Data Structures: Assignment1: (Due by 12:30pm on Jan 31, 2000)

Goal: The objectives of this assignment are the following.

- It will familiarize you with the design of a simple array based data structure.
- You will learn to follow the ADT Use Rule and the ADT Implementation Rule.
- You will learn about how the implementation details of an ADT are hidden from the application program.
- You will learn about simple image processing algorithms such as image differencing, thresholding, and the morphological operations of erosion, dialation, opening and closing.
- You will learn to document your programming experience and to describe your program.

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, InputImage_1 and InputImage_2 as follows:

$$\text{OutputImage}(i,j) = |\text{InputImage}_1(i,j) - \text{InputImage}_2(i,j)|$$

   (b) Threshold a image based on **double** thresholds, T_1 and T_2 (user specified). Given a gray level image, InputImage, the function creates an output image, OutputImage, such that for each pixel location (i,j):

$$\text{OutputImage}(i,j) = \begin{cases} 0 & \text{if InputImage}(i,j) > T_1 \text{ and InputImage}(i,j) <= T_2 \\ 255 & \text{otherwise} \end{cases}$$

   The user should be able to specify which image to threshold. It can be either the input images or the subtracted image.

(c) Morphological *Erosion* operation given by the following logic

$$\text{OutputImage}(i,j) = \begin{cases} 0 & \text{if } \textbf{ALL} \text{ neighbors of InputImage(i, j) within } m \text{ pixels} == 0 \\ 255 & \text{otherwise} \end{cases}$$

(d) Morphological *Dialation* operation given by the following logic

$$\text{OutputImage}(i,j) = \begin{cases} 0 & \text{if } \textbf{ANY} \text{ neighbors of InputImage(i, j) within } m \text{ pixels} == 0 \\ 255 & \text{otherwise} \end{cases}$$

(e) Morphological *opening* operation, which is defined as the Erosion operation followed by the Dialation operation.

(f) Morphological *closing* operation, which is defined as the Dialation operation followed by the Erosion operation.

(g) 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 (`application.c`), one header file `image.h` that specifies the Image ADT (given to you), and a file `image.c` that actually implements the Image ADT functions. To compile all these files together, simply type:

   ```
   gcc driver.c image.c -o driver lm -lc
   ```

   Remember to include the `image.h` files in `driver.c`.

2. All user interaction routines and image manipulations routines such as image differencing, thresholding, and morphological operations should be implemented in `application.c`. *There should be no user interaction from* `image.c`

3. Note that for the morphological operations 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.

4. Note that the code in `application.c` should not access the image data structure, which has been implemented as an array, directly. You should use only the functions specified in `image.h` for this purpose.

5. In your report, using example images with results, comment on the operation of each of the image processing operations that you implemented. Also, experiment with the program parameters and functions, viz. the threshold parameters, the morphological m parameter, and the morphological operations. Is there a combination that enables you to segment each moving object or person as *one* connected black blob?

```
typedef struct image {
    int NumRows, NumCols;
    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 (int Row, int Col, Image *I)
/*    Returns the pixel value at I->Pixels(Row*NCols + Col)
      Returns zero if the (Row, Col) falls outside the image
    ---------------------------------------------------------------------*/
int SetPixel (int Row, int Col, int Value, Image *I)
/*   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 (int *NRow, int *NCol, Image *I);
/*  Returns the total number of Rows and Columns in the Image
    The values are returned through *Nrows and *Ncol
    ---------------------------------------------------------------------*/
int Initialize (int NRow, int NCol, Image *I);
/*  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 (int Row, int Col, Image *I);
/*  Returns 1 if the pixel (Row, Col) is inside the image
                boundary, return 0, otherwise
    ---------------------------------------------------------------------*/
```