2/28/2017

Steganography with PPM Images

F28HS2 Hardware Software Interface Coursework 1

Mark Gordon
H00228757

Chapter 1 - Introduction

This coursework involves writing a simple steganography system in C.

Steganography is a technique for hiding text in images, by replacing successive random pixels in the image with letters from the text. The text is then retrieved by comparing the new image with the old image, and extracting letters from the new one where it differs from the old one.

The Image format of choice was PPM which is a very simple open source RGB colour bitmap format. According to the PPM documentation (http://netpbm.sourceforge.net/doc/ppm.html) each image follows this format:-

1. A "magic number" for identifying the file type. A ppm image's magic number is the two characters "P6".
2. Whitespace (blanks, TABs, CRs, LFs).
3. A width, formatted as ASCII characters in decimal.
4. Whitespace.
5. A height, again in ASCII decimal.
6. Whitespace.
7. The maximum color value (Maxval), again in ASCII decimal. Must be less than 65536 and more than zero.
8. A single whitespace character (usually a newline).
9. A raster of Height rows, in order from top to bottom. Each row consists of Width pixels, in order from left to right. Each pixel is a triplet of red, green, and blue samples, in that order. Each sample is represented in pure binary by either 1 or 2 bytes. If the Maxval is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first.

A row of an image is horizontal. A column is vertical. The pixels in the image are square and contiguous.

Following these guidelines, we can create a program to hide text in an image.

Chapter 2 - Algorithms and Methods

Structs

Our first task was to obtain the PPM image information and store it into a struct. The struct would need to store: -

- P3 – 2 letter code for PPM format
- *width* – integer number of columns
- *height* – integer number of rows
- *max* – integer maximum colour value – usually 255
- r_i g_i b_i – integers between 0 and *max* for pixel *i*'s red, green and blue values

With this requirement the struct PPM was produced. This will store the information perfectly assuming any comments in the image do not exceed 500 characters:

```
typedef struct PPM{

   char code[3];
   char comment[500];
   int width, height;
   int max;
   pixel * pixels;

}ppm;
```

The pixel information is stored by pointing to another struct called Pixel. For every pixel in the image, a pointer in the PPM struct of index n will point to a Pixel struct n storing the rgb values.

```
typedef struct Pixel{

   int red;
   int green;
   int blue;

}pixel;
```

getPPM

A method getPPM is required for getting the image information from file and storing it into the PPM struct. This involves defining a struct pointer and allocating it memory for the size of the PPM struct:

```
ppm * image;
```

```
//allocates memory for the image
image = malloc(sizeof(ppm));
```

A check would then be done to ensure the memory was allocated. We would then read the first 3 characters into a buffer from the file using 'fscanf' to obtain the file format. A check would be done to ensure this was P3 format, if passed P3 would be stored into the struct code. After this we will need to store the comments if any. This code will store the comments by identifying if there is a '#' character and storing the characters until the end of the line. It will then skip over any whitespace until a new character is found and if this is another comment –signalled by '#' character-, it will again store until the end of the line. Once the while loop is exited, the ungetc function will be used to push the char held in c back into the stream as the function will over read by 1 character.

```
int i=0;
//Will store the comments
while(c == '#'){
  //adds characters until end of line
   do{
      image->comment[i] = c;
      c = getc(fd);
      i++;
   }while((c != '\n'));
  //adds end of line character
  image->comment[i] = c;
  //remove white space and get to next comment or dimensions
  while(c == '\n'){
     c = getc(fd);
  }
  i++;
}
//will over read by 1 character so have to add c back to the stream
 ungetc(c,fd);

//adds end of line character
image->comment[i] = '\0';
```

The width, height and maximum colour value are then read into the struct with an error returned and the program exits if fails. Next the pixels have to be read. But first, space will need to be allocated for the pixel pointer inside the PPM struct. This will have to be the (number of pixels *sizeof pixel struct):-

```
//allocates memory for pixels
image->pixels = malloc( numberOfPixels * sizeof(pixel));
```

Then the pixels are finally read into the struct. This is done by looping for the number of pixels and doing another 'fscanf' getting 3 decimals and storing them into pixel struct - via the PPM struct - at the index of the current loop. If this fails an error is displayed and the program exits. The Struct pointer is then returned.

Full getPPM method :-

```
struct PPM * getPPM(FILE * fd){

   char buffer[3];

   ppm * image;

    //allocates memory for the image
    image = malloc(sizeof(ppm));

    if(image == NULL){
      printf("Failed to allocate memory for image\n\n");
      exit(0);
    }

    //gets first and second character from file
    fscanf(fd, "%c", &buffer[0]);
    fscanf(fd, "%c", &buffer[1]);
    //gets the end of line character
    fscanf(fd, "%c", &buffer[2]);

   //checks image is of type P3 and stores information if it is
   //else show error and exit program
   if(buffer[0] != 'P' || buffer[1] != '3'){
      printf("Not P3 file format!\n\n");
      exit(0);
   }else{
      image->code[0] = 'P';
      image->code[1] = '3';
      image->code[2] = '\0';
   }

   //Will remove white space and get to comment(s) or dimensions
   int c = getc(fd);
   while(c == '\n'){
   c = getc(fd);
   }

   int i=0;
   //Will store the comments
   while(c == '#'){
      //adds characters until end of line
      do{
         image->comment[i] = c;
         c = getc(fd);
```

```
            i++;
          }while((c != '\n'));
        //adds end of line character
         image->comment[i] = c;
        //remove white space and get to next comment or dimensions
        while(c == '\n'){
           c = getc(fd);
        }
        i++;
      }
    //will over read by 1 character so have to add c back to stream
     ungetc(c,fd);

    //adds end of line character
    image->comment[i] = '\0';

    //reads width, height and max and returns error if fails
    if(fscanf(fd, "%d %d %d", &image->width, &image->height, &image->max) != 3){
          printf("Image size is invalid! Width: %d, Height: %d\n\n", image->width,
          image->height);
        exit(0);
     }

     int numberOfPixels = (image->height * image->width);
     //allocates memory for pixels
     image->pixels = malloc( numberOfPixels * sizeof(pixel));

     //reads in pixels and returns error, exiting the program, if it can't read the read,
     green and blue pixel
     for(int i = 0; i<numberOfPixels; i++){

       if(fscanf(fd, "%d %d %d", &image->pixels[i].red, &image->pixels[i].green,
      &image->pixels[i].blue) != 3){
      printf("Error reading pixels!");
      exit(0);
     }

   }

   return image;

 }//end of getPPM method
```

showPPM

A method to display the PPM image information stored in the struct is required. Simply done by printing out the code, comment, width, height and max colour value from the PPM struct pointer. Then the pixels are printed out by looping for the number of pixels in the image,

printing out the pixel values of each pixel index. If the loop index and width give an absolute value, then take a new line as this means all the pixels on this row has been printed.

Full showPPM method:-

```
void showPPM(struct PPM * image){


  int numberOfPixels = (image->height * image->width);

  printf("\nImage Code: %s\n", image->code);
  printf("Comment: %s", image->comment);
  printf("Image width: %d\n", image->width);
  printf("Image height: %d\n", image->height);
  printf("Maximum colour value: %d\n", image->max);

   //loop for number of pixels displaying each one
   for(int i = 0; i<numberOfPixels; i++){
     //take new line if end of row
     if((i % image->width) == 0){
        printf("\n");
     }
    printf("%3d %3d %3d\t", image->pixels[i].red, image->pixels[i].green, image-
   >pixels[i].blue);

  }

  printf("\n");

}//end of showPPM
```

encode

An encode method is required to hide the message within the red value of the pixels in the PPM image. First of all, a check if done to ensure the message length doesn't exceed the number of pixels. Then we loop for the size of the message in order to generate random numbers from 0 to the number of pixels – 1. A check is then done and if its unique its then added to an array of random numbers, else if its not unique, the number is discarded and another random number is generated and checked. After this, the C library function 'qsort' (worst case $O(n^2)$) is used to sort the array of random numbers into ascending order as the hidden characters need to be placed in order to be decoded: -

```
//performs quicksort on pixelIndex array to sort numbers in ascending order
 qsort(pixelIndex, sizeOfMessage, sizeof(int), cmpfunc);
```

a function called cmpfunc is passed into this function which just returns the result of a comparison: -

```
int cmpfunc (const void * a, const void * b){
        return ( *(int*)a - *(int*)b );
}
```

Then we loop for the size of message, replacing the red value of the pixel at the index of the number stored in the random number array at the current loop index with the ASCII of the character at the current loop index: -

```
for(int i=0; i < sizeOfMessage; i++){
    image->pixels[pixelIndex[i]].red = text[i];
 }
```

The image is then returned.

Full encode method:-

```
struct PPM * encode(char * text, struct PPM * image){

    int numberOfPixels = (image->height * image->width);

   int sizeOfMessage = strlen(text);

    //checks if encoding is possible
    while(sizeOfMessage > numberOfPixels){
       printf("Not enough pixels to hide message. Please use a different picture or
       enter a smaller message.\n");
       exit(0);
    }

   //stores the randomly generated numbers for where to store the characters
   int pixelIndex[sizeOfMessage];

   srand(time(NULL));

   //generates random numbers for the amount of characters in the message
   for(int i=0; i < sizeOfMessage; i++){

     bool hasBeenGeneratedBefore = false;

     //produces random number between 0 and numberOfPixels - 1
     int newRandomNumber = (int) (rand() % numberOfPixels);

      //loops for amount of numbers already generated and stored so can checks
     //if the newly generated number has already been generated
     for(int y =0; y < i; y++){

        if(newRandomNumber == pixelIndex[y])
```

```
            hasBeenGeneratedBefore = true;

          }

        if(hasBeenGeneratedBefore == true){
            i--;
        }else{
            pixelIndex[i] = newRandomNumber;
        }

      }

       //performs quicksort on pixelIndex array to sort numbers in ascending order
      qsort(pixelIndex, sizeOfMessage, sizeof(int), cmpfunc);


     //replaces the red value of the pixel at the index of the number stored in the
     //random number array at the current loop index with the ASCII of the
     //character at the current loop index
     for(int i=0; i < sizeOfMessage; i++){
         image->pixels[pixelIndex[i]].red = text[i];
     }

     return image;

}//end of encode
```

decode

The decode method is used to compare the original image, to the altered image to obtain the hidden message. In this, we define a char pointer to hold the message. We then loop for the number of pixels, comparing the red values of the pixels in the original image and altered image. If they differ, then this value is converted to a character and added to the char pointer using the function stradd:-

```
char* stradd(const char* a, const char* b){
    //length of new string
    size_t len = strlen(a) + strlen(b);
    //creates pointer with length of both pointers combined + 1(for end of line
    character)
    char *ret = (char*)malloc(len * sizeof(char) + 1);
   *ret = '\0';
   //returns the concatenation of both pointers
   return strcat(strcat(ret, a) ,b);
```

```
}
```

The char pointer is then returned.

Full decode method: -

```
char * decode(struct PPM * i1, struct PPM * i2){

    int numberOfPixels = (i2->height * i2->width);
    char * secret = "";


    for(int i = 0; i<numberOfPixels; i++){

      //compares red pixels between original image and altered image
      if(i1->pixels[i].red != i2->pixels[i].red){

        char * c;
        c = malloc(sizeof(char));

         //changes the ascii value to character
        c[0] = i2->pixels[i].red;
        //adds the character to the char pointer
        secret = stradd(secret, c);

       }

     }//end of loop

    return secret;

  }//end of decode method
```

writePPM

This method will write the struct information to file creating a PPM image. We first of all have to define a file pointer then open/create the file for writing with the user defined filename and check if this was successful. We then write the code, comment, width, height and max colour depth to the file using 'fprintf'. We then loop for the number of pixels in the image, writing the information in each Pixel struct to the file, taking a new line when the loop index and image width is of absolute value as this means we've written all the pixel information in this row. The file is then closed.

Full writePPM method:

```
void writePPM(const char *filename, struct PPM *img){
    FILE *fp;

    fp = fopen(filename, "wb");

    if (!fp) {
        fprintf(stderr, "Unable to open file '%s'\n", filename);
       exit(1);
    }

    fprintf(fp,"%s\n%s%d %d\n%d", img->code, img->comment,img->width, img-
    >height, img->max);


   int numberOfPixels = (img->height * img->width);

   for(int i=0; i < numberOfPixels; i++){

      //take new line if width is reached
       if((i % img->width) == 0){
          fprintf(fp, "\n");
        }

       //write pixel values
       fprintf(fp, "%3d %3d %3d\t", img->pixels[i].red, img->pixels[i].green, img-
       >pixels[i].blue);

    }

    fclose(fp);
}
```

handleEncoding

We require a method for setting up the program for encoding, if the user has chosen the encode command. In this method it will open the PPM image for reading, perform a check on the file to ensure it was opened properly and get the information from the file by calling the getPPM method storing it into a PPM struct. Then, we get the message to be encoded into the image from user input, encode this into the image by calling the encode method, display the new PPM file information to the user with the message encoded, and save the new image with the encoded message by calling the writePPM method.

Full handleEncoding method : -

```
void handleEncoding(const char * originalFilepath, const char * alteredFilepath){


    FILE * originalImageFile;

   //opens file for the original image
    originalImageFile = fopen(originalFilepath, "r");


    if (!originalImageFile) {
       fprintf(stderr, "Unable to open file '%s'\n", originalFilepath);
       exit(1);
    }

     //gets information from image and stores it in struct
     struct PPM * originalImage = getPPM(originalImageFile);

      int numberOfPixels = (originalImage->height * originalImage->width);

      //allocates memory for message and gets message from user
     char *message = malloc(sizeof(char) * (numberOfPixels + 1));
     printf("\nPlease enter a secret message: ");

     //stores message
    scanf("%[^\n]s", message);

    //encodes image with message
    encode(message, originalImage);

    //shows original image information
    showPPM(originalImage);

    //writes new image to file (originalImage info is changed hence the name but is
    //saved with a different filename so the original file will still contain the original
   //information)

   writePPM(alteredFilepath, originalImage);

}//end of handleEncoding
```

handleDecoding

We need a method for setting up the program for decoding if the user chooses the decode command. We first open up two files for reading – the original image and the altered image- and check that they were both successfully opened. We then call the getPPM on both images to store their information into structs. A check is done to ensure the images have the same

dimensions. Then we define a char pointer and set its value to the result of calling the decode method on the 2 images and then we print the message which was encoded.

Full handleDecoding method:-

```
void handleDecoding(const char * originalFilePath, const char* alteredFilePath){

    //opens file for original image
    FILE * originalImageFile = fopen(originalFilePath, "r");
    //opens file for image with hidden message
    FILE * alteredImageFile = fopen(alteredFilePath, "r");

    if (!originalImageFile) {
        fprintf(stderr, "Unable to open file '%s'\n", originalFilePath);
        exit(1);
    }

    if (!alteredImageFile) {
        fprintf(stderr, "Unable to open file '%s'\n", alteredFilePath);
        exit(1);
    }

    //gets information from image and stores it in struct
    struct PPM * originalImage = getPPM(originalImageFile);
    struct PPM * alteredImage = getPPM(alteredImageFile);

    //ensures equal dimensions to avoid weird results
    if(originalImage->width != alteredImage->width && originalImage->height !=
    alteredImage->height){
        printf("\nThis is not the original image as the dimensions differ!\n\n");
        exit(0);
    }

    //decodes and prints out hidden message by comparing both structs
    char * secret = decode(originalImage, alteredImage);
    printf("\nSecret message is : %s\n\n", secret);

}//end of handleDecoding
```

main

The main method is what runs at program start. We first check the user has entered 4 arguments. Passing this we check if the 2nd argument is ‘d’ or ‘e’. If d then the handleDecoding method is called passing in argument 2 and 3. If e then the handleEncoding method is called also passing in argument 2 and 3. If the user fails to enter d or e for the 2nd argument, an error is displayed and the program exits

Full main method:-

```
int main( int argc, const char* argv[] ){


    //if the number of arguments is not 4 exit the program
    if(argc != 4){
       printf("\nThere should be 4 arguments!\n\n");
         exit(0);
    }

    //d is the command for decoding
     if(strcmp(argv[1],"d") == 0){
         handleDecoding(argv[2], argv[3]);
     }//e is argument for encoding
    else if (strcmp(argv[1],"e")==0){
       handleEncoding(argv[2], argv[3]);
     }
     else//will warn user of an unrecognised character argument
        printf("\nWrong character entry for 2nd argument. Enter 'E' for encoding, 'D' for
         deconding.\n\n");


     return 0;
    }
```

Chapter 3 - Use of Program

Encoding

We shall encode a PPM image with the message “Hello”. We will use a small 4 x 4 image so the difference can be seen clearly.

We get the program to encode by calling:

./Steg e originalImage.ppm alteredImage.ppm

The 1st argument is the program path, the 2nd is the command character, the 3rd is the original image used to encode, and the 4th the filename you would like to give the new encoded image. The program will then ask for a message to encode which we have responded with ‘Hello’: -

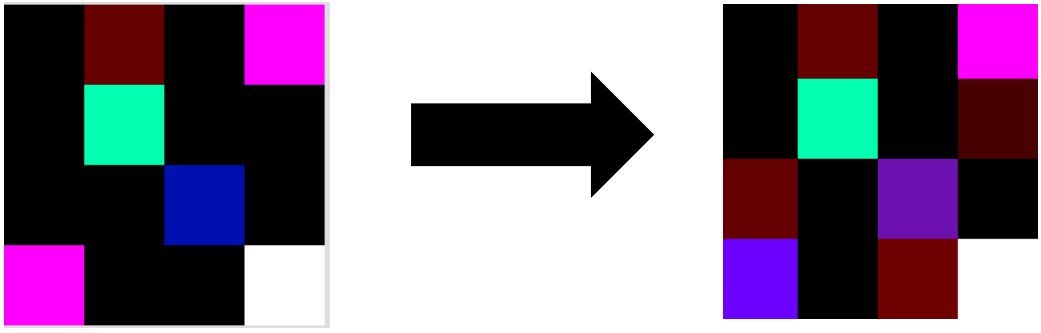
```
Please enter a secret message: Hello
```

It will then display the file information of the newly created PPM image with the encoded message. The values that have been changed have been highlighted: -

```
Image Code: P3
Comment:
Image width: 4
Image height: 4
Maximum colour value: 255

 72   0   0     100   0   0     101   0   0     255   0 255
108   0   0       0 255 175       0   0   0       0   0   0
  0   0   0       0   0   0     108  15 175       0   0   0
255   0 255       0   0   0     111   0   0     255 255 255
```

The new image will be saved. We can now compare the original to the new image:



With such a low pixel size we can clearly see the difference displayed here.

Decode

Decoding is done by making the call

./Steg d originalImage.ppm alteredImage.ppm

Using the previous images, this would return ‘Hello’:

```
Secret message is : Hello
```

It has noticed that the differing values are 72, 101, 108, 108 and 111 which converts as shown:

72 – H
101 – e
108 – l
108 – l
111 - o