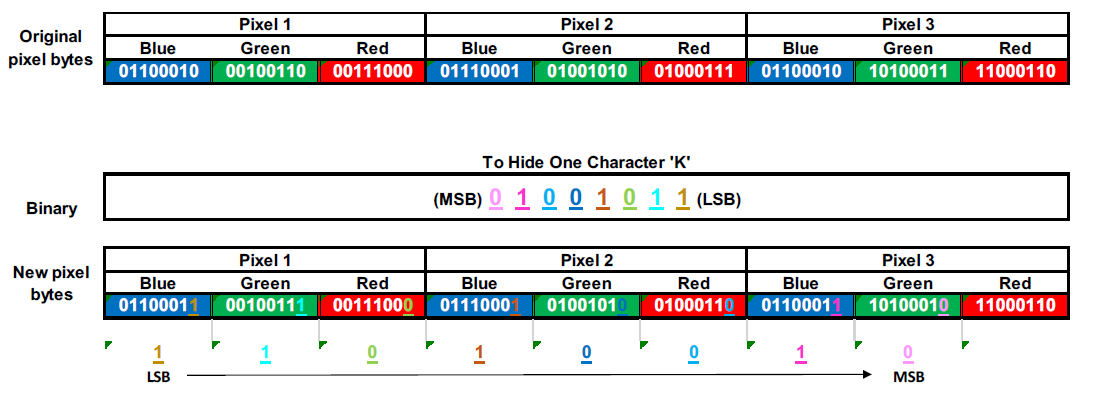
Assignment 3

Problem Identification and Statement:

We must make a program that uses Least Significant Bit image steganography which is used to encode a secret message into the bitmap image by replacing the least significant bit of a pixel to have the least visible effect on the color of the pixel. The program can also decode the secret message from an encoded bitmap file.

Gathering Information:

When encoding the Least Significant Bit (LSB) image steganography 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.



The message can then be decoded by taking the last bit of each byte (which is the LSB) and then combining it into 1 byte by using bitwise manipulation and shifting the values to the left until the integer reaches 8 digits. This is repeated until a delimiter value is reached. The bytes can then be represented as letters and then it can be displayed as the message.

Menu description:

Option 1: 1. Encode

Option 2: 2. Decode

Option 3: 3. Exit

Input/Output:

Encoding:

Input:1

Input: Secret message

Input: File name to encode

Then a file gets created in the project directory

Decoding:

Input: 2

Input: File name to decode

Output: Decoded secret message

Exit:

Input:3

Program exits

Test Cases:

Case 1: Checking the Encoding functionality

Input 1 to Encode

Input the secret code to be encoded

Input Bitmap file name

Creates encoded file in the directory

Case 2: Checking the Decoding functionality

Input 2 to Decode

Input Bitmap file name

Output secret message

Case 3: Checking the Exit functionality

Input 3 to Exit

The program exits

Case 4: Checking the Invalid input functionality

Input an invalid value. e.g.: 5

Output Error message

Loop back to the main menu

Algorithm:

Source.cpp

Assign exit\_program to false

Main function

Declare variable input

Declare variable file\_name

Declare variable text\_to\_encode

Declare variable decoded\_text

Assign ret to false

Declare variable tmp

Do

Print “Image Steganography”

Print “1. Encode”

Print “2. Decode”

Print “3. Exit”

Print “Input [1-3] : “

Assign the input to input

If input is not equal to 1and input is not equal to 2 and input is not equal to 3

Print “Input is invalid. Please try again.”

If input less than 1 or input is greater than 3

Continue

If input is equal to 1

Print “Enter the secret message to encode : “

Assign input into tmp

Assign input to text\_to\_encode

Print “Enter image file name: “

Assign input to file\_name

Assign ret to the Encode function

If ret is assigned to encode

Then Print “Encoding successful”

Otherwise

Print “Encoding Failed”

If input is equal to 2

Print “Enter image file name to retrieve hidden text : “

Assign input to file\_name

Assign ret to the Decode function

If ret is assigned to encode

Then Print “Decoding successful”

Print”Decoded message: “ “’decoded\_text’””

Otherwise

Print “Encoding Failed”

If input is equal to 3

Call the Exit function

While exit\_program is false

Exit function

Print “Program exited successfully”

Assign exit\_program to true

Encoder.cpp

Encode(s, file\_name) function to encode text into a bitmap file

Assign imageWidth to 0

Assign imageHeight to 0

Assign b1 to ReadBitmapImage(file\_name,imageData,imageWidth,imageHeight)

If b1 is false

Return false

Assign n to the size of s

Assign val to 0

Assign bit\_val to 0

Assign counter to 0

Assign row to 0

Assign col to 0

Assign channel to 0

Assign i to 0

If i is less than n

Then assign c to s[i]

Assign val to the integer value of c

Assign j to 0

If j is less than 8

Then assign bit\_val to val % 2

Assign val to val/2

Assign channel to counter % 3

Assign col to (counter/3)% imageWidth

Assign row to (counter/(3 \* imageWidth)) % imageWidth

Assign imageData[row][col][channel] to (Bitshift 1 to the left (Bitshift imageData 1 to the right)) + bit\_val

Increment counter

Increment j

Increment i

Assign file\_name\_size to the size of file\_name

Declare array encoded\_file\_name[MAX\_FILE\_NAME\_SIZE]

If file\_name\_size is greater than 4

Then Assign encoded\_file\_name[file\_name\_size -1] to ‘\0’

Assign encoded\_file\_name[file\_name\_size -2] to ‘\0’

Assign encoded\_file\_name[file\_name\_size -3] to ‘\0’

Assign encoded\_file\_name[file\_name\_size -4] to ‘\0’

Declare end\_file\_name(encoded\_file\_name)

Assign end\_file\_name to end\_file\_name + “\_encoded.bmp”

Assign b2 to function WriteBitmapImage(end\_file\_name, imageData, imageWidth, imageHeight)

Call function ReleaseMemory(imageData, imageHeight, imageWidth) to write the data into the bitmap image

If b2 is false

Then return false

Print ”Encoded image file name : “, end\_file\_name

Return true;

Decoder.cpp

Decode(file\_name, ss)to decode text hidden text in a bitmap image

Assign imageWidth to 0

Assign imageHeight to 0

Assign b1 to ReadBitmapImage(file\_name,imageData,imageWidth,imageHeight)

If b1 is false

Return false

Assign row to 0

Assign col to 0

Assign channel to 0

Declare c

Assign collected\_bit\_count to 0

Assign curr\_bit to 0

Assign val to 0

Assign counter to 0

Assign ss to “”

Assign string\_index to 0

While true

Assign channel to counter % 3

Assign col to (counter/3)%imageWidth

Assign row to (counter/(3 \* imageWidth)) % imageHeight

Assign curr\_bit to imageData[row][col[channel] % 2

Increment val by (current\_bit bitshifted to the left by collected\_bit\_count)

Increment collected\_bit\_count

If collected\_bit\_count is equal to 8

Assign temp to character version of val

Assign ss to ss + temp

If val is equal to 0

Skip to next iteration

Assign val to 0

Assign collected\_bit\_count to 0

Increment counter

Call function ReleaseMemory(imageData, imageHeight, imageWidth) to delete the dynamic memory

Return true

BitmapHelper.h ReleaseMemory Function

Assign row to 0

If row is less than imageHeight

Then Assign col to 0

If col is less than imageWidth

Delete imagedata[row][col]

Increment col

Delete imageData[row]

Increment row

Delete imageData

Code:

Source.cpp

#include <iostream>

#include <string>

#include "Decoder.h"

#include "Encoder.h"

using namespace std;

bool exit\_program = false;

void Exit();

int main() {

int input;

string file\_name = "";

string text\_to\_encode = "";

string decoded\_text = "";

bool ret = false; //Boolen to keep track of the state of the function

string tmp;

do {

cout << endl; //Main menu

cout << "Image Steganography" << endl;

cout << endl;

cout << "1. Encode" << endl;

cout << "2. Decode" << endl;

cout << "3. Exit" << endl;

cout << endl;

cout << "Input [1-3] : ";

cin >> input; //User input

cout << endl;

if (input != 1 && input != 2 && input != 3) {

cout << "Input is invalid. Please try again."; //If the user inputs an invalid value an error message is printed

}

if (input < 1 || input > 3) {

continue;

}

switch (input)

{

case 1:

cout << "Enter the secret message to encode : ";

getline(cin, tmp);

getline(cin, text\_to\_encode);

cout << "Enter image file name: ";

getline(cin, file\_name);

ret = Encode(text\_to\_encode, file\_name); //Call the Encoding funciton in the header file

if (ret) {

cout << "Encoding successful" << endl;

}

else {

cout << "Encoding failed" << endl;

}

break;

case 2:

cout << "Enter image file name to retrieve hidden text: ";

cin >> file\_name;

ret = Decode(file\_name, decoded\_text); //Call the Decode function in the header file

if (ret) {

cout << "Decoding successful" << endl;

cout << "Decoded message: \"" << decoded\_text << "\"" << endl; //Print the decoded message in quotes

}

else {

cout << "Decoding failed" << endl;

}

break;

case 3:

Exit(); //Exits the program

//the memory was allocated in the header files and deleted after use

}

} while (!exit\_program);//Run the program until "exit\_program" boolen is true.

return 0;

}

void Exit() { //Function to be used as a condition for looping the main body.

cout << "Program exited successfully" << endl;

exit\_program = true;

}

Encoder.cpp

#pragma once

#include "BitmapHelper.h"

#define MAX\_FILE\_NAME\_SIZE 1024

bool Encode(string s, string file\_name) {

unsigned char\*\*\* imageData; //Image data pointer

int imageWidth = 0; //Image width

int imageHeight = 0; //Image height

bool b1 = ReadBitmapImage(file\_name.c\_str(), imageData, imageWidth, imageHeight); //Read the image data via the given library

if (!b1) { //Checks if the image read operation was a success.

return false;

}

int n = s.size(); //Size of the secret text

int val = 0; //Variable to store numerical ASCII value of the characters of the secret string

int bit\_val = 0; //Variable to store bit values of the selected ASCII character

int counter = 0; //A counter that keeps track of the count of imageData bytes updated

int row = 0; //Current row

int col = 0; //Current column

int channel = 0; //Current channel

for (int i = 0; i < n + 1; i++) { //Loop to go through all the characters of the string

char c = s[i]; //Set current characeter in the string to char 'c'

val = (int)c; //Get ASCII value of 'c'

for (int j = 0; j < 8; j++) { //Loop for 8 times to extract the 8 bits representing the character.

bit\_val = val % 2; //Get LSB of val.

val = val / 2;

channel = counter % 3; //Get the current channel.

col = (counter / 3) % imageWidth; //Get the current column.

row = (counter / (3 \* imageWidth)) % imageHeight; //Get the current row.

imageData[row][col][channel] = ((imageData[row][col][channel] >> 1) << 1) + bit\_val; //Calculate the value that needs to be written.

counter++; //Increase counter to move to next byte in the image

}

}

int file\_name\_size = file\_name.size();

char encoded\_file\_name[MAX\_FILE\_NAME\_SIZE]; //File name for the encoded image

memcpy(encoded\_file\_name, &file\_name[0], file\_name\_size); //Copy the already given file name

if (file\_name\_size > 4) { //This is to remove the ".bmp" part of the file name given.

encoded\_file\_name[file\_name\_size - 1] = '\0'; // if have the filename string : "Earth.bmp"

encoded\_file\_name[file\_name\_size - 2] = '\0'; // after removing the .bmp part. Now we have : "Earth"

encoded\_file\_name[file\_name\_size - 3] = '\0'; // so later we can add "\_encoding.bmp" part, so the final name would be "Earth\_encoding.bmp"

encoded\_file\_name[file\_name\_size - 4] = '\0';

}

string end\_file\_name(encoded\_file\_name);

end\_file\_name += "\_encoded.bmp";

bool b2 = WriteBitmapImage(end\_file\_name.c\_str(), imageData, imageWidth, imageHeight); //Write the new .bmp file

ReleaseMemory(imageData, imageHeight, imageWidth);//Release the memory

if (!b2) { //If .bmp file write was unsuccessful, return false

return false;

}

cout << "Encoded image file name : " << end\_file\_name << endl;

return true;

}

Decoder.cpp

#pragma once

#include "BitmapHelper.h"

using namespace std;

bool Decode(string file\_name, string& ss) {

unsigned char\*\*\* imageData; //Image data holding pointer

int imageWidth = 0;

int imageHeight = 0;

bool b1 = ReadBitmapImage(file\_name.c\_str(), imageData, imageWidth, imageHeight); //Read the .bmp file

if (!b1) { //If read .bmp fails return false

return false;

}

int row = 0;//current row

int col = 0;//current col

int channel = 0;//current channel

char c;

int collected\_bit\_count = 0; //counter to verify that all 8 bits were collected

int curr\_bit = 0;//current bit collected

int val = 0;//variable to store the number represented by the 8bits collected

int counter = 0;//counter to navigate the image.

ss = "";

int string\_index = 0;

while (true) {

channel = counter % 3;//calculate the current channel

col = (counter / 3) % imageWidth; //calculate the current column

row = (counter / (3 \* imageWidth)) % imageHeight;//calculate the current row

curr\_bit = imageData[row][col][channel] % 2; //get the LSB of current image byte

val += (curr\_bit << collected\_bit\_count); //get the value of the ASCII by shifting the current bit to the left each iteration of the loop

collected\_bit\_count++; //collected\_bit\_count will increase by one each loop.

if (collected\_bit\_count == 8) { //Once 8 bit is collected, we found one character.

string temp(1, (char)val);

ss += temp;

if (0 == val) {

break; //If the character is a NULL character, end the loop.

}

val = 0;

collected\_bit\_count = 0;

}

counter++;

}

ReleaseMemory(imageData, imageHeight, imageWidth); //Release memory

return true;

}

BitmapHelper.h ReleaseMemory Function

void ReleaseMemory(unsigned char\*\*\* imageData,

int imageHeight, int imageWidth) {

for (int row = 0; row < imageHeight; row++)

{

for (int col = 0; col < imageWidth; col++)

{

delete[] imageData[row][col];

}

delete[] imageData[row];

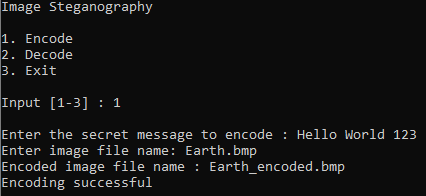
}

delete[] imageData;

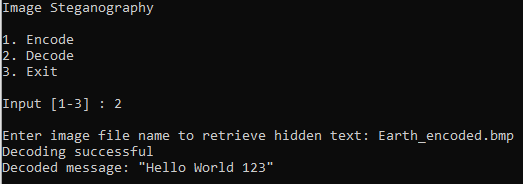
}

Test and Verification:

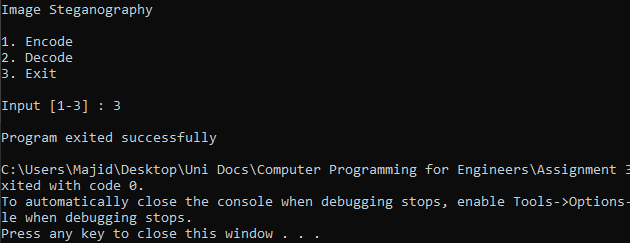
Case 1:



Case 2:



Case 3:



Case 4:

