МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего образования

«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»

ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ

Кафедра компьютерной инженерии и моделирования

**ТЕМА КУРСОВОЙ РАБОТЫ**

Курсовая работа

по дисциплине «Программирование»

студента 1 курса группы ИВТ-б-о-191(1)

Новикова Валерий Олеговича

направления подготовки 09.03.04 «Информатика и вычислительная техника»

|  |  |  |
| --- | --- | --- |
| Научный руководитель  ассистент кафедры компьютерной инженерии и моделирования | \_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) | Чабанов В.В. |

Симферополь, 2020

**РЕФЕРАТ**

Разработка проекта «Система тестирования знаний». – Симферополь – ФТИ КФУ им. Вернадского, 2020. 25 с, 21 ил, 5 ист.

Объект разработки – симулятор шифровальной машины Энигма.

Цель работы – приложение, которое имеет интерфейс и функционал, схожий с шифровальной машиной Энигма.

Для создания проекта использовались языки программирования С++ для клиентской части и C# для серверной. Средой разработки являлась Microsoft Visual Studio 2019. Был использован .NET Framework, где исполняющей средой был выбран CLR.

Результаты и их новизна – в результате выполнения проекта «SimEnigma» было создано приложение, которое выполняет шифрование введенного вами сообщения по алгоритму Энигма

Значимость работы и выводы – шифрование данных одна из самых острых проблем человечества и при усовершенствовании данного проекта он может быть использован для сокрытия информации от злоумышленников

В ходе выполнения курсовой работы

изучены следующие языки программирования: C++ и С#, был получен опыт работы с клиент-серверной архитектурой.

Область применения, рекомендации по использованию результатов работы - шифрование текстовых сообщений

ПРОГРАММИРОВАНИЕ,С++,КЛИЕНТ,СЕРВЕР,С#

**О**ГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc51519085)

[ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ 5](#_Toc51519086)

[1.1 Цель проекта 5](#_Toc51519087)

[1.2 Существующие аналоги 5](#_Toc51519088)

[1.3 Основные отличия от аналогов 5](#_Toc51519089)

[1.4 Техническое задание 6](#_Toc51519090)

[ГЛАВА 2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ 8](#_Toc51519096)

[2.1 Анализ инструментальных средств 8](#_Toc51519097)

[2.2 Описание алгоритмов, описание структур данных, описание основных модулей 8](#_Toc51519098)

[ГЛАВА 3 ТЕСТИРОВАНИЕ ПРОГРАММЫ 17](#_Toc51519099)

[3.1 Тестирование интерфейса пользователя и юзабилити 17](#_Toc51519100)

[3.2 Тестирование исходного кода. 20](#_Toc51519101)

[ГЛАВА 4 ПЕРСПЕКТИВЫ ДАЛЬНЕЙШЕГО РАЗВИТИЯ ПРОЕКТА 21](#_Toc51519102)

[4.1 Перспективы технического развития 21](#_Toc51519103)

[4.2 Перспективы монетизации 21](#_Toc51519104)

[ЗАКЛЮЧЕНИЕ 22](#_Toc51519105)

[ЛИТЕРАТУРА 23](#_Toc51519106)

ВВЕДЕНИЕ

Проблема сохранения информации стояла перед людьми всегда, хотя для подавляющего большинства членов общества она была мало видна. С появлением информационных и коммуникационных технологий задача многократно усложнилась — помимо сохранения информации на традиционных аналоговых носителях (что всегда было делом нелегким и дорогостоящим), необходимо сохранять еще и электронную (цифровую) информацию, рожденную как в электронном виде, так и полученную в результате оцифровки информации на аналоговых носителях. И для защиты всей этой информации было придумано шифрование.

Шифрование — обратимое преобразование информации в целях сокрытия от [неавторизованных](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) лиц, с предоставлением, в это же время, [авторизованным](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) пользователям доступа к ней. Главным образом, шифрование служит задачей соблюдения [конфиденциальности](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%84%D0%B8%D0%B4%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C) передаваемой информации. Важной особенностью любого алгоритма шифрования является использование [ключа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)), который утверждает выбор конкретного преобразования из совокупности возможных для данного алгоритма.

В своем проекте я поставил задачу рассмотреть один из самых известных механизмов шифрования информации под названием «Энигма», который был известен во время Второй Мировой Войны. Главной целью является создать симулятор этой шифровальной машины, который будет работать по тому же принципу, но в цифровом виде, так как в современном мире это более удобно, нежели использовать граммосткий аппарат.

ГЛАВА 1  
ПОСТАНОВКА ЗАДАЧИ

**1.1 Цель проекта**

В ходе разработки данного проекта планируется создать цифровой аналог шифровальной машины «Энигма», который будет более легок в использовании, нежели его оригинал. Приложение должно быть компактное, но иметь возможность выполнять шифровку и дешифровку информации. Во время работы над этим проектом также планируется закрепить знания в языке программирования C++ и применить C# для написания сервера к программе.

**1.2 Существующие аналоги**

Существует множество различный онлайн симуляторов, которые можно найти в интернете. Одним из лучших симуляторов является сайт: <http://enigmaco.de/enigma/enigma.html>.

Но также имеются и симуляторы на android устройства, как например Enigma simulator.

Из приложений на ПК существует данный аналог:

<http://www.radioscanner.ru/files/miscsoftware/file15849/>

**1.3 Основные отличия от аналогов**

Сама шифровальная машина «Энигма» имеет достаточно сложное строение, и разобраться в принципе ее работы достаточно проблематично. И большинство цифровых аналогов этого аппарата имеют схожий интерфейс, что может ввести в заблуждение многих людей, решивших заняться шифрованием этим методом. Поэтому я считаю одной из главных особенностей своей программы простоту интерфейса. В оригиналах шифровальной машины имеется две клавиатуры, где первая эта сами кнопки для ввода, вторая клавиатура - это набор индикаторов, где подсвечивается соответствующий нажатой вами кнопки зашифрованный символ. В ходе шифрования просто можно проглядеть зашифрованный символ, что может доставить неудобства. В своей же программе я убрал данную клавиатуру. С одной стороны, мой проект отличается от оригинала шифровальной машины, но с другой стороны это, как по мне, удобнее в использовании. В моем проекте используется одно «окошко», в котором и будет выводиться результат работы программы. Также есть окно для ввода сразу нескольких символов, которые программы разделяет по одному символу и проводит всё те же действия шифрования и выводит в тоже «окошко» для вывода. Причем пока вы не нажмете на следующее действие, результат прошлого будет отображаться в приложении.

## 1.4 Техническое задание

В этом разделе вы составляете для себя ТЗ, так, как если бы вы были заказчиком продукта. Тут перечисляется структура будущего ПО, основные возможности, привила (для игры), желаемый внешний вид, хотелки, ограничения и т.д.

Начнем с необходимых для использования системных требований:

Для корректной работы программы необходим персональный компьютер со следующими требованиями к составу и параметрам технических средств:

* Версия Windows - XP и выше.
* процессор Intel Pentium или совместимый,
* объем свободной оперативной памяти 256 Мб,
* объем необходимой памяти на жестком диске 200 Мб,
* стандартный монитор,
* стандартная клавиатура,
* манипулятор «мышь».

## Требования к функциональным характеристикам

### Программа должна корректно работать, без вылетов.

### Должно запускаться окно с приложением.

### Кнопки роторов должны прокручивать алфавит

### Все кнопки клавиатуры в приложении должны выполнять свои функции.

* При нажатии кнопки "V", введенное сообщение должны быть закодировано в соответствии с алгоритмом

Данная программа имеет одно, но при этом достаточно весомое ограничение: человек, использующий данное приложение должен минимально понимать принцип устройства шифровальной машины для успешного шифрования и дешифрования.

Сама программа имеет достаточно узкую специализацию, которой является шифровка и дешифровка сообщений через алгоритм «Энигма». В самом алгоритме используются заранее забитые в код роторы, имеющие определенную последовательность символов на барабане. Если использовать другую программу для расшифровки сообщения, зашифрованного в моей программе, то понадобиться знать порядок расстановки этих роторов и набор их элементов. Внешний вид у программы прост, не имеет отвлекающего интерфейса, чтобы можно было полностью сосредоточиться на основном функционале программы.

ГЛАВА 2  
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

## 2.1 Анализ инструментальных средств

Тут перечисляются библиотеки, модули, фреймворки и другие программные средства, применяемые в процессе выполнения курсовой работы, с указанием причины выбора именно этих инструментальных средств.

В начале работы с проектом основным вопросом стал выбор фреймворка для разработки приложения и приложение для разработки. Для разработки своего проекта я выбрал .NET Framework; .NET Framework — [программная платформа](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0#%D0%BA%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), выпущенная компанией [Microsoft](https://ru.wikipedia.org/wiki/Microsoft) в [2002 году](https://ru.wikipedia.org/wiki/2002_%D0%B3%D0%BE%D0%B4). Основой платформы является общеязыковая среда исполнения [Common Language Runtime (CLR)](https://ru.wikipedia.org/wiki/Common_Language_Runtime" \o "Common Language Runtime), которая подходит для разных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду. В этом фреймворке я смог совместить клиентскую часть, написанную на C++ и серверную часть, написанную на языке C#. В качестве среды разработки я выбрал Microsoft Visual Studio, так как для меня это является наиболее привычной средой разработки. Клиентская часть заключается в создании интерфейса приложения и возможностях переключения режимов шифровки и дешифровки, а также для переключения стартового положения роторов. На сервере же у нас храниться сам алгоритм шифрования, куда мы и отправляем введенное сообщение. Так как в этом проекте приходиться много работать с различными символами и строками, то нам нужна библиотека <string>

## 2.2 Описание алгоритмов, описание структур данных, описание основных модулей

В первую очередь, в создании приложения, я решил создать примерный интерфейс, в котором будут присутствовать кнопки и лейблы для вывода на них текста, помимо этого будет присутствовать одно окно для ввода текста с клавиатуры вашего устройства и два RadioButton для переключения режимов шифровки и дешифровки. В итоге получен данный интерфейс:

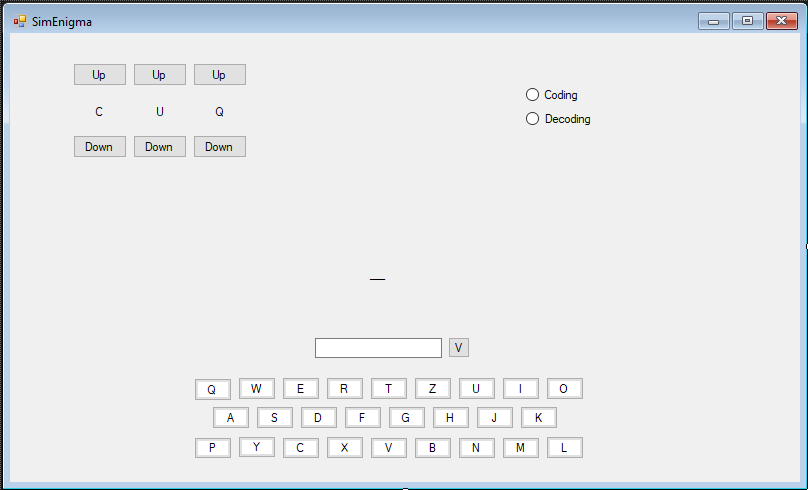


Рисунок 2.2.1 - внешний вид приложения

При этом, при создании каждого объекта, в коде автоматически прописываются его характеристики, в которое входит размер, расположение в программе, текст в данном объекте и еще некоторые параметры. Как пример приведу описание второй созданной кнопки в моей программе:

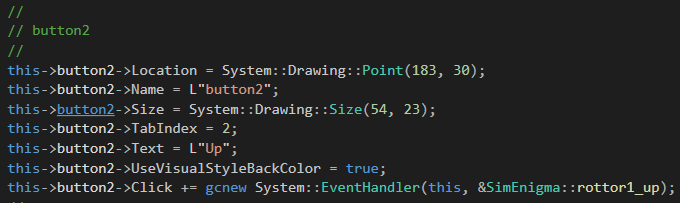


Рисунок 2.2.2 - описание созданной второй кнопки

Первым блоком в моей программе рассмотрим роторы, которые находятся в верхнем правом углу. Там имеется 3 лейбла, в которых будет выведен символ ротора, в зависимости от его положения. И сверху и снизу этих лейблов имеется по две кнопки, на которых написано «Up» и «Down». Это кнопки будут переключать наши роторы вперед и назад по алфавиту соответственно. Вот так будет выглядеть код для переключения буквы вперед для первого ротора:

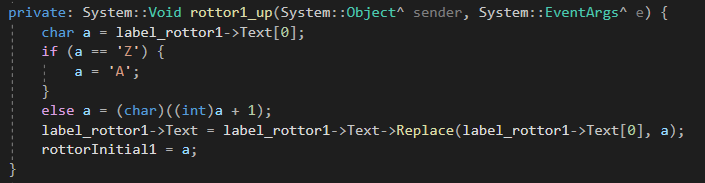


Рисунок 2.2.3 - код прокрутки ротора пользователем

Далее рассмотрим блок кода, в котором описывается переключения между режимами кодирования и декодирования. Алгоритм кодирования и декодирования абсолютно одинаков, но для декодирования зашифрованного сообщения необходимо начать его вводить при таком же начальном положении роторов, что и было при начале шифровки, то есть переключение режимов кодирования и декодирования - это возвращение начального выставленного пользователем положения роторов. В коде это выглядит так:

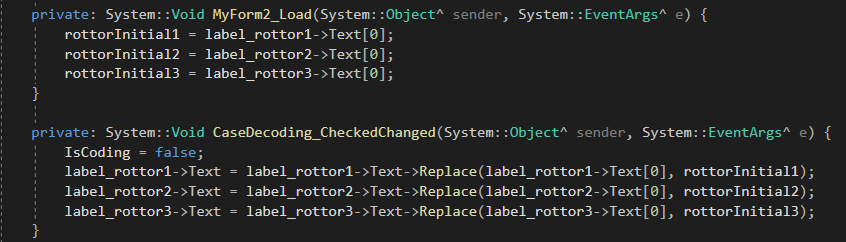


Рисунок 2.2.4 - блок кода о переключение режимов

Теперь перейдем к самому алгоритму шифрования, который является достаточно сложным для новичков, но на самом деле алгоритм требует работу с индексами букв в алфавите, если учесть, что алфавит является массивом. Так как алгоритм подразумевает переход с одного ряда символов на другой, в соответствии с выставленными роторами, то на C# был написан класс, в котором оформляется переход между этими столбцами. В моем случаем скрытые роторы выглядят так:

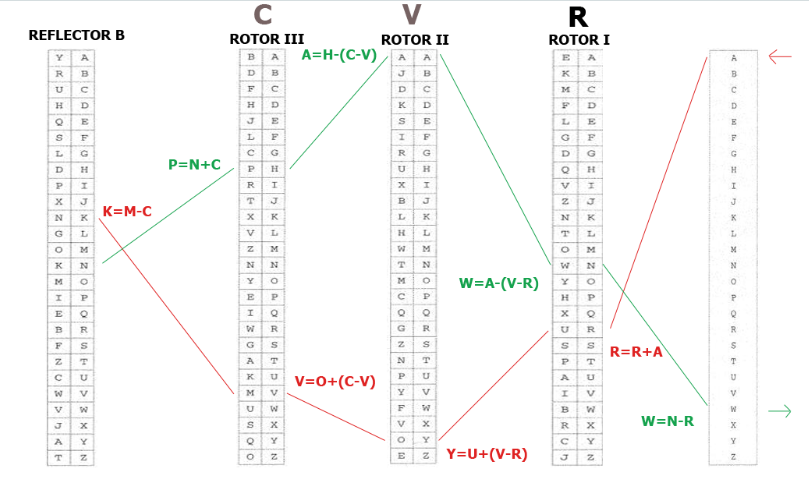


Рисунок 2.2.5 - соответствующие роторы в моем случае

А сам код для перехода будет выглядеть так и находится он в отдельном файле «rottor.cs». Эти функции будут выполнять переход при их вызове, только одна будет работать при первом проходе по всем роторам, другая при обратном проходе через них, ну и 3 функция возвращает текущее значение.

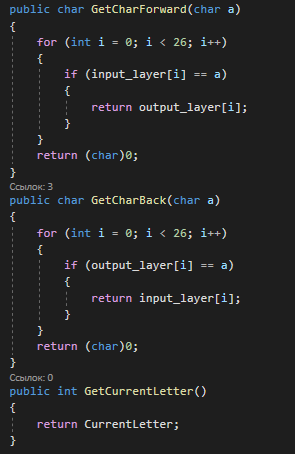


Рисунок 2.2.6 - переход символов в соответствующих роторах

Собственно, все скрытые роторы и записываем в виде строки, которая представляет собой массив символов, и в том же районе кода мы вызываем функцию прокрутки роторов при введении символов и там, внутри функции указываем символы, при проходе которых будут вращаться соседние символы, пишем мы это в основном классе нашего сервера, в котором и собраны все функции алгоритма шифровки. Все это лежит в файле «ClientObject.cs»

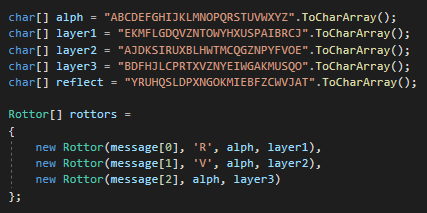


Рисунок 2.2.7 - блок кода для работы с роторами

Далее рассмотрим файл на сервере, в котором описан сам алгоритм шифрования и дешифрования. Файл имеет название «EnigmaAlg.cs». Этот файл разложим также на несколько пунктов. В первую очередь рассмотрим вспомогательные функции, которые будут работать со сложением соответствующих индексов по алгоритму шифровки.

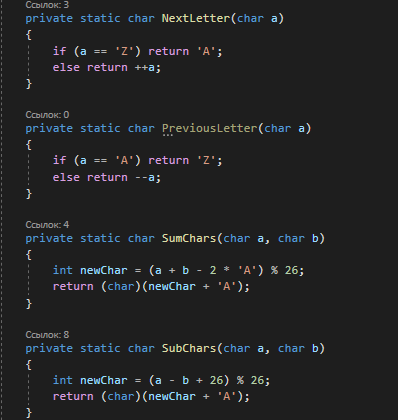


Рисунок 2.2.8 - вспомогательные функции сложения индексов символов

Далее рассмотрим блок кода, в котором мы уже начинаем вызывать все эти вспомогательные функции для шифрования, в котором надо проделать несколько однотипных действий несколько раз с незначительными изменениями:

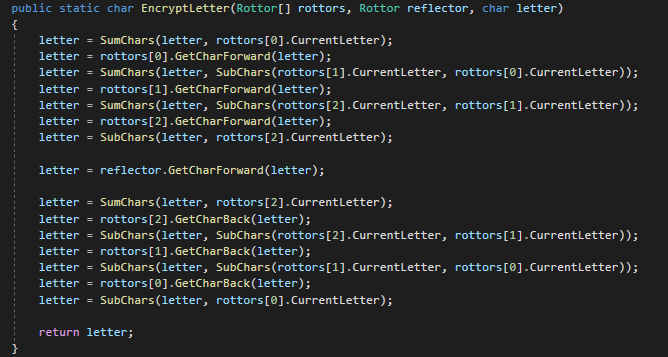


Рисунок 2.2.9 - шифровка одного символа

Также тут присутствует вспомогательная функция для прокрутки роторов при введении одного символа:

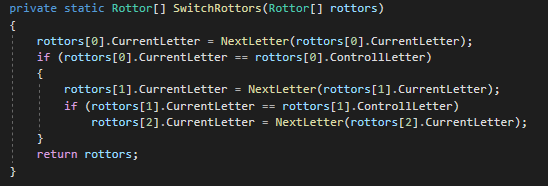


Рисунок 2.2.10 - автоматическая прокрутка роторов при введении одного символа

Имея все эти функции, мы можем использовать их такое количество раз, какое количество символов мы написали в окно ввода. Данная функция будет выглядеть так:

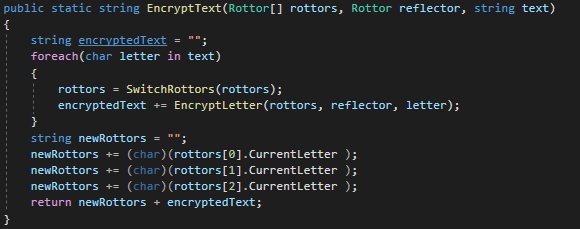


Рисунок 2.2.11 - шифрование нескольких символов

Из классов, написанных на C# остался один под название «Program.cs». Тут мы и соединяем нашу серверную и клиентскую часть. Соединение будем проводить по порту 8888, хотя можно это делать и по другому любому порту, если он не занят иными процессами. И используем ip 127.0.0.1. Пока не будет выполнена команда, которая будет вызывать сервер, будет сообщение "Ожидание подключений...".

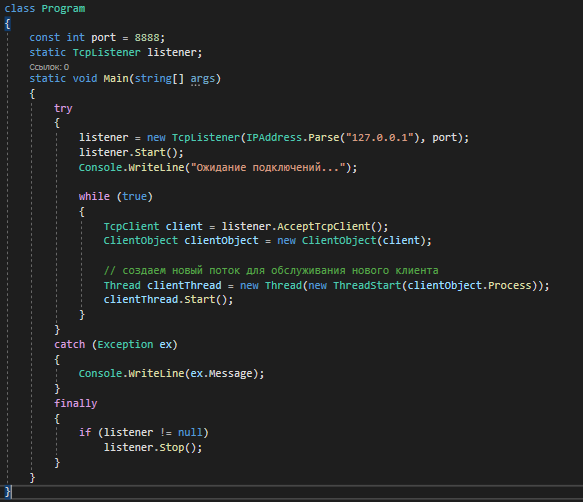


Рисунок 2.2.12 - подключение к серверу

И в итоге мы в классе «ClientObject.cs» собираем все это функции воедино и выводим в консоль полученный результат, а также отправляем этот результат в нашу клиентскую часть, где и будем выводить это все в главное окно приложения.



Рисунок 2.2.13 - отправка результатов клиенту и в консоль

Последним пунктом рассмотрим часть серверное части, написанной на С++. В этой части кода происходит отправка начального условия на сервер, где будет производиться обработка информации и отправлена обратно. В конце данной функции мы выводим в соответствующие лейблы полученные с сервера данные. В коде будет прописано два похожих блока кода, где один будет отвечать за ввод одного символа с клавиатуры в приложении, в другой работает со строкой данных:

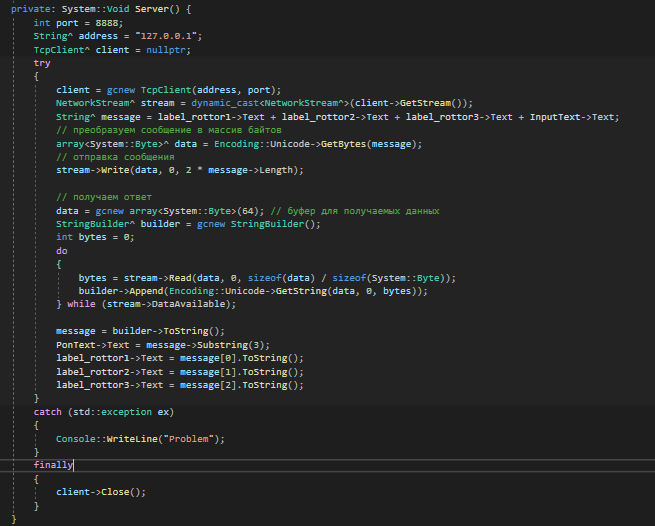


Рисунок 2.2.14 - сервер со стороны клиента

ГЛАВА 3  
ТЕСТИРОВАНИЕ ПРОГРАММЫ

**3.1 Тестирование интерфейса пользователя и юзабилити**

Результат белого тестирования проекта «SimEnigma»

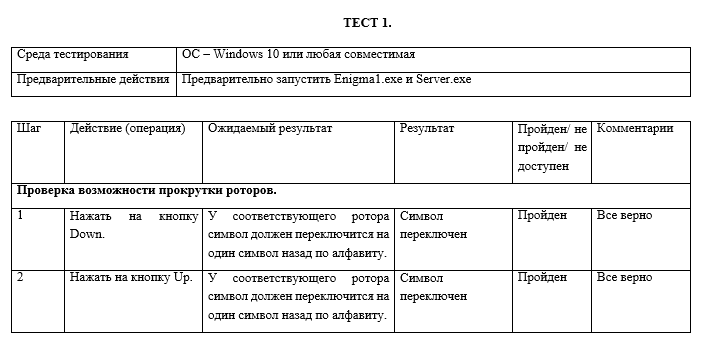


Рисунок 3.1.1 результат тестирования №1

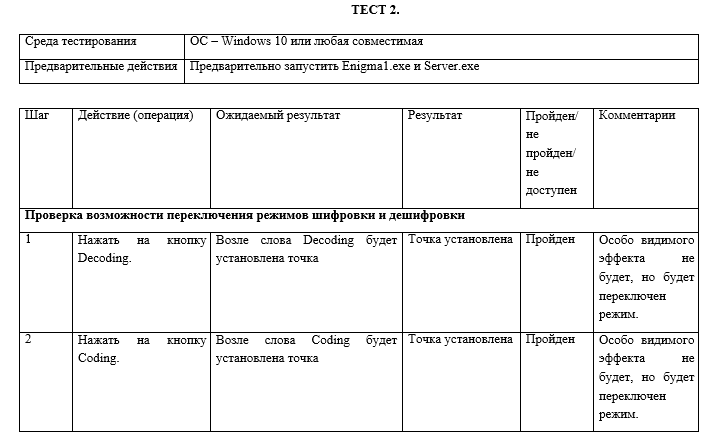


Рисунок 3.1.2 результат тестирования №2

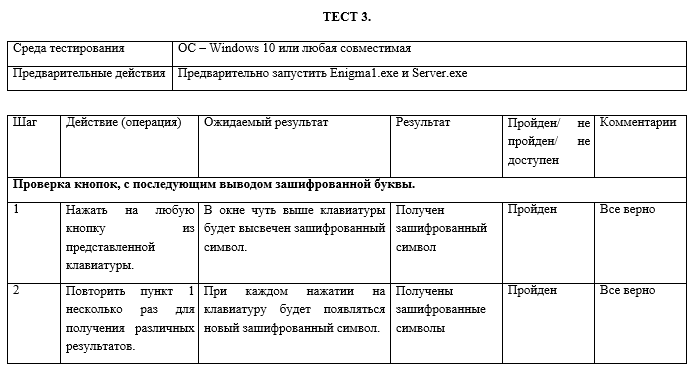


Рисунок 3.1.3 результат тестирования №2

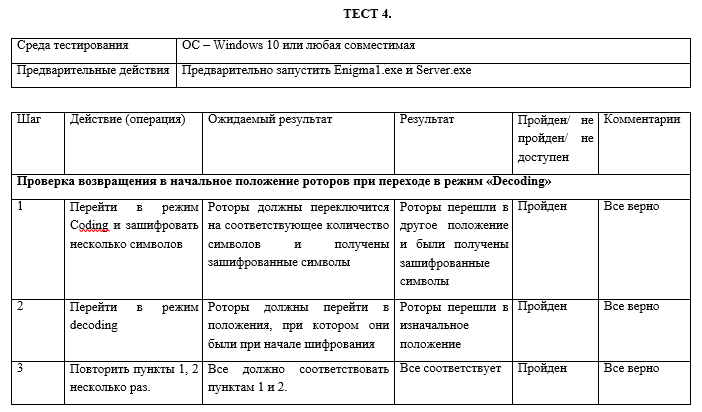


Рисунок 3.1.4 результат тестирования №4



Рисунок 3.1.5 результат тестирования №5

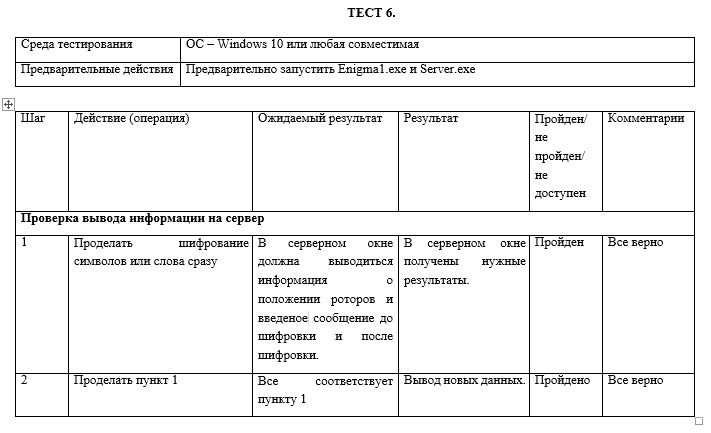


Рисунок 3.1.6 результат тестирования №6

В ходе тестирования был применен метод тест - кейсов, который проверяет

работу Сервиса тестирования знаний. Критических ошибок во время выполнения тестов не обнаружено.

**Результат черного тестирования:** в проверенной мною части программы ошибок не выявлено. Весь код написан аккуратно и в едином стиле.

**3.2 Тестирование исходного кода.**

В ходе тестирования были выявлены следующие замечания:

Код клиента имеет массивный заголовочный файл (SimEnigma.h) с неоднократно повторяющимися строчками с разницей только в объектах, например в строчках 81-115 на каждой строчке прописывается «Private:», хотя в C++ достаточно одного раза, кажется что неудачно был использован конвертер с C# в C++.

Вывод: В целом код написан понятно, присутствует комментирование, которое способствует этому. Переменным и функциям даны понятные имена, соответствующие их назначению.

То что основной код на 905 строк прописан в заголовочном файле трудно для понимания, но на работе программы это не сказывается. В целом код можно было бы оптимизировать в некоторых повторяющихся действиях, циклах.

ГЛАВА 4  
ПЕРСПЕКТИВЫ ДАЛЬНЕЙШЕГО РАЗВИТИЯ ПРОЕКТА

**4.1 Перспективы технического развития**

Данный проект сделан с достаточно простым интерфейсом, который передает основной функционал шифровальной машины энигма, но при этом нет полного сходства. В дальнейшем можно усовершенствовать данный проект, добавив к объектам программы текстуры, которые будут смотреться как и на физической модели шифровальной машины.

Так же можно будет добавить звуковое сопровождение при нажатии на соответствующие кнопки.

В моем приложении реализованы только некоторые из существующих роторов шифрования и в дальнейшем возможно дополнить их количество и сделать возможным произвольное переключение между ними.

Возможно создание отдельного окна, в котором будет описание работы шифровальной машины для того, чтобы люди могли предварительно ознакомиться с принципом работы и алгоритмом, а уже за тем преступать к работе с программой.

Данное приложение создано для персональных компьютеров, но возможно сделать и версию, которая будет доступна на мобильных устройствах.

**4.2 Перспективы монетизации**

Так как данной приложение возможно транспартировать на мобильные устройства, то после совершения этого действия проект можно будет выложить в мобильные магазины(Play Market для операционной системе Android, и в App Store для операционной системы IOS). Хоть приложение будет бесплатным, но при этом возможно использование рекламы, а также возможны различные платные функции, такие как наличие дополнительных роторов шифрования, и еще некоторые.

ЗАКЛЮЧЕНИЕ

В результате работы было создано приложение, которое имеет интерфейс, схожий с физической моделью шифровальной машины Энигма. Данная программа выполняет функционал оригинальной шифровальной машины. Помимо клиентской части была выполнена серверная часть, на которой и лежела логика проекта и в сетевую консоль при использовании кнопок выводилась соответствующая информация. Для реализации серверной части был выбран язык программирования C#. Для создания клиентской части был выбран C++. Был использован .NET Framework, где исполняющей средой был выбран CLR. В ходе проекта был получен опыт работы с клиент-серверной архитектурой.

ЛИТЕРАТУРА

1. Оформление выпускной квалификационной работы на соискание квалификационного уровня «Магистр» («Бакалавр»): методические рекомендации. / сост. Бержанский В.Н., Дзедолик И.В., Полулях С.Н. – Симферополь: КФУ им. В.И.Вернадского, 2017. – 31 с.
2. <https://metanit.com/>
3. <https://stackoverflow.com/>
4. <https://habr.com/ru/post/269519/>
5. https://habr.com/ru/post/217331/