Covert Imaging

Assignment 2 for 8505 By Ramzi Chennafi

About this Project

This is a program for the encoding of data into an image file. This was performed by using the LSB of each byte. The main focus of this design was 32 bit bitmaps, and this means that each pixel stores 4 bits of file data.

The encoded data is stored with its own variable size header. This header contains a filename and file size. Each is ended with a null terminator, and using the size the full file data is retrieved.

Requirements

-Linux

-gcc compiler

Using the program

TO Encode a data file into an image

```
covertdata encode [image/to/encode/into] [file/to/encode]
```

TO Decode a data file from an image

```
covertdata decode [image/to/decode] [output/name]
```

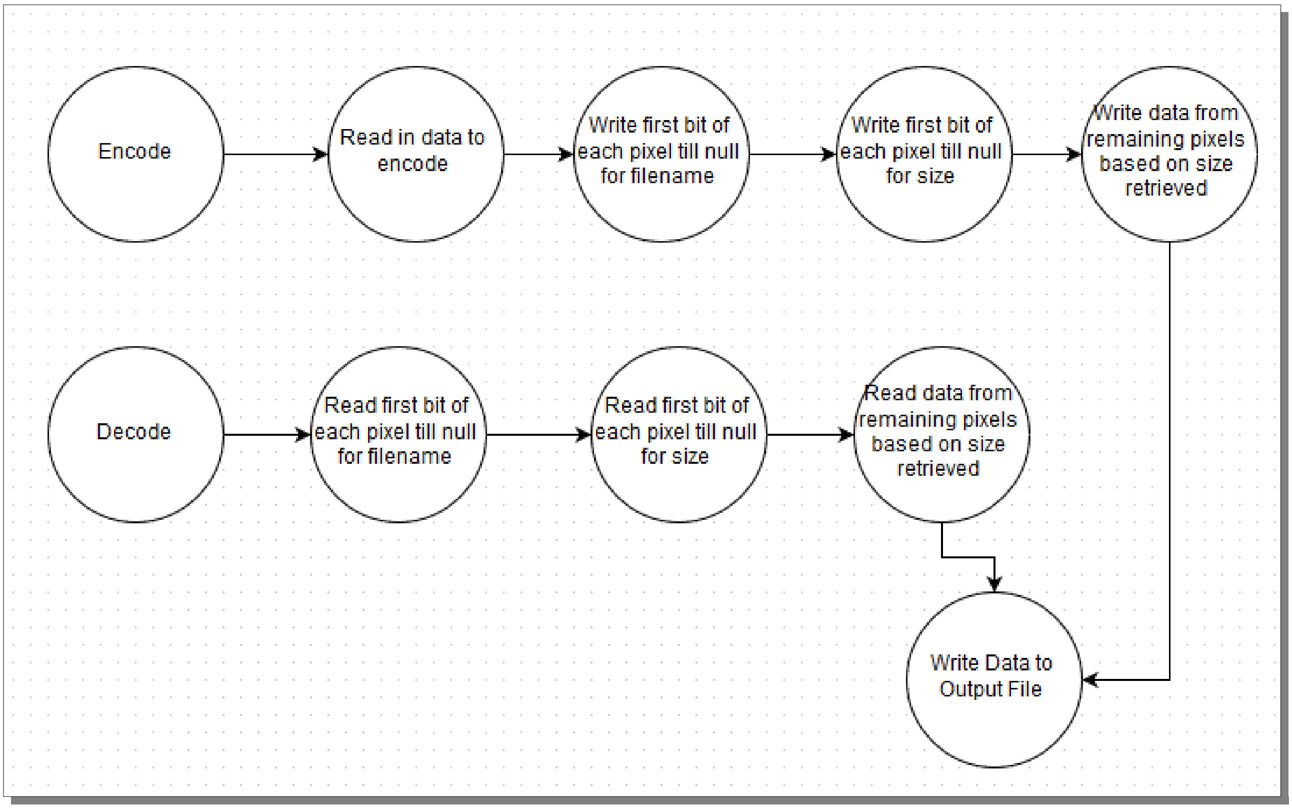
Encode will create a file named encoded_data.bmp containing the embedded data. Decode will create a file named [output/name] containing the data embedded into the image.

Compiling the program

TO compile the program execute the following on the main directory

```
gcc *.c -o covertdata
```

Design


Encode
Read in data to encode
Write first bit of each pixel till null for filename
Write first bit of each pixel till null for size
Write data from remaining pixels based on size retrieved
Decode
Read first bit of each pixel till null for filename
Read first bit of each pixel till null for size
Read data from remaining pixels based on size retrieved
Write Data to Output File

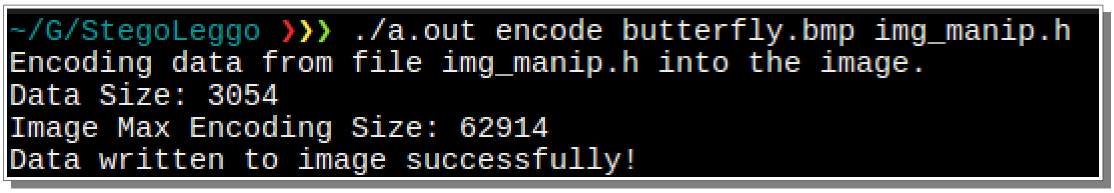
Pseudocode

```
int main{
    read in argument
    grab pixel data
    grab header data
    checkbmp_type(bmp header)
    if encode specified
        encode_data) into image
    if decode specified
        decode_data() out of image
}

encode_data(){
        open data to encode into image
        read header of bitmap image into struct
        read pixels of bitmap image into an array
        insert_encode_data() for filename into pixels ensure null termination
        insert_encode_data() for filename into pixels ensure null termination
        insert_encode_data() for the file to encode into the pixels
        write_bmpi()
}

insert_encode_data(){
        for each c character of data
                for each p bit in a byte
                        set the 0th bit of pixels[position] to the  p bit of data[c]
        return new pixel array
}

decode_data_basic(){
        grab_decode_header() for the filename
        grab_decode_header() for the size
        grab_decode_header() for the data
        write retrieved data to a file with the specified output name
}
 checkbmp_type(){
        if signature of FHDR is not « BM »
                exit program, invalid image
        if size matches a bmpinfo file
                return 1
        return 0
}
```

Testing

Test No	Name	Desc.	Tools Used	Pass/Fail
1	Data Hidden in Image	Check if data is noticeable within image	Eyes, covertdata	Pass
2	Data Retrieved	Check if data can be retrieved from encoded image	Less, ls, covertdata	Pass
3	Filename	Check if encoded data file name is properly written to image	Ls, covertdata	Pass

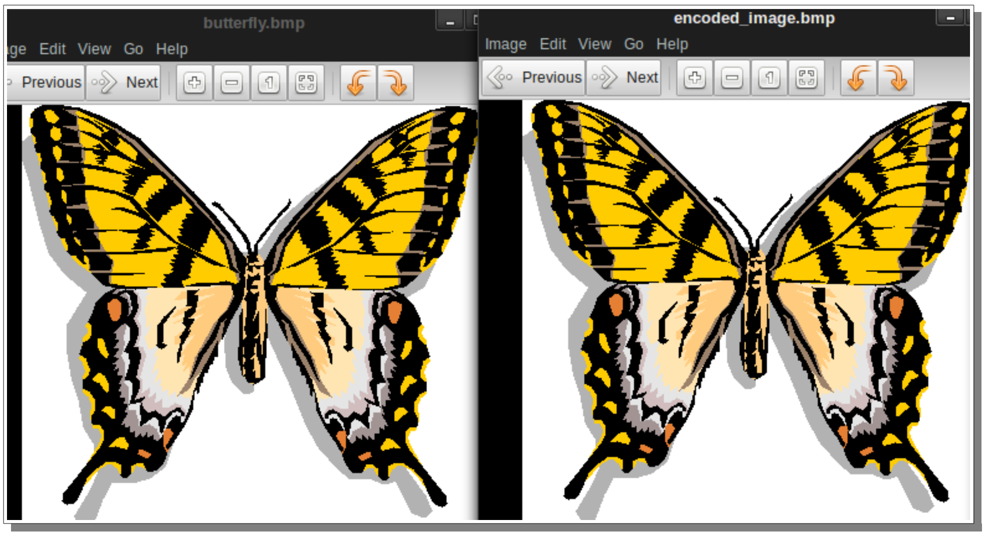
Test 1 – Data Hidden in Image



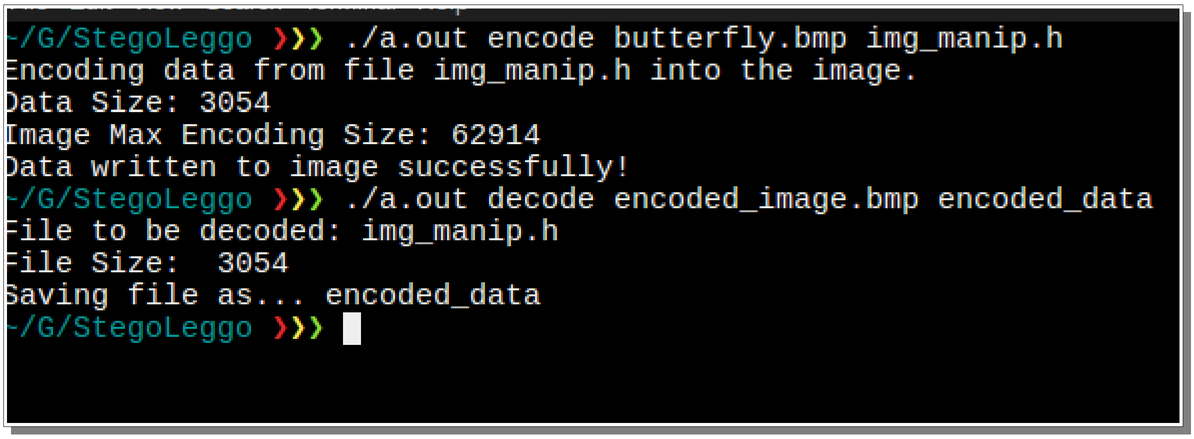
```
~/G/StegoLeggo ❯❯❯ ./a.out encode butterfly.bmp img_manip.h
Encoding data from file img_manip.h into the image.
Data Size: 3054
Image Max Encoding Size: 62914
Data written to image successfully!
```

Here we successfully encoded 3054 bytes of data into the image. If we compare the images we cannot discern any change with our eye.

Here the encoded image is on the right, and the regular image on the left. This test is successful.



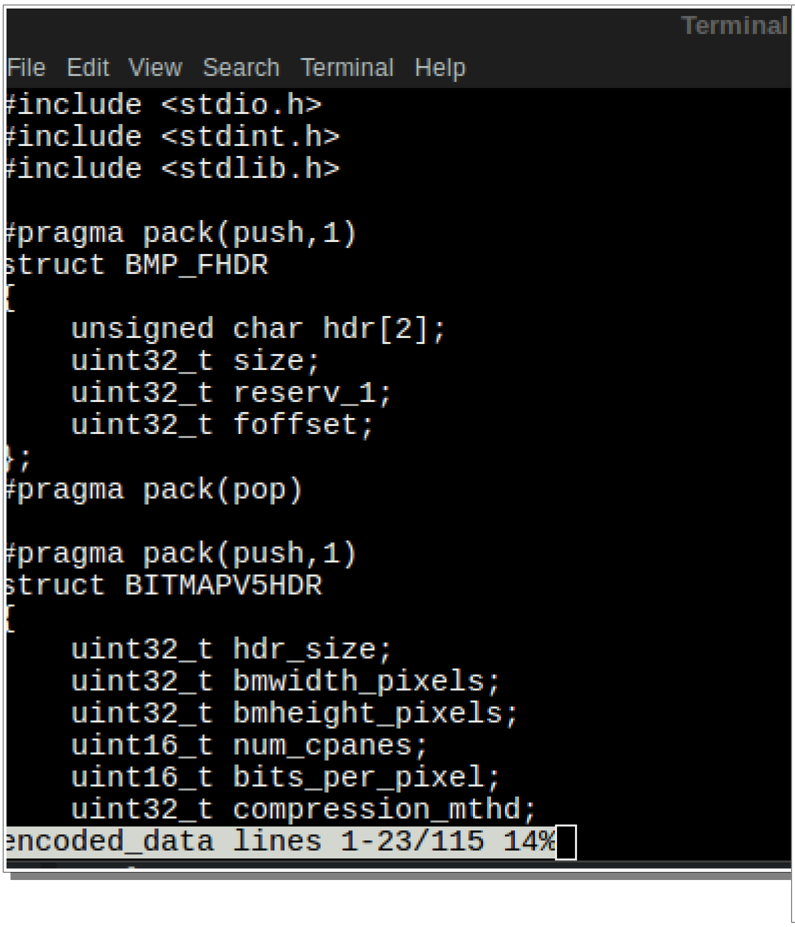
Test 2 – Data Retrieved



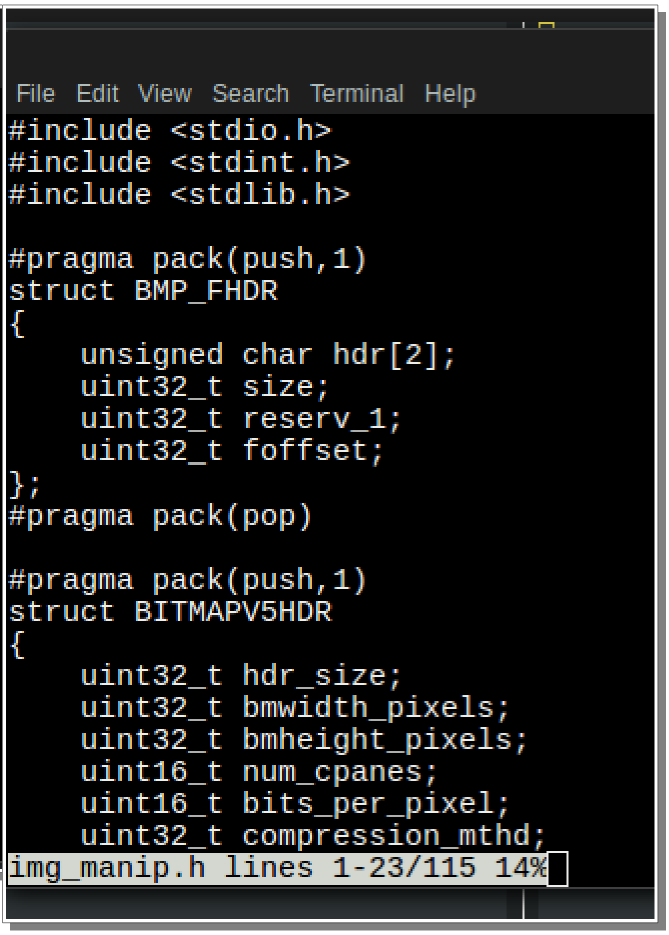
```
~/G/StegoLeggo ❯❯❯ ./a.out encode butterfly.bmp img_manip.h
Encoding data from file img_manip.h into the image.
Data Size: 3054
Image Max Encoding Size: 62914
Data written to image successfully!
~/G/StegoLeggo ❯❯❯ ./a.out decode encoded_image.bmp encoded_data
File to be decoded: img_manip.h
File Size:  3054
Saving file as... encoded_data
~/G/StegoLeggo ❯❯❯
```

Here we test to see if the data encoded is the same as the data decoded. Below we intiated an encode and decode.

As you can see, the same filename and file size was retrieved when we decoded the image. If we compare the data output of encoded_data and img_manip.h, we find them to be the same.



```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

#pragma pack(push,1)
struct BMP_FHDR
{
    unsigned char hdr[2];
    uint32_t size;
    uint32_t reserv_1;
    uint32_t foffset;
};
#pragma pack(pop)

#pragma pack(push,1)
struct BITMAPV5HDR
{
    uint32_t hdr_size;
    uint32_t bmwidth_pixels;
    uint32_t bmheight_pixels;
    uint16_t num_cpanes;
    uint16_t bits_per_pixel;
    uint32_t compression_mthd;
encoded_data lines 1-23/115 14%
```

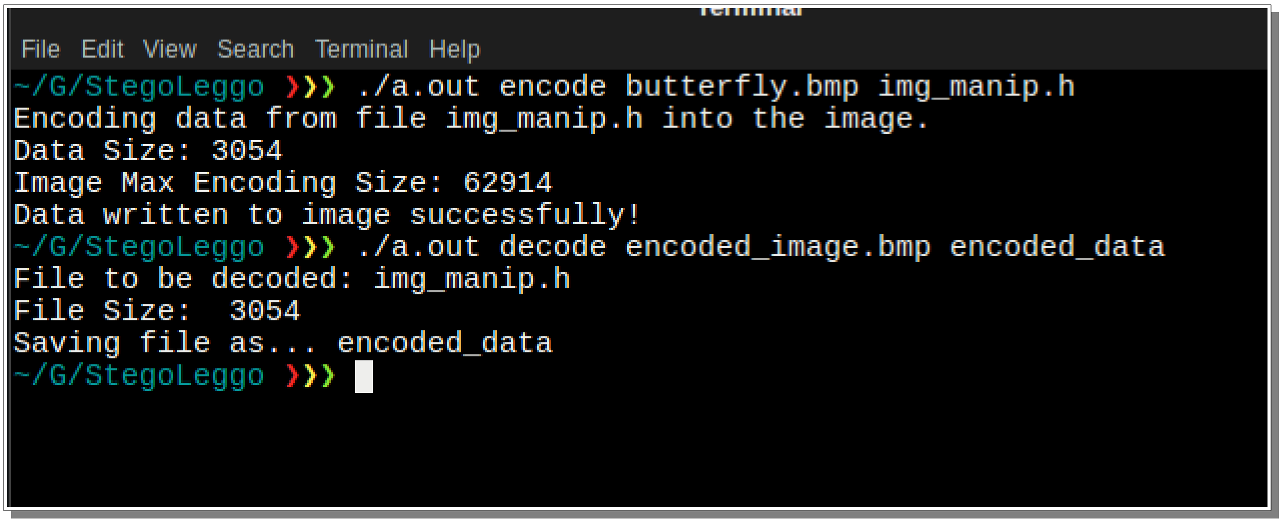


```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

#pragma pack(push,1)
struct BMP_FHDR
{
    unsigned char hdr[2];
    uint32_t size;
    uint32_t reserv_1;
    uint32_t foffset;
};
#pragma pack(pop)

#pragma pack(push,1)
struct BITMAPV5HDR
{
    uint32_t hdr_size;
    uint32_t bmwidth_pixels;
    uint32_t bmheight_pixels;
    uint16_t num_cpanes;
    uint16_t bits_per_pixel;
    uint32_t compression_mthd;
img_manip.h lines 1-23/115 14%
```

This shows that the data encoding was successful.

Test 3 – Filename

In this I will demonstrate that the filename is properly stored. Consider the following image.



```
~/G/StegoLeggo ❯❯❯ ./a.out encode butterfly.bmp img_manip.h
Encoding data from file img_manip.h into the image.
Data Size: 3054
Image Max Encoding Size: 62914
Data written to image successfully!
~/G/StegoLeggo ❯❯❯ ./a.out decode encoded_image.bmp encoded_data
File to be decoded: img_manip.h
File Size:  3054
Saving file as... encoded_data
~/G/StegoLeggo ❯❯❯ 
```

As you can see, img_manip.h was encoded, and if we look to decode, the file to be decoded is shown to be img_manip.h. This is a success.