### Алгоритм работы программы:

Программа выполняет стенографическую обработку BMP-файлов. Она может:

1. **Кодировать сообщение** в изображение (встраивать его в младшие биты пикселей).
2. **Декодировать сообщение** из изображения (восстанавливать данные из младших битов пикселей).

### Алгоритм кодирования сообщения (encodeMessage):

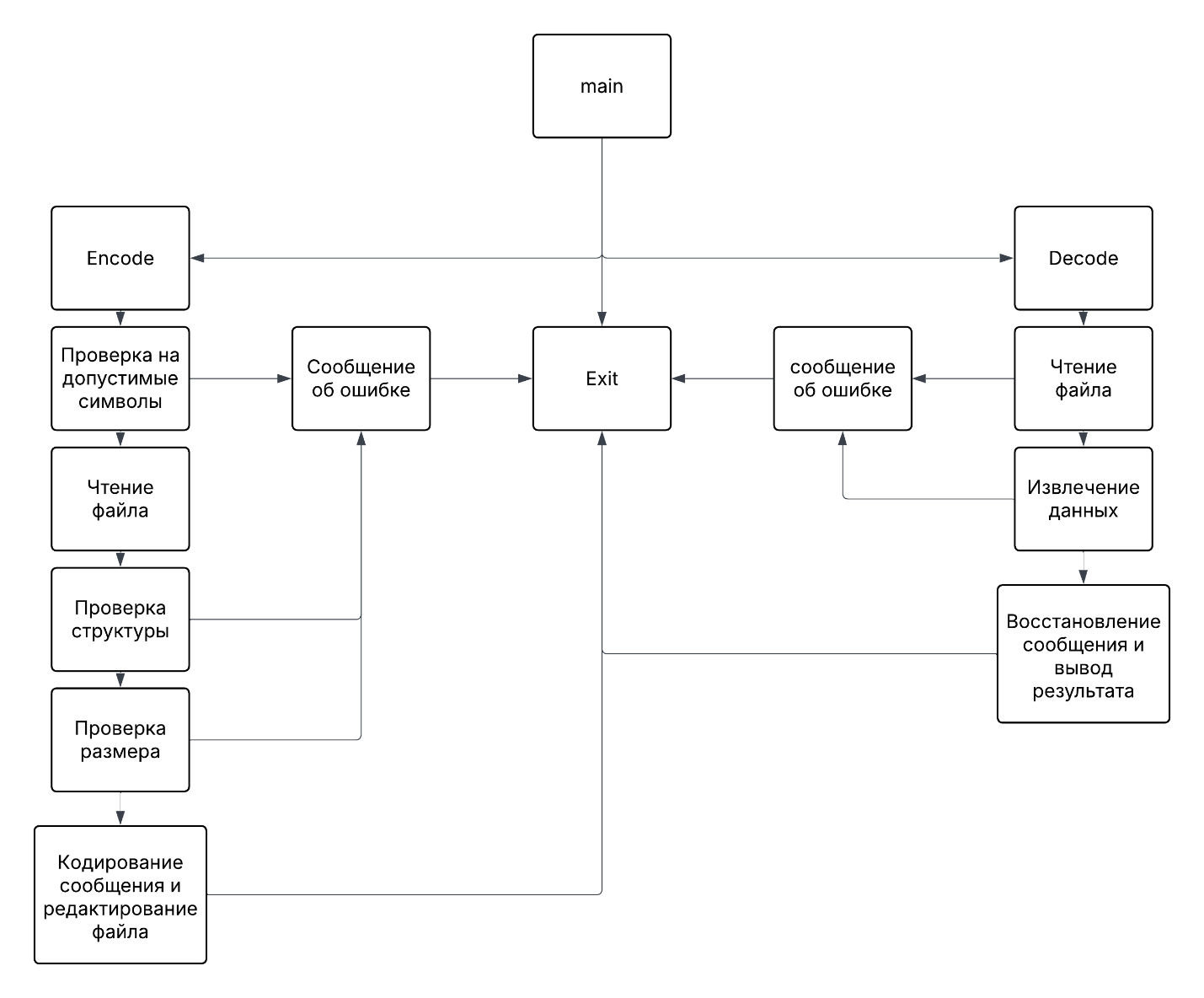
1. **Проверка допустимых символов сообщения**:
   * Сообщение проверяется на наличие только латиницы (A-Z, a-z) и пробелов. Если обнаружены недопустимые символы, программа завершается с ошибкой.
2. **Чтение исходного BMP-файла**:
   * Загружается содержимое BMP-файла в память в виде массива байтов.
3. **Проверка структуры файла**:
   * Проверяется минимальный размер файла (54 байта для заголовка BMP).
   * Определяется смещение начала данных изображения (считывается из байтов 10–13 заголовка).
4. **Проверка достаточного размера для записи**:
   * Подсчитывается количество битов, необходимых для записи сообщения (длина сообщения в символах × 8).
   * Проверяется, хватит ли места в данных изображения для размещения сообщения.
5. **Кодирование сообщения**:
   * Сообщение дополняется нулевым байтом ('\0') для обозначения конца.
   * Каждый символ сообщения преобразуется в 8 бит, которые записываются в младшие биты пикселей изображения.
6. **Запись результата**:
   * Изменённый массив данных записывается в новый BMP-файл.

### Алгоритм декодирования сообщения (decodeMessage):

1. **Чтение BMP-файла**:
   * Загружается заголовок BMP и определяется смещение начала данных изображения.
2. **Извлечение данных**:
   * Читаются младшие биты пикселей, которые используются для восстановления сообщения.
3. **Восстановление сообщения**:
   * Каждый символ собирается из 8 младших битов.
   * Сообщение завершается, если обнаружен нулевой байт ('\0').
   * Проверяется, что символы являются латиницей или пробелами. Если встречается недопустимый символ, программа выводит ошибку.
4. **Вывод результата**:
   * Восстановленное сообщение отображается в консоли.

### Основная функция (main):

1. **Выбор режима работы**:
   * Пользователь выбирает один из трёх вариантов: кодирование, декодирование или выход.
2. **Взаимодействие с пользователем**:
   * Для кодирования пользователь вводит путь к исходному BMP-файлу, путь для сохранения результата и сообщение.
   * Для декодирования пользователь указывает путь к BMP-файлу с внедрённым сообщением.
3. **Обработка результатов**:
   * В случае успешного выполнения задачи (кодирования или декодирования) выводится сообщение об успехе.
   * При ошибке пользователю сообщается о проблеме.



**Реализация:**

#include <iostream>

#include <fstream>

#include <vector>

#include <cstdint>

#include <string>

using namespace std;

// Проверка, что символ латиница A-Z или a-z или пробел

bool isLatinChar(char c) {

return (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z') || (c == ' ');

}

// Кодируем сообщение в BMP-файл, создавая новый BMP с внедрённым сообщением

bool encodeMessage(const string& inputBmpPath, const string& outputBmpPath, const string& message) {

// Проверяем сообщение на латиницу

for (char c : message) {

if (!isLatinChar(c)) {

cerr << "Wrong alphabet." << endl;

return false;

}

}

// Открываем исходный BMP-файл

ifstream inputFile(inputBmpPath, ios::binary);

if (!inputFile) {

cerr << "I can't open your file." << endl;

return false;

}

// Читаем весь файл в память

vector<uint8\_t> fileData((istreambuf\_iterator<char>(inputFile)), {});

inputFile.close();

// Проверяем размер файла (минимум заголовок BMP)

if (fileData.size() < 54) {

cerr << "I don't think it is the right format." << endl;

return false;

}

// Получаем смещение начала данных изображения из заголовка (байты 10-13)

uint32\_t dataOffset = \*reinterpret\_cast<uint32\_t\*>(&fileData[10]);

if (dataOffset >= fileData.size()) {

cerr << "Error." << endl;

return false;

}

// Подготовка сообщения с завершающим нулём

string msgWithTerminator = message + '\0';

// Подсчитываем сколько битов нужно записать

size\_t bitsToWrite = msgWithTerminator.size() \* 8;

// Проверяем, что в данных изображения достаточно байтов для записи

if (dataOffset + bitsToWrite > fileData.size()) {

cerr << "Too long." << endl;

return false;

}

// Внедрение сообщения в младшие биты пикселей

size\_t bitIndex = 0;

for (char c : msgWithTerminator) {

for (int bit = 0; bit < 8; bit++) {

uint8\_t bitVal = (c >> bit) & 1;

size\_t dataPos = dataOffset + bitIndex;

fileData[dataPos] = (fileData[dataPos] & 0xFE) | bitVal;

bitIndex++;

}

}

// Записываем результат в новый файл

ofstream outputFile(outputBmpPath, ios::binary);

if (!outputFile) {

cerr << "I can't make this file." << endl;

return false;

}

outputFile.write(reinterpret\_cast<char\*>(fileData.data()), fileData.size());

outputFile.close();

cout << "Congratulations, your secrets are a little bit safer now here: " << outputBmpPath << endl;

return true;

}

// Декодер сообщения из BMP (только латиница + пробел)

string decodeMessage(const string& filePath) {

ifstream inputFile(filePath, ios::binary);

if (!inputFile) {

cerr << "Error: I can't open your image." << endl;

return "";

}

// Читаем заголовок BMP (54 байта)

vector<uint8\_t> header(54);

inputFile.read(reinterpret\_cast<char\*>(header.data()), header.size());

if (inputFile.gcount() != 54) {

cerr << "Nope, this is not BMP." << endl;

return "";

}

// Определяем начало данных изображения

uint32\_t dataOffset = \*reinterpret\_cast<uint32\_t\*>(&header[10]);

// Переходим к данным изображения

inputFile.seekg(dataOffset, ios::beg);

if (!inputFile) {

cerr << "Error: I can't." << endl;

return "";

}

// Читаем все данные пикселей

vector<uint8\_t> pixelData((istreambuf\_iterator<char>(inputFile)), {});

if (pixelData.empty()) {

cerr << "Seems like this is just the picture." << endl;

return "";

}

string decodedMessage;

uint8\_t currentByte = 0;

int bitIndex = 0;

for (uint8\_t pixelByte : pixelData) {

uint8\_t lsb = pixelByte & 1;

currentByte |= (lsb << bitIndex);

bitIndex++;

if (bitIndex == 8) {

if (currentByte == '\0') {

// Нулевой байт — конец сообщения

break;

}

// Проверяем символ: латиница или пробел

if ((currentByte >= 'A' && currentByte <= 'Z') ||

(currentByte >= 'a' && currentByte <= 'z') ||

(currentByte == ' ')) {

decodedMessage += static\_cast<char>(currentByte);

}

else {

cerr << "Hmm, I see something I don't know." << endl;

return "";

}

currentByte = 0;

bitIndex = 0;

}

}

return decodedMessage;

}

int main() {

string filePath, inputPath, outputPath, message;

int i;

cout << "What is my purpose" << endl << "1. Encode" << endl << "2. Decode" << endl << "3. exit" << endl;

cin >> i;

switch (i) {

case 1:

cout << "What do you need to change: ";

cin >> inputPath;

cout << "Where do you need your image: ";

cin >> outputPath;

cout << "FEED ME YOUR SECRETS (only latin alphabet): ";

cin.ignore(); // Сбрасываем символ новой строки после cin

getline(cin, message);

if (!encodeMessage(inputPath, outputPath, message)) {

cerr << "Something went terribly wrong." << endl;

return 1;

}

break;

case 2:

cout << "FEED ME YOUR SECRETS (BMP): ";

cin >> filePath;

cout << " " << filePath << endl;

string message = decodeMessage(filePath);

if (!message.empty()) {

cout << "Congratulations, here is your message: " << message << endl;

}

else {

cerr << "Something is wrong." << endl;

}

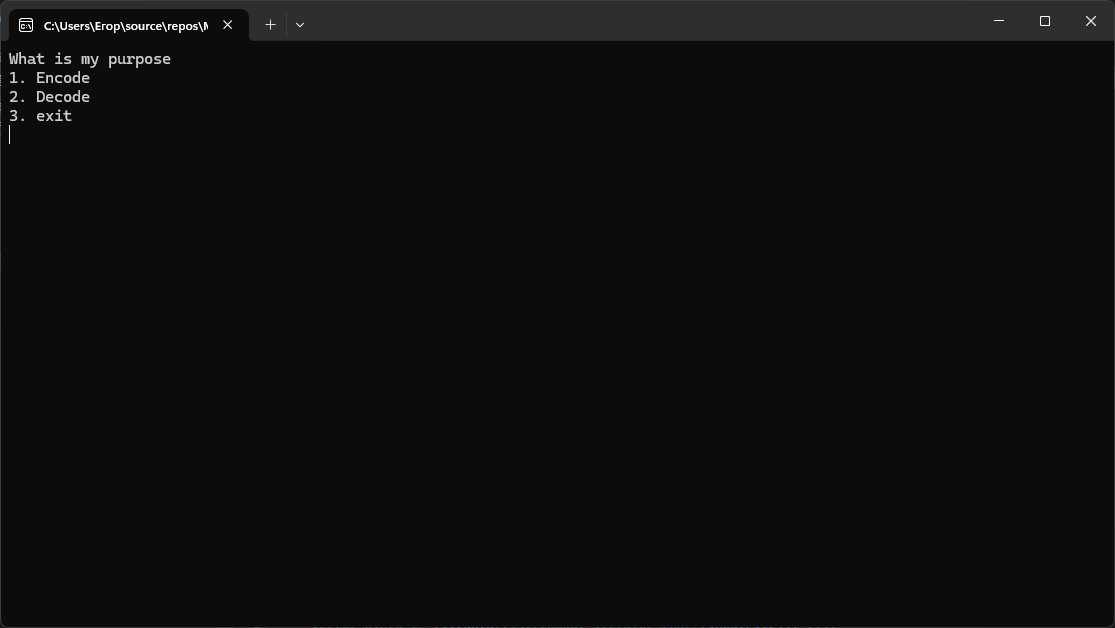
}

return 0;

}

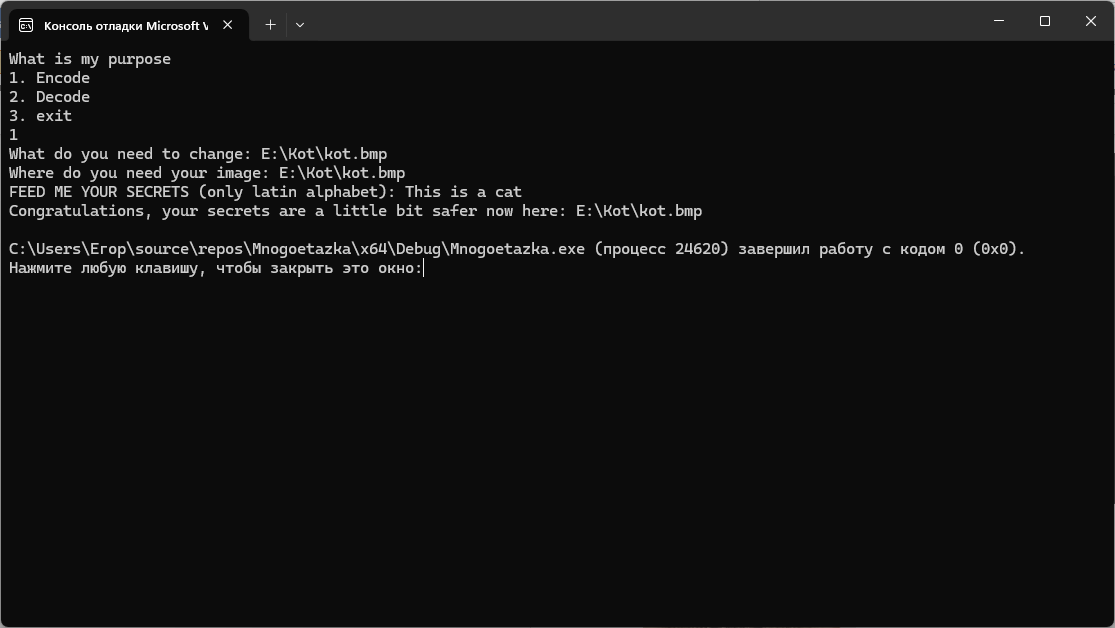
**Тестирование:**

**Шифрование**



Изображение до:

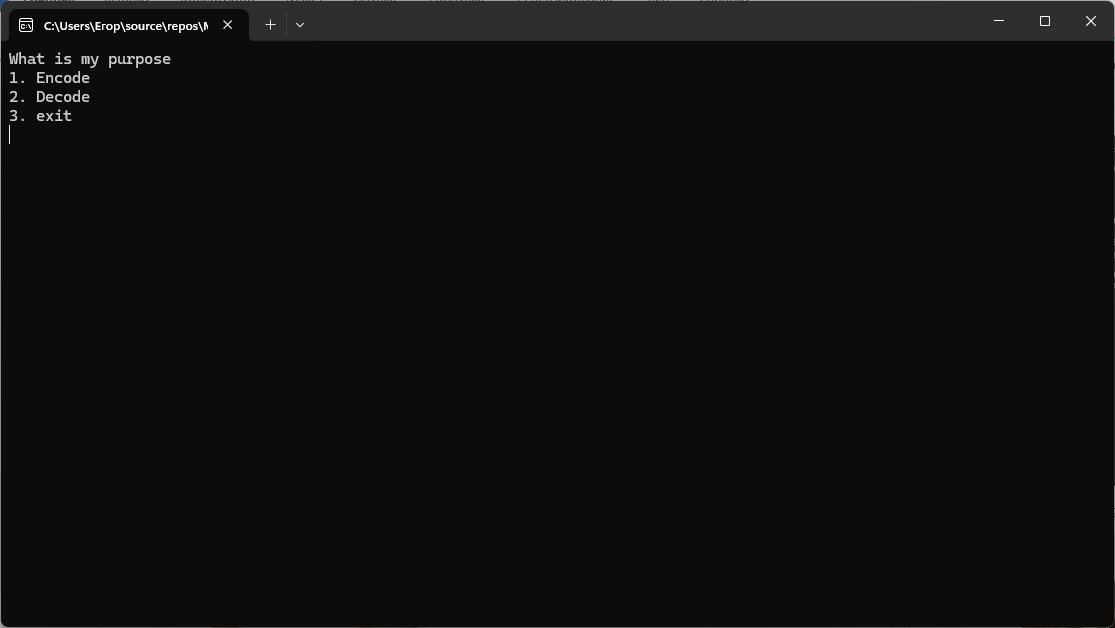


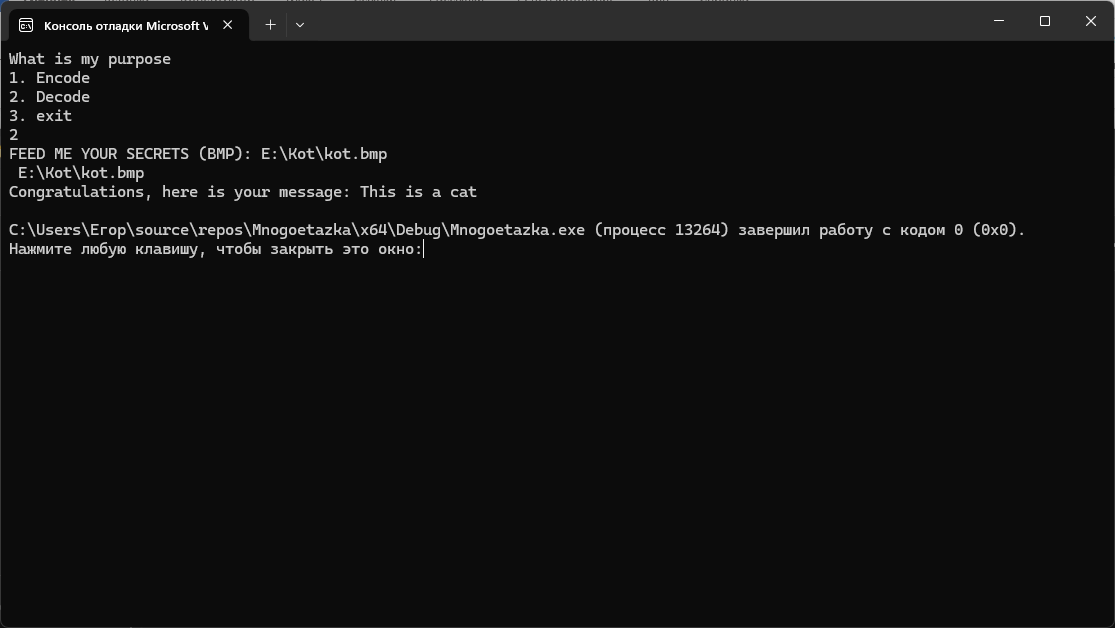


Изображение после:



**Расшифровка:**

****

****