CS 350: COMPUTER SYSTEM CONCEPTS

PROGRAMMING ASSIGNMENT 2: C-PART
ANNOUNCED: **OCT 16, 2020**
DUE DATE: **OCT. 30, 2020, 11:59 PM (VIA ECAMPUS – Documentation, program outputs; program listings)**

MAXIMUM MARK: 100

A copy of this document and other required files for the assignment can be found via the course web page on eCampus.

1. STRING PROCESSING IN C (45 MARKS)

Exercise 8.31, pp. 375 (Deitel and Deitel, *C: How to Program*, 2016).

Additional Requirement: Your program should provide **additional capabilities**:

Please read the extra information on this sheet carefully, and consider it as you write and submit your programs.

Additional Requirements:
The name of your program file should be **analysis.c**

The results should be in the form of five tables:
Table 1a: alphabet versus occurrences
Table 1b: word lengths versus occurrences (the example table given on page 375)
Table 1c: words versus occurrences
Table 1d: words versus occurrences (sorted by words)
Table 1e: words versus occurrences (sorted by occurrence)

Note: Your program should read input from a *text file* and should be able to write output to another text file, depending on user's input. If the user does not specify any output file, the results will be displayed on the screen.

Example command line input after appropriate compilation:

```
$> analysis inText.txt outText.txt /*input from inText.txt, output to outText.txt
$> analysis inText.txt /*input from inText.txt, output to the screen
```

See Problem 2 on Image Filtering next page …

2. Image Processing in C (Image Edge Detection) (45 marks)

Note: this particular assignment will form the basis for the OS-assignment, which will involve image manipulation using many processes or threads. Thus, an understanding of (and correct solution to) this problem will significantly reduce the time you will need in the OS assignment.

Given an $\boldsymbol{m \times m}$ image **A**, the required task is to detect edges in the image. This type of operation is a standard practice in image processing and computer vision, and is a required step in tasks, such as automated object recognition by a robot or by a machine, in shape retrieval and classification, and in biometrics. It is also performed during the stage of image and video compression, for edge-based compression schemes. For simplicity, we assume that the input image is always square (but this does not have to be the case). The required edge detection is performed using a 2D *filtering mask* as follows.

Step 1: Filtering

Define a 2D mask as an $\boldsymbol{n \times n}$ matrix**.** For now we assume n is a small odd number, example n=3, 5, or 7. For each $\boldsymbol{n \times n}$ block we compute the product of the values in the mask with the corresponding position in the small block. We replace the value at the center of the small block from the image with the sum of the products. Specifically, if $I(x,y)$ is the pixel value at position (x,y) in the image I, and $E(x, y)$ is resulting edge response after the filtering using the mask $\boldsymbol{w}$**.** The edge response is obtained as:

$$E(x,y) = \sum_{p=-a}^{a} \sum_{q=-a}^{a} w(p,q) I(x+p, y+q),$$

where $a=(n-1)/2$.

The filtering is performed using a horizontal mask (w_h) and a vertical mask (w_v).
Each mask produces a component image, which are then combined to obtain the final edge response image.

$$E(x,y) = |E_h(x,y)| + |E_v(x,y)|$$

For this assignment, we perform simple Sobel edge detection, with the masks defined by the following matrices:

$$w_h = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad w_v = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}.$$

Step 2: Binary Image

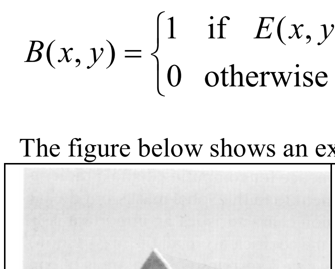
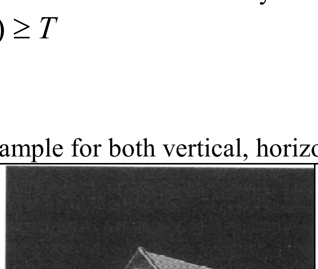
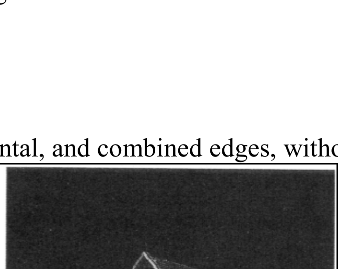
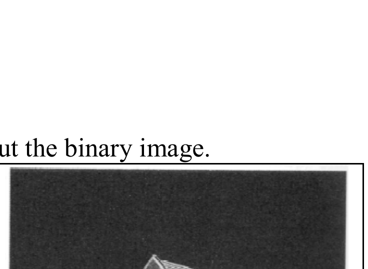
The edge response image is then binarized using a threshold to obtain a binary edge image. To determine a threshold, we compute the mean (μ) and standard deviation (σ) of the edge responses. We define the threshold simply as follows:

$$T = \mu + K\sigma,$$

where K is a constant. We set K=1 for now. The binary image is then defined as :

$$B(x,y) = \begin{cases} 1 & \text{if} \quad E(x,y) \geq T \\ 0 & \text{otherwise} \end{cases}$$

The figure below shows an example for both vertical, horizontal, and combined edges, without the binary image.

			
Original image	Edge response image (E_h) using horizontal mask (w_h)	Edge response image (E_v) using horizontal mask (w_v)	Combined edge response E from E_h and E_v.

Note: Images are from Gonzalez & Woods, *Digital Image Processing*, 2nd Edition, 2001.

Your program should produce the respective edge response images (as above), and also the corresponding binary edge images.

The main function will thus do the following:

i. Read the input image (**imageIn.ppm**)
ii. Select smaller overlapping subimages (or blocks) each of size $n \times n$
iii. Send each image block to another function, (say an image-analysis function), which then analyzes the image and returns the results.
iv. Wait for the called function to return the results
v. Arrange the returned results to form the edge response images.
vi. Compute the final respective binary images for the horizontal, vertical, and combined edge responses.
vii. Write the resulting images to disk as output files
viii. Calculate and printout the overall mean and standard deviation for the original image, and for each edge response image.

The actual task that the image-analysis function will perform is to compute the horizontal, vertical and combined edge responses using the image block passed to it by the client. That is, the function will:

i. Receive the input image block from the calling function
ii. Perform that image analysis task
iii. Send back the results to the client

Note: given a collection of numbers, $x_1, x_1, x_3, ..., x_n$ the mean and standard deviation are defined respectively as:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \, ; \qquad \sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2} \, .$$

The Input Data

The input data will be an image in `ppm` (portable pixel format) to be provided by the user. (Some sample images are provided in the project directory). Also provided are two functions, one to read images in `ppm` format, and the other to create a `ppm` image from a given data. These functions are in the image library: **iplib2New.c,** available in **imageInfo** directory. A sample program for using the image library is given in the file: **ip03mainSample.c**. You are not required to create your own program for reading or writing image files. But there will be no penalty for doing so :-). However, if you use a different image format, you will have to be able to read from and write to such formats. Also, you do not need a complete understanding of how the image read and write functions work. All you need is to understand the interface to the functions – i.e. how to pass parameters to the functions (essentially, the function prototypes). Assume that images are only gray-level images.

Example Command Line

```
ourShell$> imEdge  imageIn.ppm  outEh.ppm outEv.ppm outE.ppm  outBh.ppm outBv.ppm outB.ppm
```

PROGRAMMING STYLE (10 MARKS)

Marks will be given for good programming style, exception handling, error checking, and special considerations for improving the program beyond the stated requirement. Examples here will be handling arbitrary image dimensions, handling color images, handling image boundaries, etc. The maximum here will be **10 marks**.

Algorithm Design Phase. For each problem above, you are required to include a documentation of the algorithm you used to solve the problem, before you started to code. The algorithm will carry about **30% of the mark**, before we start to consider the programs. This is different from the 10% mark for programming style. Submit algorithm design phase as a text file, or a file in .pdf or Word format. Also, you are required to submit hard copies of your assignment in class as indicated above.

Your program should be in C, and should be able to run on a Unix/Linux platform. The program should be called **imEdge.c** .

Please submit your assignment through eCampus.

Special Section: Advanced String-Manipulation Exercises

The preceding exercises are keyed to the text and designed to test the reader's understanding of fundamental string-manipulation concepts. This section contains intermediate and advanced problems that you should find challenging yet enjoyable. They vary considerably in difficulty. Some require an hour or two of programming. Others are useful for lab assignments that might require two or three weeks of study and implementation. Some are challenging term projects.

8.31 *(Text Analysis)* The availability of computers with string-manipulation capabilities has resulted in some rather interesting approaches to analyzing the writings of great authors. Much attention has been focused on whether William Shakespeare ever lived. Some scholars find substantial evidence that Christopher Marlowe actually penned the masterpieces attributed to Shakespeare. Researchers have used computers to find similarities in the writings of these two authors. This exercise examines three methods for analyzing texts with a computer.

a) Write a program that reads several lines of text and prints a table indicating the number of occurrences of each letter of the alphabet in the text. For example, the phrase

```
To be, or not to be: that is the question:
```

contains one "a," two "b's," no "c's," and so on.

b) Write a program that reads several lines of text and prints a table indicating the number of one-letter words, two-letter words, three-letter words, and so on, appearing in the text. For example, the phrase

```
Whether 'tis nobler in the mind to suffer
```

contains

Word length	Occurrences
1	0
2	2
3	1
4	2 (including 'tis)
5	0
6	2
7	1

c) Write a program that reads several lines of text and prints a table indicating the number of occurrences of each different word in the text. The program should include the words in the table in the same order in which they appear in the text. For example, the lines

```
To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
```

contain the words `"to"` three times, `"be"` two times, `"or"` once, and so on.

8.32 *(Printing Dates in Various Formats)* Dates are commonly printed in several different formats in business correspondence. Two of the more common formats are

`07/21/2003` and `July 21, 2003`

Write a program that reads a date in the first format and prints it in the second format.

8.33 *(Check Protection)* Computers are frequently used in check-writing systems, such as payroll and accounts payable applications. Many stories circulate regarding weekly paychecks being printed (by mistake) for amounts in excess of $1 million. Weird amounts are printed by computerized