Overview

This homework processes graphical image files. The skeleton provided to you reads and writes the files. Your task is to write several filters to modify the image content while it is in memory.

This homework is due online before 9:30 AM Friday, April 23. Turn in the source file filterPGM.s and the log file recording a final test run as described at the end of this writeup.

Program Structure

The structure of the program is this:

data segment

- miscellaneous text strings
- filter procedure address table

text segment (code)

main	interpret command line arguments
filter0, filter1, ...	modify the image buffer as requested
minPixel, maxPixel	find min and max pixel values
putPGM	write a pgm formatted file
getPGM	read a pgm formatted file
skipToWhitespace	advance over a string token
skipOverWhitespace	advance to a string token
encodeInt	translate from integer to string
parseInt	translate from string to integer

The only procedures that you will edit are the filter functions and minPixel and maxPixel. The other procedures can be used without modification.

Data Format

This program reads an input source image file, modifies it, then writes a new output destination image file. The file format that it reads is called binary pgm (Portable Gray Map), a simple format for describing gray scale (monochrome) images. You don't need to know the details of this format in order to do this homework, but you may find it interesting to see how simple it is and how it can be used to easily convert numeric data into images. There is a link in the references section that describes the pgm format.

The procedures getPGM and putPGM are used by main to read and write the files. The procedures skipToWhitespace, skipOverWhitespace, encodeInt, and parseInt are support procedures used by getPGM and putPGM.

The getPGM procedure reads a file into memory and builds a small header block to describe the image. It also reads all the pixel values and stores them in a large integer valued array (the image buffer).

The header block contains the number of columns and rows in the image. It also contains an entry giving the maximum allowable value in the image (ie, the brightest possible pixel value). The header block contains an entry for the address of the image buffer. Assuming that $a0 points to the image header block, the elements of the header are formatted as follows:

`0($a0)`	column count
`4($a0)`	row count
`8($a0)`	max value
`12($a0)`	image buffer address

The image buffer contains one integer value per pixel. These are positive 32-bit integers. The number of entries is (column count) * (row count). The buffer starts with the entry for the pixel in the upper left corner of the image, then continues with the data for the rest of the first row. Following that is the data for the second row from left to right, and so on through all the rows of the image.

Running the program

This program uses the parameters passed to it with the SPIM "go" command. The format of the command is:

<srcfile.pgm> <dstfile.pgm> <filter> <filter> ...

<srcfile.pgm>	Name of an existing binary format Portable Gray Map file
<dstfile.pgm>	Name of the output file. An existing file will be overwritten.
<filter>	Number of a filter procedure, eg, 0 or 1 or ... As many filters as desired can be specified.

Console output from a sample run might look like this.

```
Author: Doug Johnson
gradient-b.pgm out.pgm 0 1 0 2 0 3 0 4 0 5 0
(col x row) : (15 x 39), max allowed value: 255
(min,max) : (0,190)
(min,max) : (65,255)
(min,max) : (55,245)
(min,max) : (65,255)
(min,max) : (64,240)
(min,max) : (0,255)

Run is complete.
```

Important notes

The image buffers take up a lot of room in memory. The default memory allocation for SPIM is not large enough to hold these images. You need to modify the command line that is used to start SPIM to give it more memory.

On my Windows machines, I did this by going to the shortcut that I use to start SPIM and opening the Properties dialog. That dialog has an entry labeled "Target" containing the path to pcspim.exe. I added the -ldata switch to the command line so the Target field now contains:

```
C:\apps\PCSpim\pcspim.exe -ldata 9000000
```

If you don't do this, SPIM will print an error message when you try to read large image files. As far as I know, this same switch should work for all versions of SPIM.

Be sure to reload the program file between each run of your program using the menu item Simulator -> Reload filterPGM.s. SPIM does not reinitialize between runs unless you do this, and the program will not work correctly.

Requirements

There is a total of 30 points for this homework, with an additional 2 pts extra credit available. The point allocations are described below.

You should read over the main procedure to understand the flow of control and the contents of the image header block. The main procedure prints out values from the header block, so you can use it as an example of how to retrieve these values.

Change the author string to your name instead of mine before you do any editing.

Your task is to write several procedures to process the images. Each of these procedures takes the address of an image header as its only argument in $a0. None of the filters return a value, they just analyze or modify the contents of the image buffer. In brief, the filters provide the following capabilities.

filter0	stats (print to console)
filter1	invert image
filter2	darker by 10
filter3	brighter by 10
filter4	posterize
filter5	stretch gray range

Several pgm files are supplied in the homework zip file for you to experiment with. The image viewer program Irfanview can be used to display these files, and also to convert other image files to the binary pgm format.

The specific capabilities of each filter are as follows.

0. (10 pts) minPixel, maxPixel, and filter0 - statistics

This procedure prints out statistics describing the pixel data in the image. Your implementation is required to print out the minimum and maximum values found in the image.

As with all the filter procedures, this filter takes the address of an image header as its only argument in $a0. It does not return a value.

You should write two support procedures minPixel and maxPixel to actually do the work of finding these values, and then call them from filter0 and print the results. This structure means that filter0 is a non-leaf procedure and must create and manage a simple stack frame.

The procedures minPixel and maxPixel can easily be implemented as leaf procedures with no stack requirements. Each of these procedures takes the address of the image header block in $a0, and returns the requested pixel value (either the min or the max) in $v0. Both procedures loop over all the pixels in the image buffer to find the requested minimum or maximum. Spend some time coding this looping process carefully - you will be able to use similar (or identical) code in the other filter procedures.

You can store the null-terminated strings needed for the filter0 printout near the filter0 instructions by using the .data assembler directive, followed by the strings, followed by the .text assembler directive, followed by the executable instructions.

Specifying filter 0 on the command line between calls to the other filters is an easy way to check that your filters are performing as expected since you can see how the min and max values are changed after each filter operates.

1. (5 pts) filter1 - invert image

This procedure inverts the image (creates a negative of the image). For a gray scale image, this is done by subtracting the current value of the pixel from the maximum allowed value and storing the result. The maximum allowed value is provided in the image header.

As with all the filter procedures, this filter takes the address of an image header as its only argument in $a0. It does not return a value.

As an example of how this filter operates, consider the following. If the maximum allowed value is 255 (typical for an 8-bit image), and a particular pixel has the value 1 (very close to black), then the inverted value would be 255-1 = 254 (very close to white). Similarly, if the original pixel had a value of 128 (middle gray) then the inverted value would be 255-128 = 127 (also middle gray).

The filter1 procedure loops over all the pixels in the image buffer, retrieves the existing value of the pixel, inverts it using the maximum allowed value, and stores it back in the same location.

2. (5 pts) filter2 - darker

This procedure darkens an image by subtracting 10 from every pixel value. If the result of the subtraction is less than zero, then the pixel value is set to zero. This is known as clamping the value at 0.

As with all the filter procedures, this filter takes the address of an image header as its only argument in $a0. It does not return a value.

As an example of how this filter operates, consider the following. If a particular pixel has the value 15, then the darkened value would be 15-10 = 5. If the filter was applied again, then for this pixel the procedure would calculate 5-10 = -5, and would set the new value to 0.

3. (5 pts) filter3 - brighter

This procedure brightens an image by adding 10 to every pixel value. If the result of the addition is greater than the maximum allowed value, then the pixel value is set to the maximum allow value, thus clamping the value at the maximum.

As with all the filter procedures, this filter takes the address of an image header as its only argument in $a0. It does not return a value.

The operation of this filter is nearly identical to the operation of filter2, but it uses a different delta value and clamps at the other end of the range. A reasonable design would be to write a separate filterBrightness procedure that took an image header and a delta value, and then just call that from filter2 and filter3 with the appropriate values. You are not required to implement a separate procedure, this is just a comment for your consideration.

4. (5 pts) filter4 - posterize

This procedure modifies an image by zeroing out the four low order bits of each pixel value using a mask word and the "and" instruction. Zeroing out the low order bits of a pixel means that several different shades of gray are all mapped to the same value that has zeroes in the low order bits. This reduces the number of colors in the image and is sometimes useful for simplifying an image.

As with all the filter procedures, this filter takes the address of an image header as its only argument in $a0. It does not return a value.

The operation of this filter is similar to those already described. It loops over all the pixels in the image and modifies the existing value. In this case, it uses the "and" instruction to combine the existing value with a mask value of 0xFFFFFFF0 and stores the result in the pixel location.

5. (Extra credit 2 pts) filter5 - stretch image range

This procedure modifies an image by recalculating all the pixel values so that they cover the entire range of possible values. This is useful when the image data is all at one end of the range and the contrast between levels is too small to perceive. Stretching the range makes the differences more visible.

As with all the filter procedures, this filter takes the address of an image header as its only argument in $a0. It does not return a value.

As with the previously described filters, this procedure loops over all the pixels in the image and modifies the existing value. However, the modification requires a little more calculation. The desired result is that the minimum value in the original image is mapped to zero, the maximum value in the original image is mapped to the maximum allowable value, and all the pixels in between are scaled in a linear fashion.

Thus each pixel value is replaced using the formula

$pixel_{new}$ = (($pixel_{old}$-min) * (max allowed value)) / (max - min)

Turn in

When you have completed the implementation, run your program using a command line that exercises all the filters that you implemented on a sample file. Save the log file after the run.

Turn in your source file filterPGM.s and the log file on line. The link is on the class calendar page. If you build any particularly interesting image files to illustrate the operation of your filters, you can put them in a zip file and turn them in too.

References

netpbm file formats

http://netpbm.sourceforge.net/
http://netpbm.sourceforge.net/doc/pgm.html

I used the file gradient.xls and Excel to build a simple ascii pgm file using Save As [tab separated text], then converted the resulting gradient.pgm to binary pgm using Irfanview. This is an example of how you might convert numeric data from any application to an image for further analysis.