1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Институт кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА № 8**

1. «**Основы стенографической защиты информации**»
2. по дисциплине «Основы информационной информации»
3. Выполнил
4. студент гр. 4851001/20001 Козлов О. И.

<*подпись*>

1. Преподаватель
2. профессор Калинин М. О.

<*подпись*>

1. Санкт-Петербург
2. 2023

Оглавление

[1) Цель работы 3](#_Toc129725928)

[2) Постановка задачи 3](#_Toc129725929)

[3) Ход работы 4](#_Toc129725930)

[4) Контрольные вопросы 18](#_Toc129725931)

[5) Выводы 20](#_Toc129725932)

[*Приложение 1* 21](#_Toc129725933)

## Цель работы

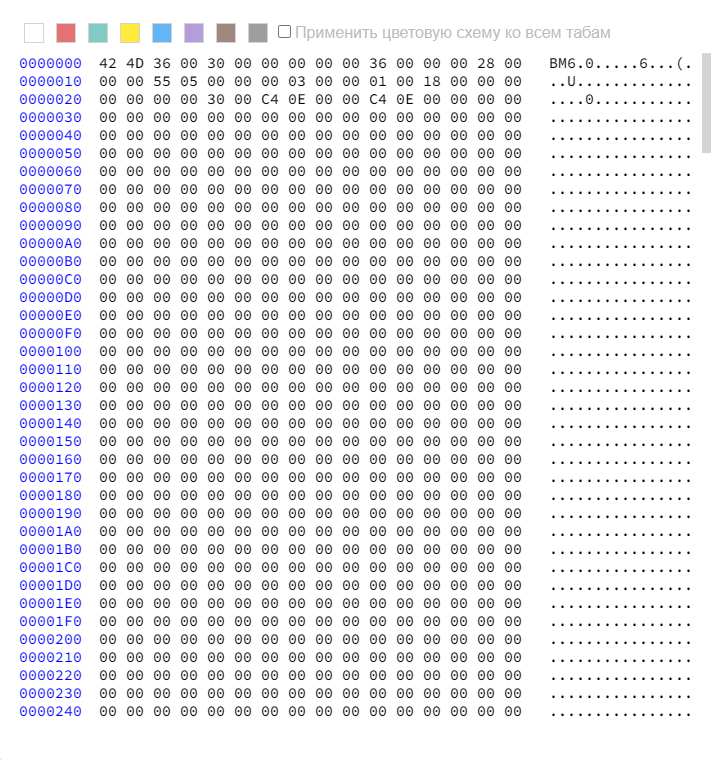
Приобретение навыков исследования свойств стегоконтейнеров, разработки стегосистем и их применение для сокрытия данных при передаче с помощью графических изображений.

## Постановка задачи

1. Создать произвольный графический файл формата BMP, который будет использоваться в качестве стегоконтейнера.
2. Открыть файл изображения с помощью HEX-редактора (например HIEW). Найти основные блоки, входящие в файл соответствующего формата. Отметить наличие или отсутствие глобальной палитры. При наличии глобальной палитры указать ее размер.
3. Создание программу – кодер, реализующую стенографическое сокрытие «секретного» текстового файла в созданный графический файл.
4. Запустить разработанную программу-кодер на выполнение, указывая различные значения степени упаковки в качестве параметра командной строки исполняемого файла. При этом визуально сравнить полученные графические файлы, содержащие упакованное «секретное» сообщение, с исходным изображением. Сравнить размер графического файла (стегоконтейнера) до упаковки в нем стегосообщения и после. Отметить влияние степени упаковки на качество изображения, получаемого в результате сокрытия в нем стегосообщения.
5. Создать программу-декодер, осуществляющую извлечение «секретного» сообщения из стегоконтейнера.
6. Запустить разработанную программу-декодер, используя в качестве входного параметра графический файл, который был получен в п. 4.
7. Запустить программу-кодер на выполнение, задав в качестве входных параметров текстовый файл, размер которого превышает объем стегоконтейнера. Отметить полученные результаты.
8. Открыть с помощью графического редактора файл изображения, содержащий «секретное» сообщение. Выполнить преобразования изображения: растяжение, поворот, отражение. Отметить параметры преобразование, которые были выполнены над графическим файлом. Сохранить результирующее изображение в тот же файл.
9. Запустить программу-декодер, используя в качестве входного параметра графический файл, который был получен в результате преобразований в п.8. Сравнить расшифрованное и исходное сообщение.

## Ход работы

В работе в качестве стегоконтейнера используется изображение формата BMP. Открыв файл такого типа в HEX редакторе, увидим следующее:



*Рисунок 1 ­– представление файла BMP в HEX редакторе.*

*Заголовок файла BMP*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Адрес (смещение) | Размер | HEX значение | Значение | Описание |
| 0h | 2 | 42 4D | BM | Код формата, в данном случае «BM» |
| 2h | 4 | 36 00 30 00 | 3145782 байт | Размер BMP Файла |
| 6h | 2 | 00 00 | - | Специальное приложение |
| 8h | 2 | 00 00 | - | Специальное приложение |
| Ah | 4 | 36 00 00 00 | 54 байт (14 + 40) | Адрес, по которому находится информация о пикселях |

*Таблица 1 – заголовок файла BMP*

Заголовок изображения включает в себя 40 байт, адреса в заголовке файла занимают 14 байт

*Заголовок файла BMP, в котором содержится информация об изображении:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Адрес (смещение) | Размер | HEX значение | Значение | Описание |
| Eh | 4 | 28 00 00 00 | 40 байт | Размер информационного заголовка |
| 12h | 4 | 55 05 00 00 | 1365 пикселей | Ширина изображения в пикселях |
| 16h | 4 | 00 03 00 00 | 768 | Высота изображения в пикселях |
| 1Ah | 2 | 01 00 | 1 плоскость | Количество используемых цветовых плоскостей |
| 1Ch | 2 | 18 00 | 24 бит | Количество битов на пиксель |
| 1Eh | 4 | 00 00 00 00 | 0 | Сжатие массива пикселей не используется |
| 22h | 4 | 00 00 30 00 | 3 145 728 | Количество байт для пикселей на 1 горизонтальную строку (с заполнением) |
| 26h | 4 | C4 0E 00 00 | 3780 | Разрешение изображения |
| 2Ah | 4 | C4 0E 00 00 | 3780 |
| 2Eh | 4 | 00 00 00 00 | 0 | Количество цветов в палитре |
| 32h | 4 | 00 00 00 00 | 0 | «0» означает, что все цветы важны |

*Таблица 2 – заголовок файла BMP, в котором содержится информация об изображении*

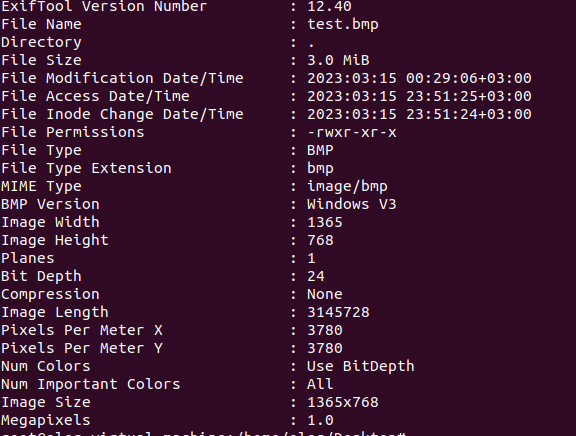
Глубина изображения файлов BMP в данной работе 24bit. Битовая глубина относится к количеству битов, используемых для представления цветов пикселя. Эти 24 бита разделены на три канала RGB (красный, зеленый, синий) по 8 бит каждый. Таким образом для представления изображения может быть использовано 2^24 = 16777216 (или же цветовая модель RGB). При такой глубине палитра цветов отсутствует.

Для изображения будем реализовывать замену младших битов изображения на биты сообщения. По этой причине, разница между исходным изображением и изображением – стегоконтейнером будет невелика и незаметна для чело веского глаза.

В качестве стегоконтейнера возьмем созданное мною в графическом редакторе Paint изображение:



*Рисунок 2 – созданное мною графическое изображение*

**

*Рисунок 3 – Свойства файла, содержащего исходное графическое изображение*

Если говорить о максимальном размере стегосообщения, которое можно вложить в данное изображение, то можно подвести несложные математические расчёты: 1365\*768 = 1 048 320 пикселей. Поскольку у нас используется цветовая модель RGB, то значит всего можно сохранить 1 048 320 \* 3 байта (3 цвета, в RGB каждый цвет кодируется 1 байтом) = 3 144 960 байт. Чтобы не было видно, что в стегоконтейнер спрятано сообщение, необходимо использовать степень упаковки 2 (меняются только самые младшие биты). Мы будем кодировать текстовое сообщение, в котором, каждый символ кодируется типом данных char, который занимает 1 байт = 8 бит. Поэтому будут меняться только последние 2 бита в стегоконтенере: закодировать один символ = 4 байта изображения.

Необходимо сказать, что при увеличении степени упаковки будет меняться количество возможной информации для шифрования, а также качество изображения.

|  |  |  |
| --- | --- | --- |
| **Размер изображения** | **Степень упаковки** | **Количество символов (байт)** |
| 3 144 960 | 1 | 98280 символов  (393120 байт) |
| 3 144 960 | 2 | 196 560 символов  (786240 байт) |
| 3 144 960 | 4 | 393120 символов (1 572 480 байт) |
| 3 144 960 | 8 | 786240 символов  (3 144 960 байт) |

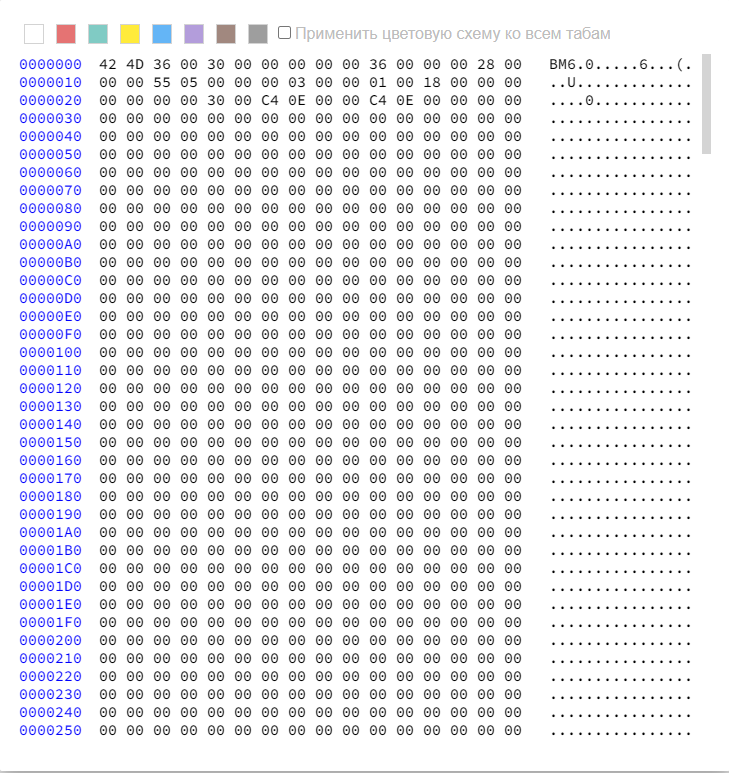
*Таблица 3 – Количество символов в скрываемом сообщении для разных степеней упаковки*

Максимальная длина скрываемого сообщения – 786240 символов, однако при высокой степени упаковки изменения картинки будут заметны на глаз, поэтому главная цель – скрытая передача информации не будет достигнута.

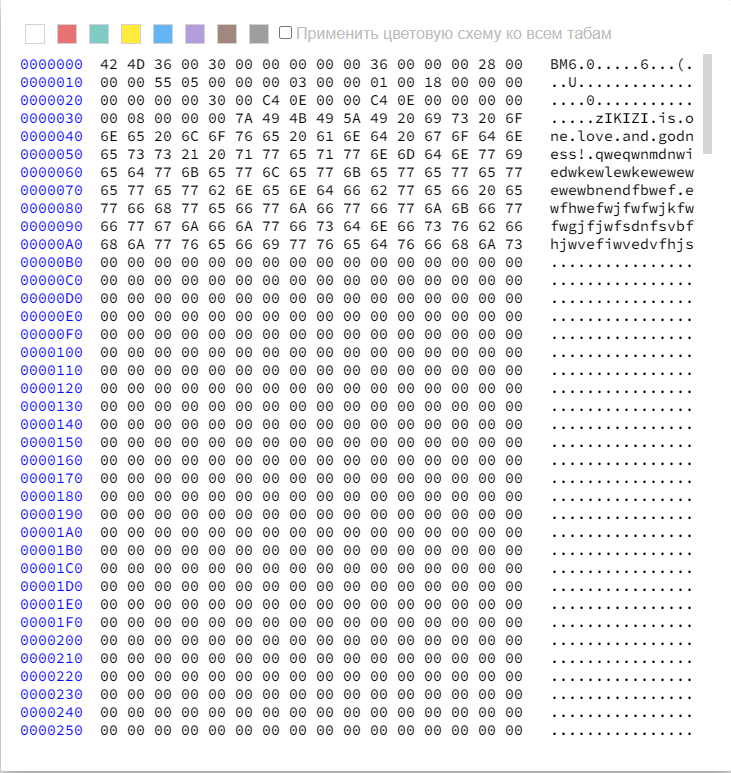
*Декодер*

В данной функции используется два файла. Один файл – стегоконтейнер (открывается файл, в который было вложено стегосообщение через кодер), другой файл, в который записывается декодированное сообщение. Из заголовка считывается степень упаковки и размер закодированного сообщения. Они используются для декодирования. Считывается байт из изображения, сохраняются последние биты (количество равно степени упаковки). Так происходит с каждым байтом. Эти биты преобразуются в символы и записывают в текстовое изображение.

*Разные степени упаковки и сравнение изображений*

**

*Рисунок 4 – HEX - код исходного изображения*



*Рисунок 5 – HEX - код измененного изображения*

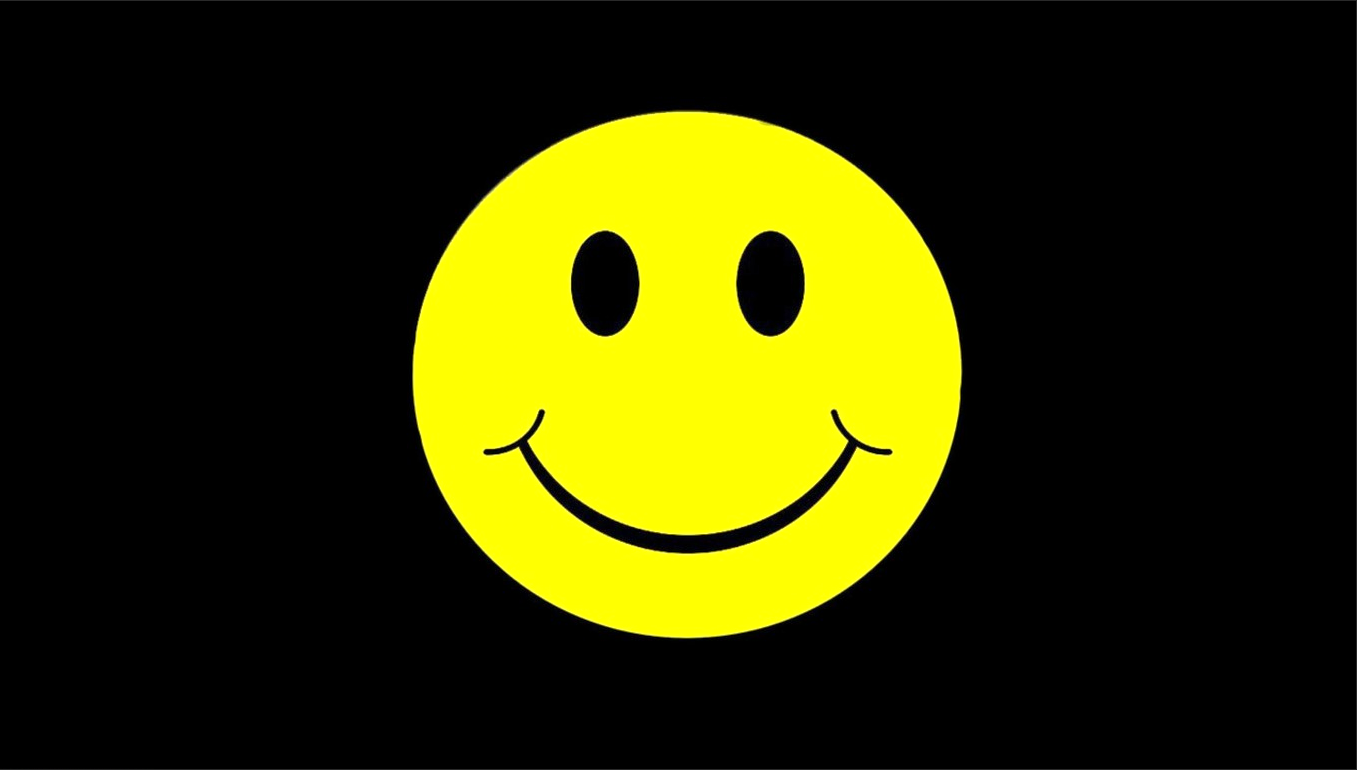
Используя HEX – редактор мы можем убедиться, что код изображения изменился при том, что на первый взгляд изображения одинаковые. Однако размер изображения не изменился, что можно объяснить тем, что мы изменяем исходные биты изображения, а не создаем новые.

Теперь необходимо сравнить изображения с разными степенями упаковки



*Рисунок 6 – Изображения со стегосообщением. Степень упаковки 1*

При степени упаковки равной 1 изменения не заметны



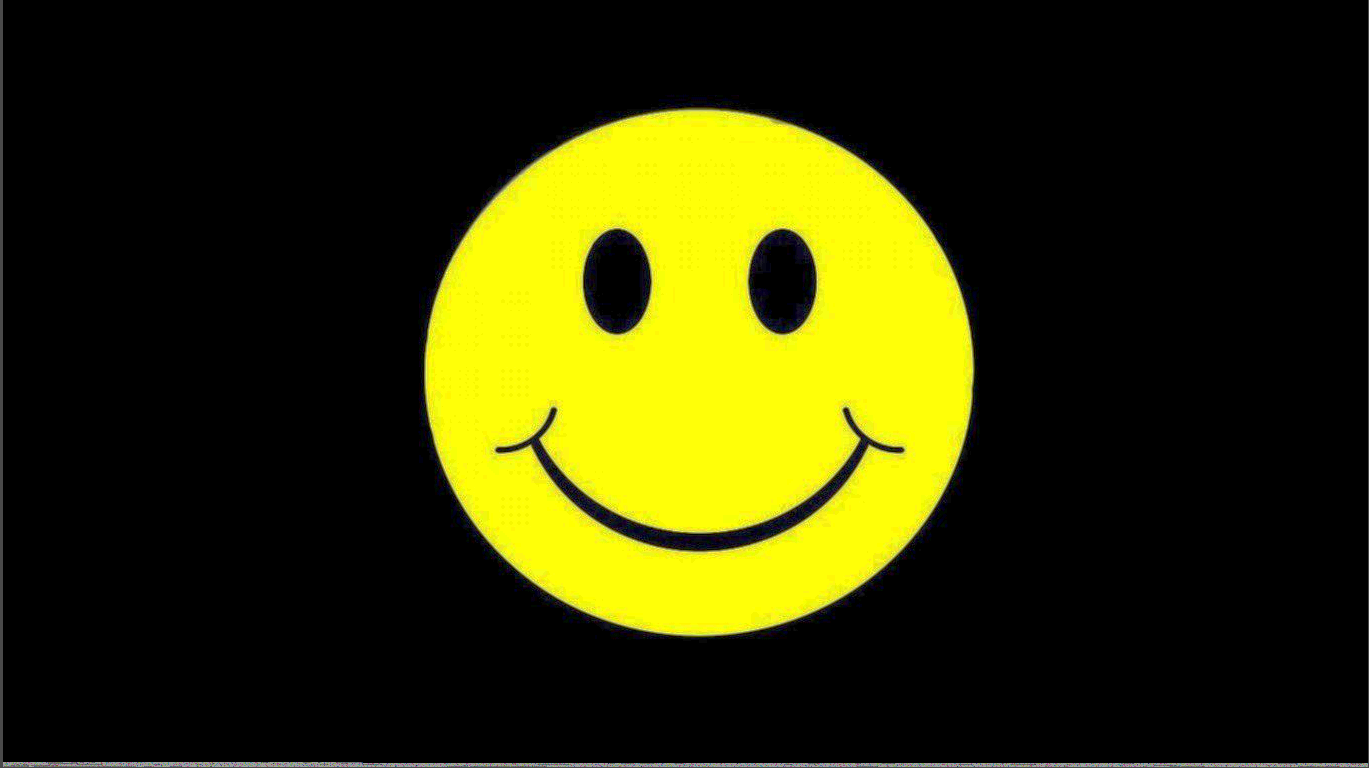
*Рисунок 7 – Изображения со стегосообщением. Степень упаковки 2*

При степени упаковки равной 2 желтый цвет стал чуть более блеклым по сравнению с оригиналом



*Рисунок 8 – Изображения со стегосообщением. Степень упаковки 4*

При степени упаковки равной 4 желтый цвет стал еще чуть более блеклым по сравнению со степенью упаковки 2.

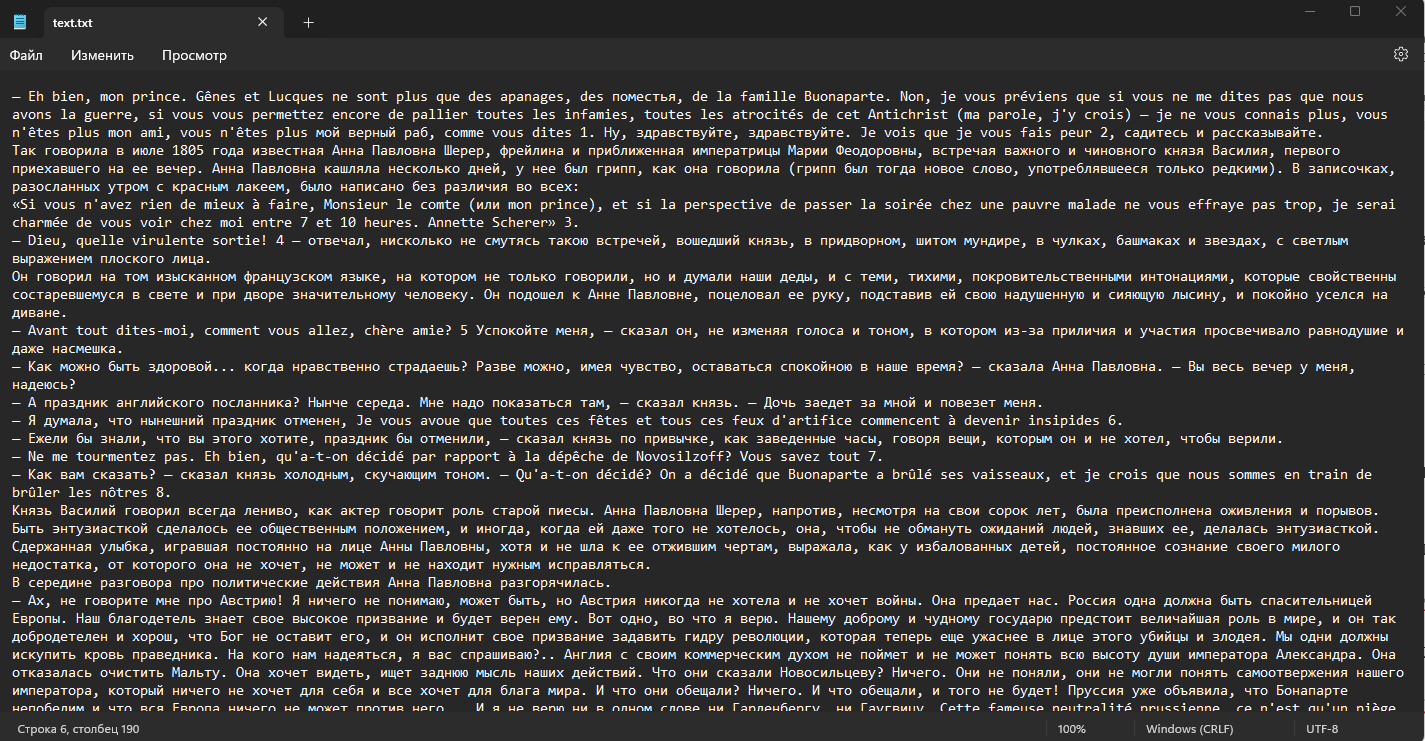


*Рисунок 9 – Изображения со стегосообщением. Степень упаковки 8*

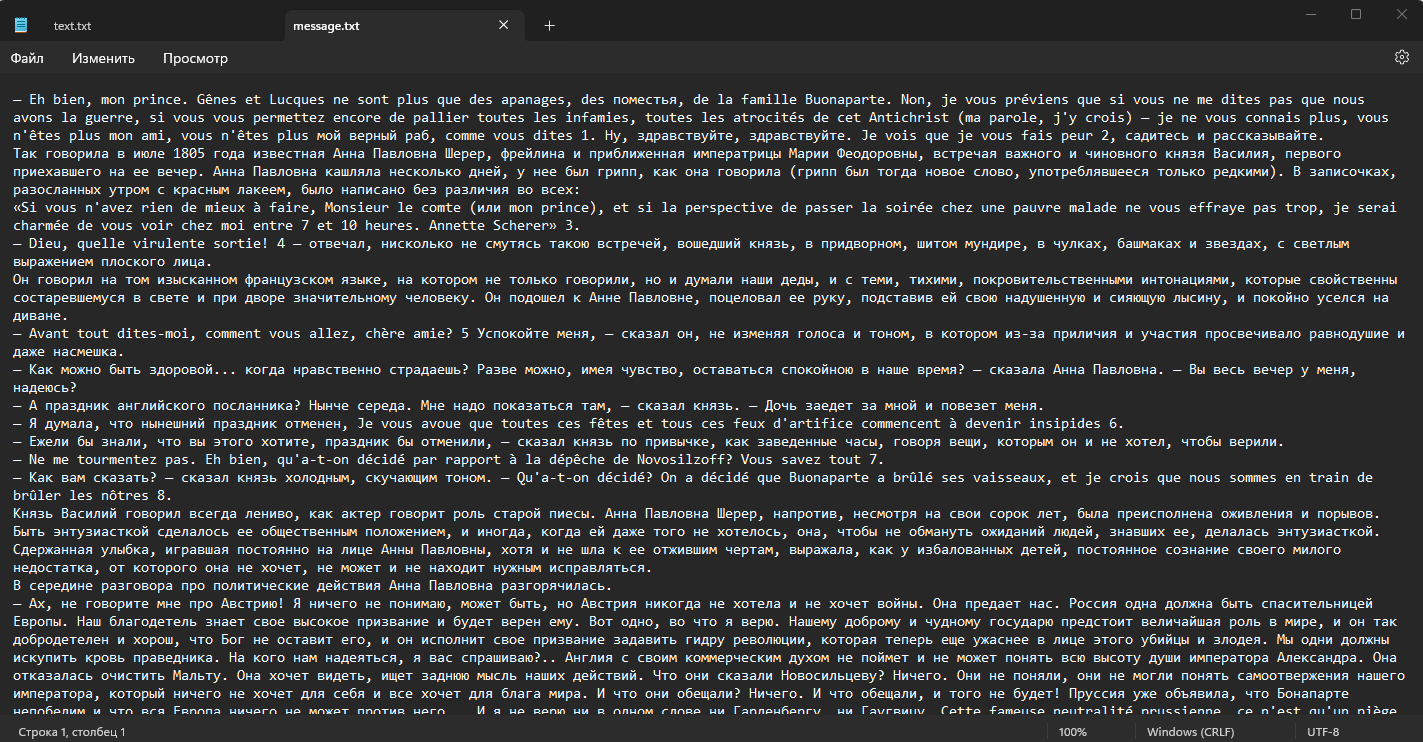
При степени упаковки равной 8 качество изображения сильно снижается, заметны артефакты изображения, пиксели имеют совершенно другой цвет по сравнению с оригиналом.

***Декодер***

Ниже представлены изображения исходного тестового файла и текста, который был получен при декодировании.



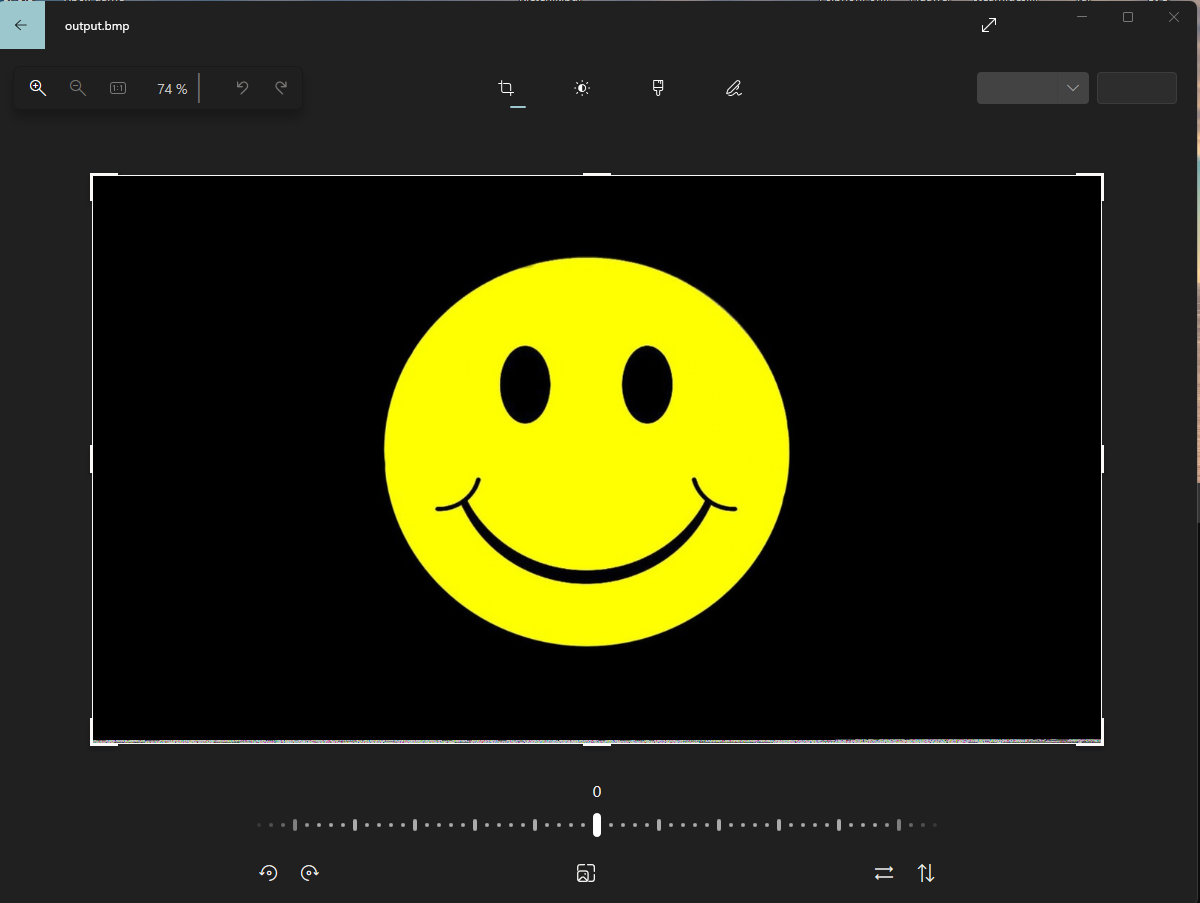
*Рисунок 10 – Исходный текст, который находится в text.txt*



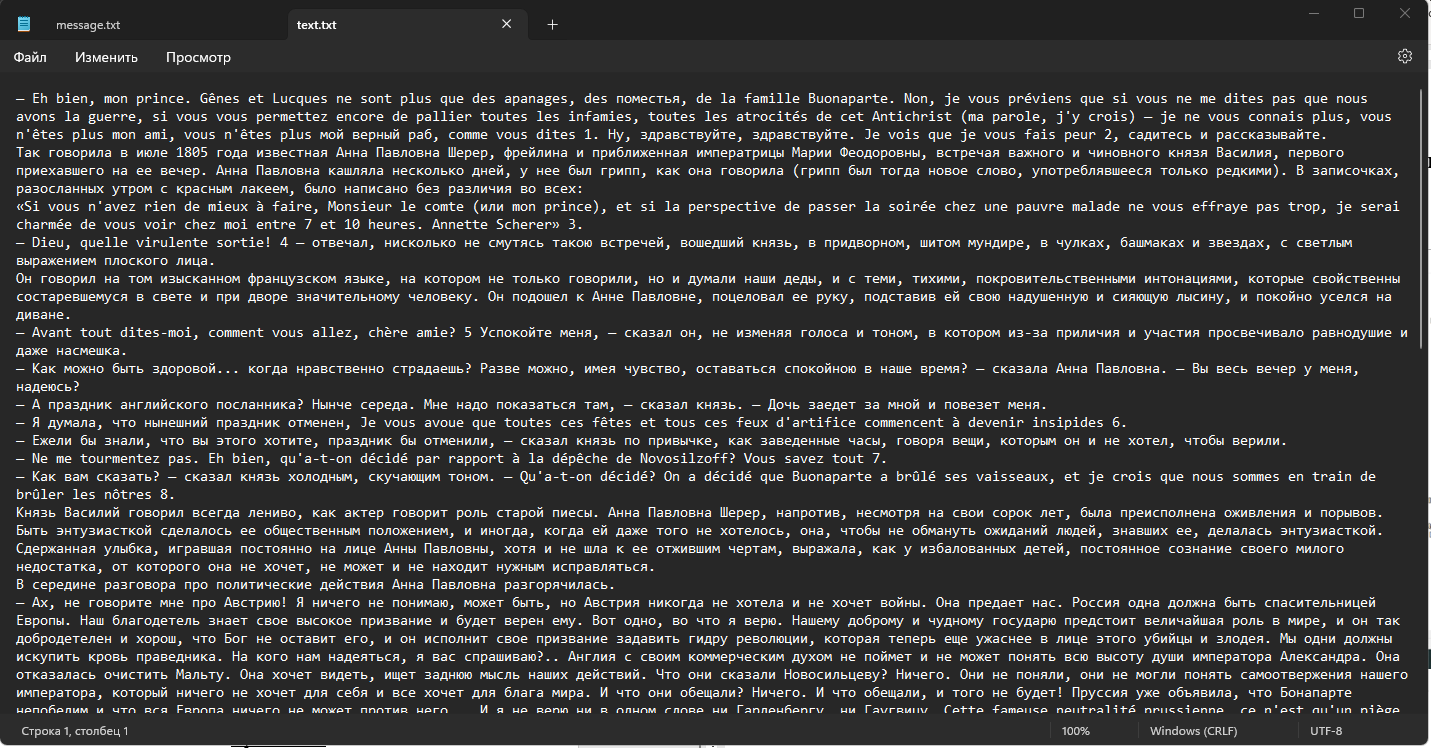
*Рисунок 11 – Текст, который находится в файле message.txt*

***Преобразование изображения***

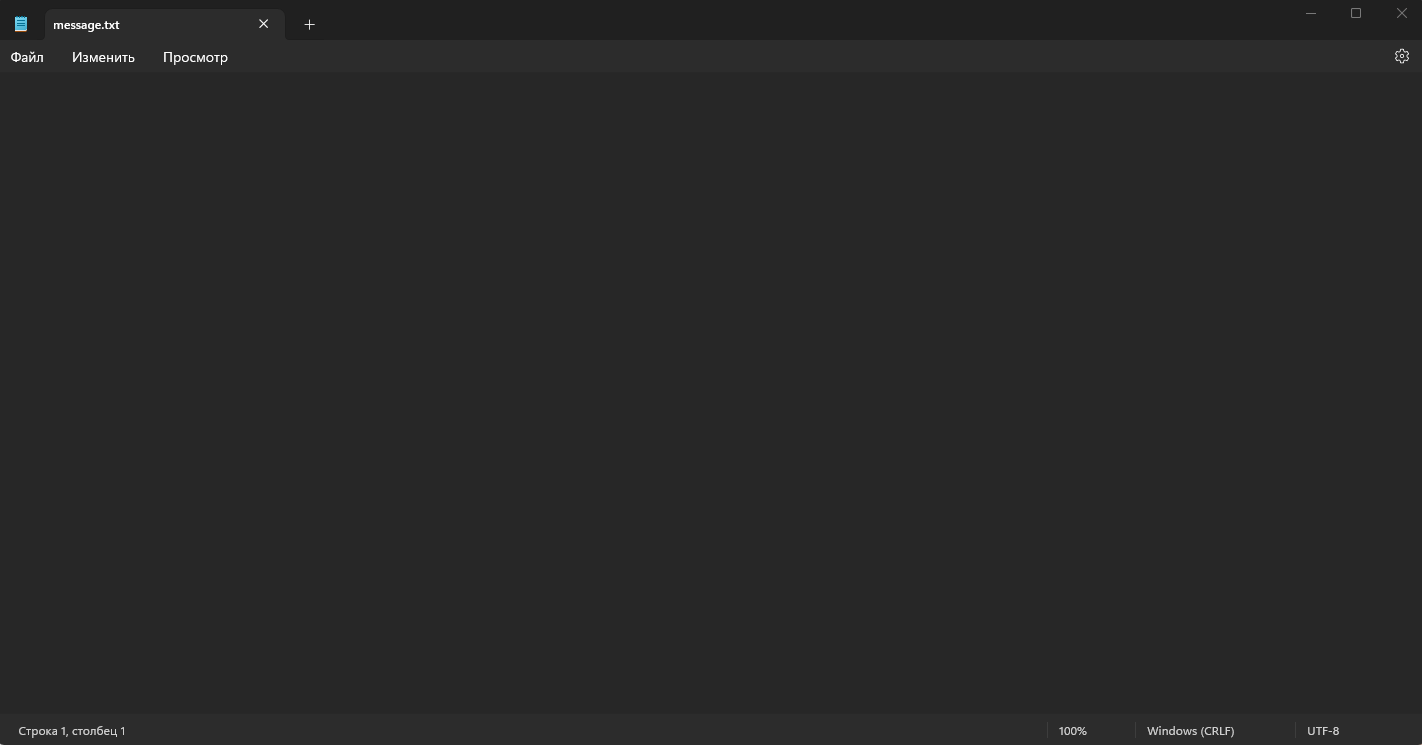
1. Отобразим изображение по горизонтали



*Рисунок 12 – Отобразим изображение по горизонтали*

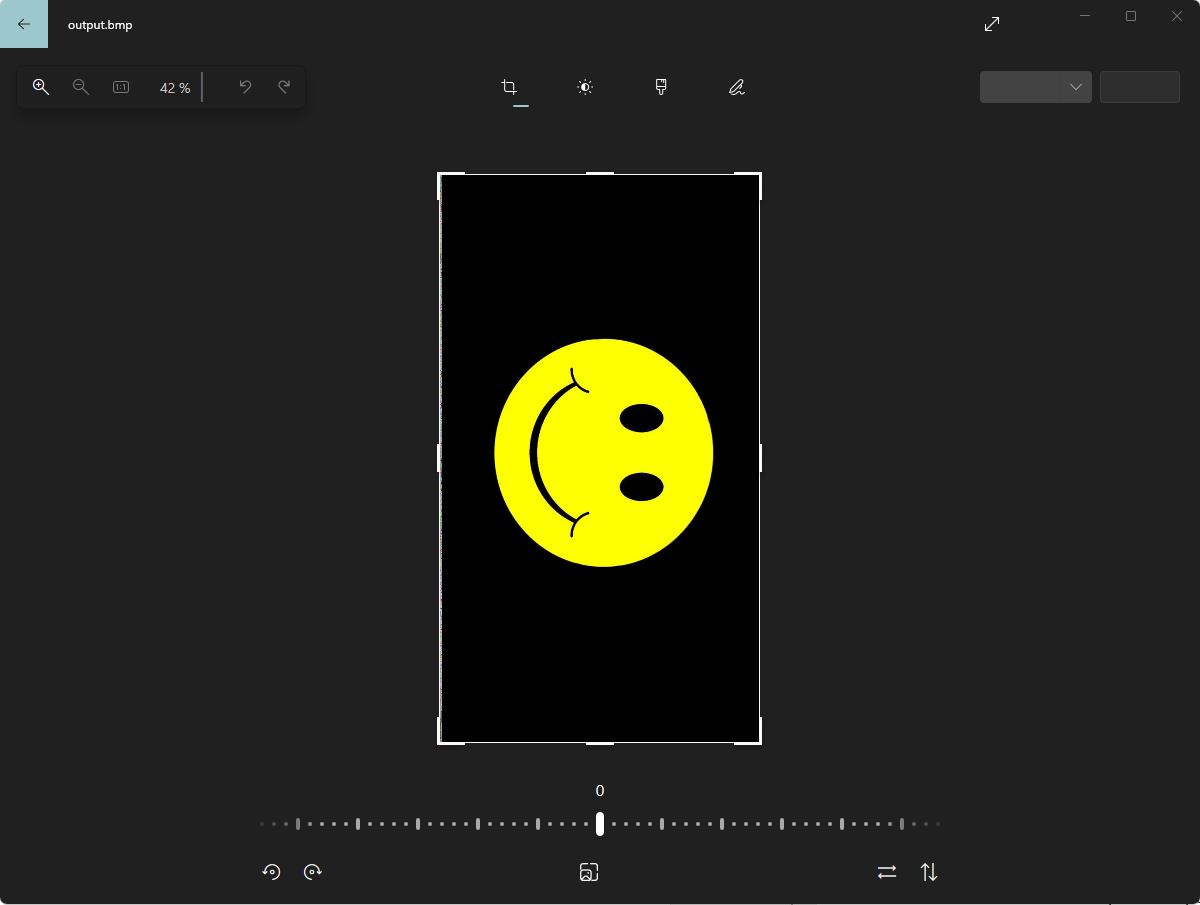


*Рисунок 13 – изначально закодированный текст*

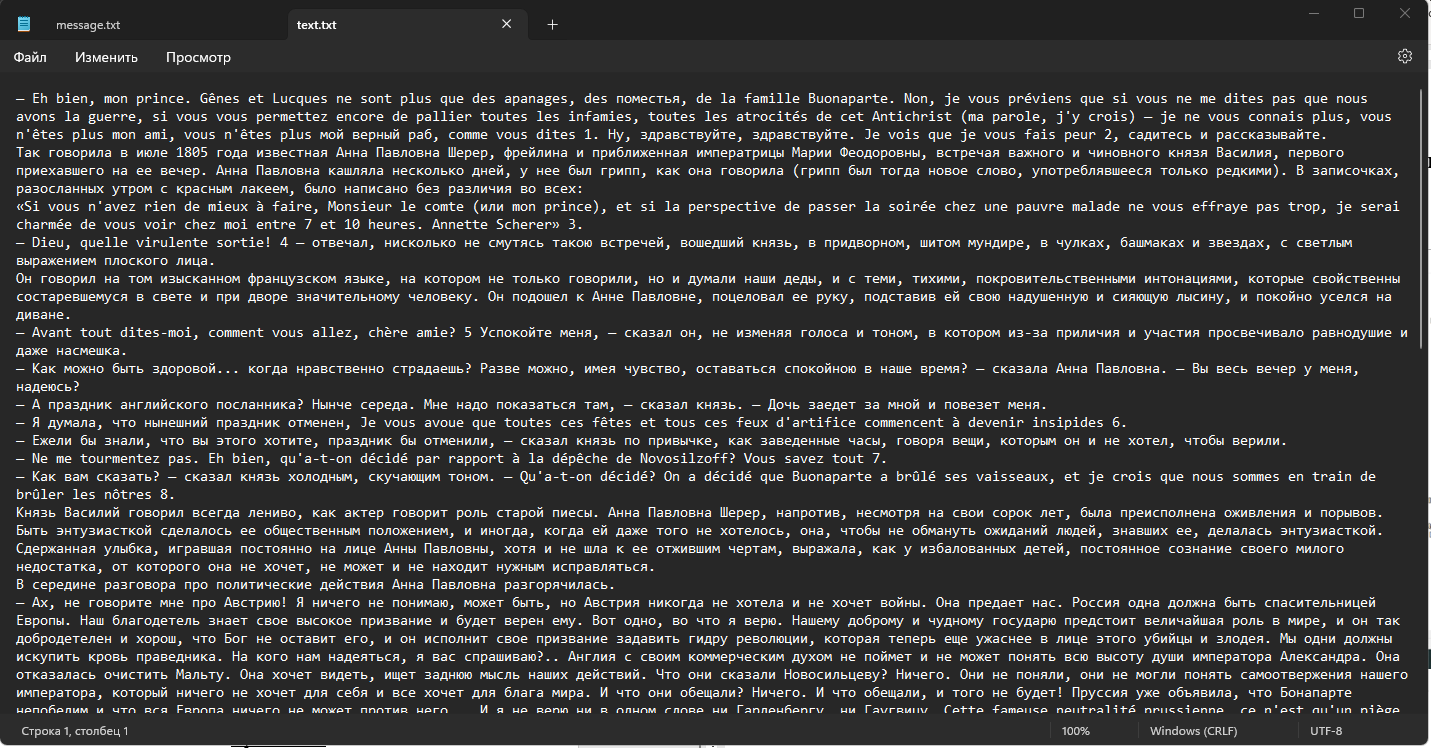
**

*Рисунок 14 – текст, полученный при расшифровке изображения, отраженного по горизонтали.*

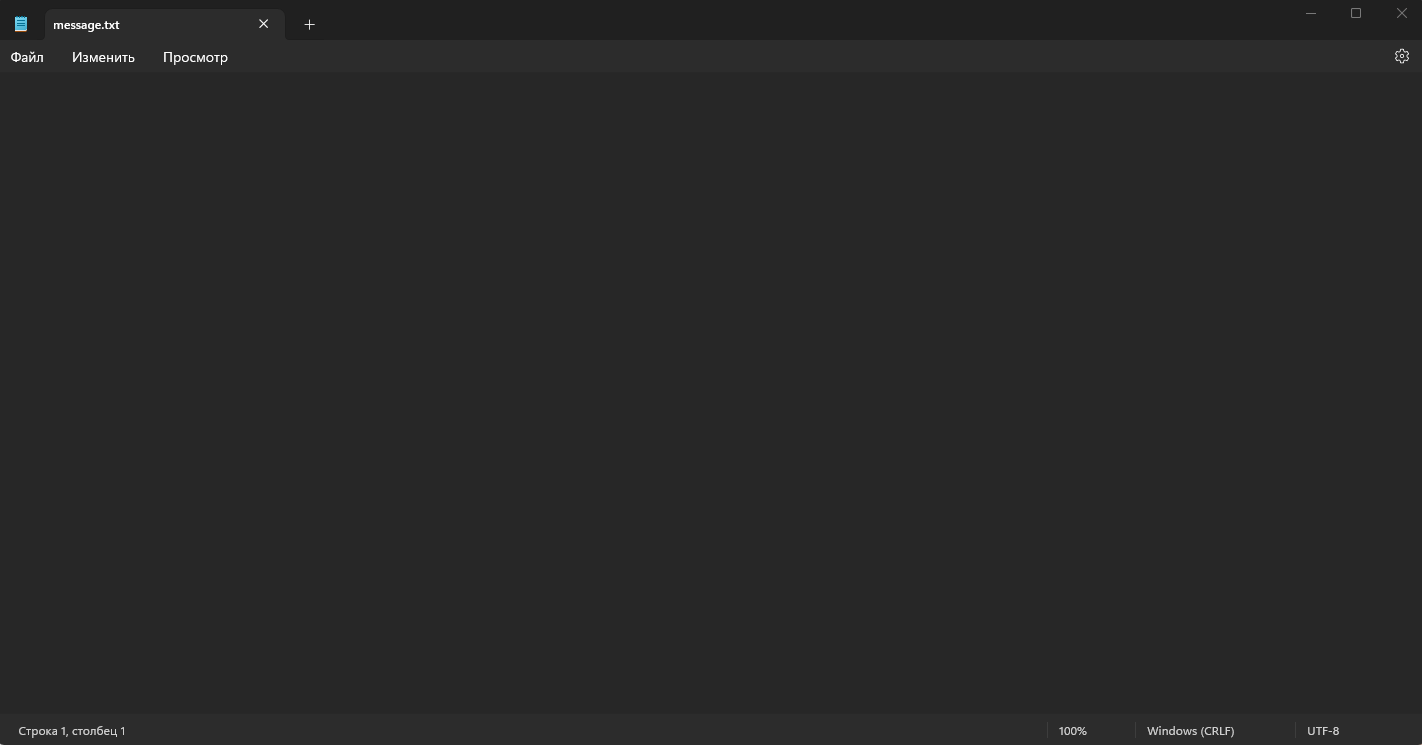
1. Текст, полученный при расшифровке изображения, повернутого на 90 градусов.



*Рисунок 15 – Повернем изображение на 90 градусов*

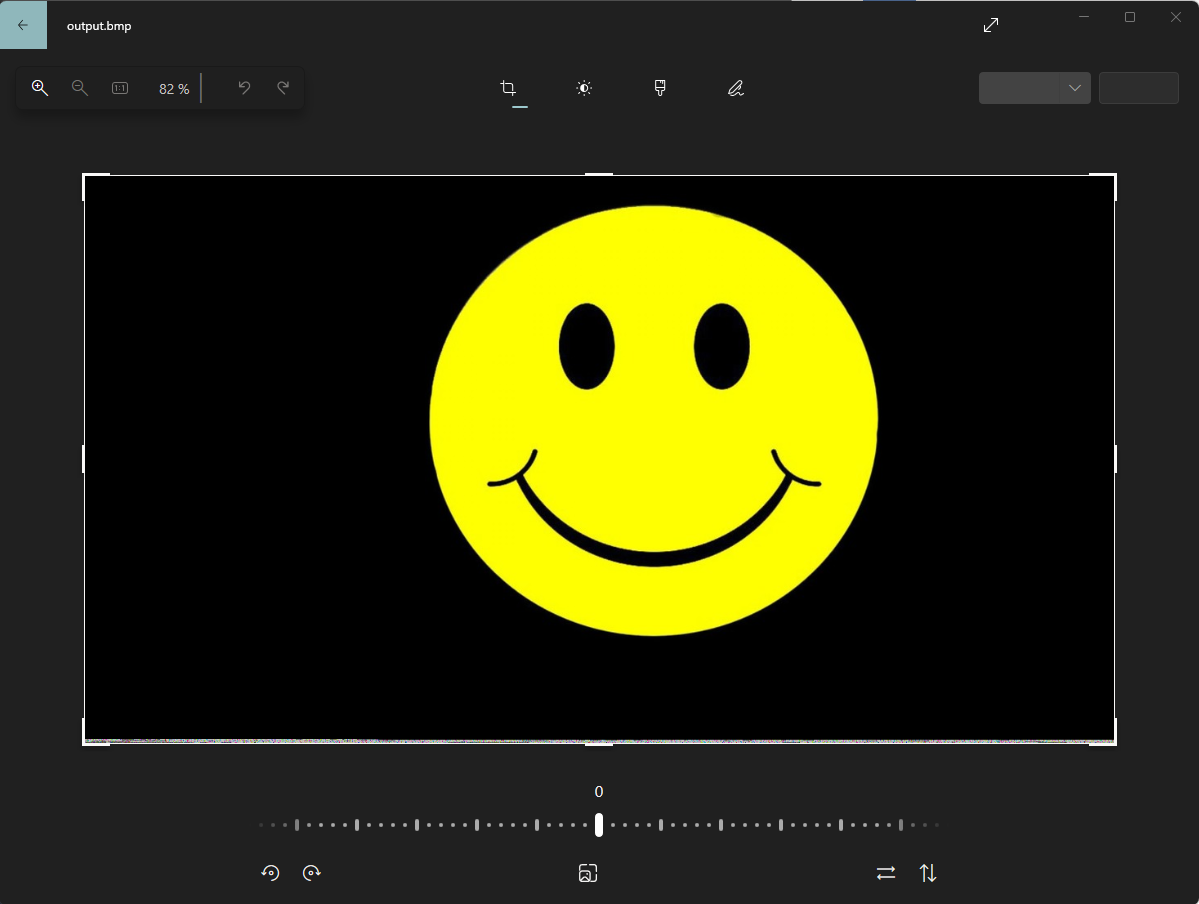


*Рисунок 16 – изначально закодированный текст*

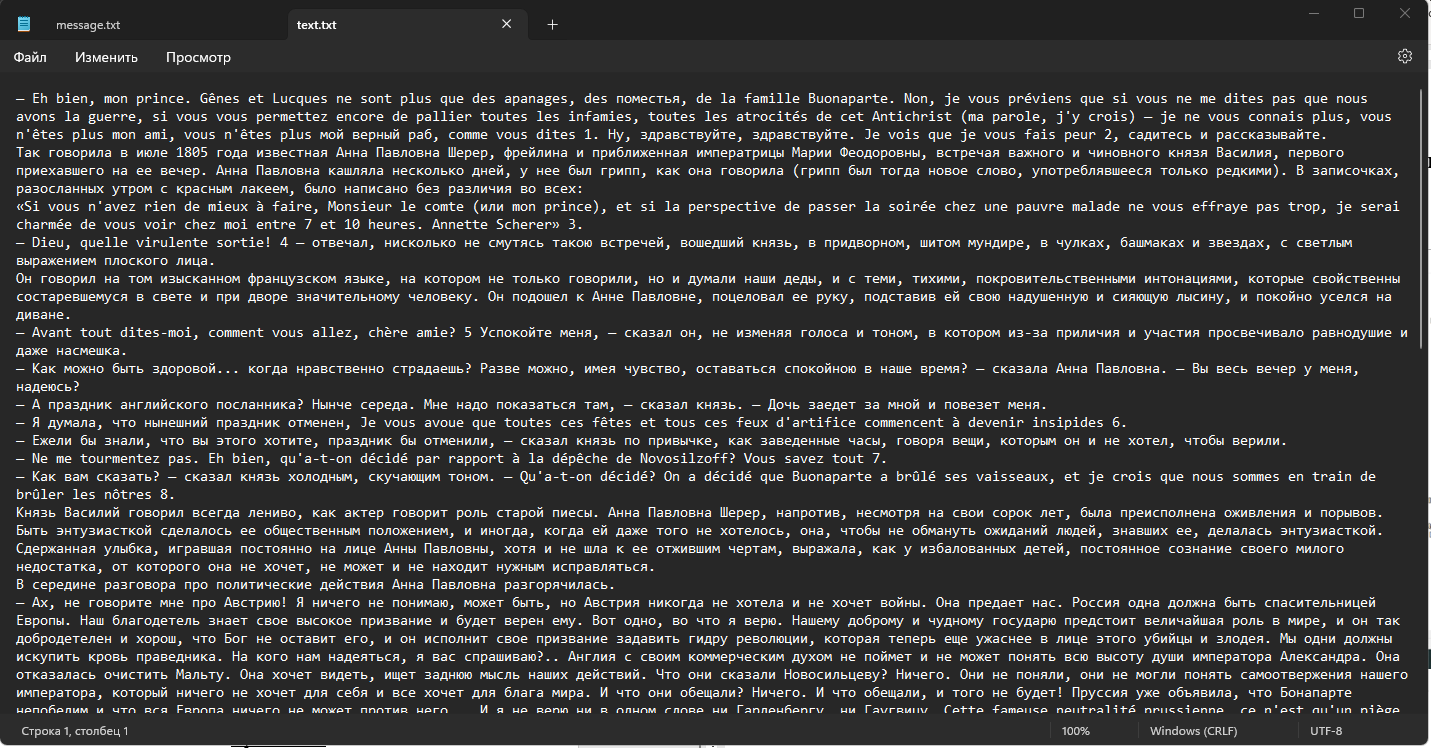
**

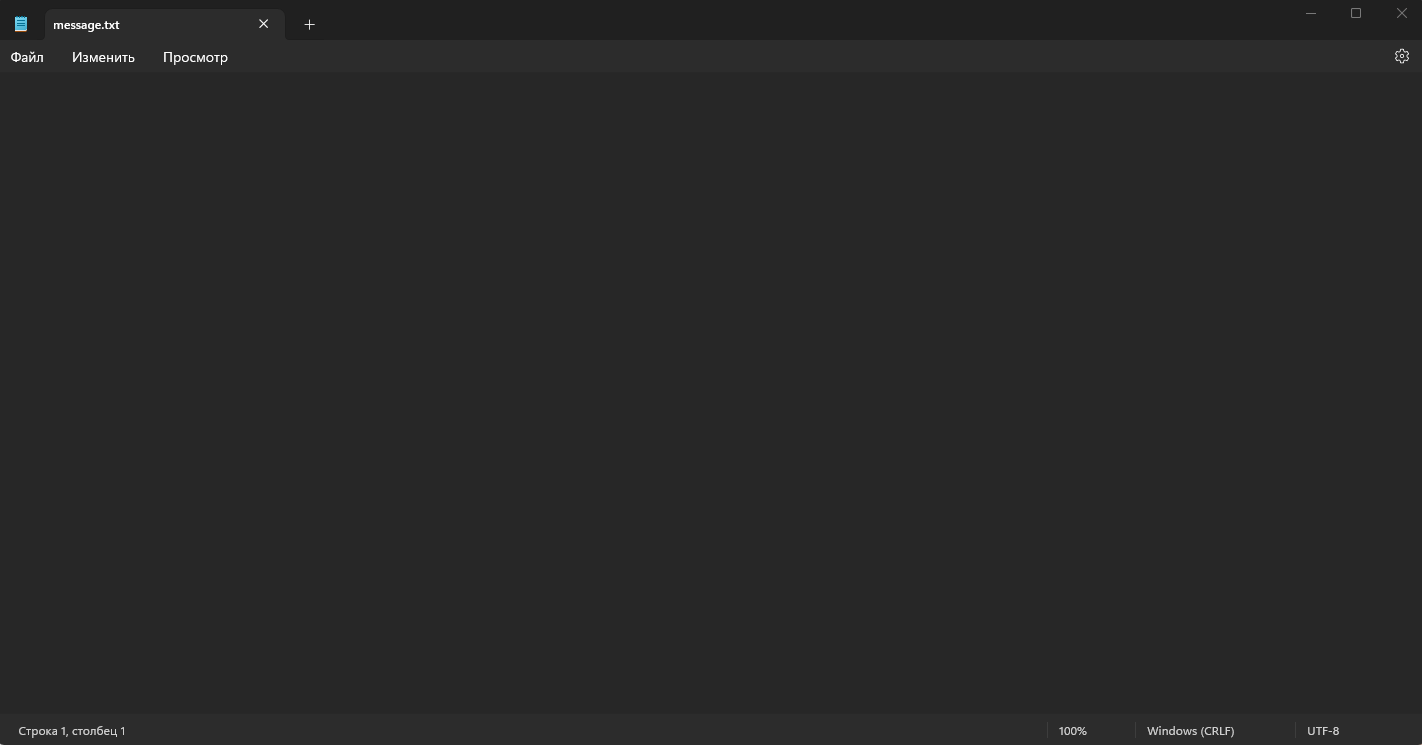
*Рисунок 17 – текст, полученный при расшифровке изображения, повернутого на 90 градусов.*

1. Текст, полученный при уменьшении размеров изображения.



*Рисунок 18 – Изменим размеры изображения*

 *Рисунок 19 – изначально закодированный текст*

**

*Рисунок 20 – текст, полученный при расшифровке изображения, с изменными размерами.*

Программа декодер не работает с данными изображениями, поэтому восстановить стегосообщение не предоставляется возможным. Поэтому можно сделать вывод, что любое изменение изображение приводит к разрушению сообщению.

## Контрольные вопросы



1. *Стегоконтейнер* – информационная последовательность, в которой скрывается сообщение. Например, в его качестве выступает изображение, в котором и будет сокрыто сообщение.
2. *Предварительный кодер* – устройство, предназначенное для преобразования скрываемого сообщения к виду, удобному для встраивания в стегоконтейнер.
3. *Кодер* – устройство, предназначенное для осуществления вложения сообщения в другие данные с учетом их модели.
4. *Детектор* – устройство, предназначенное для определения наличия стегосообщения.
5. *Декодер* – устройство, восстанавливающее скрытое сообщение. Нужно понимать, что данный узел может отсутствовать.
6. *Стеганографический канал* ─ канал передачи стегоконтейнера.



* *Атаки на сообщение.* Направлены на извлечение стегосообщения без искажения контейнера. Примеры атаки: очистка сигналов от шумов, нелинейная фильтрация
* *Атаки на стегоконтейнер.* Направлены на удаление или порча сообщения путем различных преобразований, проводимых над стегоконтейнером. Примеры атак: изменение контрастности, сжатие изображения, добавление шумов
* *Атаки на детектор.* Направлены на затруднение работы или блокировка детектора. При этом сообщение в стегоконтейнере остается, но теряется возможность его приема. Примеры атак: масштабирование, сдвиги, повороты, перестановка пикселей.
* *Атаки на механизм использования стегосообщений.* В основным связаны создание ложных стегосообщений, добавление нескольких сообщений.
* *Атаки на дополнительный ЦВЗ.* Противостояние попыткам защиты от атак аффинного преобразования (масштабирование, сдвиг, поворот, усечение изображения).



Безопасность стегосистем оценивается их *стойкостью*, т.е. способностью стегосистемы скрывать от квалифицированного нарушителя факт передачи сообщений, способностью противостоять попыткам нарушителя разрушить, исказить, удалить скрытно передаваемые сообщения, а также способностью подтвердить или опровергнуть подлинность скрытно передаваемой информации.

Стегосистема является *стойкой*, если нарушитель, наблюдая информационный обмен между отправителем и получателем, не способен обнаружить, что под прикрытием контейнера передается то или иное сообщение.



Сжатие графического изображение, как и другие аффинные преобразования, может привести к разрушению части или всего строенного сообщения. Декодировать начальное стегосообщение будет невозможно полностью.



Для защиты от атак аффинного преобразования (масштабирование, сдвиг, поворот, усечение изображения) можно использовать дополнительные ЦВЗ. Эти ЦВЗ не несут в себе информации, но используются для регистрации выполняемых нарушителем преобразований.

Другой альтернативой является вложение ЦВЗ в визуально значимые области изображения, которые не могут быть удалены из него без существенной его деградации.

Другим методом защиты от подобных атак является блочный детектор. Модифицированное изображение разбивается на блоки размером или пикселей, и для каждого блока анализируются все возможные искажения, т.е. пиксели в блоке подвергаются поворотам, перестановкам и т.д. Для каждого изменения определяется коэффициент ЦВЗ. Преобразование, после которого коэффициент оказался наибольшим, считается реально выполненным нарушителем. Таким образом, появляется возможность обратить внесенные нарушителем искажения.

## Выводы

В данной работе был изучен один из метод сокрытия информации, а именно стенография. В ходе работы была реализована программа, которая способна кодировать сообщение в изображение и декодировать сообщение из стегоконтейнера. Таким образом были преобретены навыки исследования свойств стегоконтейнеров, разработки стегосистем и их применения для сокрытия данных при передаче с помощью графических изображений.

## *Приложение 1*

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale.h>

#define COMPLETED 0

#define CODER 1

#define DECODER 2

#define EXIT 3

#define ERROR\_OPENING\_THE\_FILE 4

#define ERROR\_LARGE\_TEXT\_SIZE 5

typedef struct {

FILE\* Text;

FILE\* Input;

FILE\* Output;

} Files;

unsigned int FileSize(FILE\* File) {

int Size = 0;

fseek(File, 0, SEEK\_END);

Size = ftell(File);

fseek(File, 0, SEEK\_SET);

return(Size);

}

void Heading(FILE\* Input, FILE\* Output) {

char Symbol = 0;

for (int i = 0; i < 49; i++) {

fread(&Symbol, 1, 1, Input);

fwrite(&Symbol, 1, 1, Output);

}

}

int PackingDegree() {

int Degree = 0;

while (1) {

printf("Введите степень упаковки сообщения: 1, 2, 4 или 8\n");

scanf\_s("%d", &Degree);

while ((getchar()) != '\n');

if (Degree == 1 || Degree == 2 || Degree == 4 || Degree == 8) break;

}

return Degree;

}

void InformationText(int Degree, int SizeText, FILE\* Output) {

unsigned int Size = 0;

fwrite(&Degree, 1, 1, Output);

for (int i = 3; i >= 0; i--) {

Size = SizeText;

Size = Size >> (8 \* i);

Size = Size & 255;

fwrite(&Size, 1, 1, Output);

}

}

void TextEncoding(FILE\* Input, FILE\* Output, FILE\* Text, int Degree) {

char TextSymbol, Symbol, ImgSymbol;

unsigned char ImgMask = 255 << Degree;

unsigned char TextMask = 255 >> (8 - Degree);

while (!feof(Text)) {

fread(&TextSymbol, 1, 1, Text);

if (!feof(Text)) {

for (int i = 8 / Degree - 1; i >= 0; i--) {

Symbol = TextSymbol;

Symbol = Symbol >> Degree \* i;

Symbol = Symbol & TextMask;

fread(&ImgSymbol, 1, 1, Input);

ImgSymbol = ImgSymbol & ImgMask;

ImgSymbol = ImgSymbol | Symbol;

if (!feof(Text)) fwrite(&ImgSymbol, 1, 1, Output);

}

}

}

while (!feof(Input)) {

fread(&ImgSymbol, 1, 1, Input);

if (!feof(Input)) fwrite(&ImgSymbol, 1, 1, Output);

}

}

int MaxSize(unsigned int SizeText, unsigned int SizePicture, int Degree) {

if (SizeText >= (SizePicture \* (unsigned int)Degree / 8 - 54)) {

printf("Размер выбранного текста больше, чем возможно упаковать в контейнер\n");

return ERROR\_LARGE\_TEXT\_SIZE;

}

return COMPLETED;

}

int CheckFiles(FILE\* File) {

if (File == NULL) {

printf("Ошибка при открытии файла\n");

return ERROR\_OPENING\_THE\_FILE;

}

return COMPLETED;

}

int NumberOfCharacters(FILE\* Output) {

int Quantity = 0;

unsigned char Symbol = 0;

unsigned int Size = 0;

for (int i = 3; i >= 0; i--) {

fread(&Symbol, 1, 1, Output);

Size = (unsigned int)Symbol;

Size = Size << (8 \* i);

Quantity = Quantity | Size;

}

return Quantity;

}

void PrintInformation(unsigned int Quantity, int Degree, FILE\* Output, FILE\* Text) {

char Symbol = 0, ImgSymbol = 0;

char ImgMask = ~(255 << Degree);

for (unsigned int j = 0; j < Quantity; j++) {

Symbol = 0;

for (int i = 8 / Degree - 1; i >= 0; i--) {

if (!feof(Output)) fread(&ImgSymbol, 1, 1, Output);

if (!feof(Output)) {

ImgSymbol = ImgSymbol & ImgMask;

ImgSymbol = ImgSymbol << Degree \* i;

Symbol = Symbol | ImgSymbol;

}

}

fwrite(&Symbol, 1, 1, Text);

}

}

int Coder() {

int Degree;

unsigned int SizeText, SizePicture;

Files OpenFile;

Degree = PackingDegree();

OpenFile.Input = fopen("test.bmp", "rb");

OpenFile.Output = fopen("output.bmp", "wb+");

OpenFile.Text = fopen("text.txt", "rb");

if (CheckFiles(OpenFile.Input)) return ERROR\_OPENING\_THE\_FILE;

if (CheckFiles(OpenFile.Output)) return ERROR\_OPENING\_THE\_FILE;

if (CheckFiles(OpenFile.Text)) return ERROR\_OPENING\_THE\_FILE;

SizeText = FileSize(OpenFile.Text);

SizePicture = FileSize(OpenFile.Input);

if (MaxSize(SizeText, SizePicture, Degree)) return ERROR\_LARGE\_TEXT\_SIZE;

Heading(OpenFile.Input, OpenFile.Output);

InformationText(Degree, SizeText, OpenFile.Output);

fseek(OpenFile.Input, 5, SEEK\_CUR);

TextEncoding(OpenFile.Input, OpenFile.Output, OpenFile.Text, Degree);

fclose(OpenFile.Input);

fclose(OpenFile.Output);

fclose(OpenFile.Text);

return COMPLETED;

}

int Decoder() {

int Degree;

unsigned int Quantity = 0;

Files OpenFile;

OpenFile.Output = fopen("output.bmp", "rb");

OpenFile.Text = fopen("message.txt", "wb");

if (CheckFiles(OpenFile.Output)) return ERROR\_OPENING\_THE\_FILE;

if (CheckFiles(OpenFile.Text)) return ERROR\_OPENING\_THE\_FILE;

fseek(OpenFile.Output, 49, SEEK\_SET);

Degree = fgetc(OpenFile.Output);

Quantity = NumberOfCharacters(OpenFile.Output);

PrintInformation(Quantity, Degree, OpenFile.Output, OpenFile.Text);

fclose(OpenFile.Output);

fclose(OpenFile.Text);

return COMPLETED;

}

void Menu() {

int Choice = 0;

while (1) {

printf("------------------------------------------\n");

printf("- +++++++++++++++++++++++++++++++++ -\n");

printf("- Что вы хотите сделать? -\n");

printf("- +++++++++++++++++++++++++++++++++ -\n");

printf("------------------------------------------\n");

printf("------------------------------------------\n");

printf("- -\n");

printf("- |1| для выбора сокрытия текста -\n");

printf("- -\n");

printf("- |2| для выбора извлечение сообщения -\n");

printf("- -\n");

printf("- |3| выход из программы -\n");

printf("- -\n");

printf("------------------------------------------\n");

printf("Ввод: ");

scanf\_s("%d", &Choice);

system("cls");

while ((getchar()) != '\n');

if (Choice == CODER || Choice == DECODER || Choice == EXIT) {

if (Choice == EXIT) break;

if (Choice == CODER) {

Coder();

continue;

}

if (Choice == DECODER) {

Decoder();

continue;

}

}

else printf("Ошибка ввода, попробуйте еще раз:");

}

}

int main() {

setlocale(LC\_ALL, "Rus");

Menu();

return COMPLETED;

}