Goal:

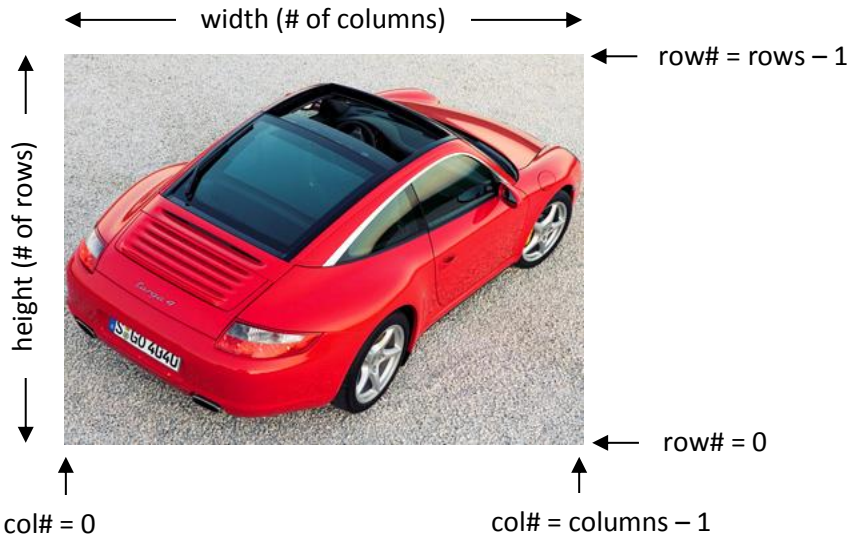
Manipulation of BMP image files:
(1) Create a "negative" by inverting the colors in an image.
(2) Create a "posterized" image using thresholding.

Objective: A first gentle refresher on C using loops and function calls.

Background: This lab uses 24-bit BMP image files. An image is a two-dimensional array of "pixels", where each pixel is represented by a triple of three 8-bit numbers (3x8=24), corresponding to the relative strengths (0-255) of the three primary colors: red, green, and blue. For example:

Black:	RGB = { 0, 0, 0}
Bright Red:	RGB = {255, 0, 0}
Bright Purple:	RGB = {255, 0, 255}
White:	RGB = {255, 255, 255}

An image has a height (# of rows) and a width (# of columns). The columns are numbered from left to right, starting at column number 0; the column on the far right is number (columns - 1). The rows are numbered from bottom to top, starting at row number 0; the top row is number (rows - 1).



Download: Download and unpack file lab1.zip from Camino. It contains a partially completed program (main1.c), a pre-compiled library file (bmp1.a) for manipulating BMP image files, and an associated include file (bmp1.h).

BMP Library: The library file (bmp1.a) provides several functions needed for simple manipulation of BMP image files:

```
IMAGE *ReadBMP24(char *filespec) ;
```

Allocates memory to hold an image and fills it from a 24-bit BMP file. Parameter *filespec* is the name of the file given as a character string. The return value is a pointer to the image in memory and must be used as an argument to all image manipulation functions.

```
void WriteBMP24(char *filespec, IMAGE *image) ;
```

Stores an image from memory into a 24-bit BMP file. Parameter *filespec* is the filename specified as a character string.

```
void FreeImage(IMAGE *image) ;
```

Releases the memory used by an image.

```
unsigned GetRows(IMAGE *image) ;
unsigned GetCols(IMAGE *image) ;
```

Returns the height (# of rows) and width (# of columns) in the image.

```
unsigned GetRed(IMAGE *image, unsigned row, unsigned col) ;
unsigned GetGrn(IMAGE *image, unsigned row, unsigned col) ;
unsigned GetBlu(IMAGE *image, unsigned row, unsigned col) ;
```

Returns the RGB color components of a pixel at a specific row and column.

```
void PutRed(IMAGE *image, unsigned row, unsigned col, unsigned red) ;
void PutGrn(IMAGE *image, unsigned row, unsigned col, unsigned grn) ;
void PutBlu(IMAGE *image, unsigned row, unsigned col, unsigned blu) ;
```

Replaces the RGB color components of a pixel at a specific row and column.

Assignment: Complete the source code for each the following three functions that are located within the provided main program (main1.c):

```
unsigned Brightness(unsigned red, unsigned grn, unsigned blu) ;
```

Returns the relative brightness of a specific RGB color as perceived by the human eye according to the formula:

$$brightness = 0.2126\ red + 0.7152\ grn + 0.0722\ blu$$

Note: The result should be rounded to the nearest integer.

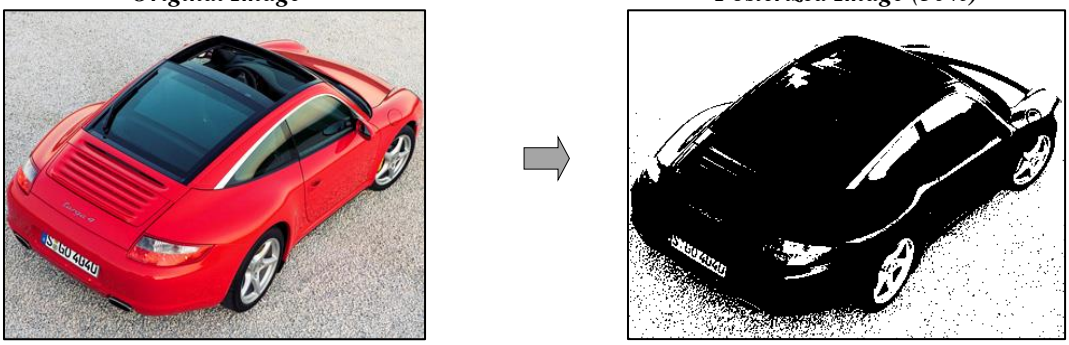
```
IMAGE  *PosterizeImage(IMAGE *image, double percent) ;
```

Replaces the color of each pixel in the image by either RGB(black) = {0,0,0} or RGB(white) = {255,255,255} and returns a pointer to the resulting image.

The parameter *percent* is a real number in the range 0 to 100. It should be converted to an unsigned integer threshold between 0 and 255; any pixel whose *perceived brightness* is above the threshold should be replaced by white, and all other pixels replaced by black.

When done correctly, the result should look similar to the example below:

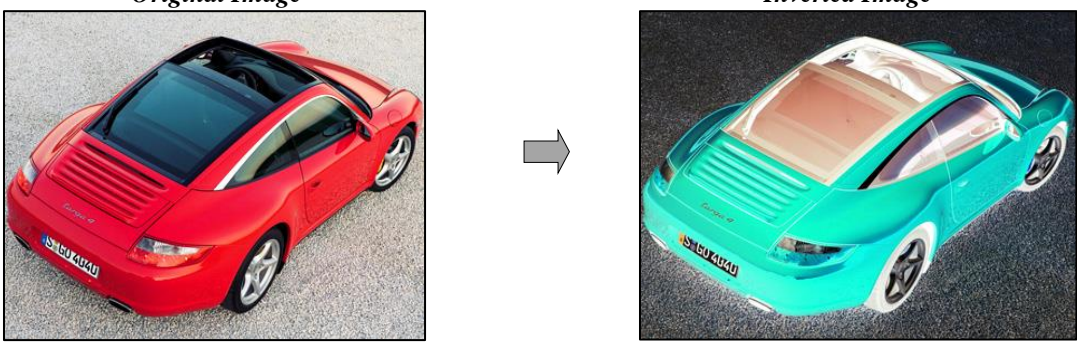
Original Image ***Posterized Image (50%)***



`IMAGE *InvertColors(IMAGE *image) ;`
Replaces the color of each pixel in the image by the inverse color and returns a pointer to the resulting image. To invert the color of a pixel, simply replace each of the RGB color component values by 255 minus the original value, as in: red = 255 – red ; grn = 255 – grn ; blu = 255 – blu ;

When done correctly, the result should look similar to the example below:

Original Image ***Inverted Image***



Compilation: Compile and link your program using the following command line:

gcc –o lab1 main1.c -L. -lbmp1

Execution: Execute your program using the following command syntax:

./lab1 src-file dst-file {option#}

When Done: Demonstrate proper operation of your program to the teaching assistant and upload the completed source code for file main1.c to the lab drop box on Camino. Do not upload any other files.