SPL-1 Project Report, 2022

Invisible Ink & Animated Frame Encoder

SE 305: Software Project Lab

Submitted by

Shifat Jahan Shifa

BSSE Roll No.: 1301

BSSE Session: 2020-2021

Supervised by

Dr. B M Mainul Hossain

Professor

Institute of Information Technology

Institute of Information Technology

University of Dhaka

21-05-2023

Project:	Invisible Ink & Animated Frame Encoder
Author:	Shifat Jahan Shifa
Submission Date:	21 May, 2023
Supervised By:	Dr. B M Mainul Hossain
	Professor
	Institute of Information Technology
	University of Dhaka

Supervisor's Signature: ____

Acknowledgement

I would like to express my heartfelt and sincere gratitude to my Software Project Lab -01 supervisor, Dr. B M Mainul Hossain Sir, Associate Professor, Institute of Information Technology, University of Dhaka, for giving me the opportunity to do this project and his invaluable guidance throughout the entire course. His dynamism, vision, motivation have deeply inspired me towards my project. It was a great privilege and honor for me to work under his supervision.

Table of Contents

1. Introduction

"**Invisible Ink & Animated Frame Encoder**" is divided into two parts. First part "invisible ink" is identical to **Steganography**. Steganography is the technique used for concealing information into some digital media e.g., image, audio, video, text, network protocols. In this project "**image**" has been used as covered media and "**text**" has been used as hidden information. For embedding the text file into image, "**LSB**" technique has been used here.

The second part "Animated Frame Encoder" is identical to "**GIF Encoder**". GIF files are made of 8-bit BMP images. It can be consisted of static image or frames of images enabling them to be animated. The animation is achieved by displaying a sequence of frames in a specific order and at specified time intervals. In this project an animated GIF will be generated. Choosing a set of pixel values, putting them into each frame and implementing motion between each frame, GIF file will be made. During the encoding, "**LZW**" algorithm is implemented for lossless compression in GIF file and advanced data structure "**Trie**" is used for that purpose.

1.1. Background Study

- <u>BMP Structure</u>:
  In my project, BMP image is used wherever image is needed. For concealing information, BMP image is used as cover media. Even extracting information is also done from BMP image. Even in GIF encoder, I had to apply the knowledge of BMP Image. So, to start the project, at first, I learned about BMP image structure in details.

- <u>Image Read and Write</u>:
  For embedding the text information and extracting out the hidden information, image read and write are vital parts. For hiding the information, at first, I had to read the original image and make a copy of it by writing it to a new file. While writing, I did the encoding part by manipulating Least Significant Bits of each pixel with the hidden message. Thus, Stego image has been created. When extracting the message, the Stego image is read, doing the decoding desired message has been extracted. So, after learning BMP structure, I learned how to read and write image.

- <u>LSB Technique:</u>

  The most commonly method used for steganography technique is Least Significant Bit (LSB) algorithm. LSB algorithm performs the embedding operation of message along with the image file where each pixel has a size of 3 bytes. In my project, I have used 24-bit BMP image. For hiding the message perfectly without leaving any sign for any intruder, the image size and message size played an important role. As in LSB technique, I can use at best 3 bits of each pixel, there is a relation between the size of image and information. The relation may vary based on implementation. For my project, I developed the formula:

  Suppose, height=image height, width=image width
  Text file size= x bits
  Marker (the length of the message) =32 bits
  As height and width of the image are of 4 byte or 32 bits int and RGB values (red, green, blue) eat up 8 bits for each color (in total 24 bits), I can make use of at most 3 Least Significant Bits of each color (red, green, blue) from a single pixel value.
  So, if**((height*width*3)>(x+32))** satisfies, then steganography using LSB technique can be done with the image and message. Otherwise, we can stay on the same technique continuing with other image or change the technique.

- <u>Marker:</u>

  While decoding the concealed message, it must be known of what is the actual size or location of the message in the image. Otherwise, message can't be retrieved correctly. For that purpose, I need to have a "delimiter" which can give me the information of the hidden message location in the image. Delimiter can be implemented in many ways. I have done it by embedding the actual size of the message into the image prior embedding the actual message. The size is of 32 bits integer number. So, when embedding, the first 32 Least Significant Bits are used to store the size of the message. After that message are encoded. When extracting the message, my program first extracts the first 32 Least Significant Bits from sequential pixels. Converting the bit streams into integer value, I get the size of the message which indicates how much bits further would be retrieved from the pixels to get the hidden information correctly and fully.

- <u>GIF File Structure:</u>

  GIF stands for "Graphics Interchange Format". For encoding a GIF file, firstly, I learned about GIF structure which is more complex compared to BMP structure. GIF is different from many other common bitmap formats in the sense that it is **stream-based**. There are currently only two versions of GIF: 87a (the original GIF format) and 89a (the new GIF format). Just as with version 87a, the 89a version also begins with a Header, a Logical Screen Descriptor, and an optional Global Color Table. Each image also contains a Local Image Descriptor, an optional Local Color Table, and a block of image data. The trailer in every GIF89a file contains the same value as 87a files. Version 89a added a new feature to the GIF format called "Control Extensions". The four Control Extensions introduced by GIF89a are the "Graphics Control Extension", "Plain Text Extension", "Comment Extension" and "Application Extension". In my project, I have created GIF file of version GIF89a.

➢ LZW Algorithm:

The image data stored in a GIF file is always compressed. In a GIF file, pixel value compression is applied to reduce the file size and optimize the storage or transmission of the image. GIF files use a form of lossless compression, meaning that the image can be reconstructed exactly as it was before compression. The compression algorithm used in GIF is called **LZW (Lempel-Ziv-Welch)**. Here's a brief explanation of how LZW compression works:

1. Dictionary Creation*:* The LZW algorithm begins by creating an initial dictionary of pixel values or color codes used in the image.
2. Scanning and Encoding: The image is then scanned pixel by pixel, and a sequence of pixel values is encoded into a series of codes. Initially, each pixel value is represented by a code with the same value.
3. Code Expansion: As the algorithm progresses, it encounters repetitive sequences of pixel values. Instead of encoding each pixel value separately, LZW replaces these repetitions with shorter codes, which refer to previously encountered sequences.
4. Dynamic Dictionary Update: LZW dynamically updates the dictionary as it encounters new sequences of pixel values. It adds new codes to the dictionary to represent the unique sequences. By replacing repetitive pixel sequences with shorter codes and using a dynamically expanding dictionary, LZW compression achieves compression ratios for GIF images.

When the GIF is displayed, the decoding process reverses the compression, reconstructing the original image with the help of the dictionary. Overall, this algorithm is capable of reducing the size of typical 8-bit pixel data by **40 percent or more**. So, I learned how to implement LZW algorithm to use in my project which creates GIF of version "89a".

- Trie:

  A Trie, also known as a **prefix tree**, is a tree-based data structure. Trie is used in my project for efficient storage and retrieval of dictionary words. Trie helps to reduce space consumption by allocating same memory space for repetitive pattern. It suits best to achieve the functionality of "LZW" algorithm in a comparatively fast time.

  Time complexity: O(n); for insertion

  Time complexity: O(n); for deletion

  Time complexity: O(n); for searching

1.2. Challenges

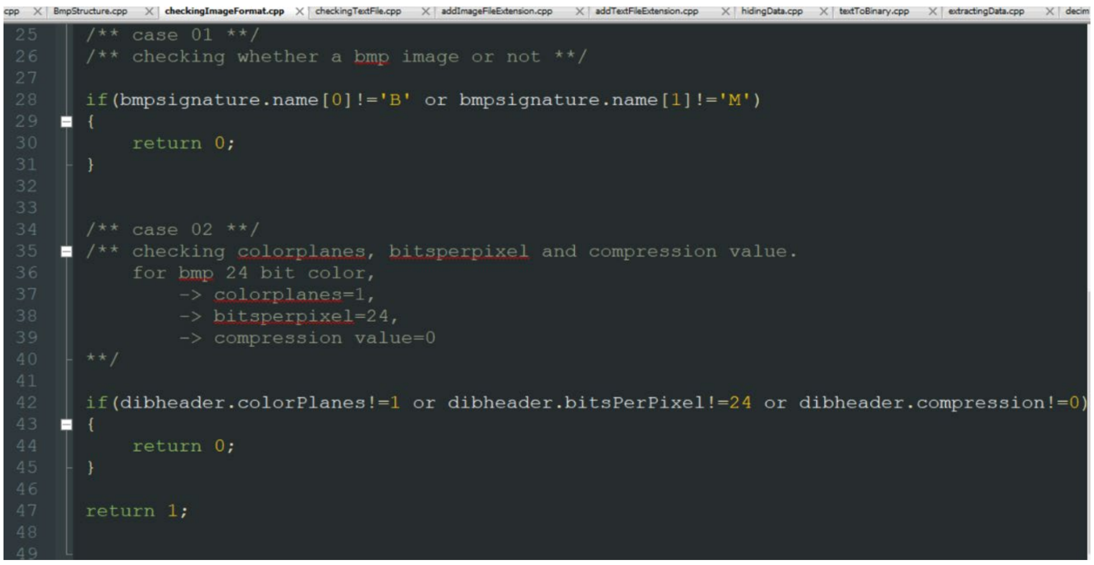
There are a lot of tasks in my project which were challenging and exciting for me. I found it complex to implement image related part in steganography. Working with GIF, was the most challenging part for me as I had to implement every header not knowing whether the parsing is right or not. Below are the parts of my project which I found as challenging for me:

- Perfectly copying an image.
- Converting the pixel value from hex format to int format by doing Bitwise operation.
- Generating correct Bit Streams for hidden message.
- Putting the marker properly.
- Embedding the message in right order without losing information.
- Generating "stego" image perfectly leaving no sign for any intruder.
- Maintaining the BMP format properly, opening the "stego" image with Windows Photos.
- Retrieving the actual message properly and in order as it was.
- Understanding the GIF file format as it more complex for being byte-streamed data.
- Implementing each header in GIF documentation specially;
  - Graphics Control Extension
  - Bounding Box
  - Creating Frame
  - Putting image
- Implementing Trie for LZW algorithm.
- Manipulating data with pointer.
- Without corrupting, opening the GIF file with Windows Photos.

2. Project Overview

➢ <u>Checking Image File and Text File:</u>

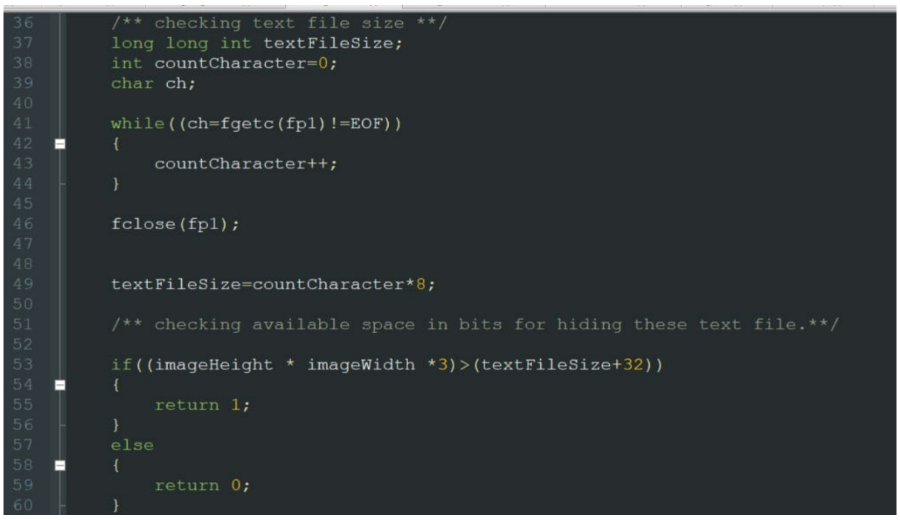
At very first, I have checked the format of image file by inspecting the "**Magic Number**" of BMP image ('**BM**') along with color Planes, bits Per Pixel, compression value.



```
/** case 01 **/
/** checking whether a bmp image or not **/

if(bmpsignature.name[0]!='B' or bmpsignature.name[1]!='M')
{
    return 0;
}


/** case 02 **/
/** checking colorplanes, bitsperpixel and compression value.
    for bmp 24 bit color,
        -> colorplanes=1,
        -> bitsperpixel=24,
        -> compression value=0
**/

if(dibheader.colorPlanes!=1 or dibheader.bitsPerPixel!=24 or dibheader.compression!=0)
{
    return 0;
}

return 1;
```

Fig- 01: (checking image file and text file)

Then I checked the message whether it can be hidden or not. For that I used the formula **((height*width*3)>(x+32)).** If this satisfies, I proceed further.

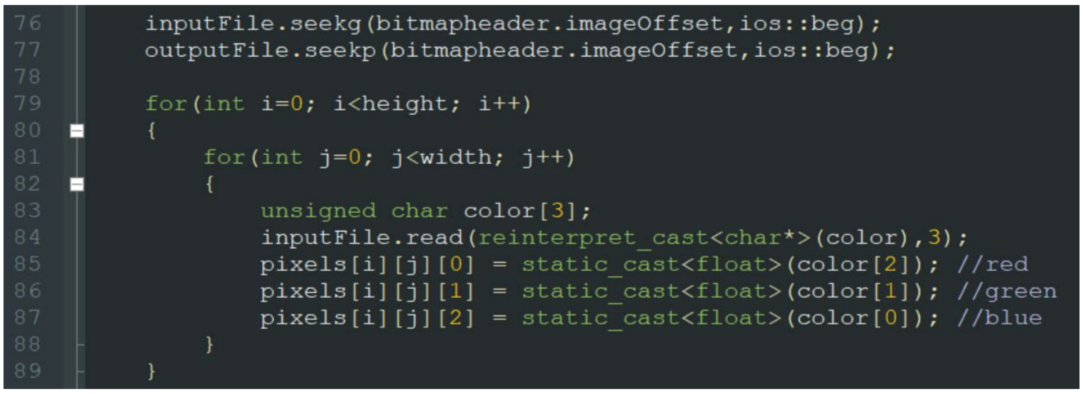


```
    /** checking text file size **/
    long long int textFileSize;
    int countCharacter=0;
    char ch;

    while((ch=fgetc(fp1))!=EOF))
    {
        countCharacter++;
    }

    fclose(fp1);


    textFileSize=countCharacter*8;

    /** checking available space in bits for hiding these text file.**/

    if((imageHeight * imageWidth *3)>(textFileSize+32))
    {
        return 1;
    }
    else
    {
        return 0;
    }
```

Fig- 02: (checking text file size)

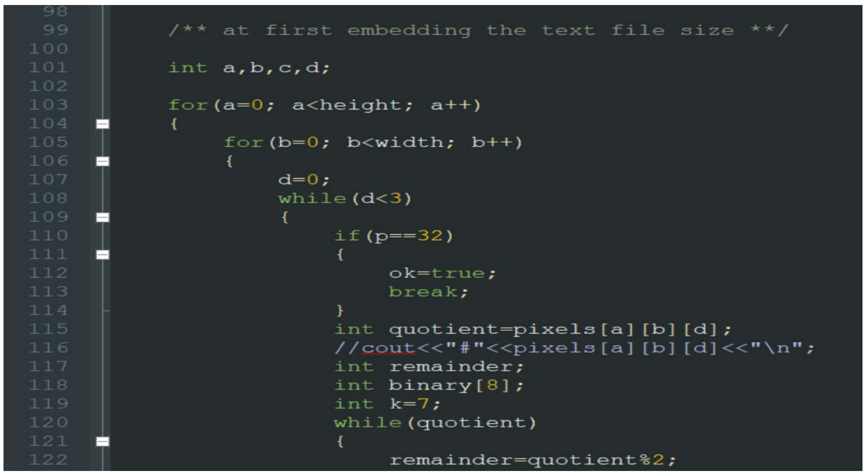
➢ <u>Embedding Information:</u>

When the image file and text file are okay to be proceeded with further, the next step is hiding the information. For that, a copy of the original image is made with the message which at the end will be treated as stego image. While copying the image, RGB values of each pixel is extracted from the original image, manipulated the Least Significant Bit of each color with the message size first (Marker Process), then with message. Finally, I wrote the manipulated pixel values with hidden message inside into the copying image. Here is the code snippet of my works.

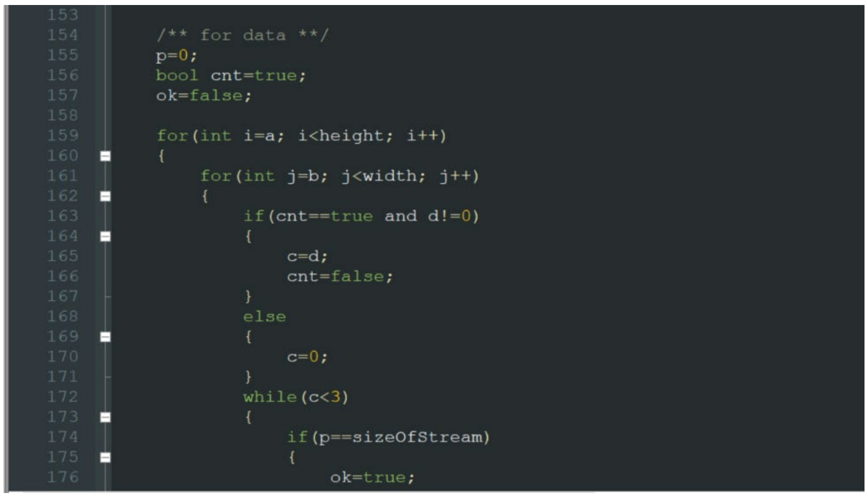


```
    inputFile.seekg(bitmapheader.imageOffset,ios::beg);
    outputFile.seekp(bitmapheader.imageOffset,ios::beg);

    for(int i=0; i<height; i++)
    {
        for(int j=0; j<width; j++)
        {
            unsigned char color[3];
            inputFile.read(reinterpret_cast<char*>(color),3);
            pixels[i][j][0] = static_cast<float>(color[2]); //red
            pixels[i][j][1] = static_cast<float>(color[1]); //green
            pixels[i][j][2] = static_cast<float>(color[0]); //blue
        }
    }
```

Fig- 03:(extracting pixel value)



```
    /** at first embedding the text file size **/

    int a,b,c,d;

    for(a=0; a<height; a++)
    {
        for(b=0; b<width; b++)
        {
            d=0;
            while(d<3)
            {
                if(p==32)
                {
                    ok=true;
                    break;
                }
                int quotient=pixels[a][b][d];
                //cout<<"#"<<pixels[a][b][d]<<"\n";
                int remainder;
                int binary[8];
                int k=7;
                while(quotient)
                {
                    remainder=quotient%2;
```



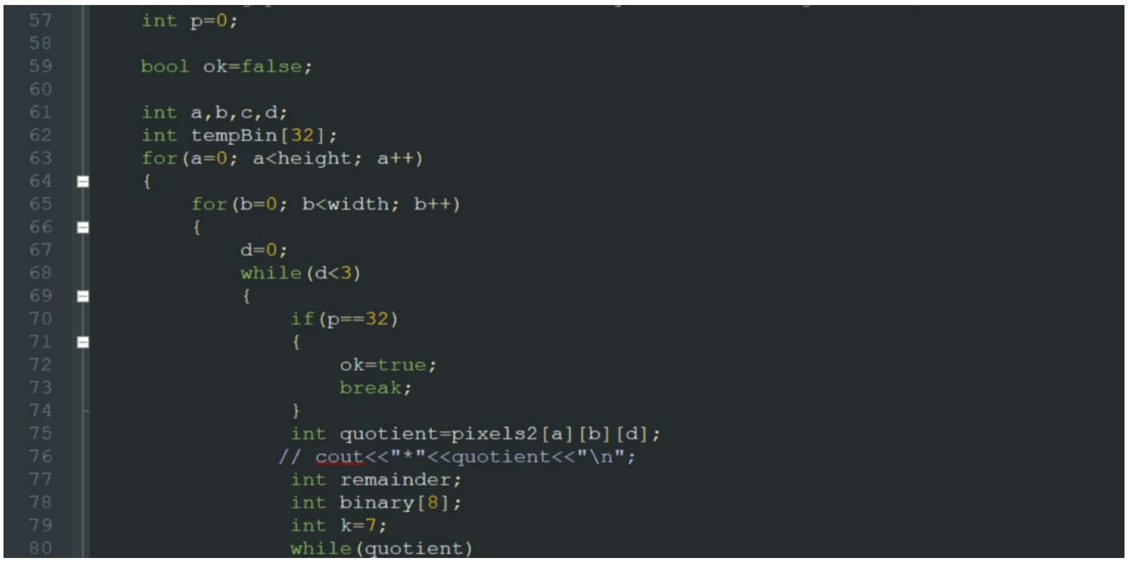
```
    /** for data **/
    p=0;
    bool cnt=true;
    ok=false;

    for(int i=a; i<height; i++)
    {
        for(int j=b; j<width; j++)
        {
            if(cnt==true and d!=0)
            {
                c=d;
                cnt=false;
            }
            else
            {
                c=0;
            }
            while(c<3)
            {
                if(p==sizeOfStream)
                {
                    ok=true;
```

Fig- 04: (embedding message size and message)

➢ <u>Extracting Information:</u>

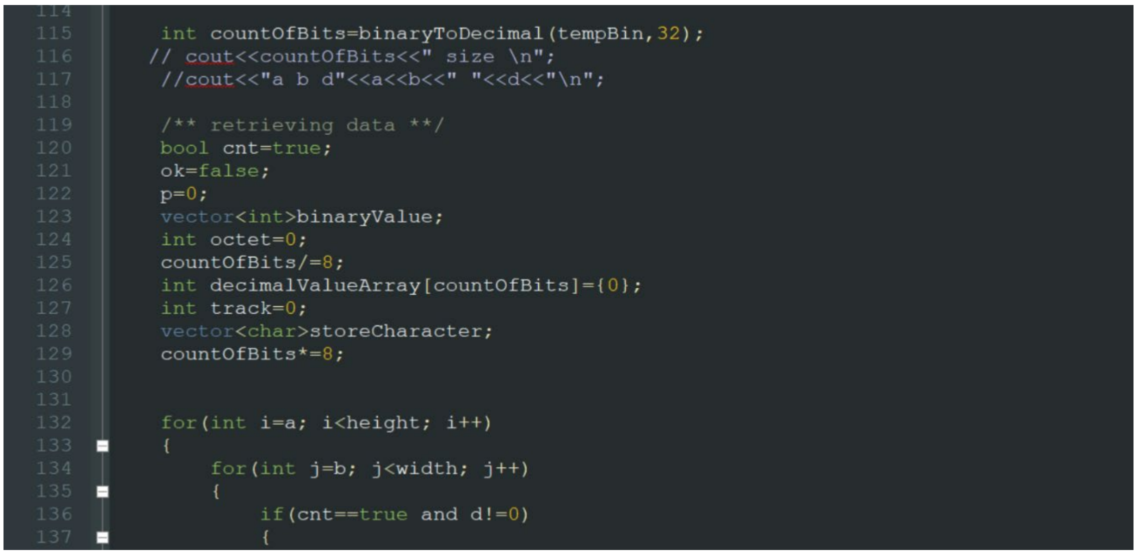
For extracting the hidden data from the stego image, image is read from the beginning. The message size is retrieved by reading LSBs of pixels values from the first. After reading first 32 LSBs, converting it to integer type, message size value is gained. After that, following the size value, original message is extracted from the subsequent LSBs.

Here are some code snippets of my work:



```
int p=0;

bool ok=false;

int a,b,c,d;
int tempBin[32];
for(a=0; a<height; a++)
{
    for(b=0; b<width; b++)
    {
        d=0;
        while(d<3)
        {
            if(p==32)
            {
                ok=true;
                break;
            }
            int quotient=pixels2[a][b][d];
           // cout<<"*"<<quotient<<"\n";
            int remainder;
            int binary[8];
            int k=7;
            while(quotient)
```

Fig- 05: (reading the size of message)

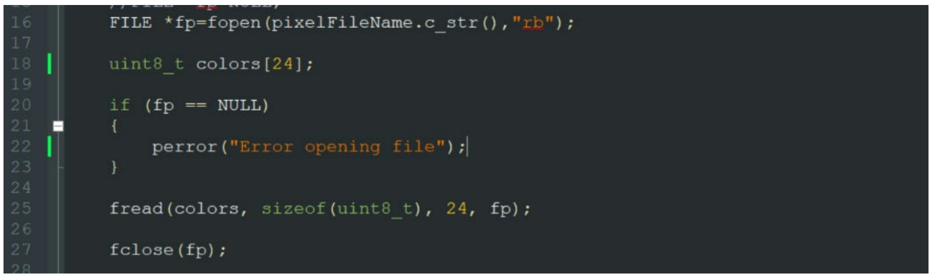


```
int countOfBits=binaryToDecimal(tempBin,32);
// cout<<countOfBits<<" size \n";
//cout<<"a b d"<<a<<b<<" "<<d<<"\n";

/** retrieving data **/
bool cnt=true;
ok=false;
p=0;
vector<int>binaryValue;
int octet=0;
countOfBits/=8;
int decimalValueArray[countOfBits]={0};
int track=0;
vector<char>storeCharacter;
countOfBits*=8;


for(int i=a; i<height; i++)
{
    for(int j=b; j<width; j++)
    {
        if(cnt==true and d!=0)
        {
```

Fig- 06: (retrieving the data)

➢ <u>Reading From a Pixel File:</u>

For GIF encoder, there are some input pixels in binary file format. My program first read one of the pixels sets and store them in a colors array.

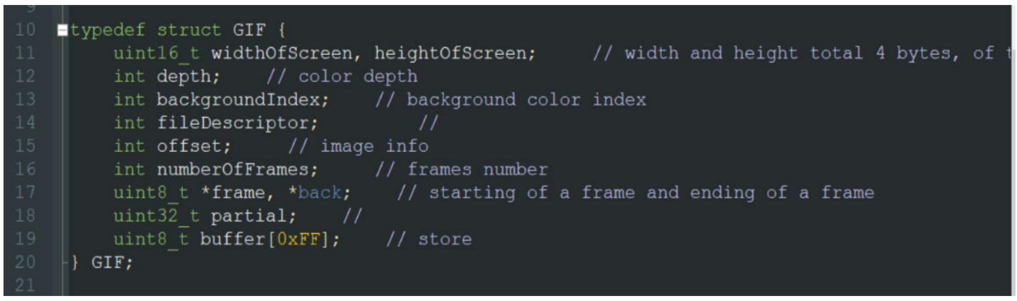


```
FILE *fp=fopen(pixelFileName.c_str(),"rb");

uint8_t colors[24];

if (fp == NULL)
{
    perror("Error opening file");
}

fread(colors, sizeof(uint8_t), 24, fp);

fclose(fp);
```

Fig-07:(reading from pixel file)

➢ <u>Creating GIF File:</u>

For defining the GIF, in a module I put the GIF file information.



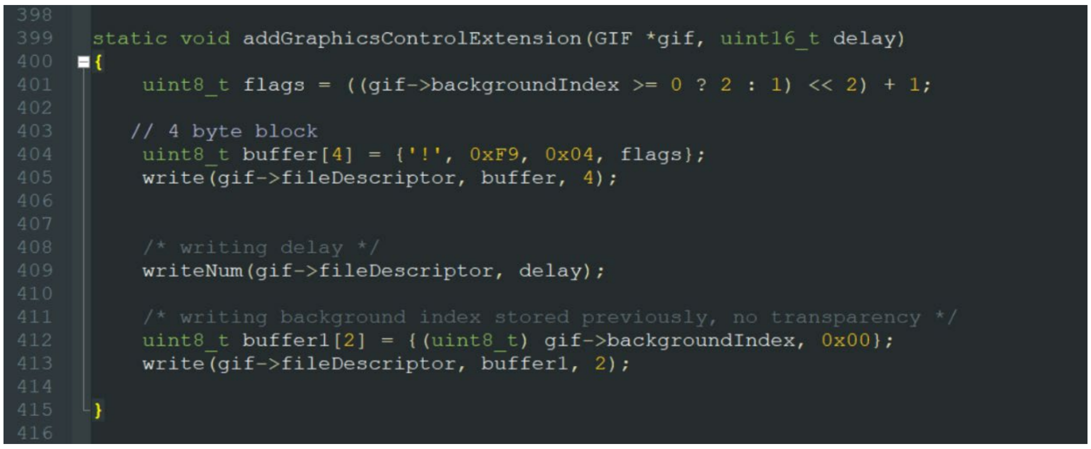
```c
typedef struct GIF {
    uint16_t widthOfScreen, heightOfScreen;     // width and height total 4 bytes, of t
    int depth;    // color depth
    int backgroundIndex;    // background color index
    int fileDescriptor;         //
    int offset;     // image info
    int numberOfFrames;     // frames number
    uint8_t *frame, *back;    // starting of a frame and ending of a frame
    uint32_t partial;    //
    uint8_t buffer[0xFF];    // store
} GIF;
```

Fig-08:(GIF structure)

For creating a GIF, there are several functions for several task. Some tasks are:

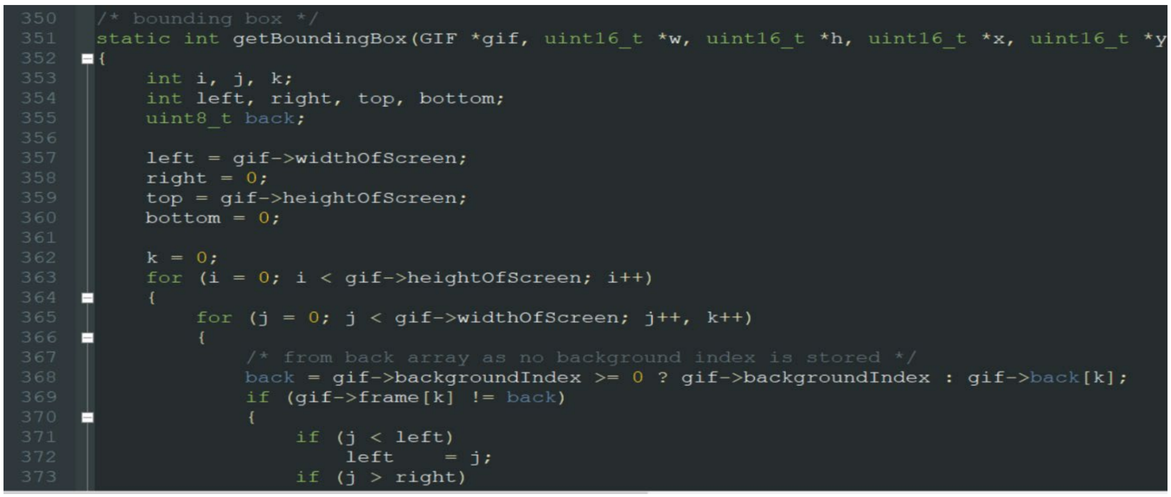
- Adding Frames.
- Insertion and deletion in Trie.
- Putting the image into frame.
- Getting bounding box for output canvas.
- Adding graphics control extension.
- Putting keys
- Ending keys

Here are some code snippets:



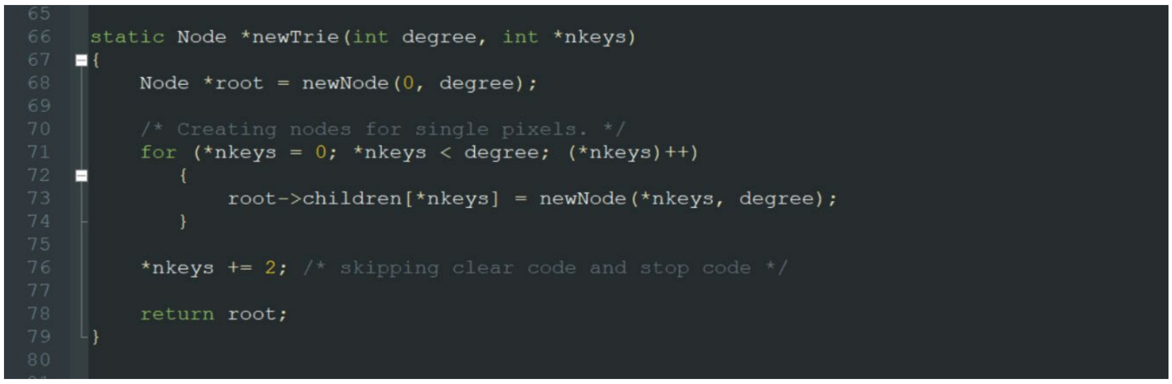
```c
static void addGraphicsControlExtension(GIF *gif, uint16_t delay)
{
    uint8_t flags = ((gif->backgroundIndex >= 0 ? 2 : 1) << 2) + 1;

   // 4 byte block
    uint8_t buffer[4] = {'!', 0xF9, 0x04, flags};
    write(gif->fileDescriptor, buffer, 4);


    /* writing delay */
    writeNum(gif->fileDescriptor, delay);

    /* writing background index stored previously, no transparency */
    uint8_t buffer1[2] = {(uint8_t) gif->backgroundIndex, 0x00};
    write(gif->fileDescriptor, buffer1, 2);

}
```

Fig-09: (adding graphics control extension)



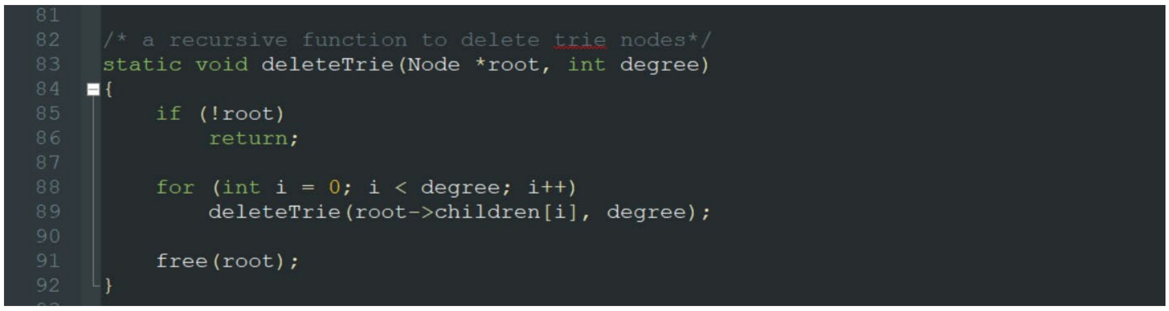
```c
/* bounding box */
static int getBoundingBox(GIF *gif, uint16_t *w, uint16_t *h, uint16_t *x, uint16_t *y
{
    int i, j, k;
    int left, right, top, bottom;
    uint8_t back;

    left = gif->widthOfScreen;
    right = 0;
    top = gif->heightOfScreen;
    bottom = 0;

    k = 0;
    for (i = 0; i < gif->heightOfScreen; i++)
    {
        for (j = 0; j < gif->widthOfScreen; j++, k++)
        {
            /* from back array as no background index is stored */
            back = gif->backgroundIndex >= 0 ? gif->backgroundIndex : gif->back[k];
            if (gif->frame[k] != back)
            {
                if (j < left)
                    left    = j;
                if (j > right)
```

Fig-10:(getting bounding box)



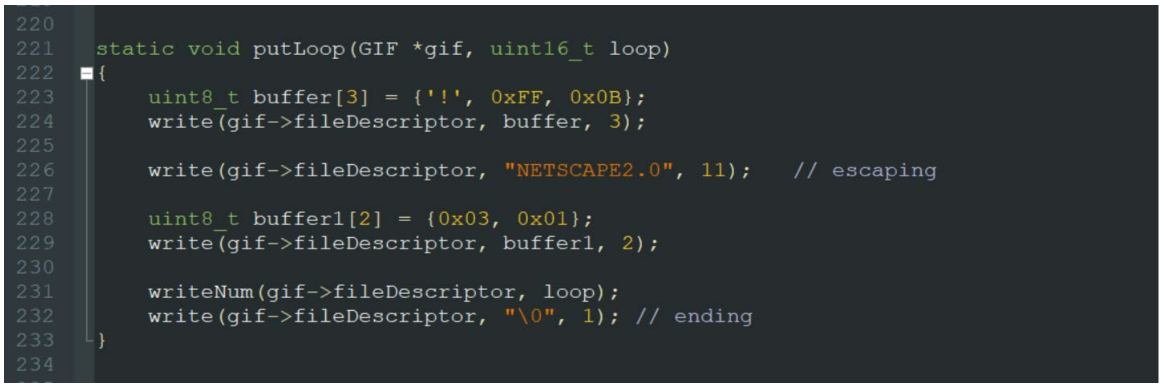
```c
static Node *newTrie(int degree, int *nkeys)
{
    Node *root = newNode(0, degree);

    /* Creating nodes for single pixels. */
    for (*nkeys = 0; *nkeys < degree; (*nkeys)++)
      {
          root->children[*nkeys] = newNode(*nkeys, degree);
      }

    *nkeys += 2; /* skipping clear code and stop code */

    return root;
}
```

Fig-11:(insertion in Trie)



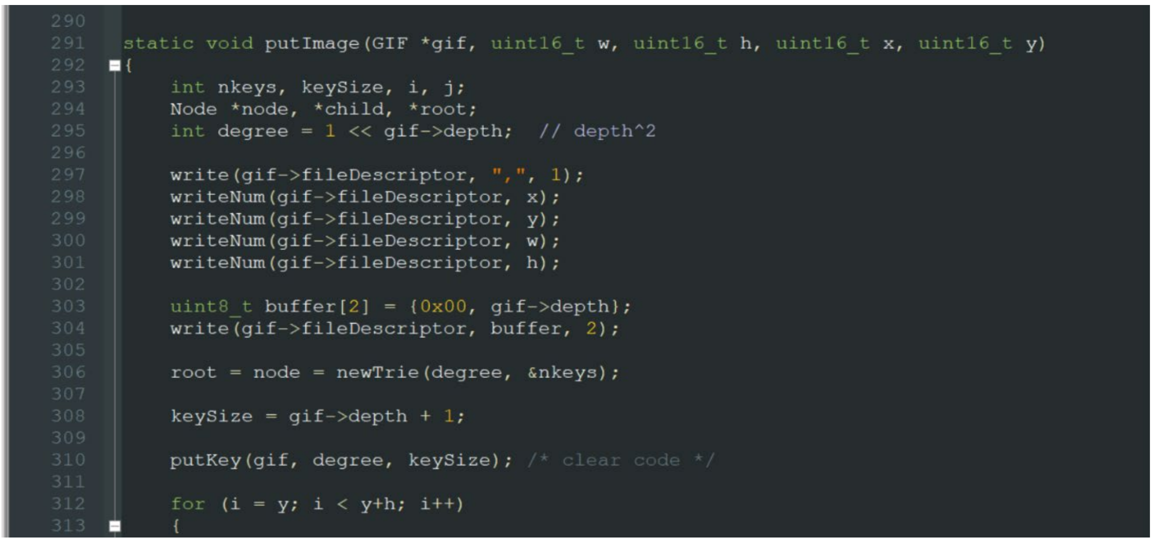
```c
/* a recursive function to delete trie nodes*/
static void deleteTrie(Node *root, int degree)
{
    if (!root)
        return;

    for (int i = 0; i < degree; i++)
        deleteTrie(root->children[i], degree);

    free(root);
}
```

Fig-12:(deletion in Trie)



```c
static void putLoop(GIF *gif, uint16_t loop)
{
    uint8_t buffer[3] = {'!', 0xFF, 0x0B};
    write(gif->fileDescriptor, buffer, 3);

    write(gif->fileDescriptor, "NETSCAPE2.0", 11);   // escaping

    uint8_t buffer1[2] = {0x03, 0x01};
    write(gif->fileDescriptor, buffer1, 2);

    writeNum(gif->fileDescriptor, loop);
    write(gif->fileDescriptor, "\0", 1); // ending
}
```

Fig-13:(put Loop function)



```c
static void putImage(GIF *gif, uint16_t w, uint16_t h, uint16_t x, uint16_t y)
{
    int nkeys, keySize, i, j;
    Node *node, *child, *root;
    int degree = 1 << gif->depth;  // depth^2

    write(gif->fileDescriptor, ",", 1);
    writeNum(gif->fileDescriptor, x);
    writeNum(gif->fileDescriptor, y);
    writeNum(gif->fileDescriptor, w);
    writeNum(gif->fileDescriptor, h);

    uint8_t buffer[2] = {0x00, gif->depth};
    write(gif->fileDescriptor, buffer, 2);

    root = node = newTrie(degree, &nkeys);

    keySize = gif->depth + 1;

    putKey(gif, degree, keySize); /* clear code */

    for (i = y; i < y+h; i++)
    {
```

Fig-14:(put image function)

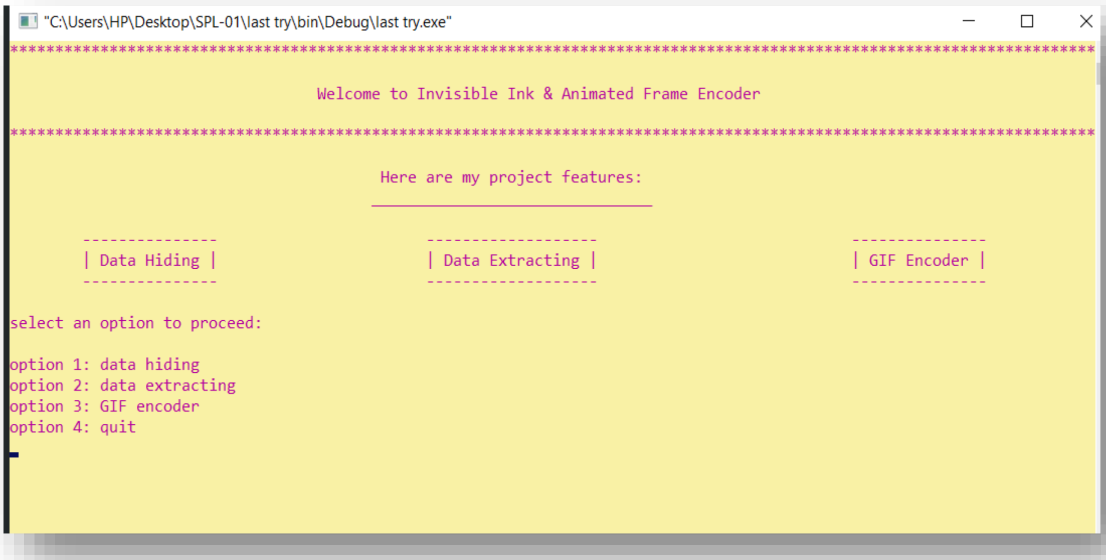
3. User Manual

- Installation:

  This project can be downloaded from my GitHub link and opened with Code Blocks or any other IDE (vscode) with 32-bit compiler.

- Usage Procedure:

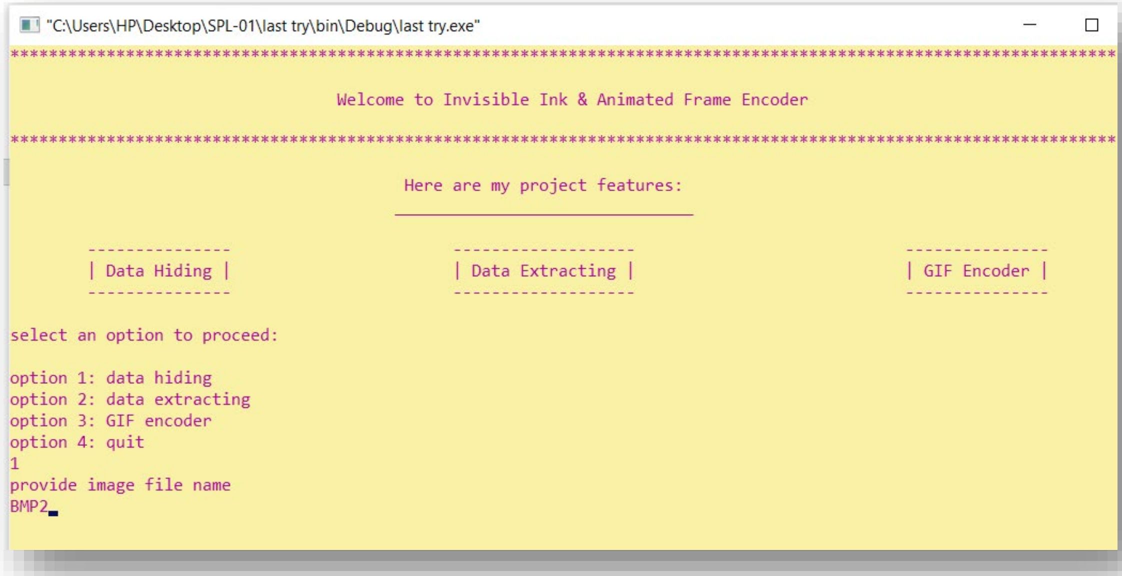
  ❖ Running the project, there will be a console like this.



```
"C:\Users\HP\Desktop\SPL-01\last try\bin\Debug\last try.exe"
****************************************************************************************************

                         Welcome to Invisible Ink & Animated Frame Encoder

****************************************************************************************************

                                    Here are my project features:
                                  _______________________________

      ---------------              -------------------                     ---------------
      | Data Hiding |              | Data Extracting |                     | GIF Encoder |
      ---------------              -------------------                     ---------------

select an option to proceed:

option 1: data hiding
option 2: data extracting
option 3: GIF encoder
option 4: quit
```

Fig-15:(console)

  ❖ For data hiding, first needs to put image file name. In my device, there are 9 sample BMP images.



```
"C:\Users\HP\Desktop\SPL-01\last try\bin\Debug\last try.exe"
****************************************************************************************************

                         Welcome to Invisible Ink & Animated Frame Encoder

****************************************************************************************************

                                    Here are my project features:
                                  _______________________________

      ---------------              -------------------                     ---------------
      | Data Hiding |              | Data Extracting |                     | GIF Encoder |
      ---------------              -------------------                     ---------------

select an option to proceed:

option 1: data hiding
option 2: data extracting
option 3: GIF encoder
option 4: quit
1
provide image file name
BMP2
```

Fig-16: (option-01)

❖ After providing the image file name, image will be opened in Windows Photos app.

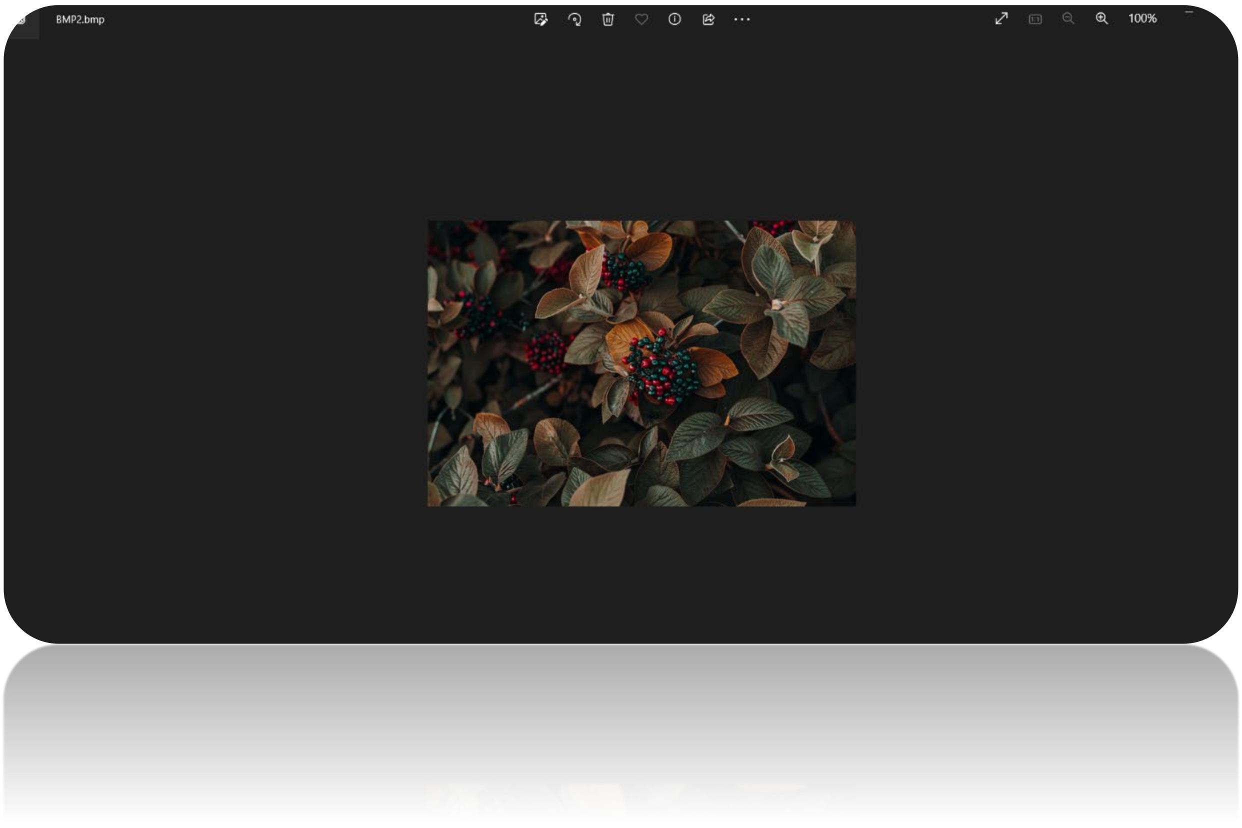
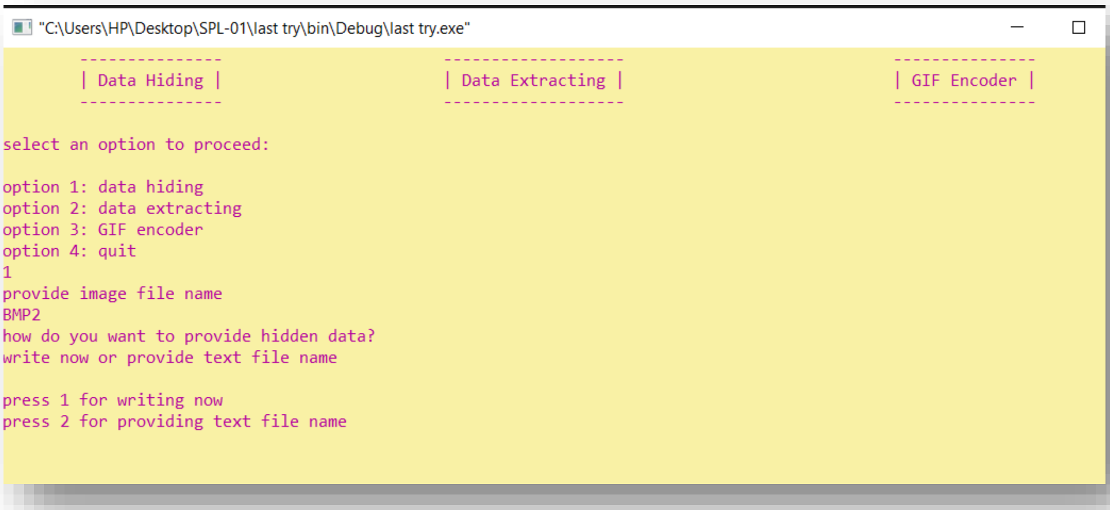


Fig-17:(opening the image)

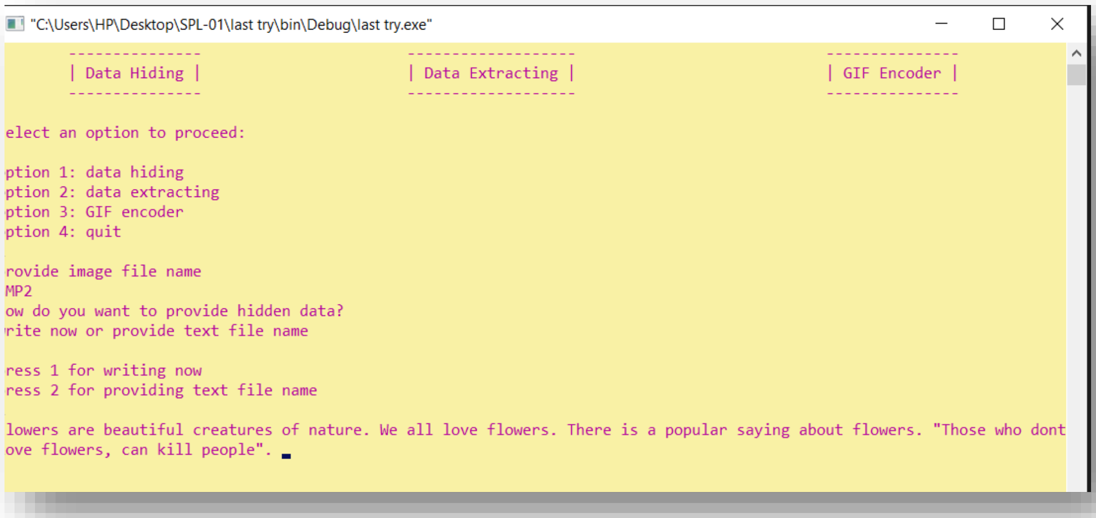
❖ For the hidden message, there are two options. User can input the file name where the message is written previously or can input the message instantly on console.



```
"C:\Users\HP\Desktop\SPL-01\last try\bin\Debug\last try.exe"

        --------------             ------------------              --------------
        | Data Hiding |            | Data Extracting |              | GIF Encoder |
        --------------             ------------------              --------------

select an option to proceed:

option 1: data hiding
option 2: data extracting
option 3: GIF encoder
option 4: quit
1
provide image file name
BMP2
how do you want to provide hidden data?
write now or provide text file name

press 1 for writing now
press 2 for providing text file name
```

Fig-18:(providing option)

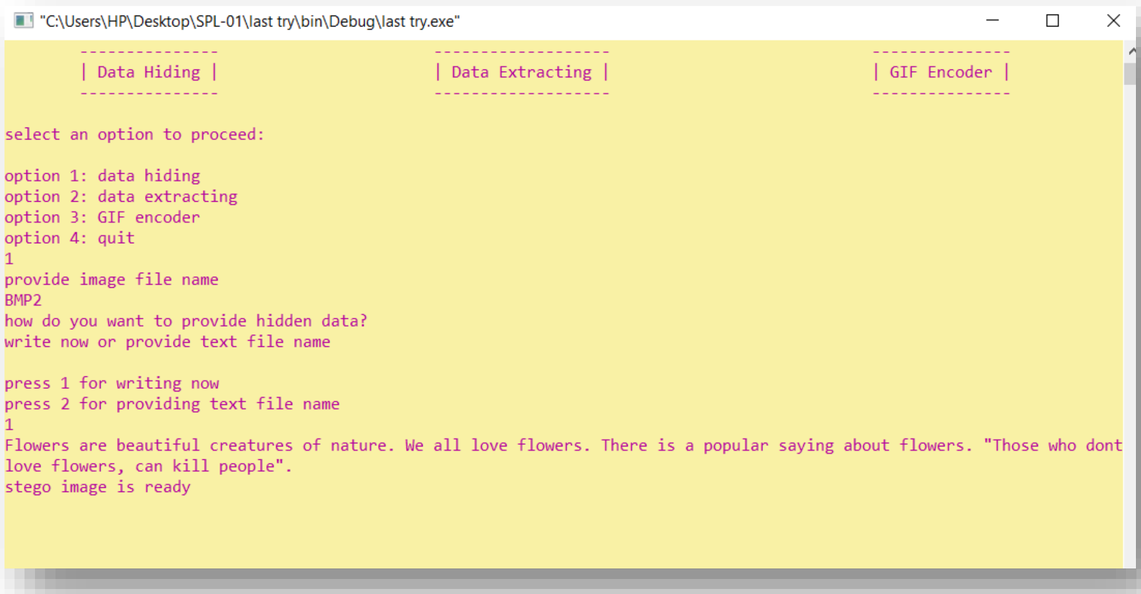
- Suppose, writing on console like this.



```
"C:\Users\HP\Desktop\SPL-01\last try\bin\Debug\last try.exe"

       ---------------                      -------------------                     ---------------
       | Data Hiding |                      | Data Extracting |                     | GIF Encoder |
       ---------------                      -------------------                     ---------------

elect an option to proceed:

ption 1: data hiding
ption 2: data extracting
ption 3: GIF encoder
ption 4: quit

rovide image file name
MP2
ow do you want to provide hidden data?
rite now or provide text file name

ress 1 for writing now
ress 2 for providing text file name

lowers are beautiful creatures of nature. We all love flowers. There is a popular saying about flowers. "Those who dont
ove flowers, can kill people".
```

Fig-19:(writing message)

- After pressing enter, the process of hiding will be done. When finished, there will be a message on console showing “stego image is ready” and instantly stego image will be opened with Windows Photos.



```
"C:\Users\HP\Desktop\SPL-01\last try\bin\Debug\last try.exe"

        ---------------                      -------------------                     ---------------
        | Data Hiding |                      | Data Extracting |                     | GIF Encoder |
        ---------------                      -------------------                     ---------------

select an option to proceed:

option 1: data hiding
option 2: data extracting
option 3: GIF encoder
option 4: quit
1
provide image file name
BMP2
how do you want to provide hidden data?
write now or provide text file name

press 1 for writing now
press 2 for providing text file name
1
Flowers are beautiful creatures of nature. We all love flowers. There is a popular saying about flowers. "Those who dont
love flowers, can kill people".
stego image is ready
```

Fig-20:(stego image is ready)

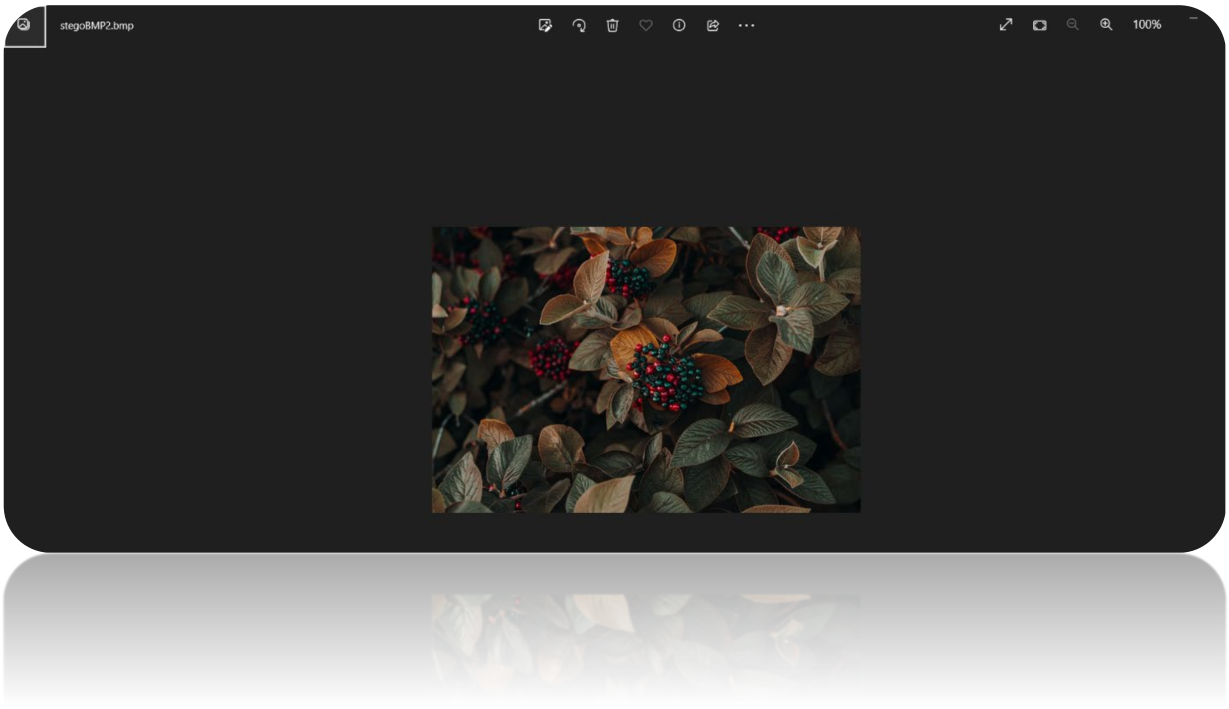
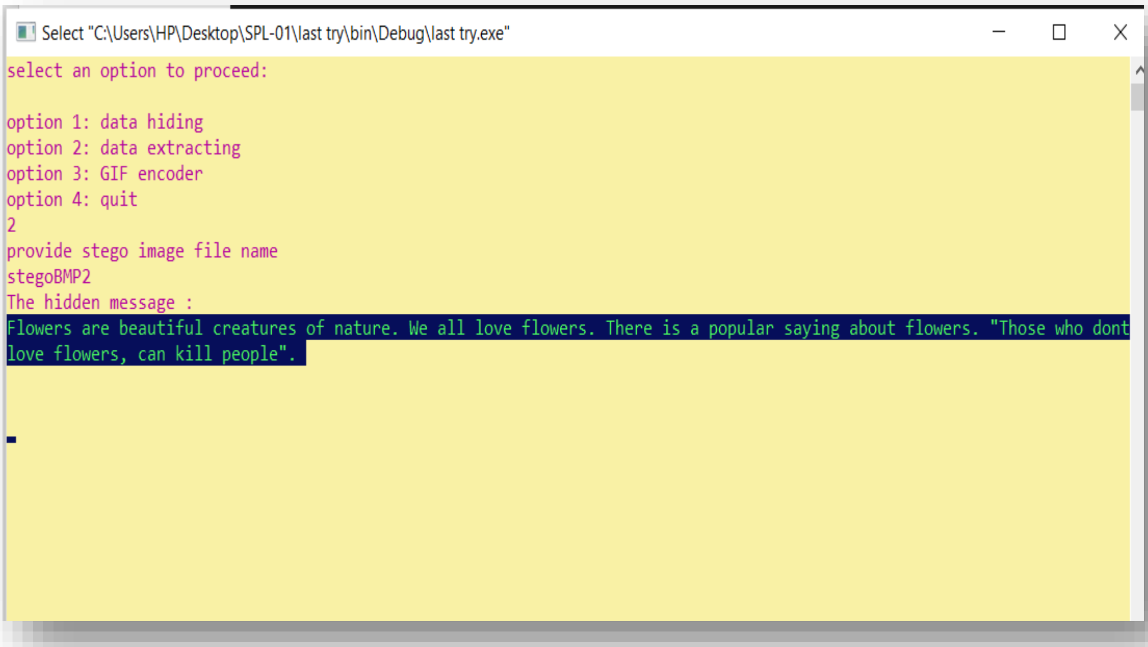


Fig-21:(stego image)

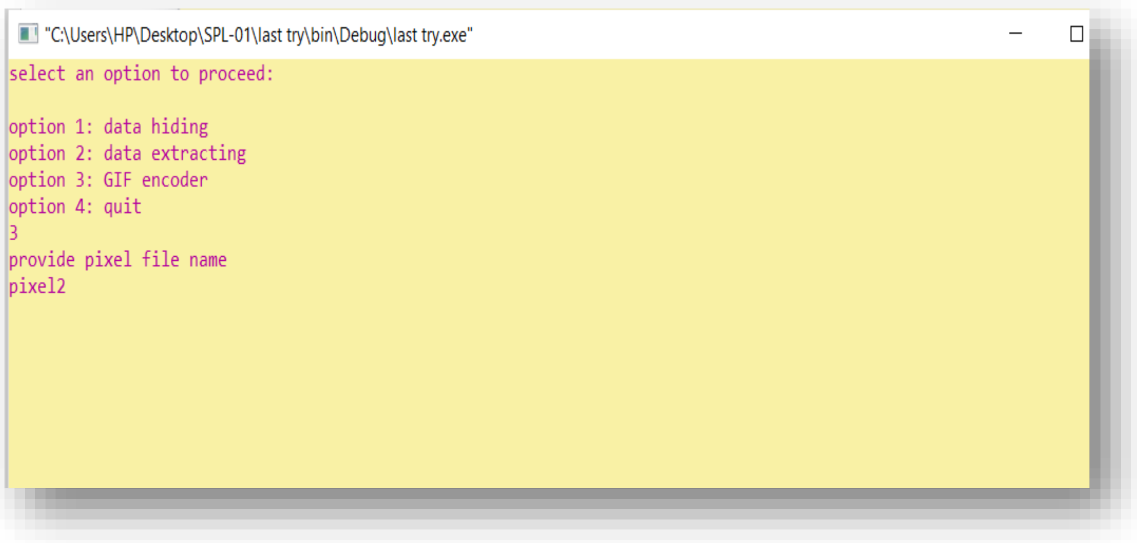
❖ If user wants to extract the hidden message from the stego image, they can choose option 2. Providing the stego image file name, user can get the output printing the message on console.


Select "C:\Users\HP\Desktop\SPL-01\last try\bin\Debug\last try.exe"

```
select an option to proceed:

option 1: data hiding
option 2: data extracting
option 3: GIF encoder
option 4: quit
2
provide stego image file name
stegoBMP2
The hidden message :
Flowers are beautiful creatures of nature. We all love flowers. There is a popular saying about flowers. "Those who dont
love flowers, can kill people".
```

Fig-22:(showing the hidden message)

- For making GIF, user can choose option 3. User needs to input pixel file name. in my device there are 3 pixels files combination 24 colors. Suppose, chose pixel2



"C:\Users\HP\Desktop\SPL-01\last try\bin\Debug\last try.exe"

```
select an option to proceed:

option 1: data hiding
option 2: data extracting
option 3: GIF encoder
option 4: quit
3
provide pixel file name
pixel2
```

Fig-23:(input of pixel file)

- After pressing enter, created GIF file will be opened with Windows Photos.

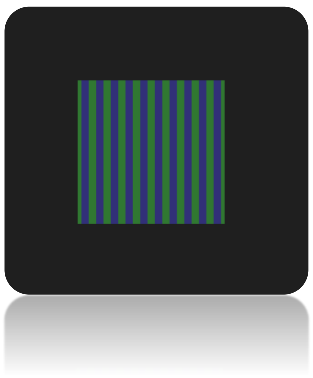


Fig-24:(output GIF)

4. Future Scope

As I have learned about steganography process using image, I can now expand my project doing steganography by other means such as audio, video, text, network protocols. For GIF encoder, I learned the structure of GIF in details, how to encode a GIF. Now, on internet, GIF media has become popular for many purposes with many features. So, my project part GIF encoder, can be expanded to add more exciting features to GIF to accommodate the concurrent need. Furthermore, I can achieve steganography using GIF media too.

5. Conclusion

Almost everything in my project was new to me. I tried to learn honestly everything and implement from scratch. I am happy that I have successfully completed my project with all of the tasks that I was committed to. It was delightful for me to see the outputs of my project as the way I thought of. I tried hard for doing everything as required. Completing the challenges, my programming and problem-solving skill has been upgraded that I'm now much confident than before. I am hopeful of getting benefits from my learned knowledge. In future I can expand my work by exploring in depth of my learned sectors. Overall, I enjoyed the whole journey with my SPL-01 project.

6. Appendix

My GitHub link: https://github.com/ShifatJahanShifa/SPL-01

References

1. **https://ieeexplore.ieee.org/document/9335027, ieeexplore.iee.org/document, 27 February, 2023**

2. **https://en.wikipedia.org/wiki/Steganography, Wikipedia.org/wiki/Steganography, 30 March, 2023**

3. **https://en.wikipedia.org/wiki/BMP_file_format, Wikipedia.org/wiki/BMP_file_format, 5 January, 2023**

4. **https://www.fileformat.info/format/gif/egff.htm, fileformat.info/format/gif/egff.htm, 15 March, 2023**

5. **https://en.wikipedia.org/wiki/Trie, Wikipedia.org/wiki/Trie, 23 March, 2023**

6. **https://en.wikipedia.org/wiki/GIF, Wikipedia.org/wiki/GIF, 31 March, 2023**

7. **https://www.scaler.com/topics/lzw-compression/, scaler.com/ topics/lzw-compression/, 2 April, 2023**

8. **https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch, Wikipedia.org/wiki/Lempel, 5 April, 2023**