Computer Engineering Department

TED University

TED UNIVERSITY

CMPE 252 C Programming, Spring 2020
HOMEWORK 3
01.05.2020, Friday
DUE DATE: 15.05.2020, Friday, 23.59
Send your C code file via Moodle.
This is an individual work. No team work is allowed.
Similarity check will be applied to submitted codes.

A DIGITAL IMAGE CROPPING TOOL

In this homework, you are expected to exercise on your C programming skills on subjects such as:

- Binary Files
- Multi-dimensional Arrays
- Pointers, Iterating over Arrays using Pointers
- Dynamic Memory Allocation
- Structures

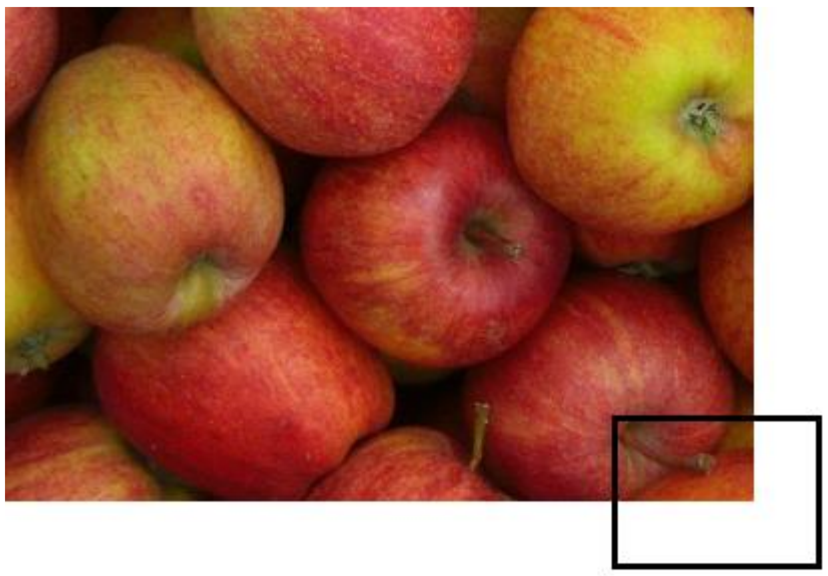
PROBLEM

You will be given an image file (a binary file) complying with a given format specification composed of a header structure and a matrix (2-dimensional) of pixel data. Your software tool will be able to:

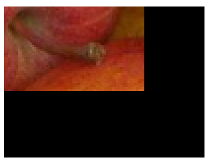
1) (100 pts) Crop* a provided rectangular region of the image and write the cropped image to an output image in the required format.
2) (☺ Bonus Feature (additional 30 pts)) Double (magnify) the cropped image in size (width and height) and write to an additional output image.

* The crop region will be provided to the tool as input arguments *(crop pixel upper/left pixel row and column id's and width/height of the region will be provided - details are in REQUIREMENT section).* If some part of the region happens to extend outside the boundaries of the image, those parts will be filled with "black" colored pixels. (See example below).

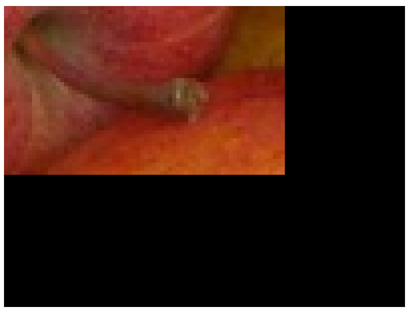
In Figure 1 an example crop region, which extends to outside the boundaries of the image is provided. When the "apples" image is given as input image, and the crop region is provided as drawn in the below Figure 1-a, then your software should generate the output crop image given in Figure 1-b and the doubled version of that crop image in size should be as given in Figure 1-c.



(a) Input "Apples" Image and the Crop Region (drawn in black)



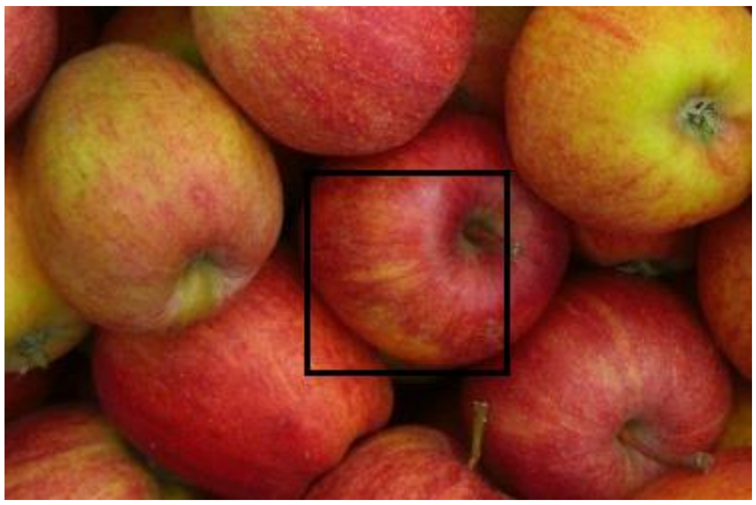
(b) Cropped Image



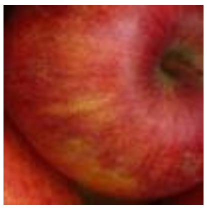
(c) Doubled Cropped Image

Figure 1 – Example depicting an input crop region that extends outside the boundaries of the image

Another example where the crop region does not extend outside the boundaries of the image is given in Figure 2.



(a) Input "Apples" Image and the Crop Region (drawn in black)


(b) Cropped Image

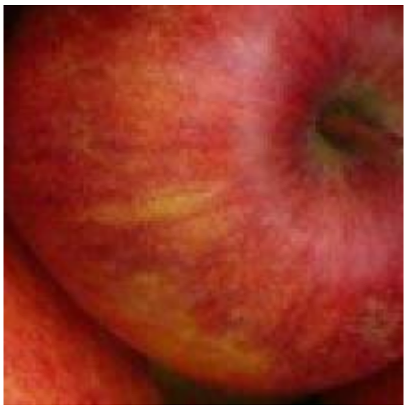
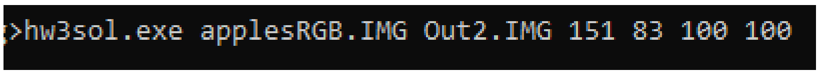

(c) Doubled Cropped Image

Figure 2 – Example depicting an input crop region that does not extend outside the boundaries of the image

REQUIREMENTS

1. **Your program shall be executed from the command line as below using 5 command line arguments:**

>program.exe *<InputImageFileName> <OutputCropImageFileName> <CropC> <CropR> <CropWidth> <CropHeight>*



```
>hw3sol.exe applesRGB.IMG Out2.IMG 151 83 100 100
```

where, each command line argument is described as below:

<InputImageFileName> : The input image file name which is a binary file in the given .IMG file format explained under Section - BACKGROUND INFORMATION

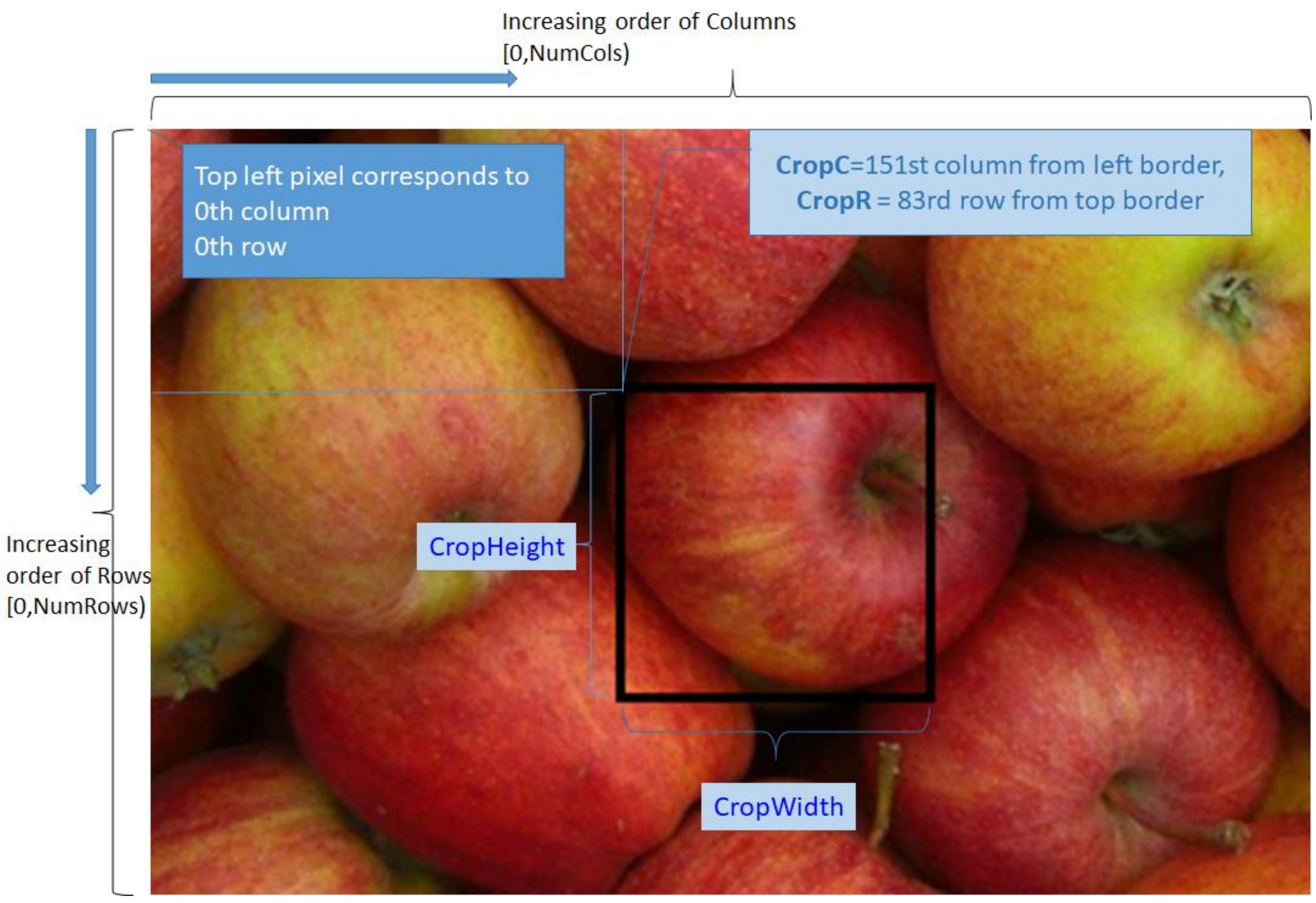
<OutputCropImageFileName> : The output file name which the cropped region image data will be written into. The file format will comply with the .IMG file format explained under Section - BACKGROUND INFORMATION

<CropC> <CropR> <CropWidth> <CropHeight> : These four arguments describe the rectangular cropping image coordinates and the width – height of the rectangular region.

For example; the below given figure depicts the meaning of CropC, CropR, CropWidth and CropHeight values that determine the cropping rectangular region for the sample execution with the below arguments:

- ➢ **`hw3sol.exe applesRGB.IMG Out2.IMG 151 83 100 100`**

!! Keep in mind that the image row and column coordinates (r,c) will always start from (0,0)th pixel and extends up to (NumberOfRows-1,NumberOfColumns-1)'th pixel for an image, where images are in fact 2 dimensional matrices (where each pixel may be composed of one or more bytes, i.e. bands or channels) (see Section-BACKGROUND INFORMATION)


Increasing order of Columns
[0,NumCols)
Increasing order of Rows
[0,NumRows)
Top left pixel corresponds to
0th column
0th row
CropC=151st column from left border,
CropR = 83rd row from top border
CropHeight
CropWidth

2. **The Doubled Cropped Image will be written to an additional output file whose file name will be determined by just concatenating "D" letter the <OutputCropImageFileName> argument.**

For example; If you call your program as below:

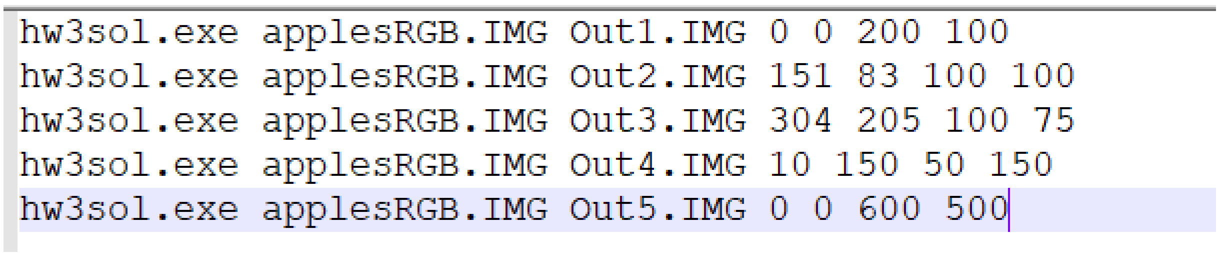
➢ `hw3sol.exe applesRGB.IMG Out2.IMG 151 83 100 100`

your executable will be able to output two files, one in "Out2.IMG" file (that includes the output cropped image) and the other is "DOut2.IMG" file which will include the Doubled Cropped Image. *(Of course, this is the applicable, if you decide to implement the bonus feature, otherwise, your executable should output only the "Out2.IMG" file)*

Refer to Section - BACKGROUND INFORMATION about how to double (magnify) an image.

3. **Along with this HW3 description, you will be given;**
   - a sample input image file ("applesRGB.IMG")
   - 5 test case output files ("Out1.IMG", "Out2.IMG", "Out3.IMG", "Out4.IMG", "Out5.IMG",)
   - 5 test case output files for Bonus Feature ("DOut1.IMG", "DOut2.IMG", "DOut3.IMG", "DOut4.IMG", "DOut5.IMG",)

which are generated for your own debugging purposes. Please make sure that your output files are "exactly the same" as these output files corresponding to below sample execution of our ground truth software so that you will be sure that your solution and the output format is correct.



```
hw3sol.exe applesRGB.IMG Out1.IMG 0 0 200 100
hw3sol.exe applesRGB.IMG Out2.IMG 151 83 100 100
hw3sol.exe applesRGB.IMG Out3.IMG 304 205 100 75
hw3sol.exe applesRGB.IMG Out4.IMG 10 150 50 150
hw3sol.exe applesRGB.IMG Out5.IMG 0 0 600 500
```

Note that the .bmp versions of all the homework data will also be provided for your manual inspection of the images.

! GRADING

For grading, your executables will be executed using ;

- **5 debug test case parameters given along with the HW description,**
- **5 more other interesting test cases that will not be provided to you, by using different input images <u>which has no size limitation requirements</u> and <u>different cropping region coordinate and size arguments.</u>**

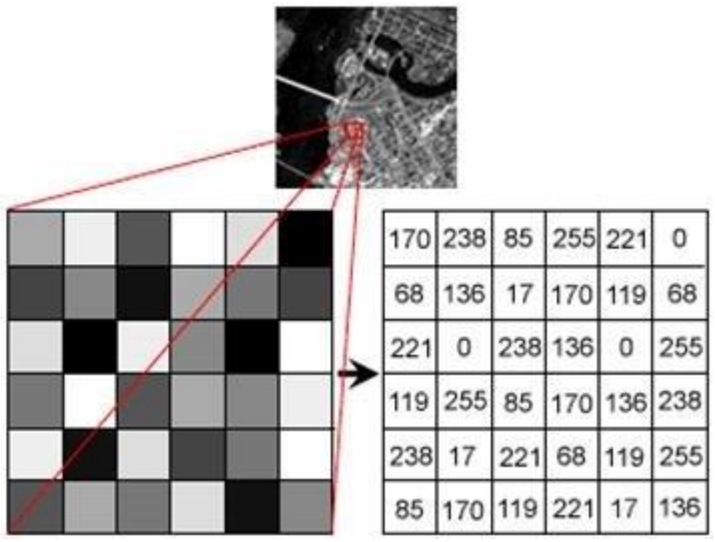
and exactly the same output with the ground truth solution will be expected from your software to get full grade.

BACKGROUND INFORMATION

WHAT IS A DIGITAL IMAGE?

A **digital image** consists of a matrix of cells, each called a picture element or **pixel**, representing the ***brightness*** of each area with a numeric value ranging from 0 to 255 in an 8-bit system ($256=2^8$).

A **grayscale image** (“siyah-beyaz görüntü” in Turkish) digital image contains a fixed number of rows and columns of pixels where each pixel have an intensity between 0 and 255, with 0 being black and 255 being white.



170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

Figure : sample grayscale image, depiction of its pixels and and pixel intensity values (Source : [2])

A **Color image** is just a simple extension of images, where pixels are composed of a vector of three values instead of one.

Using an **RGB** image as an example, the colors are constructed from a combination of Red, Green, and Blue (RGB).

Therefore, each pixel of the image has three channels (or sometimes called bands) and is represented as a 1x3 vector. Since the three colors have integer values from 0 to 255, there are a total of 256*256*256 = 16,777,216 combinations or color choices. [1]

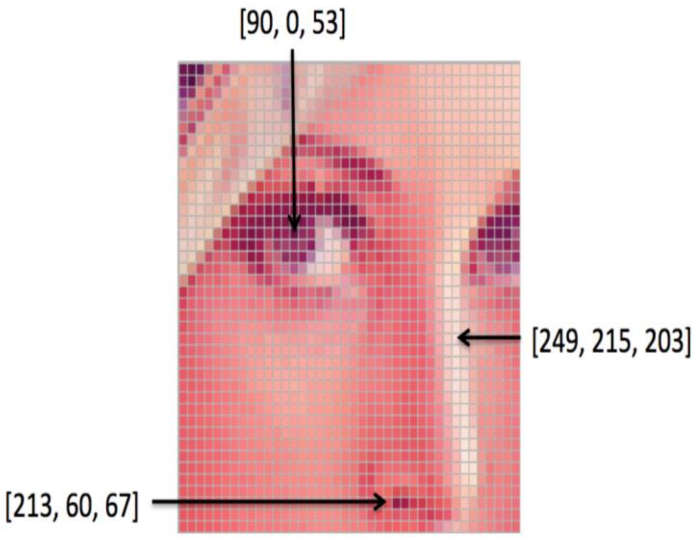


Figure : a color (RGB) image where each pixel is composed of a vector of Red, Green and Blue values (RGB) each are stored as 1 byte (8bits) in range [0,255] (Souce [1])

In computer memory and in binary image files, it is common for these RGB values to be interleaved as R-G-B. As you can see, RGB color images necessarily contain three times as much data. [3]

In colored images (0,0,0) RGB valued pixels correspond to “**Black Colored Pixels**” where (255,255,255) RGB data correspond to “**White Colored Pixels**”.

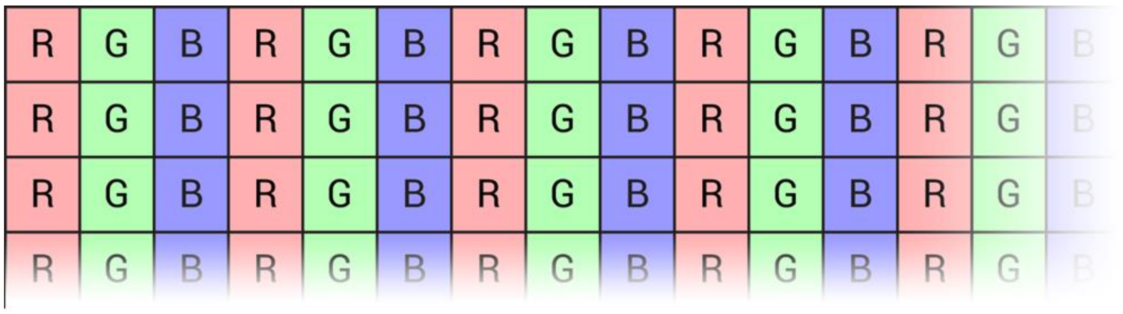


Figure:Depiction of R-G-B values in computer memory Source: [3]

[1] Introduction to Computer Vision: Tutorial 1: Image Filtering, https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

[2] "Free How to Photoshop Tutorials, Videos & Lessons to learn Photoshop training | Photoshop Course." [Online]. Available: http://www.we-r-here.com/ps/tutorials/.

[3] Image Processing and Computer Vision, By Golan Levin, https://openframeworks.cc/ofBook/chapters/image_processing_computer_vision.html

.IMG FILE FORMAT SPECIFICATION

The .IMG digital image file format is a custom file format that is created only for this homework. (Don't confuse it with the IMG file format used in the industry)

Our .IMG files are composed of a simple file header of size 14 bytes that contains information about the image properties, followed by the image pixel data array.

Summary view of an IMG file format:

Structure name	Optional	Size	Purpose
IMG FILE HEADER	No	14 bytes	To store general information about the image file
Image Pixel Data Array	No	<ImageSizeInBytes>	Contains values of pixels of the image data

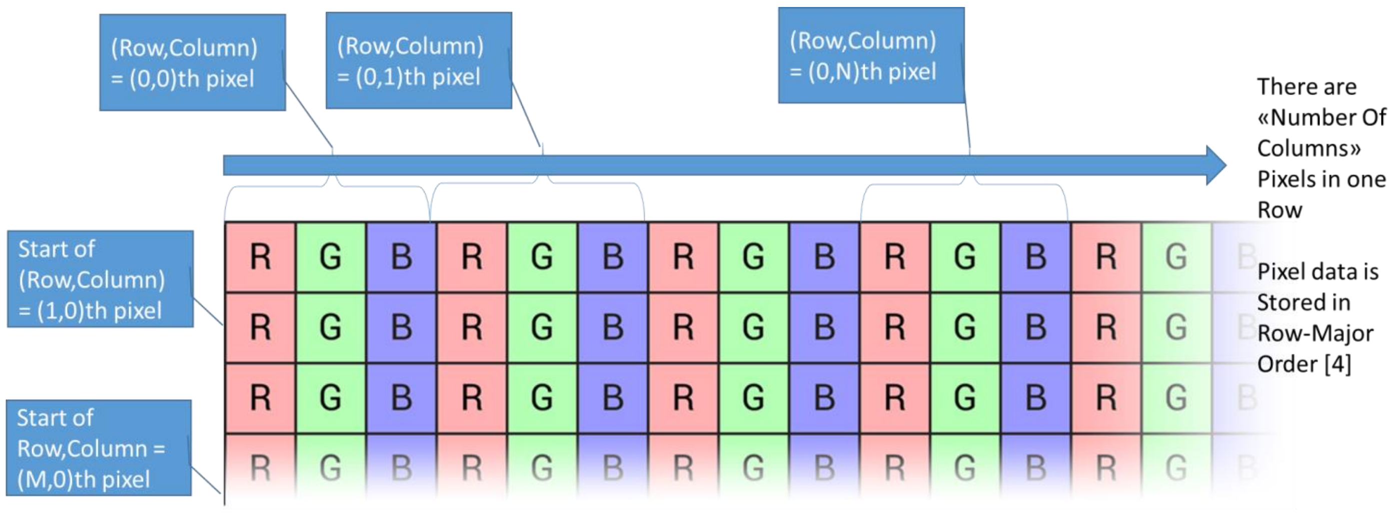
IMG FILE HEADER starts from the 0th byte offset (i.e. the beginning of) the file and is structured as given in the below table:

Field	Offset (dec)	Size	Data Type	Purpose & List of Values
ImageType String	0	4 bytes	String	One of below strings (character arrays) are stored: - 'GRY' : grayscale images - 'RGB : colored RGB images where the order of Red-Green-Blue channels are as R-G-B (R first, G next, B last) * in this homework you are only expected to work on RGB images.
Number Of Bands	4	1 byte	8-bit Integer	Provides the Number of Bands property of the image data. It should be 1 for grayscale images and 3 for RGB images.
Number Of Rows	5	4 bytes	32-bit Integer (little endian)	Number of Rows of the image pixel data array
Number Of Columns	9	4 bytes	32-bit Integer (little endian)	Number of Columns of the image pixel data array
Number Of Bits	13	1 byte	8-bit Integer	This field provides how many bits are used for representing a value of a channel (red, green or blue). This value should always be 8.

IMAGE PIXEL DATA ARRAY starts from the 14th byte of the file and recorded until the end of the file. Each pixel consists of 3 bands (or channels) for an RGB image and 1 band (channel) for a grayscale image where the values of bands are stored as given in "Number of Bits"field of the header (which is always 8-bit integers, i.e. 1 (unsigned) byte in range [0,255])

The pixels are stored in "Row Major" order, where the top left pixel of the image is stored first. Each row of the image data is stored in the file from top row to bottom row where each row of data contains all the columns of pixels of that row from left to right as depicted in the below figures.

For row major ordering of data refer to [4]


(Row,Column) = (0,0)th pixel
(Row,Column) = (0,1)th pixel
(Row,Column) = (0,N)th pixel
There are «Number Of Columns» Pixels in one Row
Start of (Row,Column) = (1,0)th pixel
Start of Row,Column = (M,0)th pixel
Pixel data is Stored in Row-Major Order [4]
R G B R G B R G B R G B R G B

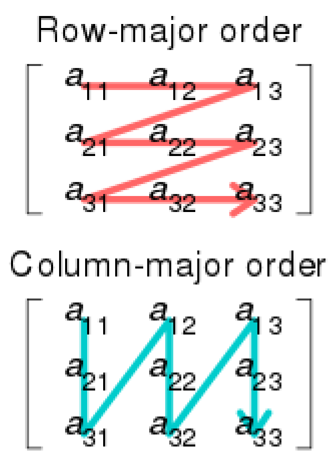

Row-major order
Column-major order
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}

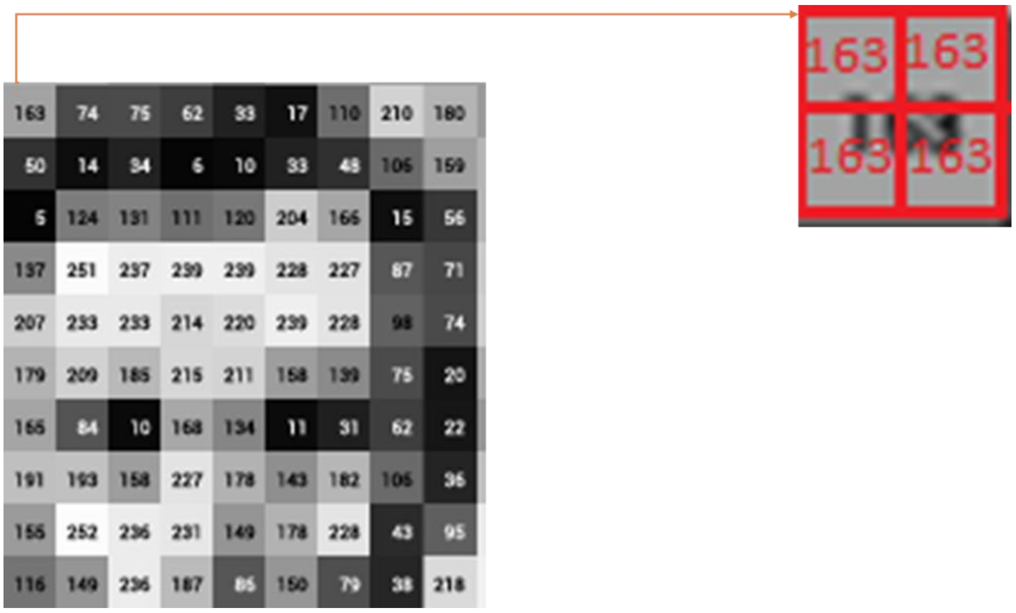
Figure – Difference between Row & Column Major Order (Source: [4])

[4] Wikipedia : Row- and column-major order https://en.wikipedia.org/wiki/Row-_and_column-major_order

DOUBLING THE IMAGE DATA:

For doubling the cropped image data, you should duplicate each pixel of the cropped image into 4 new pixels with the same values of the divided pixel as depicted below.

For example, if your input image is composed of 200x300 pixels, then this operation should generate a new image of size 400x600 pixels. (Check debug test case output files that are provided along with the homework description)



163	74	75	62	33	17	110	210	180
50	14	34	6	10	33	48	106	159
5	124	131	111	120	204	166	15	56
137	251	237	239	239	228	227	87	71
207	233	233	214	220	239	228	98	74
179	209	185	215	211	158	139	75	20
165	84	10	168	134	11	31	62	22
191	193	158	227	178	143	182	105	36
155	252	236	231	149	178	228	43	95
116	149	236	187	85	150	79	38	218

Figure – Doubling of a pixel of a grayscale image.

HINTS

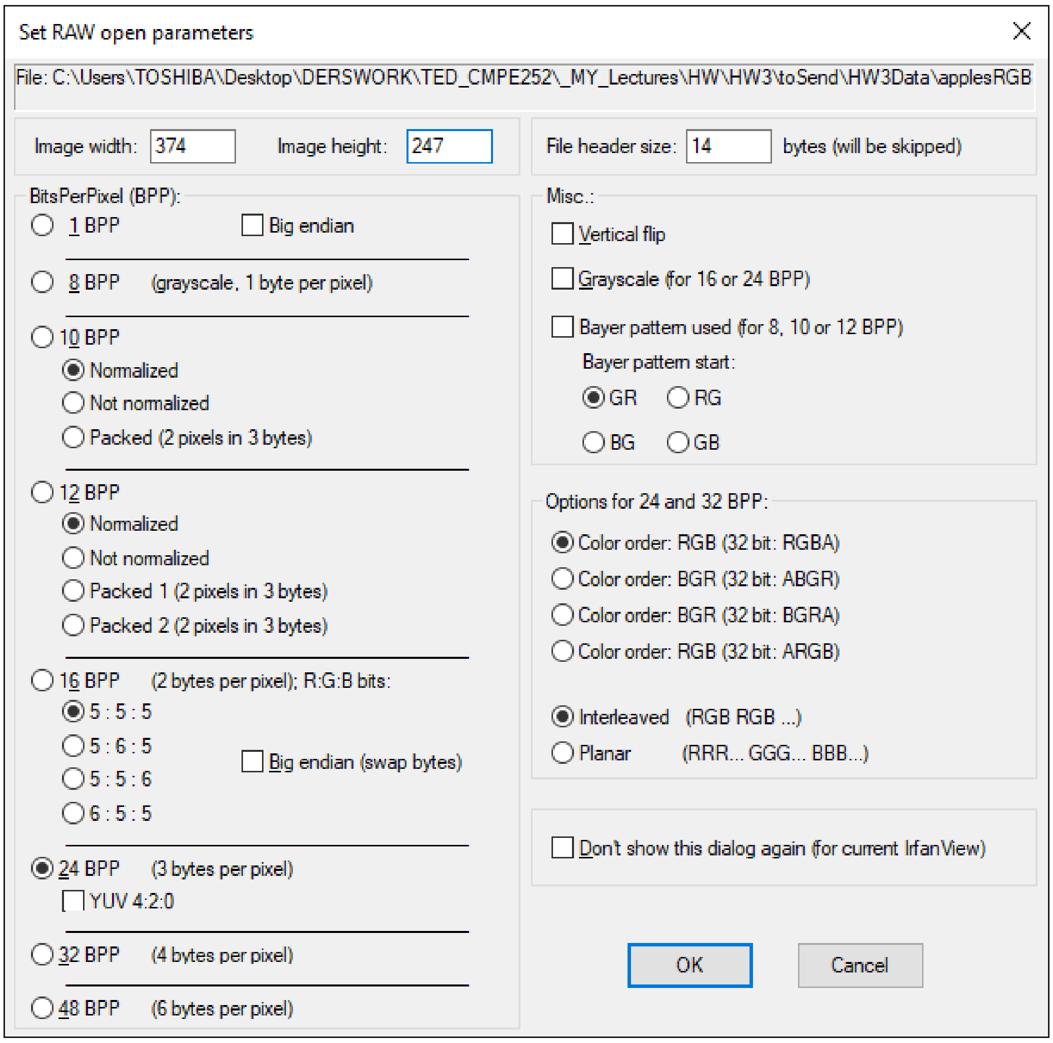
- **The test cases for grading may contain differently sized images (including large data) and crop regions, therefore you should use "dynamic memory allocation" for reading and processing over the input data and output data.**
- **For dynamically allocating multidimensional arrays and processing such array data, you have several options, which seems to be nicely described in following sources in the internet :**

  [5] https://www.geeksforgeeks.org/dynamically-allocate-2d-array-c/
  [6] https://www.tutorialspoint.com/how-to-dynamically-allocate-a-2d-array-in-c
  [7] https://stackoverflow.com/questions/2306172/malloc-a-3-dimensional-array-in-c

- **You can view your custom binary files using a Hex Editor like HxD.**

- **You can also visualise IMG image data using some freeware tools which can open RAW images. For instance Irfanview software that can be downloaded from, https://www.irfanview.com/ (download and install both the software and the plugins) , can be used to view your custom binary image files as below:**

- **Execute Irfanview software,**
- **Select from the Menu : File -> Open As -> RAW File**
- **Select your .IMG file (e.g. applesRGB.IMG)**
- **Enter "Image width", Image height", "File Header Size" (as 14 bytes) information correctly as given below, while selecting BitsPerPixel as "24 BPP" and Color Order as "RGB" as given in below screenshot:**



Set RAW open parameters

File: C:\Users\TOSHIBA\Desktop\DERSWORK\TED_CMPE252_MY_Lectures\HW\HW3\toSend\HW3Data\applesRGB

Image width: 374 Image height: 247

File header size: 14 bytes (will be skipped)

BitsPerPixel (BPP):

- ○ 1 BPP ☐ Big endian
- ○ 8 BPP (grayscale, 1 byte per pixel)
- ○ 10 BPP
  - [✓ Normalized]
  - ○ Not normalized
  - ○ Packed (2 pixels in 3 bytes)
- ○ 12 BPP
  - [✓ Normalized]
  - ○ Not normalized
  - ○ Packed 1 (2 pixels in 3 bytes)
  - ○ Packed 2 (2 pixels in 3 bytes)
- ○ 16 BPP (2 bytes per pixel); R:G:B bits:
  - [✓ 5 : 5 : 5]
  - ○ 5 : 6 : 5
  - ○ 5 : 5 : 6
  - ○ 6 : 5 : 5
  - ☐ Big endian (swap bytes)
- [✓ 24 BPP] (3 bytes per pixel)
  - ☐ YUV 4:2:0
- ○ 32 BPP (4 bytes per pixel)
- ○ 48 BPP (6 bytes per pixel)

Misc.:

- ☐ Vertical flip
- ☐ Grayscale (for 16 or 24 BPP)
- ☐ Bayer pattern used (for 8, 10 or 12 BPP)
  - Bayer pattern start:
  - [✓ GR] ○ RG
  - ○ BG ○ GB

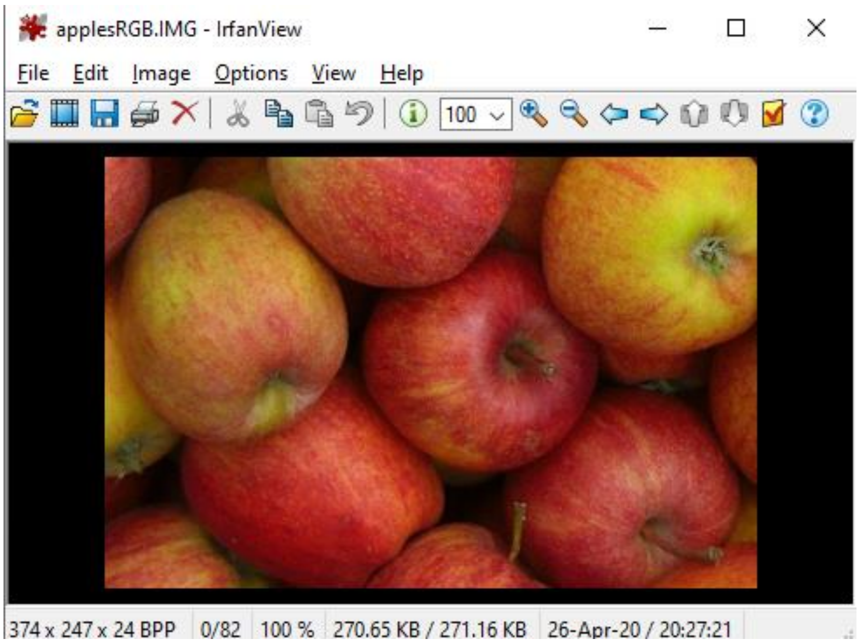
Options for 24 and 32 BPP:

- [✓ Color order: RGB (32 bit: RGBA)]
- ○ Color order: BGR (32 bit: ABGR)
- ○ Color order: BGR (32 bit: BGRA)
- ○ Color order: RGB (32 bit: ARGB)
- [✓ Interleaved (RGB RGB ...)]
- ○ Planar (RRR... GGG... BBB...)

☐ Don't show this dialog again (for current IrfanView)

OK Cancel

Then the tool will view your custom .IMG file as below:



374 x 247 x 24 BPP | 0/82 | 100 % | 270.65 KB / 271.16 KB | 26-Apr-20 / 20:27:21

Finally, you can “Save As” your file in one of the standard image file formats, for instance as a .bmp file.