CS4551 Multimedia Software Systems (Spring 2018)

Homework3 (10%) - DCT-Based Image Compression

- Due: Electronic submission via CSNS by Friday, 04/13/2018.
- What to turn in:
  - Submit ***source code only.*** You must submit all necessary files (except data files) for compile and run.
  - Do NOT submit data files.
  - You **MUST** provide a ***readme.txt*** file containing all information to help with the grading process.
- If your program produces any compile errors, you will receive 0 automatically no matter how close your program is to the solution.
- **Do not use any Java built-in image class methods, library, or tools to complete this homework.**
- You will receive penalty if you do not follow any of the given instructions.

What your program should do

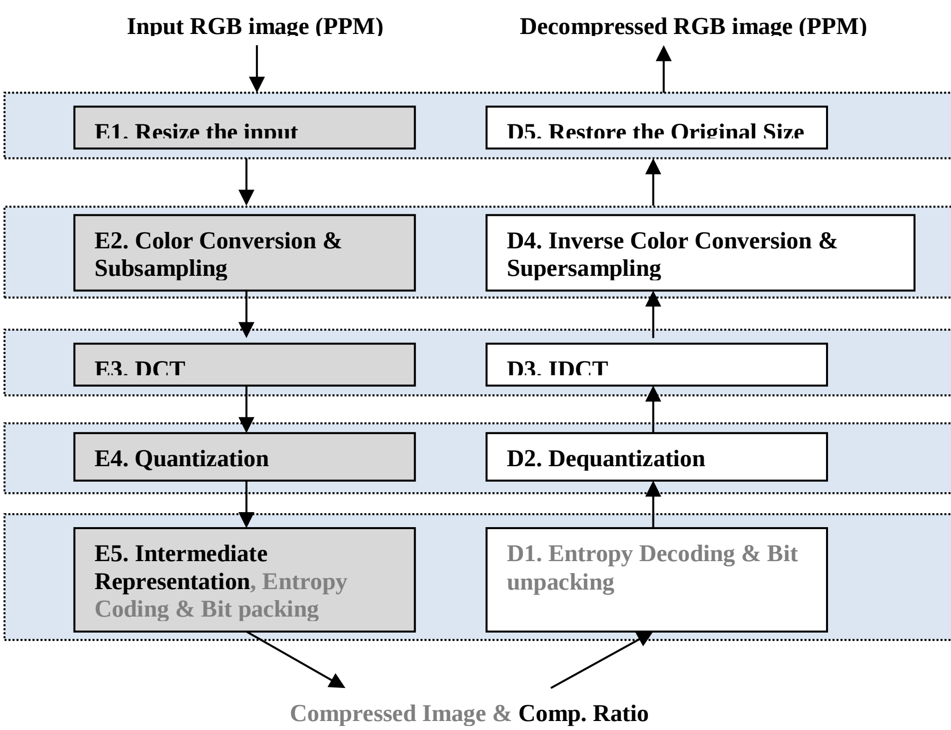
Name your main application as *CS4551_[Your_Last_Name].java*. Your program should accept one command line argument for the input PPM file name.

<eg> On Command Prompt

```
java CS4551_Doe Ducky.ppm
```

DCT-based Image Compression (100 pts + EC 20 pts) – Download SampleResults

Implement a pseudo-JPEG DCT-based compression algorithm. Notice that this algorithm is different from the standard JPEG steps. The encoder/decoder diagram is shown below. Grayed words are not required features by this homework.



<table>
<tr><th>Encoding Steps</th><th>Decoding Steps</th></tr>
<tr><td>

E1. Read and resize the input image

Read the input ppm file containing RGB pixel values for encoding. First, if the image size is not a multiple of 8 in each dimension, make (increase) it become a multiple of 8 and pad with zeros. For example, if your input image size is 21x14, make it become 24x16 and fill the extra pixels with zeros (black pixels).

</td><td>

D5. Remove Padding and Display the image

Display the decompressed image. Remember that you padded with zeros if the input image size is not multiple of 8 in both dimensions (width and height). Restore the original input image size by removing extra padded rows and columns.

</td></tr>
<tr><td>

E2. Color space transformation and Subsampling

Transform each pixel from RGB to YCbCr using the equation below:

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{bmatrix} 0.2990 & 0.5870 & 0.1140 \\ -0.1687 & -0.3313 & 0.5000 \\ 0.5000 & -0.4187 & -0.0813 \end{bmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Initially, RGB value ranges from 0 and 255. After color transformation, Y should range from 0 to 255, while Cb and Cr should range from -127.5 to 127.5. (*Truncate if necessary*.)

Subtract 128 from Y and 0.5 from Cb and Cr so that they span the same range of values [-128,127]

Subsample Cb and Cr using 4:2:0 (MPEG1) chrominance subsampling scheme. ***If Cb(Cr) is not divisible by 8, pad with zeros.***

</td><td>

D4. Inverse Color space transformation and Supersampling

Supersample Cb and Cr so that each pixel has Cb and Cr.

Add 128 to the values of the Y component and 0.5 to the values of the Cb and Cr components.

If using a color image, transform from the YCbCr space to the RGB space according to the following equation:

$$\begin{pmatrix} \mathrm{R} \\ \mathrm{G} \\ \mathrm{B} \end{pmatrix} = \begin{bmatrix} 1.0000 & 0 & 1.4020 \\ 1.0000 & -0.3441 & -0.7141 \\ 1.0000 & 1.7720 & 0 \end{bmatrix} \begin{pmatrix} \mathrm{Y} \\ \mathrm{Cb} \\ \mathrm{Cr} \end{pmatrix}$$

<u>***Common mistake:*** *After this step, you have to make sure that the resulting RGB values are in the range between 0 and 255. Truncate if necessary.*</u>

</td></tr>
<tr><td>

E3. Discrete Cosine Transform

Perform the DCT for Y image using the following steps:

- Divide the image into 8x8 blocks. Scan each block in the image in raster order (left to right, top to bottom)
- For each 8x8 block, perform the DCT transform to get the values F_{uv} from the values f_{xy}. The elements F_{uv} range from -2^{10} to 2^{10}. Check max and min and assign -2^{10} or 2^{10} for the values outside of the range so that the values range from -2^{10} to 2^{10}.

Perform the DCT for Cb image and Cr image, too.

</td><td>

D3. Inverse DCT

Perform the inverse DCT to recover the values f_{xy} from the values F_{uv} and recover Y, Cb, Cr images**.**

</td></tr>
</table>

DCT Formula

$$F_{uv} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} f_{xy} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16}$$

$C_u = 1/\sqrt{2}$ for u =0, $C_u = 1$ otherwise. $C_v = 1/\sqrt{2}$ for v =0, $C_v = 1$ otherwise. f_{xy} is the x-th row and y-th column pixel of the 8x8 image block (x and y range from 0 to 7). The element F_{uv} is DCT coefficient value in the u-th row and v-th column after DCT transformation. (u and v range from 0 to 7).

The inverse DCT Formula

$$f'_{xy} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v F'_{uv} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16}$$

E4. Quantization

Given F_{uv} in a 8x8 DCT block, quantize F_{uv} using:

Quantized(F_{uv}) = round(F_{uv} /Q_{uv}).

The intervals Δ_{uv} corresponding u and v are specified in Table 1 and Table2. The following table gives the quantization intervals for each element in the 8x8 DCT block for the luminance (Y) and chrominance (Cb and Cr).

4	**4**	**4**	**8**	**8**	**16**	**16**	**32**
4	**4**	**4**	**8**	**8**	**16**	**16**	**32**
4	**4**	**8**	**8**	**16**	**16**	**32**	**32**
8	**8**	**8**	**16**	**16**	**32**	**32**	**32**
8	**8**	**16**	**16**	**32**	**32**	**32**	**32**
16	**16**	**16**	**32**	**32**	**32**	**32**	**32**
16	**16**	**32**	**32**	**32**	**32**	**32**	**32**
32	**32**	**32**	**32**	**32**	**32**	**32**	**32**

Table 1: Luminance Y quantization table

8	**8**	**8**	**16**	**32**	**32**	**32**	**32**
8	**8**	**8**	**16**	**32**	**32**	**32**	**32**
8	**8**	**16**	**32**	**32**	**32**	**32**	**32**
16	**16**	**32**	**32**	**32**	**32**	**32**	**32**
32	**32**	**32**	**32**	**32**	**32**	**32**	**32**
32	**32**	**32**	**32**	**32**	**32**	**32**	**32**
32	**32**	**32**	**32**	**32**	**32**	**32**	**32**
32	**32**	**32**	**32**	**32**	**32**	**32**	**32**

Table 2: Chrominance (Cb and Cr) quantization table

In this homework, we want to provide a variety of compression quality options (high compression or low compression). Receive a number, n ($0 \le n \le 5$), from the user. The value n controls the quality of the compression. Use n to change Q_{uv} value such that the actual quantization is done by

Quantized(F_{uv}) = round(F_{uv} /Q'_{uv}).

$Q'_{uv} = Q_{uv} * 2^n$

D2. De-quantization

Assume that the quantization tables (basis ones) and n are available for decoding. Given the quantized value for the DCT coefficient F_{uv}, multiply it by the corresponding quantization interval Q'_{uv}.

F'_{uv} = Quantized(F_{uv}) * Q'_{uv}

Notice that the recovered F'_{uv} will be different from the original F_{uv}.

For example, if n =0, Q'_{uv} is same as Q_{uv}. If $n=1$, Q'_{uv} is double of Q_{uv} which will divide F_{uv} with bigger values and result in more compression.	

E5. Compression Ratio

Binary representation for quantized DCT coefficients and entropy coding : Compute how many bits are required to encode each 8x8 block using the following method.

Each quantized value should be represented by a binary codewords. To store any quantized coefficients, minimum (12-2-n)=(10-n) bits are required for Y and (12-3-n) = (9-n) bits for Cr/Cb because the quantized values are between -2^{10} to 2^{10} inclusive and the minimum quantization interval Q for Y is $4*2^n = 2^{2+n}$, then you need (12-2-n) bits to represent any quantized values of Y. In the same way, the minimum quantization interval for chrominance is $8*2^n = 2^{3+n}$, then you need (12-3-n) bits to represent any quantized values of chrominance. So each coefficient for Y (or Cb/Cr) will be represented by a fixed number of bits, 10-n bits (or *9*-n bits).

Enumerate ACs in zig-zag order. To achieve additional compression losslessly, the quantized DCT AC coefficients of each block are encoded using run-length encoding. The run-length coding is performed in zig-zag order. The zigzag sequence of quantized coefficients are converted into the sequence of (code, length) pairs.

For example, consider the following quantized DCT block of **Y**.

200	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Assume that we used $n=0$. We need 10 bits for DC coefficient. The AC coefficients in zigzag order are `1 1 0 0 0 0 0 0 0 0 … 0` and the corresponding sequence of (code, run-length) pairs are (1, 2), (0, 61). To encode each (code, length) pair, we need 16 bits, 10 bits for code values of ACs(of Y) and 6 bits for run-length ranging from 1 to 63. Therefore, (1, 2), (0, 61) will be represented by 16 x 2 = 32 bits. The total number bits for this Y block is 10 bits (for DC) + 32 bits (for ACs) = 42 bits. ***Notice that each block has a variable length after run-length encoding.***

Display compression ratio (to the console)

The compression ratio is computed by S/D, where S = [size of the original input image (PPM) excluding header overhead = Width*Height*24bits] and D = [total number of bits for all 8x8 blocks of Y image + total number of bits for all 8x8 blocks of Cb image + total number of bits for all 8x8 blocks of Cr image]

- E1/D5 – 15 pts
- E2/D4 – 25 pts
- E3/D3 – 30 pts
- E4/D2 – 20 pts
- E5 – 30 pts
- **Extracredit 30pts** – Implement bit packing and unpacking. It is to save/read the compressed binary stream into/from a file with a header. Define your own format for the header part and describe it in your readme.txt.

An important requirement - After each encoding step, implement the corresponding decoding step immediately and check if your output is correct or not. **<u>You will receive credits for each encoding step *if only if* you complete to implement the corresponding decoding step.</u>**