COP3223 Introduction to C – Programming, Section 1, Assignment #4

Due Date: Please refer to syllabus for section 1

Objectives

1. Learn how to use 2-dimensional arrays and files (of pictures)
2. Learn about how functions take arrays as arguments
3. Learn how to use functions whose type is an array
4. Learn how to manipulate the contents of an array

The arrays you will be working with represent pictures. In order to view the pictures you will work with, those of you using Windows, you need to install a free program called Irfanview.

1. Using your web browser of choice, navigate to <http://www.tucows.com/> and Click on tucowsDownloads

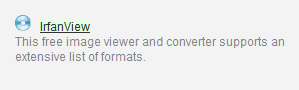
b) In the search field, enter Irfanview; click the Go button.



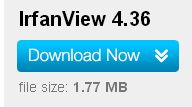
c) The first choice should be the desired program; it will have the following description:

"This free image viewer and converter supports an extensive list of formats."

Click on this entry.

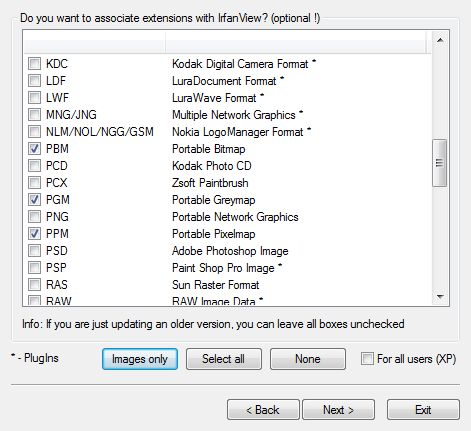


d) Click the "Download Now" button; follow the instructions provided by your web browser for the download.



e) Run the downloaded file (iview440\_setup.exe); follow all instructions.

* When the installation wizard asks what file extensions you would like Irfanview to process, make sure to select the .pbm, .pgm, and .ppm extensions.



f) In order to test your installation, download the chess1.pgm file from the class’s section 1 website and open it. A picture should appear.

**Running the provided program (assign4.c)**

Download the assign4.c, chess1.pgm, and face05.pgm files from the course page

(http://www.cs.ucf.edu/courses/cop3223/spr2019/section1)

- be sure to save them all into the same directory. assign4.c is the main file that

includes functionality for reading images; this will be explained in detail later. The other files – chess1.pgm and face05.pgm - are image files that will be processed. Open the source file in your development environment of choice, compile it, and run it. When the program prompts for an image file, enter either "chess1.pgm" or "face05.pgm". The program will then prompt for the path to which to save the processed image file. You may enter any name you wish provided you do not enter chess1.pgm or face05.pgm; a possible name is "outputchess.pgm". Once the program finishes, it'll close itself. A new picture with the output filename you provided will be created in the same directory. View the output picture and the respective input picture you entered first and try to spot the difference. This difference will be explained more rigorously later, but for simplicity we can say that the color values in each pixel have been incremented by some predetermined value. For this assignment, you will modify the assign4.c file to include new processing mechanisms. As it stands, the file provided to you has functions to read an image, store it in an array, perform the aforementioned incrementing operation on each pixel, and write the output image to a file.

**The details of the program**

To start out, we first need to understand how the .pgm image format works and how it is stored in a file. (For a more detailed description of the .pgm file format, you can look at <http://netpbm.sourceforge.net/doc/pgm.html> .) The image file starts with header information. Header information contains the image file type, in our case, “P5”, image size (width and height), and the maximum intensity value, in our case, 255.

Following the header are the details of the image pixels. Essentially, an image is nothing but a two dimensional array where each cell holds an intensity (sometimes called color or gray) value of that particular pixel in the image. For this format, the intensity value is an integer between 0 and 255, inclusive. A value of 0 indicates black, a value of 255 indicates white, and every number in between is a tone of gray.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 8 | 9 | 9 | 9 | 9 | . | . |
| 8 | 9 | 9 | 9 | 9 | . | . |
| 9 | 8 | 8 | 8 | 9 | . | . |
| 9 | 9 | 9 | 9 | 9 | . | . |
| 8 | 9 | 8 | 9 | 9 | . | . |
| . | . | . | . | . | . | . |
| . | . | . | . | . | . | . |

Intensity values of the chess1.pgm image

So, a .pgm file is simply a header followed by a list of ints (stored as chars), such as,

89999….89999….98889….99999…., etc.

When you open the assign4.c file, the first thing you should notice is the "img" variable. The two stars between the variable type and the variable name indicate to the compiler that the image is a two dimensional array that can have any size. The global variables numRows and numCols will specify how many rows and columns the image has. Following these declarations are the function prototypes; some of these, such as addtopixels, writeoutpic, and readpic, are used directly from main while others, such as readHeader, isComment, readImgID, readImgSize, readMaxVal, and readBinaryData will only be called from another function. setImage is a function that can (and will) be called both from main and other functions. To better understand these functions, we should take a closer look at the main function.

The first operation performed by the main function involves reading the name of the image. After this value has been read and stored, the readpic function is called with this filename; this function opens the file, calls the readHeader function in order to process the header information of the picture, and initially creates an empty image via the setImage function. (Aside: the setImage function’s technique of creating the empty image is interesting on its own, but beyond the scope of this class. You should think of its role as being mainly to declare an array, as in, “int somearrayname[numRows][numCols];” To make setImage practically do this, the caller, main or some other function, needs to have said “int\*\* somearrayname;”, and then say “somearrayname=setImage(); The major advantage to getting the array by this approach is that we do not need to know the values of numRows and numCols at compile time.) The function then reads in the intensity values of the provided image file and saves these values in the cells of the empty image. The image file is then closed, and the filled array is returned. We will not discuss the specifics of the readHeader and other intermediary functions, since the details behind the behaviors of these functions are beyond the scope of this class. Instead, we will focus on the functions that are called from main.

The second function that main calls is called addtopixels. This function is provided as an example of the way in which we can process the individual intensity values of the picture. addtopixels merely adds the input value - which is 103 in the example - to every element of the image array. It basically brightens the image: if you increase an intensity value, it will get closer to white. The functions you will write will be similar in nature to addtopixels - albeit a bit more complicated - so study it well. Once the image has been processed, the resulting image array is saved using the writeoutpic function. This function opens a file with the provided filename and writes to it the needed image header information. The processed image array is then written to this file after the header. Once the image has been written, the only remaining task is to free the memory previously utilized by the array; this is performed with the built-in function "free". If you can follow the order and logic behind these actions, you should be able to understand the assignment quite readily.

**The Sobel Operator Function (Read this section last)**

The sobel operator is used in image processing to detect the edges in an image; the details of this operator can be found at <http://en.wikipedia.org/wiki/Sobel_operator>. Basically, we will provide you with a function that computes the sobel operation on a picture, but we will only provide you with the function itself and an example illustrating how it can be called by main in the files sobelfunc.c and sobelcallfrag.c, respectively. Open the sobelfunc.c file and copy the topmost line (concerning math.h), add add this line to your program. This will allow your program to access C's math library. Next, add the sobel function to the bottom of your program file. sobelcallfrag.c will allow you to test this function; open this file and copy the image creation lines (the two lines following the first comment) to the segment of your code right after you create your image array. Put the rest of the sobelcallfrag.c file into your main. You can now test the sobel operator by reading in an image, applying the sobel function to it, and writing the output to an output image file, all of which are essentially provided for you in the sobelcallfrag.c file.

Once you've successfully tested the sobel function, you will be fully prepared to read images, work on image arrays, highlight the edges of a picture with the sobel operator function, and output images to files. From this point on, you are not required to, but are free to display your own talent and creativity regarding picture processing.

**Program to submit**

Start with assign4.c. Make sure it runs and produces for you the brightened picture. Then, start to modify the program as follows. Make the program prompt a menu, and read in the user’s choice.

1. Option 1 should be: read name of picture and read picture, add (could subtract by providing a negative number) brightness: the value to be added should be prompted for, and read in.
2. Option 2 should be to Subtract pictures. This operation requires two picture files (which are prompted for), and will "subtract" the second picture from the first. Before the resulting picture is outputted, ensure that the absolute value of the subtraction operation at each pixel location is computed. The subtraction option should be performed by a function whose parameters are pointers to the two pictures. You must call the setImage() function to create the array that will hold the output picture – the provided sobel code fragment illustrates the manner in which you can call setImage to do what you need to do. Then, you should call the writeoutpic function to produce the output .pgm file.
3. Option 3 should be the Sobel edge highlighter: This operation will employ a function provided to you called sobel; this technique will highlight the edges of the provided picture. For details, see the section on Sobel.

Your program should contain one function for each of the operations as per the above guidelines. Once the user has entered the desired operation and the required number of filepaths to pictures, call your function for that operation. The output of all of the three cases must be stored in a new picture array that will be outputted to three fixed files titled out1.pgm, out2.pgm and out3.pgm respectively.

A variety of functions are being provided to you; among these are the sobel edge highlighter code and functions that will read in and store pictures, initialize a two-dimensional array of integers that will serve as your picture data type, and output such an array of integers as a .pgm file. Moreover, a sample code fragment illustrating the use of the sobel function is being provided. These functions are available on the course website.

**Final Note**

If you found this assignment to be particularly enjoyable or have creative ideas for image processing that you would like to implement, please see Dr. Lobo for further directions on picture processing activities at UCF.