C++ Programming Sections 1782 & 1765

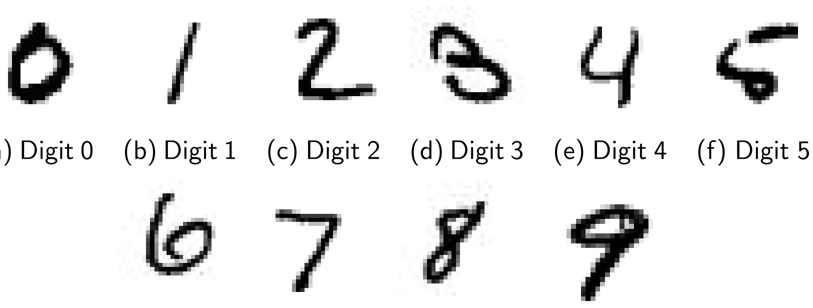
Deadline April 13, 2021 by 23:59 PM PST

Programming Assignment 3 (100 Points): Reading and Writing Images

Introduction:

The third assignment will use arrays, enums, file I/O and structs to draw to standard-out and to create an image in PPM format. In A3, we will focus our efforts on reading and writing data from and to a file. We will use ten greyscale images that represent handwritten digits from 0 to 9 (*Source: MNIST dataset*). Each image is a 28x28 greyscale image in a readable/text format called portable greymap (pgm). Figure 1 displays each image for digits 0-9 from the MNIST data set. Download the images.zip file on Canvas to complete this assignment.

The pgm format stores pixel color values in a single number that ranges from $0 - 255$, where 0 is black and 255 is white. Any other value ($0 <$ color < 255) is a shade of grey. See the section in this handout on pgm for more details about the format.



(a) Digit 0 (b) Digit 1 (c) Digit 2 (d) Digit 3 (e) Digit 4 (f) Digit 5

(g) Digit 6 (h) Digit 7 (i) Digit 8 (j) Digit 9

Figure 1: Example 28x28 images used for classification.

Reading images - a Command-Line Interface:

Create a command line menu driven application in an A03.cpp file that reads in all ten pgm images and asks a user which digit to draw to standard out. The greyscale images will be stored in a single two-dimensional array, where the first dimension is the 28x28 sized image and the second dimension is the image number. Use **ifstream** for file I/O that is defined in the **#include <fstream>** standard library.

Loading the Image Files

The images will be read into the program using a function declared as follows:

```
void readPGMImages(unsigned char images[][NUM_IMAGES], int img_size,
                   std::string filename, int digit);
```

The formal parameters: **unsigned char images[][NUM_IMAGES]** will store all ten images, **int img_size** is the image size (28x28), the **std::string filename** is the image file-name and **int digit** is the current digit being read and is used as the second dimension's index. Note that future references to an `unsigned char` will be with a `typedef unsigned char` uchar.

Command Line Interface

After reading in all 10 images, a command-line interface will allow a user to enter a digit to display. To display the digit create a function with the following prototype:

```
void draw(uchar data[][NUM_IMAGES], int img_size, int digit);
```

Where the user selected value is stored in digit. Display the digit using an asterick '*' for white (values > threshold) or a space ' ' for all other values. The threshold can be anything greater than 0. The examples in this handout use a value of 150.

The **Draw(...)** function will output an '*' for a pixel with a value ¿ 0 and three spaces “ “ for a pixel color value of 0.

An example of the user interaction for the digit 8:

```
Which MNIST handwritten digit do you want to display(0, ..., 9) ? 8

                                                            *  *
                                                            *  *
                                    *  *  *  *  *  *     *  *  *
                                    *  *  *  *  *  *  *  *  *
                                 *  *  *              *  *  *
                                 *  *              *  *  *
                                 *  *  *        *  *  *
                                 *  *  *     *  *  *
                                    *  *  *  *  *  *
                                    *  *  *  *  *
                                 *  *  *  *  *
                              *  *  *  *  *  *
                              *  *  *  *  *  *
                           *  *           *  *
                        *  *           *  *
                        *  *           *  *
                        *  *        *  *  *
                     *  *  *  *  *  *
                        *  *  *  *
                           *  *


Do you want to change the background color and font (yes or no)? yes
Enter in the background color:
64 0 128
Enter in the font color:
0 128 64
Image has been saved to digit_08.ppm
Do you want to see another digit (yes or no)?
no
goodbye
```

Colorization

The user will also be able to change the background and font color and save the image in the portable pixmap **ppm** color format. The background and font color will be entered in as a triplet of color values where each number represents the Red, Green and Blue color channel in the range $0 - 255$ for red, $0 - 255$ for green, and $0 - 255$ for blue. For example, the color red is 255 0 0, green is 0 255 0, and blue 0 0 255. See the section on ppm format in this handout for more details about RGB colors and the ppm format (Figure 2).

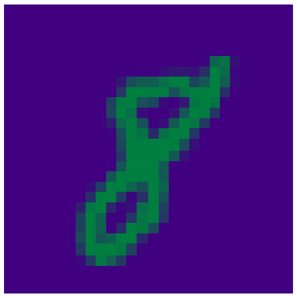


Figure 2: The 8-digit in ppm format with background and font recolored as input in the example on page 2 of this handout.

The algorithm to transform the greyscale image into a color image will be as follows:

Algorithm 1 Colorize

```
font_color ← user RGB triplet
background_color ← user RGB triplet
for all pixel_values in image do
  if pixel_value $> 0$ then
    color_pixel[0] ← font_color[0]
    color_pixel[1] ← font_color[1]
    color_pixel[2] ← font_color[2]
  else
    color_pixel[0] ← background_color[0]
    color_pixel[1] ← background_color[1]
    color_pixel[2] ← background_color[2]
  end if
end for
```

Optional Anti-Aliasing: Bluring the Edges:

Refer to Algorithm 2. We can use a simple algorithms to determing how much background and how much font should be used by normalizing the greyscale value from our pgm images. We can normalize the original number that is between 0 and 255 to a number between 0 and 1 by dividing the pgm value by 255. The result can be used as a percentage contribution **contrib** for the font color. The background color will be computed by the other percent computed as $1 -$ **contrib**. Below is a modified colorizing algorithm that incorporates the gradation for edge colors:

Algorithm 2 Optional: Colorize with anti-aliasing

```
font_color ← user RGB triplet
background_color ← user RGB triplet
for all pixel_values in image do
  for all i in (R=0,G=1,B=2) do
    if pixel_value $> 0$ then
      C ← pixel_value$/255$ {where C is the percent font contribution}
      color_pixel[i] ← (1 - C )*background_color[i] + font_color[i] $*$ C
    else
      color_pixel[i] ← background_color[i]
    end if
  end for
end for
```

File I/O: reading and writing data in pgm & ppm format:

The function **readPGMImages** will read in the ten **pgm** images and store them in a single multidimensional array where the first dimension is the color data and the second dimension is the digit number. For example, digit 0 could be stored in a $2D$ array declared as `unsigned char data[28x28][1]`.

The pgm file format (Figure 3a) is a greyscale image format that contains data about a pixel color using one color value from 0 to 255. For example, the color black is 0, white is 255, and middle-grey is 127. The header is made up of three lines. The first is a special flag **P2** that represents the greyscale format, the next line is width and height (5 5), and the third line is the maximum value for each color (255). The reminaing data are the color values in raster format in row major order. For example the first row is 5 columns,then the next row, etc.

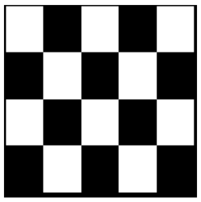
Contents of a PGM Image File (Figure 3a):

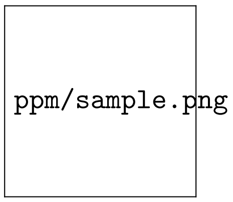
```
P2
5 4
255
255   0   255   0   255
 0   255   0   255   0
255   0   255   0   255
 0   255   0   255   0
```

The function **writeToColorPGMImages** will create an image in portable pixmap **(ppm)** color format. The full prototype is given here, where the background and font colors are stored in a Color struct:

```
struct Color{ uchar rgb[3];};
void writeToColorPGMImages(uchar images[][NUM_IMAGES], int imgsize,
                           Color bgk, Color font,std::string filename,
                           int digit);
```

The ppm file format (Figure 3b) is a text based image format that contains data about a pixel's color using three color values red (R), green (G), and blue (B). The three values (R,G,B) are used in computer graphics to represent colors. Each channel (R, G, or B) is a value from 0 to 255. For example, we represent the colors red as 255 0 0, blue as 0 255 0, green as 0 0 255, and grey as 127 127 127.


(a) Blown up 5 x 5 image. See pgm data above.



(b) Blown up 4 x 4 image. See ppm data below.

Figure 3: (a) Portable greymap (PGM) and (b) portable pixmap image formats.

PPM Image Format:

Tip: Each image is stored in row-major order. This means that when viewing the image, the top left coordinates (row,col) are (0,0) and the bottom right coordinates are (height,width). For example, the 4 x 4 image below has 16 colors. Each color is represented as three numbers. The upper left at cell 0,0 is red and is the number 255 0 0 (the first three entries in the ppm file below). The last three values in the file, 255 255 0, encode yellow and are at coordinate (4,4). We are using ASCII (text) encoding identified by the 'P3' in the file header ('P6' is binary). # are comments, Width Height, then maximum color value. Write 70 characters or less for each line in the ppm file.

Contents of the PPM Image File shown in Fig.3b:

```
P3
4 4
255
255 0     0     127 63 0      63  127  0     0   255  0
127 0    63     63  0  0      0   63   0     63  127  0
63  0   127     0   0  63     63  63   0     127 127  0
0   0   255     63  63 127    127 127  63    255 255  0
```

To view pgm/ppm files there is a free program called irfanview to load and convert image formats. Irfanview can convert ppm from ascii to binary and to png, jpg, etc. Use irfanview or a viewer of your choice. xCode displays pgm/ppm natively.

Where to do the assignment

You can do this assignment on your own computer, or using the SMC Virtual labs with Citrix. In either case, ensure the code compiles and runs on Windows. Submit one **.cpp** file named **A03.cpp**. Do not use any other name or else points will be deducted.

Submitting the Assignment

Include your name, your student id, the assignment number, the submission date, and a program description in comments at the top of your files, and use the CS52 Programming Guide for coding style guidance. Submit the assignment on Canvas (`https://online.smc.edu`) by **uploading your .cpp file** to the Assignment 3 entry as an attachment. Do not cut-and-paste your program into a text window. Do not hand in a screenshot of your program's output. Do not hand in a text file containing the output of your program.

Saving your work

Save your work often on a flash-drive or to the cloud (e.g., GoogleDrive, Microsoft OneDrive, Canvas, etc.). Always save a personal copy of your files (e.g. .cpp, .h, etc.). Do not store files on the virtual lab computers.

References:

- Image information ppm and ppgm http://davis.lbl.gov/Manuals/NETPBM/doc/ppm.html
- Raw MNIST Datasets:Binary Format MNIST