The University of Western Ontario
London, Ontario, Canada
Department of Computer Science
CS 4481b/9628b - Image Compression
Assignment 2
Due Thursday March 1, 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:
  - Answers to all questions/requirements in the assignment, if any
  - A copy of all programs that you have written
- Prepare a soft-copy submission, including:
  - A copy of your *typed PDF* report
  - All programs that you wrote: `generate_pixel_frequency.c, generate_pixel_frequency.h, generate_huffman_nodes.c, generate_huffman_nodes.h, huffman_encode_image.c, huffman_encode_image.h, store_huffman_encoded_data.c, store_huffman_encoded_data.h, read_huffman_encoded_data.c, read_huffman_encoded_data.h, huffman_decode_image.c, huffman_decode_image.h, mean_absolute_error.c, mean_absolute_error.h, pgm_huffman_encode.c, pgm_huffman_decode.c, 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 programs will be tested on the GAUL network. If you will develop your programs in any other platform, make sure that your programs are error free and produce the required outputs when compiling and running them 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.*

Before starting this assignment, you may want to review some C programming topics, including:

- Arrays
- Pointers
- Structures
- Dynamic memory allocations
- Bit operations and low-level programming
- File input/output

1. (8 Marks) Develop the following function and store it in a file named : `generate_pixel_frequency.c`

   long int *generate_pixel_frequency(struct PGM_Image *input_pgm_image, int *number_of_non_zero_values_in_the_frequency_array)

   - This function accepts two parameters:
     - input_pgm_image (a pointer to a PGM_Image structure--see pnm_library.tar)
     - number_of_non_zero_values_in_the_frequency_array (a pointer to an integer)
   - The function:
     - Dynamically allocates a long int array of (max_gray_value + 1) elements,
     - Reads ALL pixels in the provided PGM image and record the frequency of occurrence of each pixel values in the allocated array
     - Stores in the *number_of_non_zero_values_in_the_frequency_array the number of non-zero values in the frequency array
     - Returns a pointer to the allocated array

---

2. (10 Marks) Develop the following function and store it in a file named **`generate_huffman_nodes.c`**

   struct node *generate_huffman_nodes(long int *pixel_frequency, int max_gray_value, int number_of_non_zero_values_in_the_frequency_array)

   - This function accepts three parameters:
     - a pointer to a long integer array which contains the frequency of occurrence of each pixel value in an image
     - an integer representing the max_gray_value in the image
     - an integer representing the number of non-zero values in the frequency array
   - The function should utilize the input data to generate all Huffman nodes in a form of *ordered* pairs of pixel values, where a node is declared as: struct node {int first_value; int second_value;};
     The number of nodes is less by one than number_of_non_zero_values_in_the_frequency_array. The function will return a pointer to a node array which contains all Huffman nodes in order (see Huffman lecture for more details).

---

3. (10 Marks) Develop the following function and store it in a file named **`huffman_encode_image.c`**

   unsigned char *huffman_encode_image(struct PGM_Image *input_pgm_image, struct node *huffman_node, int number_of_nodes, unsigned long int *length_of_encoded_array)

   - This function accepts four parameters:
     - input_pgm_image, a pointer to a PGM_Image structure--see pnm_library.tar
     - huffman_node, a pointer to an array of struct node which contains all Huffman nodes.
     - number_of_nodes, an integer representing the number of nodes stored in *huffman_node
     - length_of_encoded_array, a pointer to a long integer which will indicate the exact length of the Huffman encoded array in bytes
   - The function should
     - generate a Huffman code for each pixel value in the provided image
     - use the generated Huffman codes to Huffman encode the provided image
     - return a pointer to an unsigned character array which will contain the Huffman encoded image after encoding it

---

4. (10 Marks) Develop the following function and store it in a file named **`store_huffman_encoded_data.c`**

   void store_huffman_encoded_data(char *compressed_file_name_ptr, int image_width, int image_height, int max_gray_value, int number_of_nodes, struct node *huffman_node, long int length_of_encoded_image, unsigned char *encoded_image)

   - This function accepts eight parameters:
     - compressed_file_name_ptr, a pointer to a string which represents the compressed file names
     - image_width, an integer representing the image width
     - image_height, an integer representing the image height
     - max_gray_value, an integer representing the maximum gray value in the image
     - number_of_nodes, an integer representing the number of nodes stored in *huffman_node
     - huffman_node, a pointer to an array of struct node which contains all Huffman nodes.
     - length_of_encoded_image, a long integer representing the exact usage of the encoded_image array in bytes
     - encoded_image, a pointer to an unsigned character array which contains the Huffman encoded image
   - The function should store in the provided file the followings:
     - image_width
     - image_height
     - max_gray_value
     - the number of Huffman nodes
     - all values in huffman_node array
     - length_of_encoded_image
     - all values in encoded_image array

---

5. (10 Marks) Develop the following function and store it in a file named **`read_huffman_encoded_data.c`**

unsigned char *read_huffman_encoded_data(char *compressed_file_name_ptr, int *image_width, int *image_height, int *max_gray_value, int *number_of_nodes, struct node **huffman_node, long int *length_of_encoded_image)

- This function accepts seven parameters:
  - compressed_file_name_ptr, a pointer to a string which represents the compressed file name
  - image_width, an integer pointer
  - image_height, an integer pointer
  - max_gray_value, an integer pointer
  - number_of_nodes, an integer pointer
  - huffman_node, a pointer of pointer to an array of struct node
  - length_of_encoded_image, a long integer pointer
- The function should read from the provided compressed file the followings:
  - image_width
  - image_height
  - max_gray_value
  - the number of Huffman nodes
  - all values in the huffman_node array
  - length_of_encoded_image
  - all values of encoded_image array
- The function should return a pointer to an unsigned character array which contains all encoded_image values.

---

6. (10 Marks) Develop the following function and store it in a file named **`huffman_decode_image.c`**

struct PGM_Image *huffman_decode_image(int image_width, int image_height, int max_gray_value, int number_of_nodes, struct node *huffman_node, long int length_of_encoded_image, unsigned char *encoded_image)

- This function accepts seven parameters:
  - image_width, an integer representing the image width
  - image_height, an integer representing the image height
  - max_gray_value, an integer representing the maximum gray value in the image
  - number_of_nodes, an integer representing the number of Huffman nodes
  - huffman_node, a pointer to an array of struct node
  - length_of_encoded_image, a long integer representing the exact usage of the encoded_image array in bytes
  - encoded_image, a pointer to an unsigned character array which contains the Huffman encoded image
- The function should
  - build a Huffman tree from the provided huffman_node
  - use the generated Huffman tree to Huffman decoded the provided compressed image (decoded_image)
  - store this decoding result as a PGM image structure
- The function should return a pointer to a PGM_Image structure which contains the uncompressed image

---

7. (8 Marks) Develop the following function and store it in a file named **`mean_absolute_error.c`**

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.

---

8. (8 Marks) Write a ***program*** and store it in a file named pgm_**`pgm_huffman_encode.c`** that calls the functions developed in Q1, Q2, Q3, and Q4, to Huffman encode an input PGM image and store the compressed image in a file.

    - Your program must accept ***two*** arguments in the command-line, which are an input PGM file name and an output compressed file name.

---

9. (8 Marks) Write a ***program*** and store it in a file named **`pgm_huffman_decode.c`** that calls the functions developed in Q5 and Q6, to Huffman decode an input compressed image and store the uncompressed PGM image in a file.

    - Your program must accept ***two*** arguments in the command-line, which are an input compressed file name and an output PGM file name.

---

10. (8 Marks) Write a ***program*** and store it in a file named **`compare_pgm_images.c`** that calls the function developed in Q7 to compare two images together.

    - Your 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.

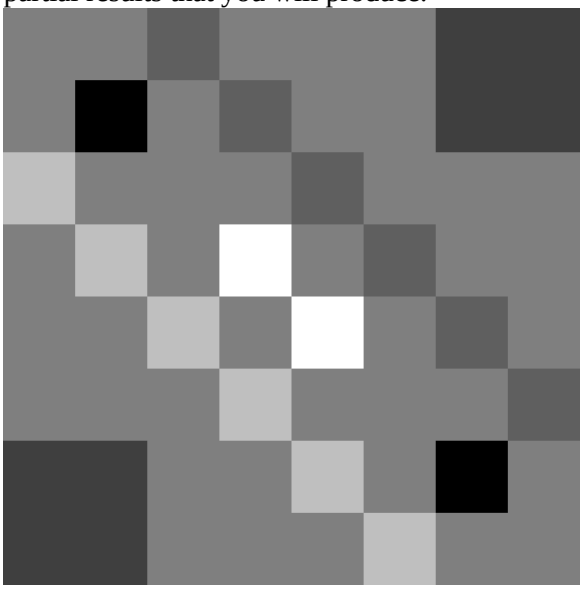
---

11. (10 Marks) Use the following images to test your programs.

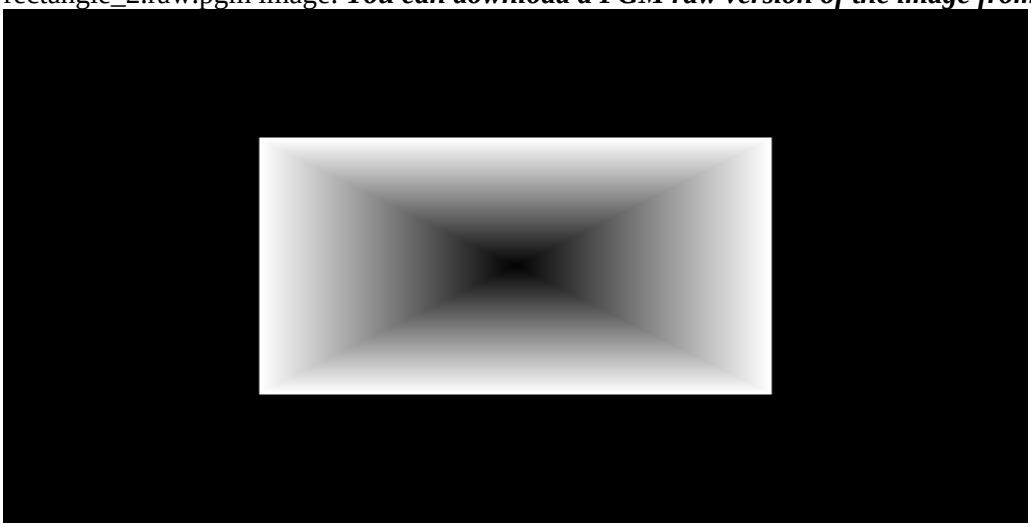
    - Make sure that the decompressed result of the compressed image is identical to the original image.
    - **You should comment on your compression result (compression performance) on each image separately.**
    - You should also comment on the size of the header of each image and how much it contributes to the average bits per pixel in the compressed image.

    Test images are:

    - test_square.raw.pgm image: ***You can download a PGM raw version of the image from the attachment list***. The probability of occurrence of the pixels in this image is as exactly as the probability of occurrence in example 1 at the Huffman encoding lecture. Hence, you can use the information provided in the lecture to verify the partial results that you will produce.

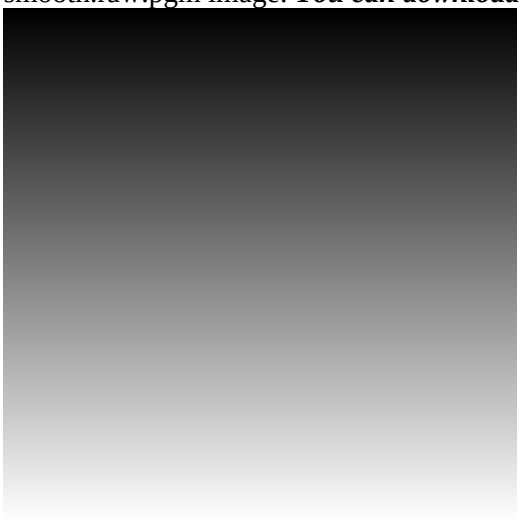


- rectangle_2.raw.pgm image: ***You can download a PGM raw version of the image from the attachment list.***



- boats.raw.pgm image: ***You can download a PGM raw version of the image from the attachment list.***

- smooth.raw.pgm image: ***You can download a PGM raw version of the image from the attachment list.***



To summarize, this assignment asks you to implement ***three*** stand-alone programs, one to Huffman encode a PGM image (Q8), one to Huffman decode a PGM image (Q9), and one to compare two PGM images (Q10). Q8 will utilize the functions developed in Q1, Q2, Q3, and Q4. Meanwhile, Q9 will utilize the functions developed in Q5 and Q6. Finally, Q10 will utilize the function developed in Q7.

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 Q3, execute "`make Q3`"
To compile Q4, execute "`make Q4`"
To compile Q5, execute "`make Q5`"
To compile Q6, execute "`make Q6`"
To compile Q7, execute "`make Q7`"
To compile Q8, execute "`make Q8`"
To compile Q9, execute "`make Q9`"
To compile Q10, execute "`make Q10`"
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 figure captions
- The neatness of the entire report
- The neatness of the written programs

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.