CS 350: COMPUTER SYSTEM CONCEPTS

PROGRAMMING ASSIGNMENT 2: C-PART
ANNOUNCED: MARCH 4, 2008
DUE DATE: MARCH 11, 2008 (IN CLASS, Q1 DOCUMENTATION)
MARCH 20, 2008 (IN CLASS, Q2 DOCUMENTATION ; Q1 & Q2 PROGRAM LISTINGS)
MARCH 20, 2008, 11:59 PM (ONLINE SUBMISSION)

MAXIMUM MARK: 100

A copy of this document and other required files for the assignment can be found via the course web page:
http://www.csee.wvu.edu/~adjeroh/classes/cs350/

Further, this description and copy's from relevant parts of the C book are available on the table outside my office.

1. STRING PROCESSING IN C (45 MARKS)

Exercise 8.34, pp. 347-348: (Deitel and Deitel, *C: How to Program*, 2001)

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 348)

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 Subsampling next page …

2. IMAGE SUBSAMPLING **(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 $m \times m$ image **A**, the required task is to reduce the image to a smaller image **B** of size $s \times s$. This type of reduction is usually performed during the stage of image and video compression. For instance, the MPEG compression scheme used for video provides a mode that can be used to produce this type of reduced image. For simplicity, we assume that m is a multiple of s. You can also assume that the input image is always square (but this does not have to be the case).

The required reduction is performed using the following steps.
Define $n=m/s$. For each $n \times n$ block of the image, we compute the average pixel value, and the standard deviation of the values in the block. (Overlapping blocks are not allowed). The average value then becomes the value for the corresponding ***single*** pixel position in the smaller $s \times s$ image. For a given block size, the standard deviation of the pixel values in each block gives us a rough indication of how well the reduced image will approximate the original image. The smaller the standard deviation the closer the reduced image will be to the original image.

The figure below shows an example result for a real image.





Original image	n=2	n=4	n=8	n=16	

Your program should produce a reduced version of an original image. For instance, the main function could partition the input image into a total of s^2 $n \times n$ smaller image blocks, and then call a function to compute some image statistics. The function returns the computed statistics to the parent, which then organizes the final result into a reduced-size image.

The main function will thus do the following:

i. Read the input image (**imageIn.ppm**)
ii. Partition the original image into s^2 smaller 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 reduced image
vi. Write the reduced image to a file (**imageOut.ppm**)
vii. Calculate and printout the overall image mean (i.e. mean of the individual averages), overall standard deviation for the original image, and the average standard deviation using the results returned by the image-analysis function.

The actual task that the image-analysis function will perform is to calculate two simple image statistics - the mean and standard deviation - 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, $x_1, x_1, x_3, \ldots, x_n$ 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. Assume that images are only gray-level images.

Example Command Line

boole$> imSubsample imageIn.ppm imageOut.ppm *a*

The parameter *a* is the reduction factor. That is reduced image dimension will be $\frac{m}{a} \times \frac{m}{a}$, where *m* is the original image size, *a* and *b* used to get the size of the reduced image.

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, 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 .ps, .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 platform. The program should be called **imSubsample.c**.

Please submit your assignment using the usual ***submit*** command:
http://www.csee.wvu.edu/~adjeroh/classes/cs350/assessment/submit.html