F28HS Hardware Software Interfaces

Coursework 1

Name: Naqeeb Mohammed Naseer

HWU ID: H00232203

Project Name: PPM Steganography

Implementation and Design of the program

1- The structs:

a. Two structs were used in the implementation of the program. One to hold the entire image (PPM) and the other to hold RGB values of the image (Pixel).
b. A one-dimensional array of type Pixel, that is declared inside the PPM struct, was used to hold RGB values of pixels. It was decided not to use a matrix for this purpose since 1D arrays are much each easier to traverse and access data.
c. Fscanf method was used for reading in the getPPM method as it is comparatively easier to use as well as supports pattern matching.

2- Encoding:

a. The random pixels at which the text would be inserted in was found using the srand and rand functions.
b. X random numbers were generated where X is the length of the string. An array of size 'pixelCount' was created to hold the random numbers where 'pixelCount' is the number of pixels in the image.
c. The array was sorted using the built-in qsort() method to make sure that pixel locations were successive.
d. Each character in the string was converted to ASCII int value and stored in the red field of their respective pixel determined by the random number.
e. If the ASCII value and the red field are equal a new random location is generated and the character is inserted there.
f. The above case is the reason why it was decided to use an array of size 'pixelCount' rather than string length since we can add more random numbers if necessary.

3- Decoding:

a. Each pixel of both images (encoded and original) is compared. If the red field of the encoded image at pixel ‘i’ is not equal to that of the original, it’s value is added to an int array. A counter variable is incremented every time a new value is added to the int array. This variable gives the length of the string.
b. Each value in the int array is converted to a char and inserted into a char array. Finally, a ‘\0’ character is added to mark the end of the string.

4- Main program:

a. The program takes 3 arguments; the run mode, input/original PPM image and output/encoded PPM image.
b. The run modes are “e” for encode and “d” for decode.
c. Error case is when the user types in another value for run mode other than “e” or “d”.

Running the Program (On Windows IDE)

1- Encoding the data (entering data):

```
C:\Users\naqee\CLionProjects\steganographyCW\cmake-build-debug\steganographyCW.exe e aar(1).ppm out.ppm
Type the message you want to encode:
hello world
```

2- Original Image vs Encoded image:

Original

Encoded

3- Displaying the encoded message by comparing images:

```
C:\Users\naqee\CLionProjects\steganographyCW\cmake-build-debug\steganographyCW.exe d aar(1).ppm out.ppm
The hidden message is: hello world


Process finished with exit code 0
```

4- Error case when argument used is incorrect:

```
C:\Users\naqee\CLionProjects\steganographyCW\cmake-build-debug\steganographyCW.exe a aar(1).ppm out.ppm
Wrong input.

Process finished with exit code 1
```

Program listing

```c
#include <stdio.h>
#include <stdlib.h>
#include <mem.h>

#define COMMENTMAX 5
#define COMMENTSIZE 39

typedef struct {
    int red,green,blue;
} Pixel;

typedef struct {
    char type[2];
    char comment[COMMENTMAX][COMMENTSIZE];
    int width, height;
    int max;
    Pixel* pixelData;
} PPM;

//method for reading data from PPM file and storing in a struct
PPM * getPPM(FILE * fin)
{
    PPM *img;

    if (fin == NULL) {
        printf("Unable to open file \n");
        exit(1);
    }

    //allocate memory for image
    img = (PPM *)malloc(sizeof(PPM));
    if (!img) {
        printf("Unable to allocate memory\n");
        exit(1);
    }

    //read image type
    fscanf(fin,"%2s\n",img->type);
    if(strcmp(img->type,"P3") != 0){
        printf("Not in P3 format");
        exit(1);
    }

    //reading comments
    int i=0;
    while (i<COMMENTMAX){
        fscanf(fin," #%40[^\n]",img->comment[i]); //pattern matching
        i++;
    }

    //read image size information
    fscanf(fin,"%d %d", &img->width, &img->height);

    //read rgb max value
    fscanf(fin,"%d", &img->max);
```

```
    //memory allocation f or pixel data
    int pixCount = img->width * img->height;
    img->pixelData = (Pixel*)malloc(pixCount * sizeof(Pixel));

    if(!img->pixelData){
        printf("Unable to allocate memory \n");
        exit(1);
    }

    //reading pixel data
    int loop=0;
    while(loop<pixCount){
        fscanf(fin,"%d %d %d",&img->pixelData[loop].red,&img-
>pixelData[loop].green,&img->pixelData[loop].blue);
        loop++;
    }

    fclose(fin);
    return img;
}

//method for printing PPM image.
void showPPM(PPM * i){
    //image format
    printf("P3 \n");

    //comments
    int j=0;
    while(j<COMMENTMAX){
        if(strlen(i->comment[j])<2){
            break;
        }
        printf("#%s \n",i->comment[j]);
        j++;
    }

    //image size
    printf("%d %d \n",i->width,i->height);

    //rgb max value
    printf("%d\n",i->max);

    //pixel data
    int pixCount= i->width * i->height;
    int printer;
    for(printer=0;printer<pixCount;printer++){
        printf("%d %d %d \n",i->pixelData[printer].red,i-
>pixelData[printer].green,i->pixelData[printer].blue);
    }
}

//comparator function for qsort
int cmpfunc (const void * a, const void * b)
{
    return ( *(int*)a - *(int*)b );
}

//method for encoding text into PPM image
PPM * encode(char * text, PPM * i){
    PPM * output = i;
```

```
    int textLength = strlen(text);
    int pixCount = i->height * i->width;
    int count = 0;
    if(textLength > pixCount){
        printf("The image cannot hold your sentence.");
        exit(1);
    }
    int randArray[pixCount]; //creating sorted array of random numbers
    srand(1);
    int j=0;
    int x = rand()%pixCount+1;
    while(j < textLength){
        randArray[j]=x;
        x = rand()%pixCount+1;
        count++;
        j++;
    }
    qsort(randArray,count,sizeof(int),cmpfunc);

    j=0;
    while(j<textLength){
        if(output->pixelData[randArray[j]].red == text[j]){  //if red value and
text ASCII are same find another pixel
            randArray[count++]=rand()%pixCount+1;
            qsort(randArray,count,sizeof(int),cmpfunc);
            output->pixelData[randArray[count]].red = text[j];
        }
        else{
            output->pixelData[randArray[j]].red = text[j];
        }
        j++;
    }
    return output;
}

//method to write to a file
void writePPM(PPM *img, FILE * f)
{
    //image format
    fprintf(f,"P3 \n");

    //comments
    int j=0;
    while(j<COMMENTMAX){
        if(strlen(img->comment[j])<1){
            break;
        }
        fprintf(f,"#%s \n",img->comment[j]);
        j++;
    }

    //image size
    fprintf(f, "%d %d\n",img->width,img->height);

    //rgb max value
    fprintf(f, "%d\n",img->max);

    // pixel data
    int pixCount= img->width * img->height;
    int printer;
```

```
    for(printer=0;printer<pixCount;printer++){
        fprintf(f,"%d %d %d \n",img->pixelData[printer].red,img-
>pixelData[printer].green,img->pixelData[printer].blue);
    }

    fclose(f);
}

//method to decode the message by comparing images
char * decode(PPM * original, PPM * input){
    char * result;
    int pixCount = input->height * input->width;
    int * temp = malloc(pixCount * sizeof(int));
    int j = 0;
    int strlength = 0;
    while(j<pixCount){
        if(original->pixelData[j].red != input->pixelData[j].red){
            temp[strlength] = input->pixelData[j].red;
            strlength++;
        }
        j++;
    }
    result = malloc(strlength * sizeof(char));
    j = 0;
    while(j<strlength){
        result[j] = temp[j];
        j++;
    }
    result[j]='\0';
    return result;
}


int main(int argc,char ** argv){

    if(argv[1][0] == 'e'){ //encoding
        PPM *image1;
        FILE *f = fopen(argv[2], "r+");
        image1 = getPPM(f);
        PPM * encoded;
        char input[100];
        printf("Type the message you want to encode: \n");
        fgets (input, 100, stdin);
        encoded = encode(input,image1);
        FILE * fout = fopen(argv[3], "w+");
        writePPM(encoded,fout);
        printf("Displaying encoded PPM image \n");
        showPPM(encoded);
    }

    else if(argv[1][0] == 'd'){ //decoding
        PPM *image2;
        FILE *f = fopen(argv[2], "r+");
        image2 = getPPM(f);
        PPM * input;
        FILE * fin = fopen(argv[3],"r+");
        input = getPPM(fin);
        char * final;
        final = decode(image2,input);
        printf("The hidden message is: %s \n",final);
```

```
    }

    else{ //error case
        printf("Wrong input. \n");
        exit(1);
    }
}
```

References

1- Tutorials point was used to learn how to use qsort method. Also, various file reading methods were learnt from the same website.

2- Following link was used to learn about pattern matching used in fscanf: http://stackoverflow.com/questions/24418394/difference-between-scanf79-n-line-vs-scanf79-n-n-line-vs-scanf