The University of Western Ontario
London, Ontario, Canada
Department of Computer Science
CS 4481b/9628b - Image Compression
Assignment 4
Due Monday April 2, 2018 at 11:55 PM

INDEPENDENT WORK is required on each assignment.

After finishing the assignment, you have to do the following:

- Type your report and convert it to *PDF format*, which must include:
    - o Answers to all questions/requirements in the assignment, if any
    - o A copy of all programs that you have written
- Prepare a soft-copy submission, including:
    - o A copy of your *typed PDF* report
    - o All programs that you wrote:
      `DPCM_encoding_function.c`, **`DPCM_encoding_function.h`**,
      `DPCM_decoding_function.c`, **`DPCM_decoding_function.h`**,
      `DPCM_encoding.c`, **`DPCM_decoding.c`**,
      `mean_absolute_error.c`, **`mean_absolute_error.h`**, and **`compare_pgm_images.c`**.
- Upload the soft-copy submission file-by-file, or as an archived directory.

Late assignments are strongly discouraged.

- 10% will be deducted from a late assignment (up to 24 hours after the due date/time)
- After 24 hours from the due date/time, late assignments will not be accepted.

N.B: When marking your assignment, your program will be tested on the GAUL network. If you will develop your program in any other platform, make sure that your program is error free and produces the required outputs when compiling and running it on the GAUL network.

In this assignment, a read and write PBM/PGM/PPM library is provided (included in the attachment as **pnm_library.tar**). Carefully read and understand the code of this library. *Do NOT change anything in the provided library. Just use it.*

128	W	W	W	W	W	W	W	W	W
N	N	N	N	N	N	N	N	N	N
N	N								N
N	N								N
N	N								N
N	N								N
N	N								N
N	N								N

1. (40 marks) Develop the following function and save it in a file named **`DPCM_encoding_function.c`**

    void Encode_Using_DPCM(char *in_PGM_filename_Ptr,
                int prediction_rule,
                float *avg_abs_error_Ptr, float *std_abs_error_Ptr)

    This function should:

    - o Read from disk *in_PGM_filename_Ptr image
    - o Encode the read image using lossless DPCM encoding, where
        - ▪ if prediction_rule = 1, use W to predict encoded pixels
        - ▪ if prediction_rule = 2, use N to predict encoded pixels
        - ▪ if prediction_rule = 3, use W/2 +N/2 to predict encoded pixels
        - ▪ if prediction_rule = 4, use CALIC initial prediction ***as described in the lecture*** (*binary* mode and *continuous-tone* mode using GAP *only*) to predict pixels to be encoded
    - o Regardless of the prediction rule that you will use, you have to consider the following:
        - ▪ the pixel at **the first row and the first column (i.e., top left corner)** is always predicted as 128
        - ▪ the rest of pixels at **the first row** are always predicted as W
        - ▪ pixels at **the second row** are always predicted as N
        - ▪ the rest of pixels at **the first two columns** are always predicted as N
        - ▪ the rest of pixels at **the last column** are always predicted as N
    - o ***Do not*** apply codeword encoding

- Write the compressed image to a file with the same name as the input file name but with the *prediction rule code* and `".DPCM"` concatenated to it. This compressed file should include:
    - The header that includes ***enough information*** to allow the decoder to decompress the compressed image.
    - An array for all prediction errors generated from the DPCM compression.
- Write the ***ABSOLUTE*** prediction error histogram data (in comma separated values format, **`.csv`**) to a file with the same name as the input file name but with the *prediction rule code* and `".errors.csv"` concatenated to it.
  This file will have lines of "***ABSOLUTE*** prediction error value, frequency" where ***ABSOLUTE*** prediction error values are recorded in an increasing order. You need to omit any line with zero frequency. Note that, ***ABSOLUTE*** prediction error values ∊ [0, MAX_GRAY_VALUE].
- Finally, the function will calculate the average and the standard deviation of the ***ABSOLUTE*** prediction errors and store them in *avg_abs_error_Ptr, *std_ abs_error_Ptr, respectively.

For example, if you call the function as follow: **`Encode_Using_DPCM("image.pgm",  2, &avg, &std)`**, the function will generate the following files: **`image.pgm.2.DPCM`** and **`image.pgm.2.errors.csv`**.

It will also store 2 numbers (the average and the standard deviation of the ***ABSOLUTE*** prediction errors) in **`avg`**, and **`std`**, respectively.

IMPORTANT: Since you were not asked to do codeword encoding, it is expected that you will not achieve compression.

---

2. (20 marks) Develop the following function and save it in a file named **`DPCM_decoding_function.c`**

void Decode_Using_DPCM (char *in_filename_Ptr)
This function should:

- Read from disk *in_filename_Ptr lossless DPCM encoded image, which is generated using the Encode_Using_DPCM function.
- Interpret the header that you saved it at the beginning of the file
- Decode all pixel values
- Write the reconstructed version of the image to a file with the same name as the input file name but with `".pgm"` concatenated to it

For example, if you call the function as follow **`Decode_Using_DPCM("image.pgm.2.DPCM")`**, the function will generate a decompressed image called **`image.pgm.2.DPCM.pgm`**.

IMPORTANT:Since you are doing a lossless compression, you have to make sure that the reconstructed version of the image is EXACTLY the same (bit-by-bit) as the original image.

---

3. (15 marks) To test your functions, write ***main programs*** and save them in files named, **`DPCM_encoding.c`** and **`DPCM_decoding.c`**, where *each program calls one of the two functions developed in Q1 and Q2 separately*. Your programs *must accept all arguments in the command-line.*

a) For the main program in **`DPCM_encoding.c`**, it will **take as an input from the command line**
  an *input pgm image name* and
  a *prediction rule code*.
  Then, it will call Encode_Using_DPCM function, which will produce:
  a DPCM compressed file (as described in **`DPCM_encoding_function.c`**)
  an ***ABSOLUTE*** prediction error histogram data file (as described in **`DPCM_encoding_function.c`**)
  The main function needs to report the averages and the standard deviations of the ***ABSOLUTE*** prediction errors, as well as the ***compression time***.

b) For the main program in **`DPCM_decoding.c`**, it will **take as an input from the command line**
   an *DPCM compressed file name*.
   Then, it will call the Decode_Using_DPCM function, which will produce:
   an DPCM decompressed file

   The main function needs to report the ***decompression time***.

c) ***You will also need to develop*** **`mean_absolute_error.c`** ***function and*** **`compare_pgm_images.c`** ***program to compare the two images.***

   The prototype of the mean_absolute_error function is defined as follow:
   float mean_absolute_error(char *file_name_1_ptr, char *file_name_2_ptr)

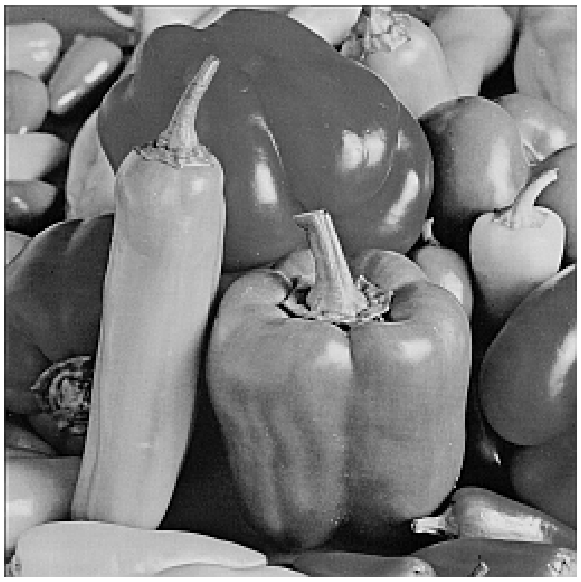
   - This function accepts two parameters:
     - file_name_1_ptr, a pointer to a string which represents a file name of a PGM image
     - file_name_2_ptr, a pointer to a string which represents a file name of a PGM image
   - The function should return a float value that represents the mean absolute error between the two provided images.
   - If the two images are not PGM of same size, the function should return -1.
   - If the max_gray_values of the two images are not the same, you need to scale the image with the smaller max_gray_value to make its max_gray_value as the larger value.

   The **`compare_pgm_images.c`** program must accept ***two*** arguments in the command-line, which are an input PGM file name number 1 and an input PGM file name number 2 and then call the mean_absolute_error function.

   You can reuse the same **`mean_absolute_error.c`** function and **`compare_pgm_images.c`** program that you developed in assignment 2 in this assignment as well.

---

4. (25 marks) Use at least the following images to test your programs:
   - Peppers.raw.pgm, a 256x256 image: ***You can download a PGM raw version of the image from the attachment list.***



- goldhill.raw.pgm, a 360x288 image: ***You can download a PGM raw version of the image the from attachment list.***

For each image mentioned above:

- Encode and decode the above images, assuming that the
  - prediction_rule number 1
  - prediction_rule number 2
  - prediction_rule number 3
  - prediction_rule number 4
- Make sure that the reconstructed image is EXACTLY the same as the original image.
- Plot *histograms* for the generated ***ABSOLUTE*** prediction error values in *semi-log scale*, i.e., plot the ***ABSOLUTE*** prediction error values against the *log(frequency)*. Note that, ***ABSOLUTE*** prediction error values ∈ [0, MAX_GRAY_VALUE].

  Generate one histogram per image per prediction_rule code, i.e., 2 images × 4 prediction_rule values × 1 histogram per prediction_rule code per image = 8 histograms in total.
- **Comment** on the shape of the offset histograms
- Report the ***average*** and the ***standard deviation*** of the data that you used to generate each histogram, i.e., report 8 histograms × 2 values (i.e., the ***average*** and the ***standard deviation***) per histogram = 16 values in total.
- Report the encoding and decoding time for each image when using various prediction_rule values, i.e., 2 images × 4 prediction_rule values per image × 2 times (one for encoding and the other for decoding) = 16 values in total. **You should do so inside your code, not by using a stop watch.**
- Present all these numbers in a **table** that is **easy to read and understand**.
- In your opinion, **which** prediction_rule is **more suitable for each image**?
  Justify your recommendation.

In this assignment, a *makefile* file is provided (included in the attachment as makefile) to facilitate the compilation and the testing processes. Carefully read and understand this makefile. *Do Not change anything in the provided makefile. Just use it.*

N.B: In your program, if you use return 1 to denote that there was an error, the makefile will not continue the rest of the testing cases. To go around this issue, you need to use return 0 instead, but you have to provide an appropriate error message before stopping the program.

To compile Q1, execute "`make  Q1`"
To compile Q2, execute "`make  Q2`"
To compile Q3a, execute "`make  Q3a`"
To compile Q3b, execute "`make  Q3b`"
To compile Q3c, execute "`make  Q3c`"
To compile all programs, execute "`make all`"

To test the input validation, execute "`make testValidation`"
To test the compression program, execute "`make testCompression`"
To test the decompression program, execute "`make testDecompression`"
To compare images, execute "`make testComparingImages`"
To perform all testing cases, execute "`make testAll`"

To delete compiled programs, execute "`make clean`"
To delete compressed images, execute "`make cleanCOMPRESSED`"
To delete decompressed images, execute "`make cleanDECOMPRESSED`"
To delete compiled programs, compressed and decompressed images, execute "`make cleanAll`"

FYI: Assignment marking scheme includes, but not limited to,

- In-line commenting
- Error free code on GAUL network (syntax)
- Correct implementation (logically)
- Efficient implementation
- Correctly acceptance of the input from the command line

- The required compressed/decompressed images
- The required comparison between decompressed image and the original image

- The neatness of caption of your figures
- The neatness of your report

If your program is not working properly, you still encouraged to submit your work for partial mark. In such case, you should include some comments explaining why your program is doing so.
