Data Structures: Assignment1: (Due by 9 am on Sept 22, 1999)

Goal: The goal of this assignment is to familiarize you with manipulating arrays. You will have to write C-code to read, write and manipulate images.

Coding: Your code writing task can be divided into two parts:

1. You have to implement the image abstract data type (ADT) using 1D arrays. The implementation should conform to the specifications on the next page. The Image ADT should allow you to
   (a) Read images from a file.
   (b) Save images to a file.
   (c) Set and get sizes of images.
   (d) Set and get pixel values of an image.
   (e) Indicate whether a pixel is inside an image or not.
   (f) Dynamically allocate and free memory space of an image.

2. Write an application program that interacts with the above Image ADT and the user to accomplish the following functions.
   (a) Subtract two images, \( I_1 \) and \( I_2 \) as follows:
      \[
      O(i, j) = |I_1(i, j) - I_2(i, j)|
      \]
   (b) Threshold a image based on double thresholds, \( T_1 \) and \( T_2 \) (user specified). Given a gray level image, \( I \), the function creates an output image, \( O \), such that for each pixel location \((i, j)\):
      \[
      O(i, j) = \begin{cases} 
      255 & \text{if } I(i, j) > T_1 \text{ and } I(i, j) \leq T_2 \\
      0 & \text{otherwise}
      \end{cases}
      \]
      The user should be able to specify which image to threshold. It can be either the input images or the subtracted image.
   (c) Morphological opening operation defined by the following sequence of two operations. Given a gray level image, \( I \), the function creates an output image, \( O \), such that for each pixel location \((i, j)\):
      \[
      O_1(i, j) = \begin{cases} 
      255 & \text{if ALL } m \text{ by } m \text{ neighbors of } I(i, j) == 255 \\
      0 & \text{otherwise}
      \end{cases}
      \]
      followed by
      \[
      O(i, j) = \begin{cases} 
      255 & \text{if ANY } m \text{ by } m \text{ neighbors of } O_1(i, j) == 255 \\
      0 & \text{otherwise}
      \end{cases}
      \]
Note, the parameter \( m \) is user specified. The user should also be able to specify which image to threshold. It can be either the input images or the subtracted image.

(d) Morphological closing operation defined by the following sequence of two operations. Given a gray level image, \( I \), the function creates an output image, \( O \), such that for each pixel location \( (i, j) \):

\[
O_1(i, j) = \begin{cases} 
255 & \text{if ANY } m \text{ by } m \text{ neighbors of } I(i, j) = 255 \\
0 & \text{otherwise}
\end{cases}
\]

followed by

\[
O(i, j) = \begin{cases} 
255 & \text{if ALL } m \text{ by } m \text{ neighbors of } O_1(i, j) = 255 \\
0 & \text{otherwise}
\end{cases}
\]

Note, the parameter \( m \) is user specified. The user should also be able to specify which image to threshold. It can be either the input images or the subtracted image.

(e) Save any of the images in memory viz., input images, thresholded, or the morphologically operated images.

Some points of importance

1. Note that you should have three source code files: one for the application program containing the main function (\texttt{driver.c}), one header file \texttt{image.h} that specifies the Image ADT (given to you), and a file \texttt{image.c} that actually implements the Image ADT functions. To compile all these files together, simply type:

   \texttt{gcc driver.c image.c -o driver lm -lc}

   Remember to include the \texttt{image.h} files in \texttt{driver.c}.

2. All user interaction routines and image manipulations routines such as image differencing, thresholding, and morphological operations should be implemented in \texttt{driver.c}.

3. Note that the code in \texttt{driver.c} should not be accessing the image data structure, which has been implemented as an array, directly. You should use only the functions specified in \texttt{image.h} for this purpose.
image.h

typedef struct image {
    int NRows, NCols;
    int *Pixels;
} Image;

int ReadImage (char *FileName, Image *I);
/* This function reads in an image from the PGM file specified by FileName.
The image is returned via variable I. Note that you will have to
dynamically allocate the amount of memory that will be required to
store this particular image.
**************************************************************************/

int SaveImage (char *FileName, Image *I);
/* Saves the image in I into a file specified by FileName in PGM format
**************************************************************************/

int GetPixel (Image *I, int Row, int Col)
/* Returns the pixel value at I->Pixels(Row*NCols + Col)
   Returns zero if the (Row, Col) falls outside the image
**************************************************************************/

int SetPixel (Image *I, int Row, int Col, int Value)
/* Sets the pixel value at I->Pixels(Row*NCols + Col) to Value
   Returns -1 if the (Row, Col) falls outside the image
   and return 1 otherwise
**************************************************************************/

int GetSize (Image *I, int *NRow, int *NCol);
/* Returns the total number of Rows and Columns in the Image
   The values are returned through *Nrows and *Ncol
**************************************************************************/

int Initialize (Image *I, int NRow, int NCol);
/* Allocates memory for the NRow X NCol image.
   Sets the NRow and Ncol field in the Image data structure.
**************************************************************************/

int Free (Image *I);
/* Frees up the memory used by the image I
**************************************************************************/

int InBounds (Image *I, int Row, int Col);
/* Returns 1 if the pixel (Row, Col) is inside the image
   boundary, return 0, otherwise
**************************************************************************/