Department of Computer Engineering
University of Peradeniya
Lab 03

Programming Methodology

February 1, 2017

1 Introduction

During this lab, you are supposed to implement a tool for processing 24-bit Bitmap images. Bitmap is a popular image format typically used for storing uncompressed image data. Therefore we can read the Bitmap images without any third party image processing library or algorithm to decode the compressed data.

When we read any 24-bit Bitmap image as a binary file, there are various information about the image (such as image width, height, and size) in pre-determined pattern. After the header section, you can access the data part of the image.

For a 24-bit bitmap image with pixel size 6×3 (columns×rows), the data has the following pattern:

BGRBGRBGRBGRBGRBGRPP
BGRBGRBGRBGRBGRBGRPP
BGRBGRBGRBGRBGRBGRPP

Each character representing a single byte in the memory. B,G and R indicates values of Blue, Green and Red stored. One BGR set is responsible for representing one pixel of the image. P means the padding value that appears after each row (if any). The number of padding values are calculated such that each row has a byte length that is a multiple of 4. For example, when there are 6 columns, each row consists of 3×6=18 bytes for BGR data. Since 18 is not a multiple of 4, we should add two bytes of padding to each row so that the total number of bytes per row becomes 20, which is a multiple of 4.

Fortunately, for this lab, you do not have to worry about the inside details of a Bitmap file structure. You are given a compiled program (**reader**) that can read any 24-bit bitmap image and print the RGB, padding along with the number of columns and rows that the data section has.

When you run the reader program, you would see the formatted output as in Fig.1 where the first row printing the number of pixel columns and pixel rows that the image has. For the given example the values are 3 and 2. Then for each pixel, the program prints three values, Red, Green, and Blue (Yes, reverse order because RGB is the normal convention). After each row, the padding is printed as three zeros because when a row has 3 pixels (3×3 = 9bytes), we need three bytes to make the row size a multiplication of 4.

```
titus@tesla:~$ ./reader small24.bmp
3 2
204 72 63
76 177 34
36 28 237
0
0
0
36 28 237
76 177 34
204 72 63
0
0
0
titus@tesla:~$
```

Figure 1: Output of the reader program for size 3×2 image

2 During the lab

2.1 Familiarize with getting input and loops

Getting input from the user is a very important part of any programme. Before you start programming lab 3, familiarize with input taking and nested loop. For nested loops, one classic example is printing a box using stars. When the program runs, it should give a prompt to the user to type the size of the box (columns rows). Then the program should print the output according to the given size.

2.2 Understand the convention

Run the reader program with very small size Bitmap image where you know the size. It would give you a clear understanding about how the reader program gives you the output and padding values. To test the program, we have given some example bitmap images. With the help of your partner, discuss how to take data and padding values (hope you would have figured out that the padding value is not the same for every image by now).

2.3 Test with the writer program

There is a program called **writer** in the lab 03 folder. To understand the reader-writer program connection, you would need some knowledge about pipes in linux shell (which will be explained in the lab). When you run the program using the following command:

```
$./reader sneha.bmp | ./writer sneha.bmp
```

It would produce an image called **conv.bmp** which is a converted image of the original one (Yes we know that it is not changed). The writer program just write the input it receive to the file called conv.bmp. Since we have done only the reading and writing, the image data has not changed.

3 Your task - Write a color inverting program

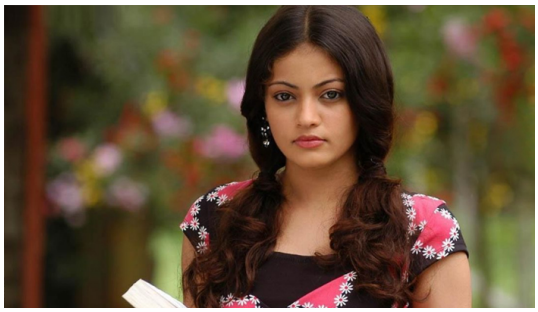
Your task is to write image processing program that can be plugged in between reader and writer programs. If your program can read the convention of the output produced by the reader program and produce a modified output to the writer program which follows the same convention, the writer program would see no different and write the modified data to **conv.bmp**. For example, the given binary **inverter** can invert the value of all three R,G,B components and print the values following the exact convention (of a Bitmap image). To invert each value, the program uses the following formula:

$$modifiedValue = 255 - Originalvalue$$

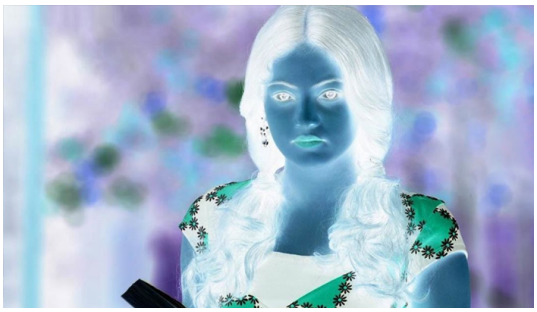
For example, if the input is 100 120 210 the inverter program would modify them as 155 135 45. You can plug the inverter program in the middle of reader and writer in the following way.

```
$./reader sneha.bmp | ./inverter | ./writer sneha.bmp
```

Which would produce image (b) from Fig. 2



(a) Original image



(b) Inverted image

Figure 2: Expected output from your program

The most important thing you should remember is that your inverter program should exactly follow the output convention the reader program follows (including first row and the padding). Otherwise the writer program would not able to read the output from your inverter program.

4 Submission

Submit a single zip file (rename it as lab03.zip) containing only your source code and the design file. Rename your source code to the following pattern where xxx is your registration number.

14xxxlab03.c

5 Important

You can start writing your program only after you are familiar enough with getting input, nested loops and the formula of the reading pattern for lab 03 excercise. I should mention that this is an extremely simple lab when you figured out what is happening and would be really hard if you just start with coding. As always, you should write the program individually because we mark the final submission individually, and under no circumstance, you should copy somebody else's code. Copying someone else's code (including your group mate's) or showing your source code to anyone else will earn you zero mark for the whole lab exercise.

6 Deadline

The deadline for the submission is Sunday (5) 23:55h.

7 Fun things to do

1. Try to see the output image from a negative filter (using mobile or desktop app) and check whether you can see the original image.
2. Rename the output file conv.bmp to another name and invert the new image again. You should get the original image again as conv.bmp
3. Use your program two times to check the output.

   ```
   $./reader sneha.bmp | ./inverter | ./inverter | ./writer sneha.bmp
   ```

4. Write a program called gray.c that can produce Fig. 3.

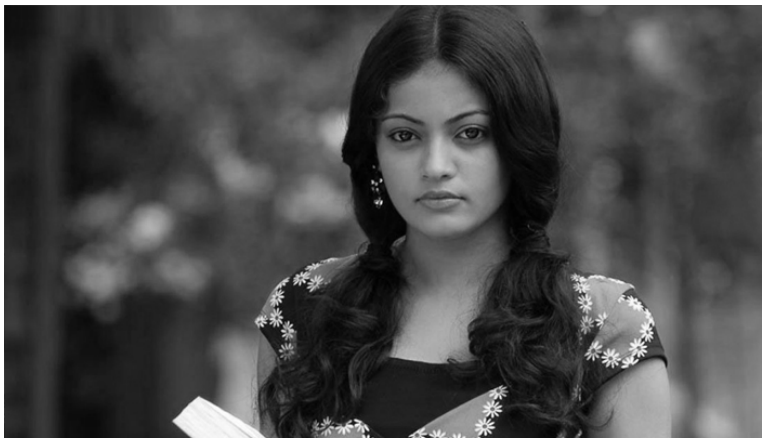


Figure 3: Output of the program gray.c (by averaging R,G,B)

5. If you have come this far, you may watch the movie **Lucky: No Time for Love (2005)** (Yes, this girl is the main actress in that movie).

8 References

- https://en.wikipedia.org/wiki/BMP_file_format/ About the Bitmap format.
- https://en.wikipedia.org/wiki/RGB_color_model RGB informations.