Image Steganography

Sagynbek Talgatuly
ENGR-AD 1000, Fall 2020

STEP 1: Problem Identification and Statement

Problem statement is to develop a software solution, which will do the following:

- Encode a secret message inside a bitmap image using the LSB image steganography
- Decode the secret message by extracting the embedded message from the encoded image and display it on the console.
- Provide a user-friendly menu of encode/decode and exit options.
- Receive names of bitmap images and the secret message from the user
- Make use of provided header script BitmapHelper.h

STEP 2: Gathering of Information and Input and Output Description

The main theme of this work is related to watermarks. A watermark is a hidden information to a document or image by placing a logo or seal in plain sight. The watermark protects the owner's rights by showing ownership. Hiding the watermark does not change the appearance of the image. This protects the owner's rights, without disturbing the image. In broad terms, it is referred as a *steganography*.

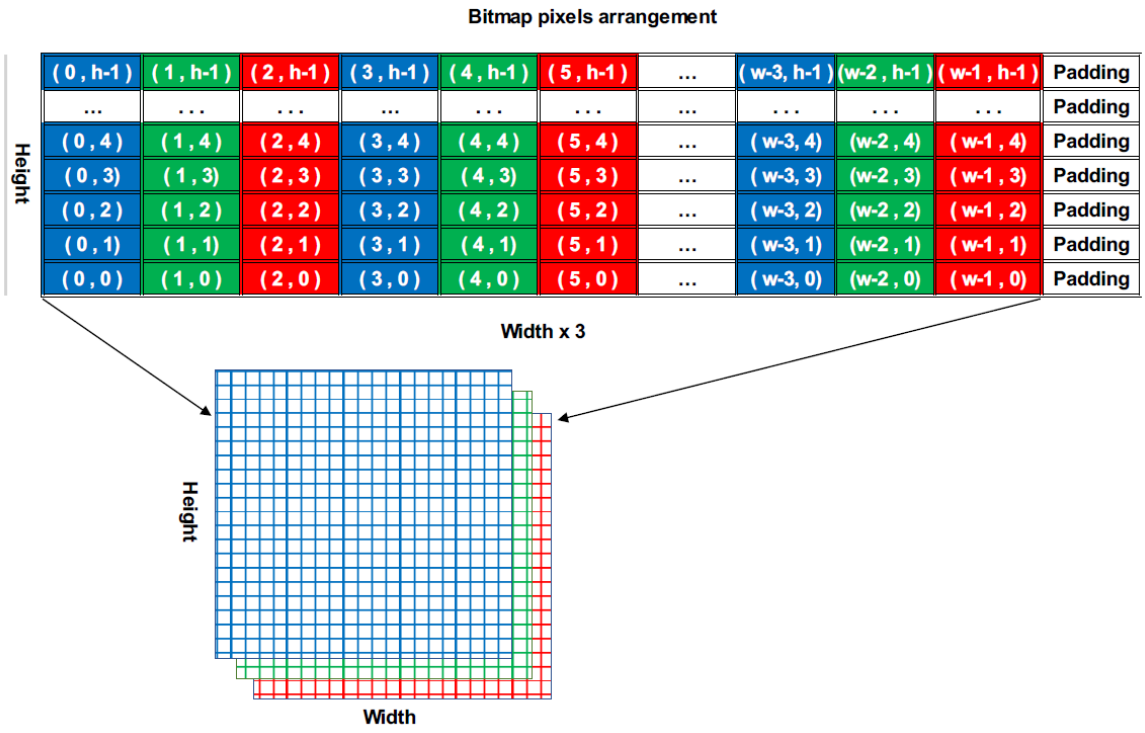
Steganography is a technique of hiding the data (file, text, image, etc.) within another data file (file, text, image, etc.). This report will describe the implementation of a solution for digital image steganography. Development of a software solution to encode a secret text message inside a bitmap image and decode the secret message by extracting the embedded message from the encoded image will be the subject.

Here, we assume that an image is represented as a three-dimensional array (M-by-N-by-3 array) where M is the number of pixels in the vertical direction (rows) and N is the number of pixels in the horizontal direction(cols), while each 3-vector corresponds to the blue, green, and red intensities of each pixel. The pixel values are 1-byte characters and are between 0 and 255.

To read and write Bitmap images a "BitmapHelper" header script is assumed to be utilized. A bitmap image mentioned above has the following structure.

Size	Contents
14 bytes	Bitmap File header
40 bytes	Bitmap info header
(variable)	Color palette
(variable)	**Bitmap pixel array**

A Bitmap pixel array is arranged in the format given below (by default the pixels are arranged in BGR order).



As stated in the problem statement, the task is to develop a software that provides the user with options to encode and decode images. The software presents the user with a main menu, with options to encode a message, decode a message, and exit the program. It is worth noting that STLs or advanced data structures for this assignment like vector, map, bit set, etc. are not allowed in the solution. However, for this assignment bitmask functions will be used. By using them, multiple bits can be set either on, off or inverted from on to off (or vice versa) in a single bitwise operation.

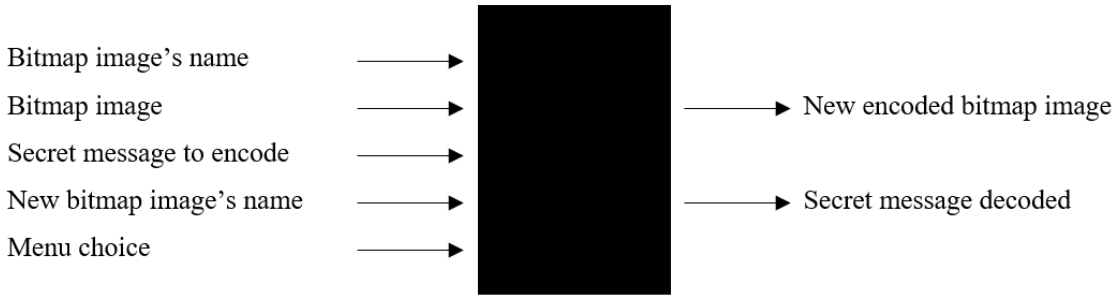
To make it clear, we can consider the step 2 by dividing it into two parts: Encoding process and Decoding process. The description of the main menu and error messages will be provided in the end of this step.

Here, it would be worth to describe the inputs/outputs of the entire program. Further in particular parts additional and more specified I/O descriptions will be included.

The inputs: Bitmap image's name, Bitmap image, Secret message to encode, New bitmap image's name, Menu choice

The outputs: New encoded bitmap image, Secret message decoded

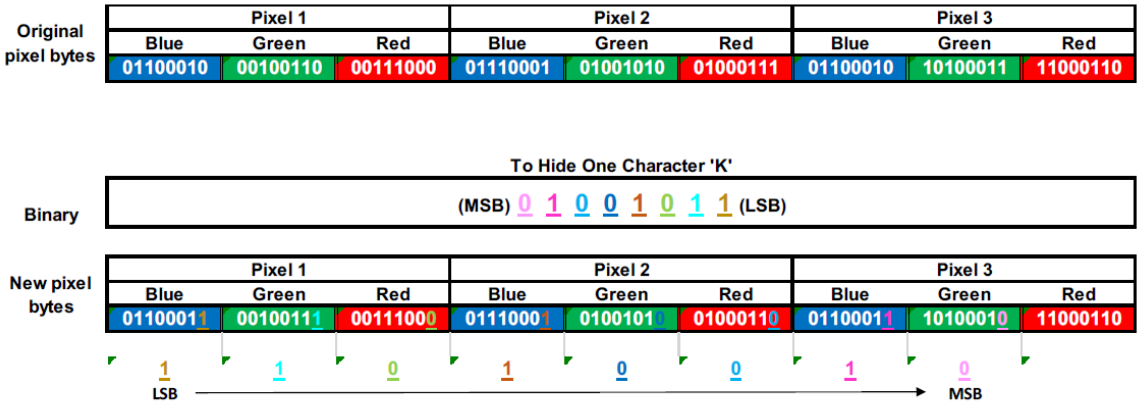
The black-box diagram of input/output of the program is provided below



1) Encoding process

First, for the encoding process the program will need a bitmap image's name and the secret message to be encoded. These data are obtained through console. If the name of the file provided by the user is valid, the program reads data about every single pixel of the image. Each pixel contains three bytes representing the intensities of the three colors (red, green, and blue). To encode the secret message into the image, the Least Significant Bit (LSB) image steganography will be implemented, where message bits

are hidden in the least significant bit of image bytes to have minimal visible effect, so it is not recognizable for a common viewer. An example of hiding one character to image pixels is illustrated through the following example.

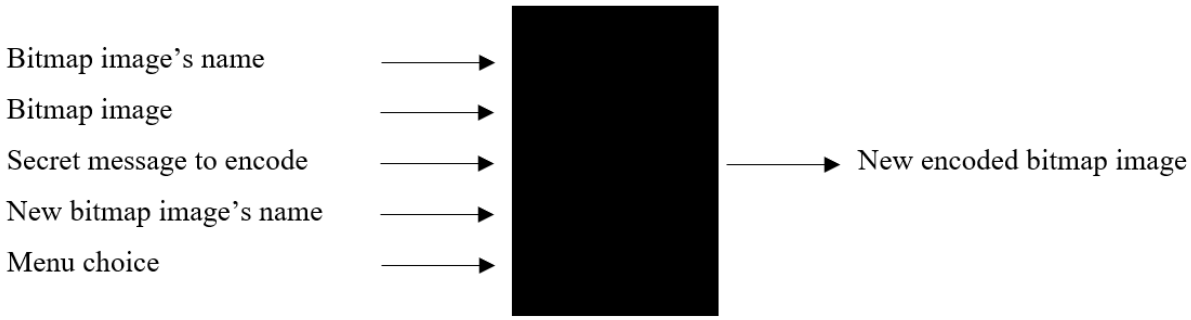


At the end of the secret message a delimiter character '\0' should be embedded to pixel bytes so the message can be extracted back in the decode process. To set the LSB of the color bytes, common bitmask functions will be used. These bitmask functions include bitwise operators as right shift, AND, OR. For this, each of the bits of each character of the secret message will be compared with 1 using the bitmask function (which includes bitwise right shift and bitwise AND). If the current bit is 1 or 0 then the LSB of the processing color byte should be altered accordingly. For this, the LSB of the color's byte will be masked to 1 or 0 (which includes bitwise OR).

The input/output description for the encoding process is provided below.

The inputs: Bitmap image's name, Bitmap image, Secret message to encode, New bitmap image's name, Menu choice
The outputs: New bitmap image

The inputs are received from the input text file.
Alternatively, the inputs and outputs could be represented using a black-box diagram



2) Decoding process

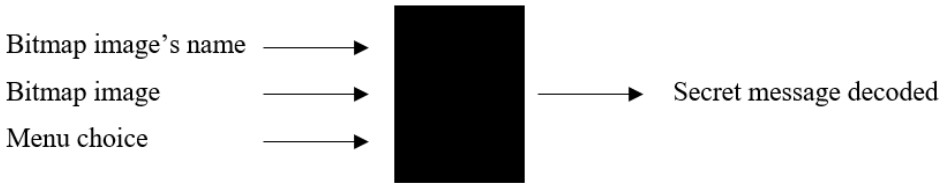
For the decoding process the program will need a bitmap image's name only, which is provided by the user. If the name of the file provided by the user is valid, the program reads data about every single pixel of the image. Each pixel contains three bytes representing the intensities of the three colors (red, green, and blue) as in the encoding process. To decode the secret message from the image, it will be assumed that the message in the bitmap image is hidden according to the Least Significant Bit (LSB) image steganography. For this, the LSB of eight consecutive color bytes will be analyzed. Again, bitmask functions will be used. If the processing LSB is 1 (which can be done using the bitmask function that includes bitwise AND operator) then the processing bit of the character of the secret message corresponding to these particular eight bytes will be set to 1 (which involves bitwise OR). In other words,

a simple masking process takes places, when particular bit of the given byte is set to 1 or 0 according to what is the LSB of the color intensity byte.

The input/output description for the decoding process is provided below.

The inputs: Bitmap image's name, Bitmap image, Menu choice
The outputs: Secret message decoded

Alternatively, the inputs and outputs could be represented using a black-box diagrams:



Apart from these two parts, one of the program's essential parts is a main menu. The menu consists of three options: encode, decode, and exit the program. When the user selects the encode option, he/she is prompted to enter the secret message and the name of the bitmap file where the secret message must be encoded in. If the file name or the provided file is invalid the corresponding error message will be displayed. Otherwise, it will show a message that the bitmap image is read successfully. Afterwards, the user is prompted to enter the name of the new encoded image name. The new encoded image will be created and saved; the confirming message will be displayed. When the user selects the decode option, the program prompts the user for the encoded bitmap image name, extracts the secret test message, and displays it on the output screen. When the user selects the exit option, the program is terminated. The menu will be shown after every encode/decode process until exit is selected. Lastly, whenever error message is displayed, the program goes back to the menu.

STEP 3: Test Cases and Algorithm Design

A) Test Cases

The expected output can be described in this step. Overall, in this step, the main menu and exit option, two cases for the encoding process and two cases for the decoding process will be described.

The following table provides a set of test cases that can be used to test the main menu:

Test Case	Choice	Result
1	*1*	*start the Encoding process*
2	*2*	*start the Decoding process*
3	*-1*	*Exit the program*
4	*Any other number*	*Print "Invalid input, please, try again" and prompt again*

Encoding process test cases:

1) NYUAD.bmp

For this test case we will try to describe the encoding process of "ThisIsTheSecretMessage!" to the NYUAD.bmp image. As stated in the Step2, according to the LSB image steganography the LSBs of the initial RGBs of first pixels will be altered. Since the LSB has the least significant effect to the pixel's color, human eye will not be able to recognize the change. The predicted picture are provided below. The picture on the left is the original one, while the picture on the right is the encoded one.

Pixel 1
Red: 00111011 Green: 01000110 Blue: 01001110

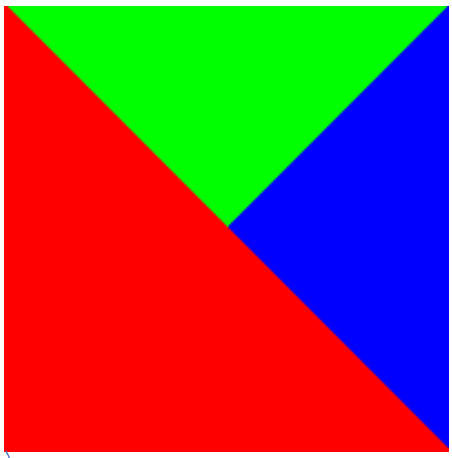
Pixel 1
Red: 00111010 Green: 01000110 Blue: 01001111

The LSBs of the Red, Green, and Blue bytes of the first pixel are assumed to be changed (They are highlighted). They should be changed in this way, because the first character in the secret message is 'T', which ASCII code is 01010100, and the first three LSBs are 0, 0, 1. This was explained more in-depth in step 2. The process goes in the same way for the pixel 2 and so forth until the program reaches the end of the secret message.

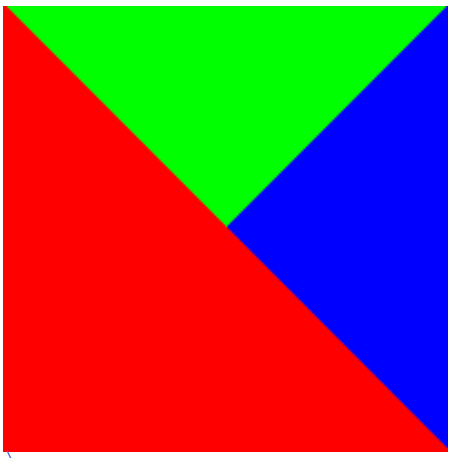
As it can be seen from here, there shouldn't be any changes in the overall image for human eye. But if we are able to check the pixel bytes, they should be different. This test case can be verified using the decoding process.

2) Figure.bmp

For this test case we will try encoding a message "This is the secret message with spaces!" As in the test case 1, according to the LSB image steganography the LSBs of the initial RGBs of first pixels will be altered. The predicted picture are provided below. The picture on the left is the original one, while the picture on the right is the encoded one.



Pixel 1
Red: 11111111 Green: 00000000 Blue: 00000000



Pixel 1
Red: 11111110 Green: 00000000 Blue: 00000001

The LSBs of the Red, Green, and Blue bytes of the first pixel are assumed to be changed (They are highlighted). The explanation of why the pixel one is altered in this way is the same as in the test case 1. The first character in the secret message is 'T', which ASCII code is 01010100, and the first three LSBs are 0, 0, 1. The process goes in the same way for the pixel 2 and so forth until the program reaches the end of the secret message.

As it can be seen from here too, there shouldn't be any changes in the overall image. But if we are able to check the pixel bytes, they should be different. This test case can be verified using the decoding process.

Decoding process test cases:

For the decoding process it is sufficient just to check the original images and encoded images from the encoding process test cases

1) NYUAD.bmp and NYUADencoded.bmp

For this test case there should be used two different pictures, one with a meaningful secret message encoded in it and the one with no such a message. Those should be gone through the decoding process of the program and the secret messages should be displayed on the console. In this test case we will suppose that the one with the secret message has an encoded message "ThisIsTheSecretMessage!". The program should display this message for the encoded message, while for the one with no message the program should display only series of meaningless characters. This happens because the program will still analyze LSBs of the pixels of the image even if they don't make sense. Whenever it reaches the series of 8 bytes all of which LSBs are 0, the program stops displaying characters.

2) Figure.bmp and Figureencoded.bmp

The similar process is expected to be seen for these two images (Figure.bmp and Figureencoded.bmp). While the first one has no meaningful encoded message, the second one is assumed to have a secret message "This is the secret message with spaces!" When the first image is decoded, some characters are expected to be seen that does not make sense. Meanwhile, when the second image is decoded, the secret message above is expected to be seen on the console.

B) Algorithm (pseudocode)

```
Function Encode
    Pass In: imageWidth, imageHeight, size, secretMessageToEncode pointer, imageData pointer
    Assign 0 to i
    Assign 0 to bit
    Assign false to flag
    Assign 0 to row
    Repeat the following while row is less than imageHeight
        Assign 0 to col
        Repeat the following while col is less than imageWidth
            Assign 0 to channel
            Repeat the following while channel is less than 3
                Assign secretMessageToEncode[i] right shifted bitwise to Value
                Assign Value bitwise AND 1 to Value
                If Value is 1 and imageData[row][col][channel] mod 2 is 0 then
                    Assign imageData[row][col][channel] bitwuse OR 1 to
                        imageData[row][col][channel]
                    Otherwise if imageData[row][col][channel] mod 2 is 1 then
                    Assign imageData[row][col][channel] bitwise AND 254 to
                        imageData[row][col][channel]
                Increment bit by 1
                If bit equals to 8 then
                    Assign 0 to bit
                    Increment i by 1
                If i equals to size then
                    Assign true to flag
```

If flag is true then exit the current loop
Increment channel by 1
If flag is true then exit the current loop
Increment col by 1
If flag is true then exit the current loop
Increment row by 1
Pass Out: flag

Function Decode

Pass In: *imageWidth, imageHeight, secretMessageDecoded pointer, imageData pointer*
*Assign (imageHeight*imageWidth*3/8) to SizeMax*
Declare pointer Temp
Dynamically allocate memory for temp as an array of characters, which size equals to SizeMax
Assign 0 to i
Repeat the following while i is less than SizeMax
Assign '\0' to Temp[i]
Increment i by 1
Assign 0 to secretMessageSize
Assign 0 to bit
Assign false to flag
Assign 0 to row
Repeat the following while row is less than imageHeight
Assign 0 to col
Repeat the following while col is less than imageWidth
Assign 0 to channel
Repeat the following while channel is less than 3
Assign imadeData[row][col][channel] bitwise AND 1 to Value
If Value is 1 then
Increment Temp[secretMessageSize] by 2^bit
Increment bit by 1
If bit equals to 8 then
Assign 0 to bit
Increment secretMessageSize by 1
If secretMessageSize equals to SizeMax then
Assign true to flag
If secretMessageSize is more than 0 then
If Temp[secretMessageSize-1] equals 0 then
Assign true to flag
If flag is true then exit the current loop
Increment channel by 1
If flag is true then exit the current loop
Increment col by 1
If flag is true then exit the current loop
Increment row by 1
Dynamically allocate memory for secretMessageDecoded as an array of size equal to secretMessageSize
Assign 0 to k
Repeat the following while k is less than secretMessageSize
Assign Temp[k] to secretMessageDecoded[k]
Increment k by 1
Deallocate the memory block ponted by Temp
Pass Out: secretMessageSize

Function ReadTheInput

Pass In: input pointer
Define max as 10000000

Declare pointer temp
Dynamically allocate memory for temp as an array of characters, which size equals to max
Assign 0 to inputSize
Assign false to flag
Repeat the following while flag is false or byte is not '\n'
Read the input character and assign it to byte
If byte is not '\n' then
Assign byte to temp[inputSize]
Increment inputSize by 1
Set flag to true
Increment inputSize by 1
Dynamically allocate memory for input as an array of characters, which size equals to inputSize
Assign 0 to i
Repeat the following while i is less than inputSize-1
Assign temp[i] to input[i]
Increment I by 1
Assign '\0' to input[inputSize-1]
Deallocate the memory block ponted by temp
Pass Out: inputSize

Function ReleaseMemory

Pass In: *imageWidth, imageHeight, imageData pointer*
Assign 0 to row
Repeat the following while row is less than imageHeight
Assign 0 to col
Repeat the following while col is less than imageWidth
Assign 0 to channel
Repeat the following while channel is less than 3
Deallocate the memory block pointed by ImageData[row][col]
Increment channel by 1
Deallocate the memory block pointed by ImageData[row]
Increment col by 1
Deallocate the memory block pointed by ImageData
Increment row by 1
Pass Out: nothing

The main function

Print "Dear User! Please, read the user guide provided in the report"
Declare a pointer ImageData that points to a 3D array
Declare pointers fileName, newFileName
Declare pointers secretMessageDecoded, secretMessageToEncode
Declare imageWidth, imageHeight
Repeat the following while the selection is not -1
Print "Enter 1 to encode a BMP file"
Print "Enter 2 to decode a BMP file"
Print "Enter -1 to exit the program"
Read the input and assign it to selection
If selection is equal to 1 then
Print "You have selected to encode a BMP file"
Print "Please, enter the name of the BMP file to encode the secret message"
Print "Do not forget to include .bmp"
Call ReadTheInput function, passing pointer to fileName
Call ReadBitmapImage function, passing imageWidth, imageHeight and pointers to fileName and imageData
Assign the returning value of ReadImage to OK
If OK is false then

Print “Please, start the process again”
Skip the current iteration of the loop
Print “The”, fileName, “was successfully read by the program”
Print “To proceed, please, enter your secret message that needs to be encoded”
Call ReadTheInput function, passing pointer to secretMessageToEncode
Assing the returning value of ReadTheInput to size
Call Encode function, passing imageWidth, imageHeight, size and pointers to secretMessageToEncode and imageData
Print “Please, enter the name of the new encoded BMP file”
Print “Do not forget to include .bmp”
Call ReadTheInput, passing newFileName
Call WriteBitmapImage function, passing imageWidth, imageHeight and pointers to newFileName and imageData
Print “The new encoded BMP file was created and saved with a name ”, newFileName
Call ReleaseMemory function, passing imageWidth, imageHeight and pointer to imageData
Deallocate the memory block pointed by fileName pointer
Deallocate the memory block pointed by newFileName pointer
Deallocate the memory block pointed by secretMessageToEncode pointer
Otherwise if selection is equal to 2 then
Print “You have selected to decode a BMP file”
Print “Please, enter the name of the BMP file to decode the secret message”
Print “Do not forget to include .bmp”
Call ReadTheInput function, passing pointer to fileName
Call ReadImage function, passing imageWidth, imageHeight and pointers to fileName and imageData
Assign the returning value of ReadImage to OK
If OK is false then
Print “Please, start the process again”
Skip the current iteration of the loop
Print “The”, fileName, “was successfully read by the program”
Call Decode function, passing imageWidth, imageHeight, pointers to secretMessageDecoded and imageData
Print “The secret message was decoded from ”, fileName, “ succesfully”
Print “The secret message is: ”, newline
Print the content pointed by secretMessageDecoded
Call ReleaseMemory function, passing imageWidth, imageHeight and pointer to imageData
Deallocate the memory block pointed by fileName pointer
Deallocate the memory block pointed by secretMessageDecoded pointer
Otherwise if selection is equal to -1 then exit the loop
Otherwise print “Invalid input, please, try again”

Exit

STEP 4: Implementation

Source.cpp

```
/*------------------------------------------------------------------------*/
/* Name: Sagynbek Talgatuly Student Number: st4121                        */
/* Date: November 13. 2020                                                */
/* Program: Source.cpp                                                    */
/* Description: This program encodes a secret message provided by the user */
/*              into the bitmap image using LSB steganography and decodes  */
/*              the secret message hidden in the bitmap image displaying   */
/*              it on the console.                                         */
/*------------------------------------------------------------------------*/
```

```
//including all the needed libraries and header files
#include <iostream>
#include "BitmapHelper.h"
#include "Encode.h"
#include "Decode.h"

using namespace std;

//prototype for the ReadInputFunction, which will be used to
//read the input from the user and store it in the char array dynamically
//this function will use pointer pointing to a char array as an argument
int ReadTheInput(char*& input);

int main() {
	//declaring the necessary pointers and variables
	unsigned char*** imageData;
	int imageWidth, imageHeight;
	char* fileName;
	char* newFileName;
	char* secretMessageToEncode;
	char* secretMessageDecoded;

	//Introduction message for the user
	cout << "Dear User!                                                    " << endl;
	cout << "Please, read the user's guide provided in the report          " << endl;
	cout << "Note: Make sure that the bitmap images you will input         " << endl;
	cout << "      passes the validation process and the names of          " << endl;
	cout << "      the bitmap images you will input have the same          " << endl;
	cout << "      format as the examples in the user's guide.             " << endl << endl;

	int selection;
	do {
		cout << endl;
		//displaying the main menu
		cout << "Enter 1 to encode a BMP file" << endl;
		cout << "Enter 2 to decode a BMP file" << endl;
		cout << "Enter -1 to exit the program" << endl;

		//reading the choice of the user
		cin >> selection;
		//if the choice is 1 then the encoding process begins
		if (selection == 1) {
			//corresponding messages are shown
			cout << "You have selected to encode a BMP file" << endl;
			cout << "Please, enter the name of the BMP file to encode the secret
				message" << endl;
			cout << "Do not forget to include .bmp" << endl;
			//the filename is read using the reading function
			ReadTheInput(fileName);
			//the provided ReadBitmapImage is called and it's return value is assigned
				to variable OK
			bool ok = ReadBitmapImage(fileName, imageData, imageWidth,
				imageHeight);
			//if the validation has not succeeded then the error message is
				displayed and the program
			//returns to main menu, otherwise it will continue the encoding process
			if (!ok) { cout << "Please, start the process again" << endl; continue; }
			//if validation is successful then the user is prompted to input the
				secret message
			cout << "The " << fileName << " was successfully read by the program"
				<< endl;
			cout << "To proceed, please, enter your secret message that needs to be
				encoded" << endl;
			//the secret message is read by the reading function and its size i
			// assigned to a variable because it will be used in the next operations
			//the maximum size of the input is the following, because it represents
				the maximum number
			//of characters that the bitmap image provided can store
			int size = ReadTheInput(secretMessageToEncode);
			//the secret message is encoded into the bitmap image
			//the imageData is altered according to the secret message
			encode(secretMessageToEncode, size, imageData, imageWidth, imageHeight);
			cout << "Please, enter the name of the new encoded BMP file" << endl;
			cout << "Do not forget to include .bmp" << endl;
			//when the encoding process is completer, the program asks for the name
			//of the new encoded bitmap image to be saved
			//the name is stored in a dynamically allocated memory pointed by
				newFileName
```

```
                        ReadTheInput(newFileName);
                        //the new encoded BMP file is created and save using the provided function
                                WriteBitmapImage
                        WriteBitmapImage(newFileName, imageData, imageWidth, imageHeight);
                        cout << "The new encoded BMP file was created and saved with a name " <<
                                newFileName << endl;
                        //the memory allocated dynamically for the imageData,
                        //fileName, newFileName, and secretMessageToEncode is released
                        ReleaseMemory(imageData, imageHeight, imageWidth);
                        delete[] fileName;
                        delete[] newFileName;
                        delete[] secretMessageToEncode;
                }

                //if the choice is 2 then the decoding process begins
                else if (selection == 2) {
                        //corresponding messages are shown
                        cout << "You have selected to decode a BMP file" << endl;
                        cout << "Please, enter the name of the BMP file to decode the secret
                                message" << endl;
                        cout << "Do not forget to include .bmp" << endl;
                        //the filename is read using the reading function
                        ReadTheInput(fileName);
                        //the provided ReadBitmapImage is called and it's return value is assigned
                                to variable OK
                        bool ok = ReadBitmapImage(fileName, imageData, imageWidth, imageHeight);
                        //if the validation has not succeeded then the error message is displayed
                                and the program
                        //returns to main menu, otherwise it will continue the encoding process
                        if (!ok) { cout << "Please, start the process again" << endl; continue; }
                        //if the validation is successful then the program starts decoding the
                                image
                        //the received secret message is stored in the dynamically allocate
                        // memory pointed by secretMessageDecoded
                        cout << "The " << fileName << " was successfully read by the program"
                                << endl;
                        decode(secretMessageDecoded, imageData, imageWidth, imageHeight);
                        //after the decoding process is completed, the secret message is
                                displayed
                        cout << "The secret message was decoded from " << fileName << "
                                successfully" << endl;
                        cout << "The secret message is:" << endl;
                        cout << secretMessageDecoded << endl;
                        //lastly, all the allocated memory is released
                        ReleaseMemory(imageData, imageHeight, imageWidth);
                        delete[] fileName;
                        delete[] secretMessageDecoded;
                }
                //if the choice is -1 then the program is exited
                else if (selection == -1) break;
                //if none of the options are chosen, then the error message is displayed
                //and the user is prompted to choose again
                else cout << endl << "Invalid input, please, try again";
        } while (selection != -1);
        return 0;
}

//the below is the definition of the special read function
//this function is used to read the input character by character and to
//save the input in the dynamically allocated char array
int ReadTheInput(char*& input) {
        //defining the maximum length og the secret message
        #define max 10000000
        //creating temporary char array dynamically with the provided maximum size
        char* temp;
        temp = new char[max];
        //the needed variables are declared and initialized
        char byte;
        int inputSize = 0;
        bool flag = false;
                //reading the character from the console and assigning it to the current
                //element of the temporary array
                //it is worth noting that the size of the input is also counted
                do {
                        cin.get(byte);
                        if (byte != '\n') { temp[inputSize] = byte; inputSize++; flag = true; }
                } while (!flag || byte != '\n');

                inputSize++;
```

```
		//the memory fot the input pointer is dynamically allocated with
		//an actual size of the input
		//the values stored in the temp is copy-pasted to the input array
		input = new char[inputSize];
		for (int i = 0; i < inputSize - 1; i++) {
			input[i] = temp[i];
		}
		input[inputSize - 1] = '\0';

		//when the input array is filled, the temporary array is simply deleted
		delete[] temp;

		return inputSize;
}
```

Encode.h

```
#pragma once
bool encode(const char* secretMessageToEncode, int size, unsigned char***& imageData, int
	imageWidth, int imageHeight) {
	//the needed variables are declared and initialized
	int i = 0;
	int bit = 0;
	bool flag = false;
	for (int row = 0; row < imageHeight; row++)
		{
			for (int col = 0; col < imageWidth; col++)
			{
				for (int channel = 0; channel < 3; channel++)
				{
					//each of the first bytes of the image undergo the
						following process
					//if the current bit of the current character of the secret
						message is 1
					//and the LSB of the byte representing the intensity of the
						color (r,g,b) is 0
					//then the LSB of this byte is altered using the masking
						with OR 1 operation
					if ((secretMessageToEncode[i] >> bit) & 1)
					{
						if ((int)imageData[row][col][channel] % 2 == 0)
							imageData[row][col][channel] |= 1;

					}
					//if the current bit of the current character of the secret
						message is 0
					//and the LSB of the byte representing the intensity of the
						color (r,g,b) is 1
					//then the LSB of this byte is altered using the masking
						with & 254 operation
					else if ((int)imageData[row][col][channel] % 2 == 1)
					{
						imageData[row][col][channel] &= 254;
					}
					//incrementing bit by 1
					bit++;
					//if bit is 8, it means that one character has been encoded
					//the bit is set to 0 and the i which represents
					//the number of characters encoded is incremented
					if (bit == 8) { bit = 0; i++; }
					//if the i equals to size, it means that all the characters
					// are encoded and we need to exit the all loops
					//for that the flag is used
					if (i == size) flag = true;
					if (flag) break;
				}
				if (flag) break;
			}
			if (flag) break;
		}
	return flag;
}
```

Decode.h

```
#pragma once
//the cmath library is included
#include <cmath>
```

```
//the maximum size is defined as imageWidth*imageHeight*3/8
//because that number represents the maximum number of characters
//the input bitmap image can hold

int decode(char*& secretMessageDecoded, unsigned char*** imageData, int imageWidth, int
        imageHeight) {
        const int sizeMax = (int)imageWidth * imageHeight * 3 / 8;
        //the temporary char array is created dynamically
        char* temp;
        temp = new char[sizeMax];
        //the temp array is filled with 0s
        for (int i = 0; i < sizeMax; i++) {
                temp[i] = '\0';
        }
        //needed variable are declared and initialized
        int secretMessageSize = 0;
        int bit = 0;
        bool flag = false;
        for (int row = 0; row < imageHeight; row++)
        {
                for (int col = 0; col < imageWidth; col++)
                {
                        for (int channel = 0; channel < 3; channel++)
                        {
                                //each of the first bytes of the image undergo the following
                                        process
                                //if the LSB of the byte representing the intensity of the color
                                        (r,g,b) is 1
                                //the current bit of the current character of the secret message is
                                        set to 1
                                //to do this the bitwise operator OR is used along with the power
                                        of 2
                                if (imageData[row][col][channel] & 1) temp[secretMessageSize] |=
                                        (int)pow(2, bit);
                                //bit is incremented
                                bit++;
                                //if bit is 8, it means that one character has been decoded
                                //the bit is set to 0 and the secretMessageSize is incremented
                                if (bit == 8) { bit = 0; secretMessageSize++; }

                                //when the character decoded is \0 or the image is analyzed
                                        entirely
                                //the program exits all the loops
                                if (secretMessageSize == sizeMax) { flag = true; }
                                if (secretMessageSize > 0) {
                                        if (temp[secretMessageSize - 1] == 0) { flag = true; }
                                }
                                if (flag) break;
                        }
                        if (flag) break;
                }
                if (flag) break;
        }

        //the secretMessageDecoded new array with an acual message size
        //is created and the values stored in tem array are copy-pasted
        //to this new array
        secretMessageDecoded = new char[secretMessageSize];
        for (int i = 0; i < secretMessageSize; i++) {
                secretMessageDecoded[i] = temp[i];
        }

        //the temporary pointer is deallocated
        delete[] temp;

        return secretMessageSize;
}
```

ReleaseMemory function in the BitmapHelper.h

```
void ReleaseMemory(unsigned char*** imageData, int imageHeight, int imageWidth) {

        for (int row = 0; row < imageHeight; row++)
        {
                for (int col = 0; col < imageWidth; col++)
                {
                        //memory used by the content pointed by each of
                        //the imageData[row][col] is deallocated
                        delete[] imageData[row][col];
                }
```

```
            //memory used by the content pointed by each of
            //the imageData[row] is deallocated
            delete[] imageData[row];
        }
        //memory used by the content pointed by each of
        //the imageData is deallocated
        delete[] imageData;

}
```

STEP 5: Tests and Verification (and Debugging)

For the Menu Test:

Test Case 1: For choice = 1, the output is:

```
Dear User!
Please, read the user's guide provided in the report
Note: Make sure that the bitmap images you will input
      passes the validation process and the names of
      the bitmap images you will input have the same
           format as the examples in the user's guide.


Enter 1 to encode a BMP file
Enter 2 to decode a BMP file
Enter -1 to exit the program
1
You have selected to encode a BMP file
Please, enter the name of the BMP file to encode the secret message
Do not forget to include .bmp
```

Test Case 2: For choice = 2, the output is:

```
Dear User!
Please, read the user's guide provided in the report
Note: Make sure that the bitmap images you will input
      passes the validation process and the names of
      the bitmap images you will input have the same
           format as the examples in the user's guide.


Enter 1 to encode a BMP file
Enter 2 to decode a BMP file
Enter -1 to exit the program
2
You have selected to decode a BMP file
Please, enter the name of the BMP file to decode the secret message
Do not forget to include .bmp
```

Test Case 3: For choice = -1, the output is:

```
Dear User!
Please, read the user's guide provided in the report
Note: Make sure that the bitmap images you will input
      passes the validation process and the names of
      the bitmap images you will input have the same
           format as the examples in the user's guide.


Enter 1 to encode a BMP file
Enter 2 to decode a BMP file
Enter -1 to exit the program
-1

C:\Users\Sagin\source\repos\Assignment3\Debug\Assignment3.e
Чтобы автоматически закрывать консоль при остановке отладки
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно…
```

Test Case 4: For choice = 45, the output is:

```
Dear User!
Please, read the user's guide provided in the report
Note: Make sure that the bitmap images you will input
      passes the validation process and the names of
      the bitmap images you will input have the same
           format as the examples in the user's guide.


Enter 1 to encode a BMP file
Enter 2 to decode a BMP file
Enter -1 to exit the program
45

Invalid input, please, try again
Enter 1 to encode a BMP file
Enter 2 to decode a BMP file
Enter -1 to exit the program
```

For the Encoding process
Test Case 1: NYUAD.bmp

```
Enter 1 to encode a BMP file
Enter 2 to decode a BMP file
Enter -1 to exit the program
1
You have selected to encode a BMP file
Please, enter the name of the BMP file to encode the secret message
Do not forget to include .bmp
NYUAD.bmp
The NYUAD.bmp was successfully read by the program
To proceed, please, enter your secret message that needs to be encoded
ThisIsTheSecretMessage!
Please, enter the name of the new encoded BMP file
Do not forget to include .bmp
NYUADencoded.bmp
The new encoded BMP file was created and saved with a name NYUADencoded.bmp
```

Test Case 2: Figure.bmp

```
Enter 1 to encode a BMP file
Enter 2 to decode a BMP file
Enter -1 to exit the program
1
You have selected to encode a BMP file
Please, enter the name of the BMP file to encode the secret message
Do not forget to include .bmp
Figure.bmp
The Figure.bmp was successfully read by the program
To proceed, please, enter your secret message that needs to be encoded
This is the secret message with spaces!
Please, enter the name of the new encoded BMP file
Do not forget to include .bmp
Figureencoded.bmp
The new encoded BMP file was created and saved with a name Figureencoded.bmp
```

For the Decoding process

Test Case 1:

```
Enter 1 to encode a BMP file
Enter 2 to decode a BMP file
Enter -1 to exit the program
2
You have selected to decode a BMP file
Please, enter the name of the BMP file to decode the secret message
Do not forget to include .bmp
NYUAD.bmp
The NYUAD.bmp was successfully read by the program
The secret message was decoded from NYUAD.bmp successfully
The secret message is:
I♀lyHrR┌VнTIb‼П┌ ♥T$┤╖╚Cy6├1╣q°_Udь8Адrbtp*♫Тнс;qт─Йm'╢У─▒У─Gь┤U┌гн
─♫ак

Enter 1 to encode a BMP file
Enter 2 to decode a BMP file
Enter -1 to exit the program
2
You have selected to decode a BMP file
Please, enter the name of the BMP file to decode the secret message
Do not forget to include .bmp
NYUADencoded.bmp
The NYUADencoded.bmp was successfully read by the program
The secret message was decoded from NYUADencoded.bmp successfully
The secret message is:
ThisIsTheSecretMessage!
```

Test Case 2:

```
Enter 1 to encode a BMP file
Enter 2 to decode a BMP file
Enter -1 to exit the program
2
You have selected to decode a BMP file
Please, enter the name of the BMP file to decode the secret message
Do not forget to include .bmp
Figure.bmp
The Figure.bmp was successfully read by the program
The secret message was decoded from Figure.bmp successfully
The secret message is:
huhm;h ftyfyudvts49598.,=IT$IT$IT$IT$IJT$IT$IT$$IT$IT$IRT$jlkjlb fагщ хрнх

Enter 1 to encode a BMP file
Enter 2 to decode a BMP file
Enter -1 to exit the program
2
You have selected to decode a BMP file
Please, enter the name of the BMP file to decode the secret message
Do not forget to include .bmp
Figureencoded.bmp
The Figureencoded.bmp was successfully read by the program
The secret message was decoded from Figureencoded.bmp successfully
The secret message is:
This is the secret message with spaces!
```

which are in agreement with the test cases expected output.

It is worth noting that the differences in the encoded and original images are non-visible:

NYUAD.bmp

NYUADencoded.bmp

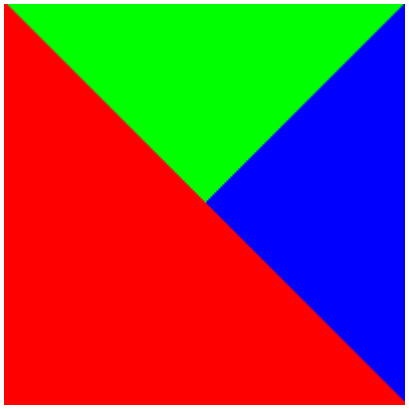
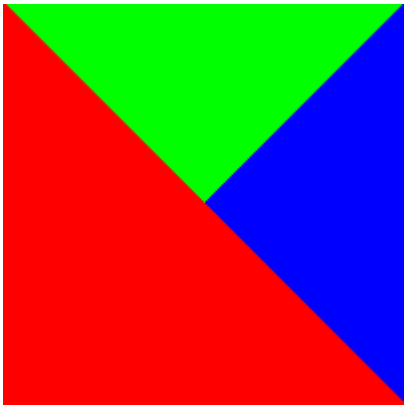


Figure.bmp



Figureencoded.bmp

Thus, we conclude that the program is functioning correctly since all test cases are verified.

The User's Guide

1. *To execute the program, the user needs to compile and run the code found in the file named Source.cpp*
2. *The user is prompted to choose among Encode, Decode, and Exit options by entering 1, 2 or -1 correspondingly. It is assumed that the user enters an integer. If none of the options are met the error message is displayed, and the menu is shown again.*
    - *If the Encode option was selected, the user will be prompted to enter the name of the bitmap image. It is assumed that the image is stored in the project folder with the Source.cpp and has the exact same format of name as the next examples. Examples: "NYUAD.bmp", "Earth.bmp". Entering empty name is impossible. The extension .bmp, its existence, and its bit value (should be 24 bit) will be validated. In case validation fails, the corresponding message will be displayed, and the program will return to the main menu. If the validation succeeds, the program will display the corresponding message and the user will be prompted to enter the secret message to be encoded. It is assumed that the secret message is a single line and again, empty secret message is impossible. Lastly, after entering the secret message, the user will be asked to enter the name for the new encoded bitmap image. It is assumed that the name format is the same as of the examples above. The new encoded bitmap image will be created and saved.*
    - *If the Decode option was selected, the user will be prompted to enter the name of the bitmap image. It is assumed that the image is stored in the project folder with the Source.cpp and has the exact same format of name as the next examples. Examples: "NYUAD.bmp", "Earth.bmp". The extension .bmp, its existence, and its bit value (should be 24 bit) will be validated. In case validation fails, the corresponding message will be displayed, and the program will return to the main menu. If the validation succeeds, the program will display the confirming message and the secret message will be decoded and displayed on the console.*
    - *If the Exit option was selected, the program will be exited.*

3. *After the chosen analysis are completed the step 2 will be repeated until exit option is chosen.*

Note: The maximum number of input characters is 10000000.