Lab: Transforming Pictures
CSC 161, “Imperative Problem Solving and Data Structures”
Department of Computer Science · Grinnell College
April 4, 2016

In this lab, we’ll get acquainted with the Picture type introduced in the MyroC library and practice writing functions that create, modify, or operate on images stored as values of this type.

The pairs
Here are the pairs for today’s lab:
• Faizaan Ali and Zachary Susag
• Saung Thuya and Anna Blindermann
• Jong Hoon Bae and Sanjay Sudhir
• Lex Martin and Jae Hong Shin
• Mattori Birnbaum and Gemma Nash
• Colin Greenman and Lilly Webster
• Josh Lavin and Yuyin Sun
• Ella Nicolson and Eli Salm
• Tanner Tufto and Erhaan Ahmad
• Ying Zhang and Sophie Gaschott
• Dennis Chan and Tyler Williams

Cory McCartan will take the place of anyone who is absent.

Exercises
Here are the definitions of the Pixel ("picture element") and Picture types from the MyroC.h header file:

```c
/**
 * @brief Struct for a pixel
 */
typedef struct
{
    unsigned char R; //!< The value of the red component
    unsigned char G; //!< The value of the green component
    unsigned char B; //!< The value of the blue component
} Pixel;

/**
 * @brief Struct for a picture object
 * @note the picture size is always 192 in height and 256 in width
 * @note Following standard mathematical convention for a 2D matrix,
 * @note all references to a pixel are given within an array as [row][col]
 */
typedef struct
{
    int height; //!< The height of the image -- set to 192 for robot camera
    int width; //!< The width of the image -- set to 256 for robot camera
    Pixel pix_array[192][256]; //!< The array of pixels comprising the image
} Picture;
```

Recall that an unsigned char is basically an integer in the range from 0 up to, but not including, 256 and occupies eight bits. A Pixel contains three such values, each indicating the contribution of one of the three primary colors to the exact color of the
pixel. For example, a “sky-blue” Pixel might have a red contribution of 108, a green contribution of 166, and a blue contribution of 205.

A Picture, then, can contain as many as 49152 such Pixels, arranged in as many as 192 rows with as many as 256 Pixels in a row. So it can store a color picture at any resolution up to 192 by 256, which happens to be the resolution of the camera in the Scribbler2 robot.

Exercise 0000: The file /home/stone/c/code/sky-blue-dot.c contains a C program that constructs and displays the image of a small sky-blue dot centered on dark-blue background. Copy this file into your directory for this course, read it once through carefully, compile it, and run the resulting executable file. This program does not use the Scribbler2 robots, so it’s not necessary to turn them on at this point. Are the effects of the program consistent with your expectations?

Exercise 0001: You will find a list of colors, with their Red-Green-Blue contributions, in /usr/share/X11/rgb.txt. With the help of information from this list, create some variants of the sky-blue-dot program: (a) one in which the background color is gold rather than dark blue; (b) one in which the dot is in the bottom right corner of the image rather than in the middle; (c) one in which the centered dot is expanded to a square seven pixels wide and seven pixels high; (d) one in which the image display window remains on screen until the user interrupts the program (hint: read the descriptive summary of the rDisplayPicture function in the MyroC.h file).

The rTakePicture function causes the Scribbler2 robot to take a picture with its camera. The return value is the Picture that it takes.

Exercise 0010: Write and test a C program that connects to the Scribbler2 robot, directs it to take a picture, and then briefly displays on screen the picture that it takes.

The “grayscale value” of a Pixel is an approximation of its visual brightness, expressed as an integer in the range from 0 to 255 (so that it can be represented in C as an unsigned char). It is computed by multiplying the red component of the Pixel by 0.30, the green component by 0.59, and the blue component by 0.11, adding the three products together, and discarding the fractional part of the result. So, for instance, the grayscale value of our sky-blue Pixel would be $\lceil 108 \cdot 0.30 + 166 \cdot 0.59 + 205 \cdot 0.11 \rceil$ (where the $\lceil \ldots \rceil$ operation means “find the greatest integer not exceeding the enclosed value”), which works out to be 152.

Exercise 0011: Write a C function that computes and returns the grayscale value of a given Pixel.

A Pixel in which all three components are equal always appears black, gray, or white, with a brightness that depends on the value of the component. So replacing all three components of a Pixel with its grayscale value gives you an equally bright Pixel with the color removed.

Exercise 0100: Write a C function with the signature

```c
void color_to_gray(Picture *ppic)
```

that overwrites all three components of every Pixel in the Picture to which ppic points with the grayscale value of that Pixel, thus producing a “grayscale” (black-and-white) version of the image. Test this function by taking a picture with the robot’s camera, applying color_to_gray to the picture it returns, and displaying the result.

Exercise 0101: Write and test a C function with the signature

```c
void purify(Picture *ppic)
```
which determines, for each Pixel in the Picture to which ppic points, which of the components is the greatest and overwrites the other two components with 0. Test this function by taking a picture with the robot’s camera, applying `purify` to the picture it returns, and displaying the result.

Exercise 0110: Write and execute a C program `flip-picture` that takes a picture from the robot (using `rTakePicture`), displays the picture for 5 seconds, flips the picture upside down, and finally displays the result for 5 seconds.

Flipping a picture upside down requires swapping pixels in one row (e.g., near the top) with pixels in a corresponding row (e.g., near the bottom). In the swap, the column index of the swapped pixels will be the same, but the corresponding pixels in two rows will be interchanged.

Exercise 0111: Write and execute a program `display-negative.c` that takes a picture with the robot’s camera, overwrites each of the component values $v$ in each Pixel with $255 - v$, and displays the result.

Exercise 1000: Write and test a C function with the signature

```
Pixel *brightestPixel(Picture *ppic)
```

that finds the Pixel with the greatest grayscale value within the Picture to which ppic points and returns a pointer to that Pixel. (If there are two or more Pixels with the same brightness, the function can return a pointer to any of them.

Exercise 1001: Using the `brightestPixel` function from the preceding exercise, write a program that uses the robot’s camera to take a Picture, identifies the brightest Pixel in it, and displays a version of the Picture in which every Pixel adjacent to the brightest one has been replaced with a bright red Pixel. (Make sure that the Pixels that you color red are actually visible and adjacent in the display, even if the brightest Pixel is at one of the edges of the Picture.

A spot is a group of nine Pixels within a Picture, in a three-by-three square. The brightness of a spot is the sum of the grayscale values of the nine Pixels.

Exercise 1010: Write a program that uses the robot’s camera to take a Picture, identifies the brightest spot in it, and displays a version of the Picture in which every Pixel adjacent to that spot has been replaced with a bright red Pixel.