и есть ключ. Например: шифрование с использованием ключа $k = 3$. Буква «E» «сдвигается» на три буквы вперёд и становится буквой «H», буква «Z», перемещённая на три буквы вперёд, становится буквой «C», и так далее:

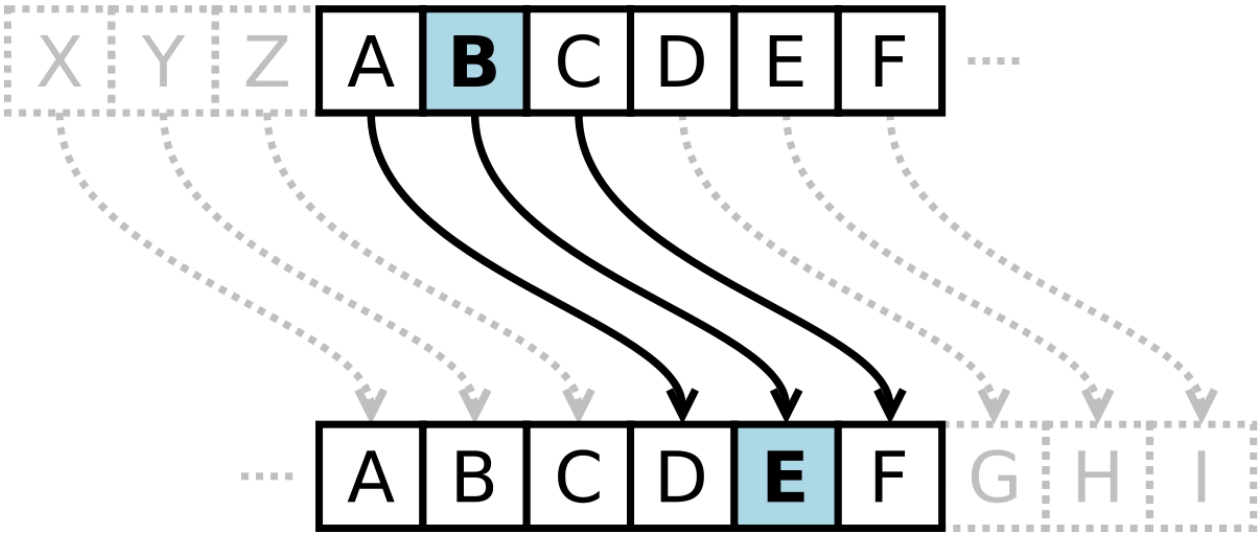


Рисунок 1 – алфавит замены с использованием ключа $k = 3$

Шифрованный текст получается путём замены каждой буквы оригинального текста соответствующей буквой шифрованного алфавита.

Оригинальный текст: *“ The quick brown fox jumps over the lazy dog.”*

Шифрованный текст: *“Qeb nrfzh yoltk clu grjmp lsbo qeb ixwv ald.”*

Обратная процедура расшифровки выполняется аналогично.

1.2. Атака

Пожалуй, лучше начать с того, что многократное шифрование никак не улучшает стойкость, так как применение шифров со сдвигом a и b эквивалентно применению шифра со сдвигом a + b. В математических терминах шифрование с различными ключами образует группу.

Шифр Цезаря может быть легко взломан даже в случае, когда взломщик знает только зашифрованный текст. Можно рассмотреть две ситуации:

1. Взломщик знает, что использовался шифр Цезаря, но не знает значение сдвига.
2. Взломщик знает (или предполагает), что использовался простой шифр подстановки, но не знает, что это — схема Цезаря.

В первом случае взлом шифра прост. Существует не так много вариантов значений сдвига (25 для английского языка), все они могут быть проверены методом грубой силы. Один из способов сделать это — выписать отрывок зашифрованного текста в столбец всех возможных сдвигов — техника, иногда называемая как «завершение простого компонента». Для обычного текста на естественном языке, скорее всего, будет только один вариант декодирования. Но, если использовать очень короткие сообщения, то возможны случаи, когда возможны несколько вариантов расшифровки с различными сдвигами. Например, зашифрованный текст «MPQY» может быть расшифрован как «aden», так и как «know» (предполагая, что открытый текст написан на английском языке). Точно также «ALIIP» можно расшифровать как «dolls» или как «wheel»; «AFCCP» как «jolly» или как «cheer».

Во втором случае шифр может быть взломан, используя те же самые методы что и для простого шифра подстановки, такие как частотный анализ. Используя эти методы, взломщик, вероятно, быстро заметит регулярность в решении и поймёт, что используемый шифр — это шифр Цезаря.

В обеих случаях применима еще один из серии грубой силы подход для взлома — проверить частоты встречаемости букв. Изобразив диаграммой (см. рис. 2 и таблицу 1) частоты встречаемости букв в зашифрованном тексте, и зная ожидаемое распределение букв для обычного текста на рассматриваемом языке, можно легко определить сдвиг, взглянув на смещение некоторых характерных черт на диаграмме. Этот метод известен как частотный анализ. Частотный анализ, частотный криптоанализ — один из методов криптоанализа, основывающийся на предположении о существовании нетривиального статистического распределения отдельных символов и их последовательностей как в открытом тексте, так и в шифротексте, которое, с точностью до замены символов, будет сохраняться в процессе шифрования и дешифрования.

Упрощённо, частотный анализ предполагает, что частота появления заданной буквы алфавита в достаточно длинных текстах одна и та же для разных текстов одного языка. При этом, в случае моноалфавитного шифрования, если в шифротексте будет символ с аналогичной вероятностью появления, то можно предположить, что он и является указанной зашифрованной буквой. Аналогичные рассуждения применяются к биграммам (двубуквенным последовательностям), триграммам и т. д. в случае полиалфавитных шифров. Утверждается, что вероятность появления отдельных букв, а также их порядок в словах и фразах естественного языка подчиняются статистическим закономерностям: например, пара стоящих рядом букв «ся» в русском языке более вероятна, чем «цы», а «оь» в русском языке не встречается. Анализируя достаточно длинный текст, зашифрованный методом замены, можно по частотам появления символов произвести обратную замену и восстановить исходный текст.

Как упоминалось выше, важными характеристиками текста являются повторяемость букв (количество различных букв в каждом языке ограничено), пар букв, то есть m (m-грамм), сочетаемость букв друг с другом,

чередование гласных и согласных, и некоторые другие особенности. Примечательно, что эти характеристики являются достаточно устойчивыми. Например, в тексте на английском языке частота букв E, T, (обычно наиболее частых), и Q, Z (обычно более редких) особенно различаются. Этот процесс можно автоматизировать, сделав, чтобы компьютерная программа оценивала, насколько хорошо фактическое распределение частот соответствует ожидаемому распределению.

Таблица 1 – Частота встречаемости букв в английском литературном тексте

Буква	M	W	F	G	Y	P	B	V	K	X	J	Q	Z
Частота, %	2,41	2,36	2,23	2,02	1,97	1,93	1,49	0,98	0,77	0,15	0,15	0,10	0,05

Буква	E	T	A	O	I	N	S	H	R	D	L	C	U
Частота, %	12,70	9,06	8,17	7,51	6,97	6,75	6,33	6,09	5,99	4,25	4,03	2,78	2,76

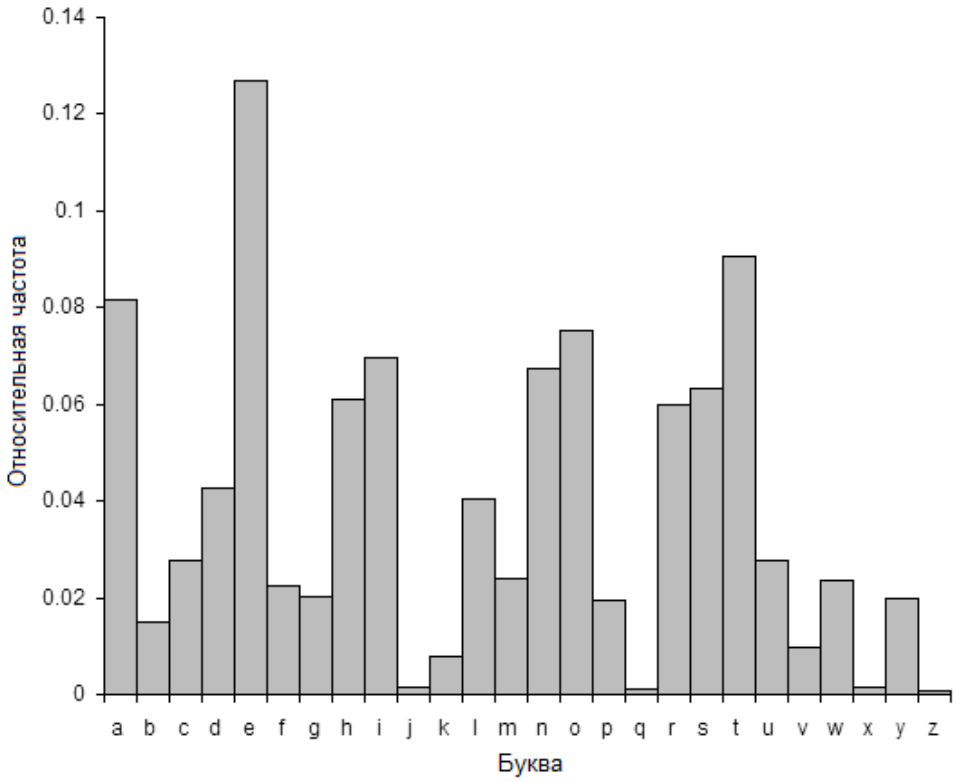


Рисунок 2 – Частота встречаемости букв в английском литературном тексте

1.3. Методы криптоанализа

Специалисты по компьютерной безопасности выделяют 4 основных и 2 дополнительных метода криптоанализа.

1.3.1. Основные методы криптоанализа

Атаки на основе шифротекста. Допустим, криптоаналитик обладает некоторым числом шифротекстов, полученных в результате использования одного и того же алгоритма шифрования. В этом случае криптоаналитик может совершить только атаку на основе шифротекста. Целью криптографической атаки в этом случае является нахождение как можно большего числа открытых текстов, соответствующих имеющимся шифротекстам, или, что ещё лучше, нахождение используемого при шифровании ключа.

Входные данные для подобного типа атак криптоаналитик может получить в результате простого перехвата зашифрованных сообщений. Если передача осуществляется по открытому каналу, то реализация задачи по сбору данных сравнительно легка и тривиальна. Атаки на основе шифротекста являются самыми слабыми и неудобными.

Атака на основе открытых текстов и соответствующих шифротекстов. Пусть в распоряжении криптоаналитика есть не только шифротексты, но и соответствующие им открытые тексты. Тогда существуют два варианта постановки задачи:

1. Найти ключ, использованный для преобразования открытого текста в шифротекст;
2. Создать алгоритм, способный дешифровать любое сообщение, закодированное с помощью этого ключа.

Получение открытых текстов играет решающую роль в осуществлении этой атаки. Открытые тексты извлекают из самых различных источников. Так, например, можно догадаться о содержимом файла по его расширению.

В случае взлома переписки можно сделать предположение, что письмо имеет структуру типа:

«Приветствие»

«Основной текст»

«Заключительная форма вежливости»

«Подпись»

Следовательно, атака может быть организована путём подбора различных видов «Приветствия» (например, «Здравствуйте!», «Добрый день» и т. д.) и/или «Заключительной формы вежливости» (таких как «С уважением», «Искренне Ваш» и т. п.). Легко заметить, что данная атака сильнее атаки на основе одного лишь шифротекста.

Атака на основе подобранного открытого текста. Для осуществления такого типа атаки криптоаналитику необходимо иметь не только какое-то количество открытых текстов и полученных на их основе шифротекстов. Помимо прочего в данном случае криптоаналитик должен обладать возможностью подобрать несколько открытых текстов и получить результат их шифрования.

Задачи криптоаналитика повторяют задачи для атаки на основе открытого текста, то есть получить ключ шифрования, либо создать дешифрующий алгоритм для данного ключа.

Получить входные данные для такого вида атаки можно, например, следующим образом:

1. Создать и отправить поддельное не зашифрованное сообщение якобы от одного из пользователей, которые обычно пользуются шифрованием;
2. В некоторых случаях можно получить ответ, в котором будет содержится зашифрованный текст, цитирующий содержание поддельного сообщения.

При осуществлении атаки подобного типа криптоаналитик имеет возможность подбирать блоки открытого текста, что при определённых

условиях может позволить получить больше информации о ключе шифрования.

Атаки на основе адаптивно подобранного открытого текста. Атака такого типа является более удобным частным случаем атаки на основе подобранного открытого текста. Удобство атаки на основе адаптивно подобранного открытого текста состоит в том, что помимо возможности выбирать шифруемый текст, криптоаналитик может принять решение о шифровании того или иного открытого текста на основе уже полученных результатов операций шифрования. Другими словами, при осуществлении атаки на основе подобранного открытого текста криптоаналитик выбирает всего один большой блок открытого текста для последующего шифрования, а потом на основе этих данных начинает взламывать систему. В случае организации адаптивной атаки криптоаналитик может получать результаты шифрования любых блоков открытого текста, чтобы собрать интересующие его данные, которые будут учтены при выборе следующих отправляемых на шифрование блоков открытого текста и так далее. Наличие обратной связи даёт атаке на основе адаптивно подобранного шифротекста преимущество перед всеми вышеперечисленными типами атак.

1.3.2. Дополнительные методы криптоанализа

Атака на основе подобранного шифротекста. Допустим, что у криптоаналитика имеется временный доступ к дешифрующему средству или устройству. В таком случае за ограниченный промежуток времени криптоаналитик может получить из известных ему шифротекстов соответствующие им открытые тексты, после чего криптоаналитику нужно будет приступать к взлому системы. При осуществлении подобного типа атаки цель взлома — получить ключ шифрования.

Сжато сформулировать эту задачу можно таким образом:

Дано: $C_1, P_1=D_k(C_1), C_2, P_2=D_k(C_2), C_3, P_3=D_k(C_3), \ldots, C_n, P_n=D_k(C_n),$

Где C_n — n-ый имеющийся шифротекст,

P_n — соответствующий C_n открытый текст

D_k — функция дешифрования при помощи ключа k.

Найти: используемый ключ шифрования k.

Интересным может быть тот факт, что атака на основе подобранного шифротекста также может носить название «Атаки в обеденное время» (lunchtime attack) или «Ночной атаки» (midnight attack). Скажем, в названии «Атаки в обеденное время» отражается тот факт, что легитимный пользователь может оставить свой компьютер с функцией дешифрования без присмотра на время обеда, а криптоаналитик может этим воспользоваться.

Атака на основе подобранного ключа. Вопреки своему названию атака на основе подобранного ключа не подразумевает под собой того, что криптоаналитик занимается простым перебором ключей в надежде найти нужный. Атака такого типа строится на том, что криптоаналитик может наблюдать за работой алгоритма шифрования, в котором используются несколько ключей. Криптоаналитик изначально ничего не знает о точном значении ключей, зато ему известно некоторое математическое отношение, связывающее между собой ключи. Примером тому может служить ситуация, когда криптоаналитик выяснил, что последние 80 битов у всех ключей одинаковы, хотя сами значения битов могут быть неизвестными.

2. РЕАЛИЗАЦИЯ ПРОГРАММЫ

2.1. Используемое программное обеспечение

Для разработки данного приложения в качестве IDE (Integrated Development Environment − Интегрированная среда разработки) была выбрана Microsoft Visual Studio Community 2015 со встроенным компилятором (Microsoft Visual C++).

Язык программирования: C++.

Операционная система: Microsoft Windows 10 Pro.

2.2. Результаты тестирования программы

Для тестирования были использованы несколько книг из мировой классической литературы:

1. “The Picture of Dorian Gray” by Oscar Wilde
2. “Pride and Prejudice” by Jane Austen
3. “On the Road” by Jack Kerouac

При тестировании все 3 книги были правильно расшифрованы. На рисунке 3 демонстрируется работа программы.

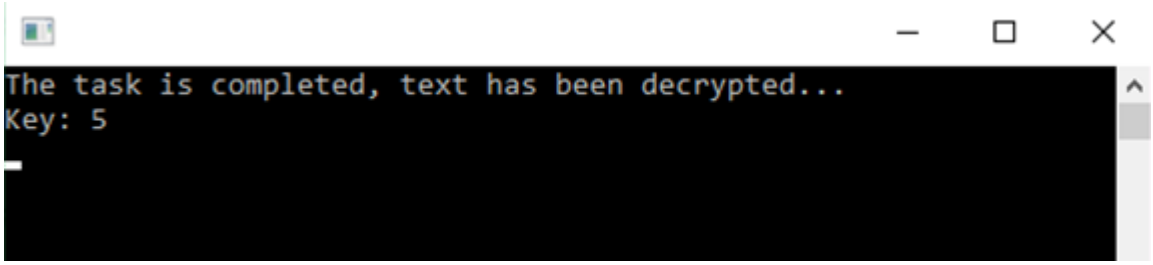


Рисунок 3 – Работа программы

В качестве входных данных для программы используется зашифрованный текст в файле crypted_text.txt, содержимое которого показано на рисунке 4.

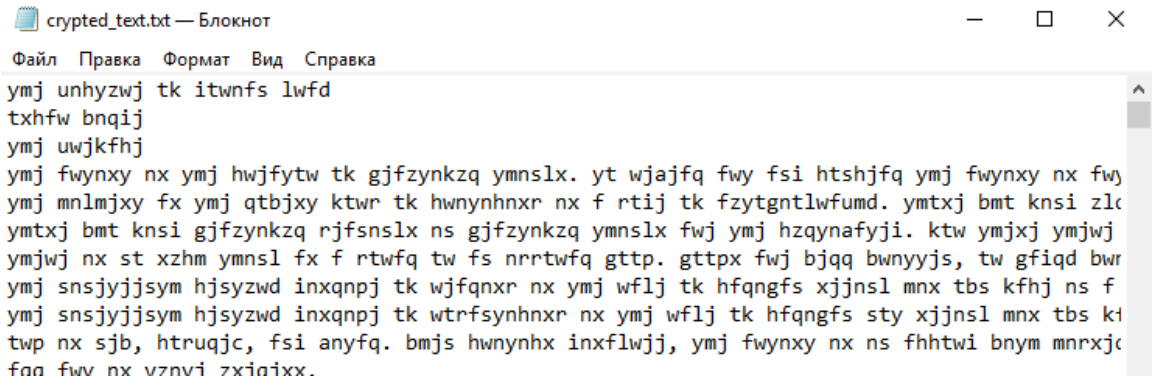


Рисунок 4 – Зашифрованный текст

Файл decrypted_text.txt (создается программой после анализа и расшифровывания) содержит в себе расшифрованный текст, показанный на рисунке 5.

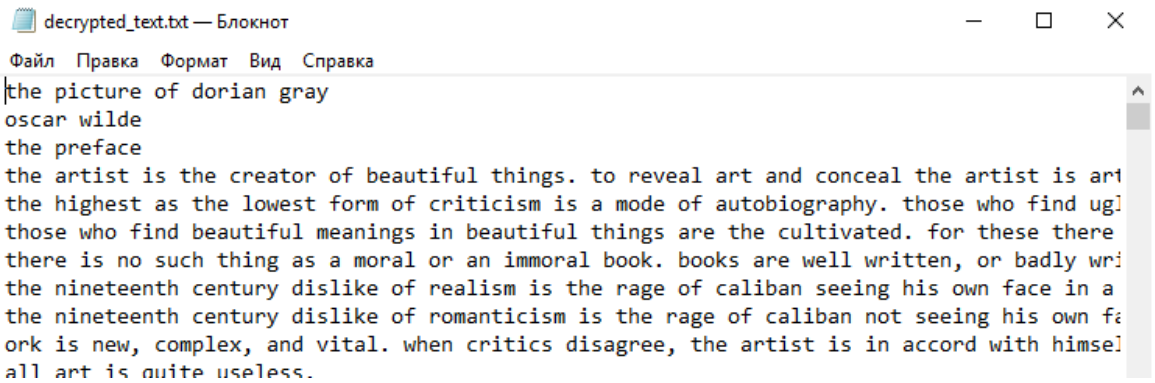


Рисунок 5 – Расшифрованный текст

2.3. Описание функций

1. Функция decode

Функция decode расшифровывает текст из файла crypted_text.txt полученным из в качестве аргументов ключом и алфавитом.

Исходный код функции находится в файле Source.cpp.

Объявление функции:

int decode(int key, string alphabet)

Тип функции:

int

Аргументы функции:

key − ключ, с которым расшифровывается текст,

тип аргумента: int

alphabet − алфавит для расшифровывания текста,

тип аргумента: string

Возвращаемое значение:

Функция возвращает 0 при корректном завершении и 1 при возникновении ошибки.

2. Функция main

Функция main получает управление при запуске программы. Исходный код функции находится в файле Source.cpp.

Объявление функции:

int main()

Тип функции:

int

Возвращаемое значение:

Функция возвращает 0 при корректном завершении программы и 1 при возникновении ошибки.

ЗАКЛЮЧЕНИЕ

В данном проекте была реализована программа частотного криптоанализа шифра Цезаря на языке C++. Программа считывает зашифрованный текст из файла, анализирует, находит ключ, расшифровывает и записывает расшифрованный текст в новый файл. Приложение имеет консольный интерфейс и корректно обрабатывает ошибки, в том числе ошибки ввода/вывода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Частотный анализ // Википедия.
   URL: https://ru.wikipedia.org/wiki/Частотный_анализ (дата обращения: 22.05.2017).
2. Шифр Цезаря // Википедия.
   URL: https://ru.wikipedia.org/wiki/Шифр_Цезаря (дата обращения: 22.05.2017)
3. Методы криптоанализа классических шифров / А.Г. Ростовцев, Н.В. Михайлова.
   URL: http://sp.sz.ru/cryptoanalysis_.html (дата обращения 22.05.2017)
4. Брюс Шнайер. Прикладная криптография, Протоколы, алгоритмы и исходные тексты на языке С — 2-е издание. М.: ТРИУМФ. 2005. 424 стр.
5. Введение в криптографию / Под общ. ред. В. В. Ященко. — 4-е изд., М.: МЦНМО, 2012. 348 стр.

ПРИЛОЖЕНИЕ 1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

1. Краткое описание программы

Программа предназначена для расшифровывания (без ключа) текста зашифрованным шифром Цезаря. Максимальная **теоретическая** длина текста 16 906 657 589 букв (примерно 9.3 млн страниц А4). Но не стоит беспокоится, так как среднестатистическая книга содержит около 400 000 букв.

2. Минимальные системные требования

Операционная система: Windows 10 Pro или более поздние версии

Процессор: 1 ГГц или SoC

Оперативная память: 1 ГБ (для 32-разрядных систем) или 2 ГБ (для 64-разрядных систем)

Место на жестком диске: 16 ГБ (для 32-разрядных систем) или 20 ГБ (для 64-разрядных систем)

Видеоадаптер: DirectX версии не ниже 9 с драйвером WDDM 1.0.

Дисплей: 800 x 600.

3. Установка программы

Копируйте исполняемый файл **decoder.exe** в удобную для вас директорию в ПК. Обратите внимание, что программа в ходе работы будет создавать новый файл в той же директории откуда запускался. Также стоит отметить, что программа не будет работать с лазерного диска (CD/DVD/Blu-Ray и т.д.).

4. Работа с программой

Создайте новую папку на рабочем столе и скопируйте туда программу (decoder.exe). Скопируйте в эту же папку текстовой файл с зашифрованным шифром цезаря текстом. Правой кнопкой мыши кликайте один раз на файл с

зашифрованным текстом (см. рис. 6). Из открывающегося меню выберите пункт переименовать, после чего введите «crypted_text.txt» (без кавычек) и нажмите кнопку Enter (см. рис. 7).

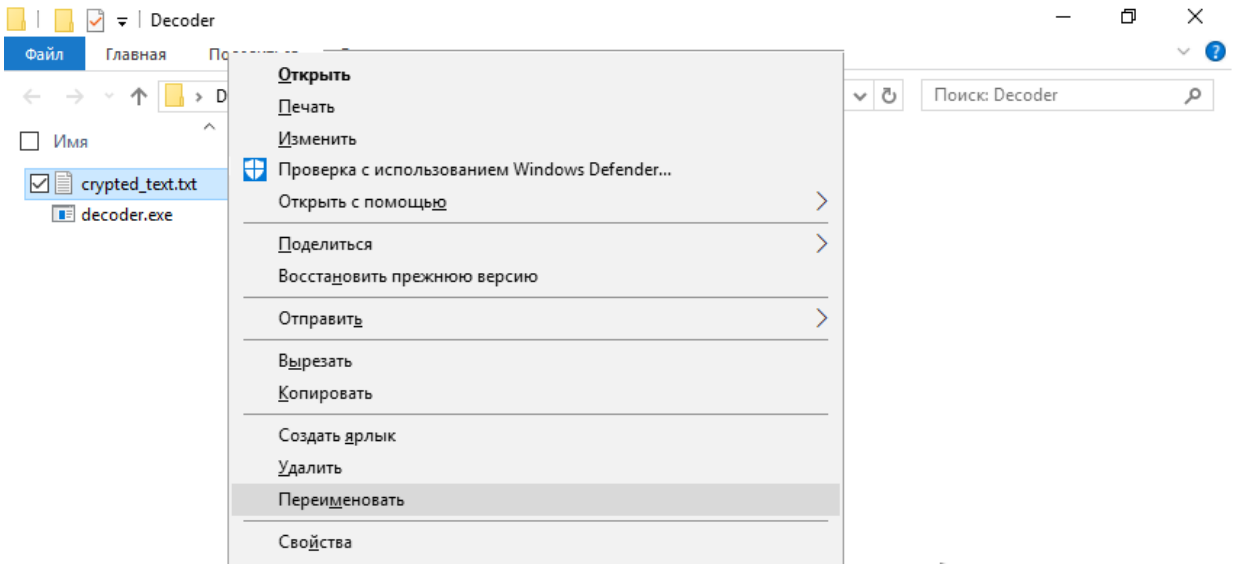


Рисунок 6 – Переименование текстового файла

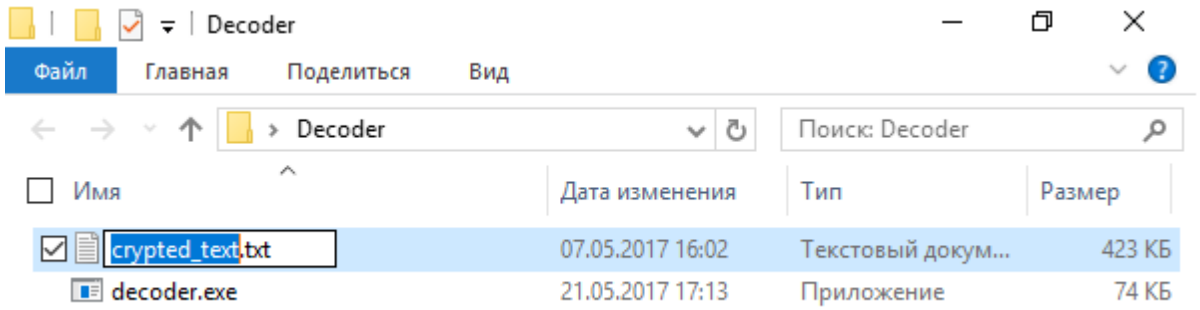


Рисунок 7 – Ввод названия текстового файла

В итоге получается результат показанная на рисунке 8.

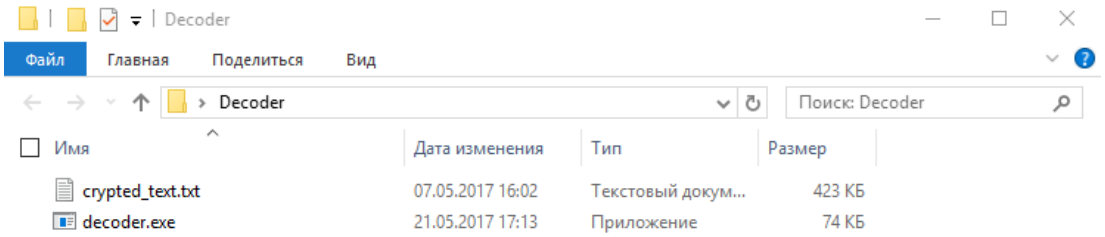


Рисунок 8 – Папка с программой и зашифрованным текстовым файлом

Запускайте программу decoder.exe двойным нажатием левой кнопки мыши. Через несколько секунд программа выводит сообщение об успешном декодировании и ключ шифрования/расшифровывания текста (см. рис. 9). При возникновении ошибок проверьте имя текстового файла с зашифрованным текстом.

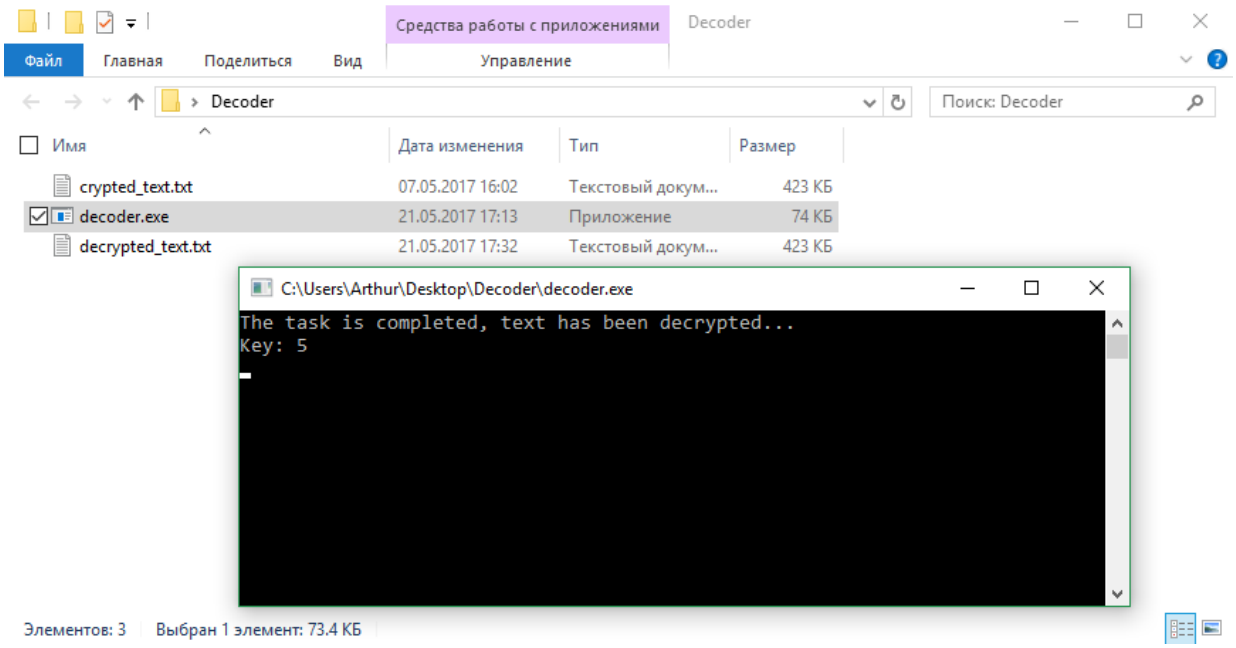


Рисунок 9 – Работа программы

В директории, где находятся программа и текстовой файл с зашифрованным текстом, появится новый текстовой файл с названием «decrypted_text.txt». В файле записан расшифрованный текст (см. рис. 10).

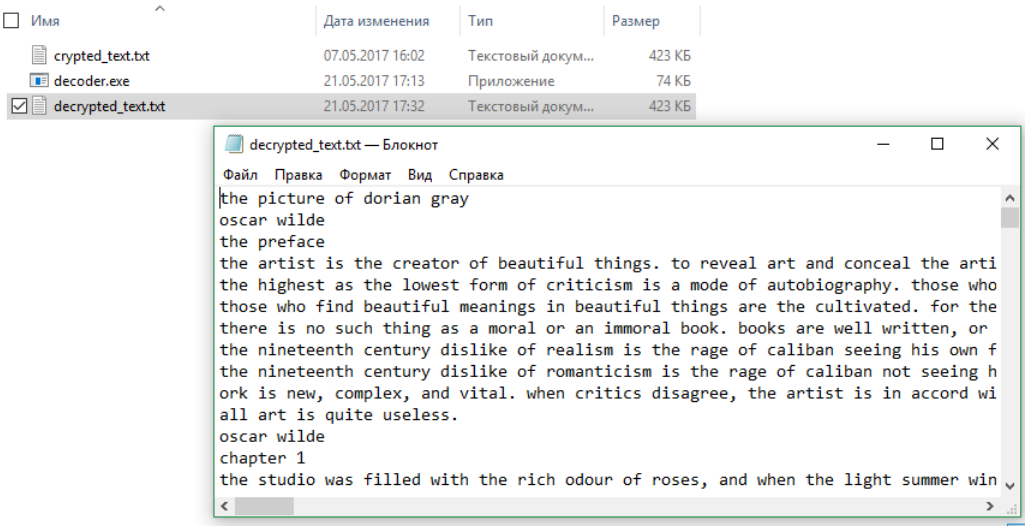
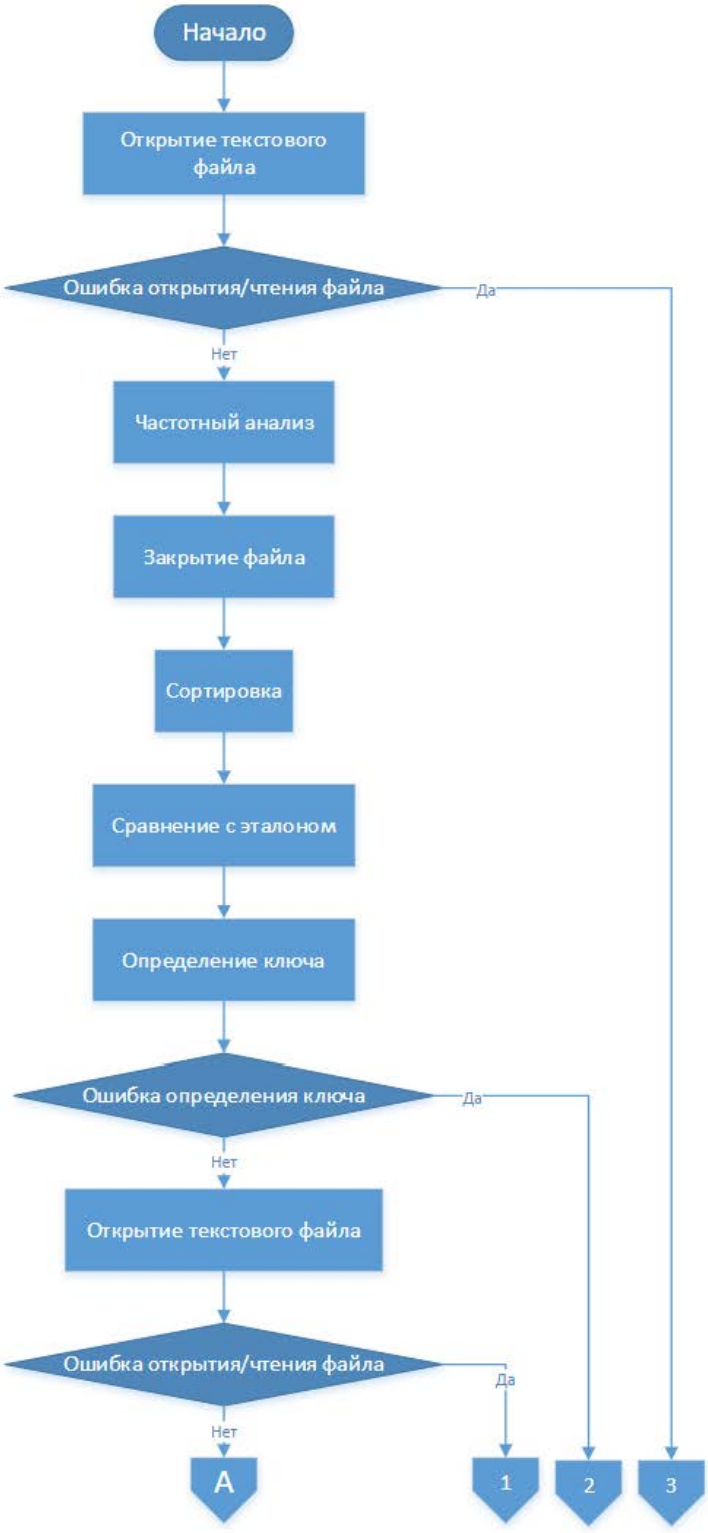
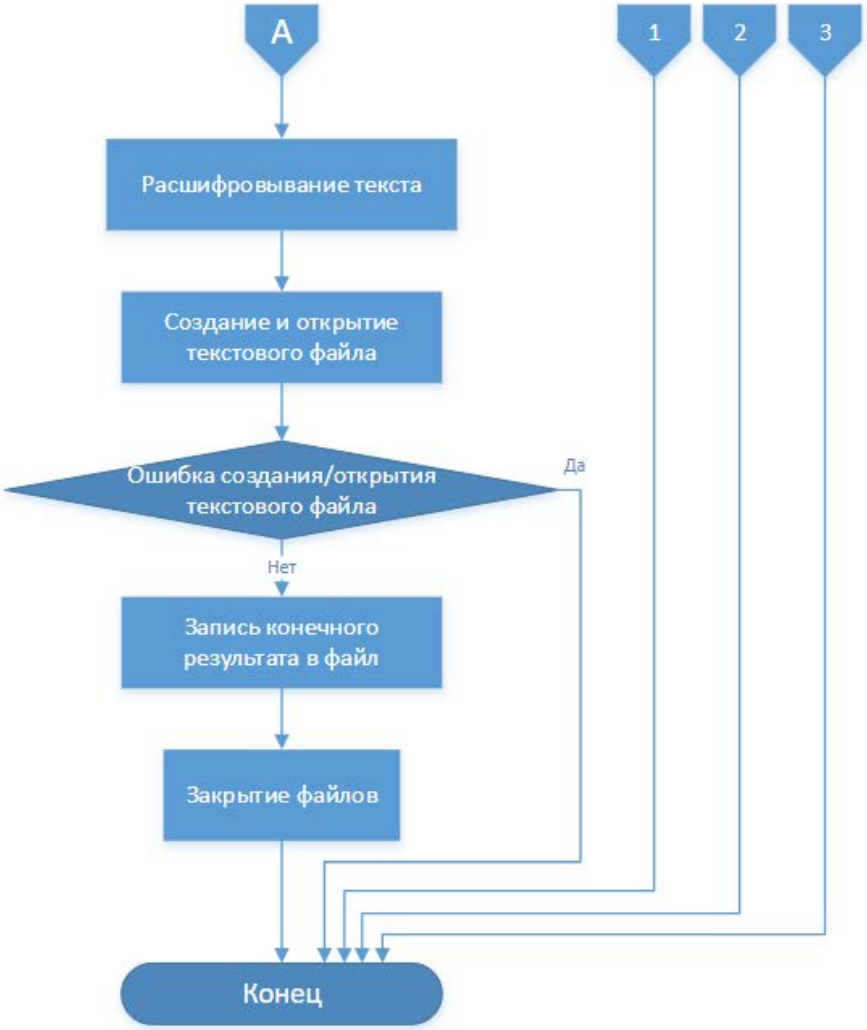


Рисунок 10 – Текстовой файл с расшифрованным текстом



					Приложение 2. Блок-схема
Изм.	*Лист*	*№ докум*	*Подпись*	*Дата*	
Разраб.		Нерсисян А.			
Пров.		Халиуллин Р.А			
Реценз.		Халиуллин Р.А			
Н. Контр.		Халиуллин Р.А			
Утв		Халиуллин Р.А			

Литера	*Лист*	*Листов*
у	*1*	*2*



					Приложение 2. Блок-схема
Изм.	*Лист*	*№ докум*	*Подпись*	*Дата*	
Разраб.		Нерсисян А.			
Пров.		Халиуллин Р.А			
Реценз.		Халиуллин Р.А			
Н. Контр.		Халиуллин Р.А			
Утв		Халиуллин Р.А			

Литера	*Лист*	*Листов*
у	*2*	*2*

ПРИЛОЖЕНИЕ 3. ИСХОДНЫЙ КОД ПРОГРАММЫ

Source.cpp:

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>

using namespace std;

int decode(int key, string alphabet)
{
     fstream crypted_text_file_1;
     crypted_text_file_1.open("crypted_text.txt");
     if (crypted_text_file_1.fail()) {
          cout << "Error! File does not open!";
          return 1;
     };
     string buffer;
     getline(crypted_text_file_1, buffer,
(char)crypted_text_file_1.eof());
     for (int i = 0; i < buffer.size(); i++) {
          for (int j = 0; j < alphabet.size(); j++) {
               if (buffer[i] == alphabet[j]) {
                    buffer[i] = alphabet[(j +
alphabet.size() - key % alphabet.size()) %
alphabet.size()];
                    break;
               };
          };
     };
     fstream decryptet_text_file;
     decryptet_text_file.open("decrypted_text.txt",
ios::trunc | ios::out);
     if (decryptet_text_file.fail()) {
          cout << "Error! File does not open!";
          return 1;
     };
     decryptet_text_file.write(buffer.c_str(),
buffer.size());
     crypted_text_file_1.close();
     decryptet_text_file.close();
```

```
return 0;
}

int main()
{
     fstream crypted_text_file("crypted_text.txt",
ios::binary | ios::in);
     if (!crypted_text_file)
     {
          printf("Error! File does not open!\n");
          return 1;
     }
     char ch;
     int i = 0, j = 0;
     long long characters[2][26] = { {
'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o'
,'p','q','r','s','t','u','v','w','x','y','z' },{
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 } };
     while (crypted_text_file.get(ch))
     {
          switch (ch)
          {
          case 'a':
               characters[1][0]++;
               break;
          case 'b':
               characters[1][1]++;
               break;
          case 'c':
               characters[1][2]++;
               break;
          case 'd':
               characters[1][3]++;
               break;
          case 'e':
               characters[1][4]++;
               break;
          case 'f':
               characters[1][5]++;
               break;
          case 'g':
               characters[1][6]++;
```

```
    break;
case 'h':
    characters[1][7]++;
    break;
case 'i':
    characters[1][8]++;
    break;
case 'j':
    characters[1][9]++;
    break;
case 'k':
    characters[1][10]++;
    break;
case 'l':
    characters[1][11]++;
    break;
case 'm':
    characters[1][12]++;
    break;
case 'n':
    characters[1][13]++;
    break;
case 'o':
    characters[1][14]++;
    break;
case 'p':
    characters[1][15]++;
    break;
case 'q':
    characters[1][16]++;
    break;
case 'r':
    characters[1][17]++;
    break;
case 's':
    characters[1][18]++;
    break;
case 't':
    characters[1][19]++;
    break;
case 'u':
    characters[1][20]++;
```

```
                break;
            case 'v':
                characters[1][21]++;
                break;
            case 'w':
                characters[1][22]++;
                break;
            case 'x':
                characters[1][23]++;
                break;
            case 'y':
                characters[1][24]++;
                break;
            case 'z':
                characters[1][25]++;
                break;
            default:
                break;
            }
        }
        long long buf, buf1;
        for (int j = 0; j < 26; j++)
        {
            for (int i = 0; i < 26 - 1; i++)
            {
                if (characters[1][i] < characters[1][i + 1])
                {
                    buf = characters[1][i];
                    characters[1][i] = characters[1][i + 1];
                    characters[1][i + 1] = buf;
                    if (buf)
                    {
                        buf1 = characters[0][i];
                        characters[0][i] = characters[0][i +
1];
                        characters[0][i + 1] = buf1;
                    }
                }
            }
        }
        double symbol_sum = 0;
```

```
    for (int numb = 0; numb < 26; numb++) symbol_sum +=
characters[1][numb];
    double native_e = 12.702, native_t = 9.056, native_a =
8.167, native_o = 7.507;
    double analyzed_per1 = 0, analyzed_per2 = 0,
analyzed_per3 = 0, analyzed_per4 = 0;
    analyzed_per1 = (characters[1][0] * 100) / symbol_sum;
    analyzed_per2 = (characters[1][1] * 100) / symbol_sum;
    analyzed_per3 = (characters[1][2] * 100) / symbol_sum;
    analyzed_per4 = (characters[1][3] * 100) / symbol_sum;
    float e_e = 0, t_t = 0, a_a = 0, o_o = 0;
    if (fabs(analyzed_per1 - native_e) < 0.600)
        e_e = analyzed_per1;
    if (fabs(analyzed_per2 - native_t) < 0.300)
        t_t = analyzed_per2;
    if (fabs(analyzed_per3 - native_a) < 0.150)
        a_a = analyzed_per3;
    if (fabs(analyzed_per4 - native_o) < 0.400)
        o_o = analyzed_per4;
    int cesar_key_0, cesar_key_1, cesar_key_2,
cesar_key_3;
    string alphabet = "abcdefghijklmnopqrstuvwxyz";
    char e_ee, t_tt, a_aa, o_oo;
    if (e_e)
    {
        e_ee = characters[0][0];
        cesar_key_0 = (e_ee - 'e') % alphabet.length();
    }
    if (t_t)
    {
        t_tt = characters[0][1];
        cesar_key_1 = (t_tt - 't') % alphabet.length();
    }
    if (a_a)
    {
        a_aa = characters[0][2];
        cesar_key_2 = (a_aa - 'a') % alphabet.length();
    }
    if (o_o)
    {
        o_oo = characters[0][3];
        cesar_key_3 = (o_oo - 'o') % alphabet.length();
```

```
    }
    int Get_key;
    if ((cesar_key_0 == cesar_key_1) && (((cesar_key_1 ==
cesar_key_2) || (cesar_key_1 != cesar_key_2)) &&
((cesar_key_2 == cesar_key_3) || (cesar_key_2 !=
cesar_key_3))))
        Get_key = cesar_key_0;
    if ((cesar_key_0 != cesar_key_1) && (cesar_key_1 ==
cesar_key_2 == cesar_key_3))
        Get_key == cesar_key_1;
    if ((cesar_key_0 != cesar_key_1) && (cesar_key_0 !=
cesar_key_2) && (cesar_key_0 != cesar_key_3) &&
(cesar_key_1 != cesar_key_2) && (cesar_key_1 !=
cesar_key_3) && (cesar_key_2 && cesar_key_3))
    {
        cout << "Error! key not found..." << endl;
        cin.get();
        return 1;
    }
    crypted_text_file.close();

    decode(Get_key, alphabet);

    cout << "The task is completed, text has been
decrypted..." << endl;
    cout << "Key: " << Get_key << endl;
    cin.get();
    return 0;
}
```