Lab 13: 2-D arrays and image filters

Goals

Understand techniques for initializing elements of structure variables. Demonstrate proficiency in: (1) reading values into a 3 x 3 array and (2) using the array as an image smoothing filter.

Background

Initializing structure variables

- We can create an unintialized instance of an image_t with the declaration:

```
image_t image;
```

- We can initialize the structure by providing a list of initializers *in the proper order* as shown.

```
image_t image =
{
   "",            // unknown filename
   "P6",          // color ppm
   640,           // width
   480,           // height
   NULL           // address of pixels
};
```

- We can also selectively initialize elements in any order by enumerating the elements we wish to initialize

```
image_t image =
{
   pixels: pixmap
   width:  3,
   height: 2,
};
```

- An entire pixmap can be statically constructed as follows: (Each pixel definition should be enclosed in { } to enhance readability.

```
pixel_t pixmap[6] =

{
   {33, 44, 51}, {31, 45, 11}, {23, 34, 41},
   {33, 44, 51}, {32, 46, 21}, {23, 34, 41},
};
```

Two dimensional arrays in C

Two dimensional arrays are declared as follows:

```
double  x[4][5];
```

The declaration above creates 4 x 5 = 20 double precision values. Specific elements are accessed as follows:

```
x[2][3] = sqrt(10.0);
```

As with single dimension arrays the legal values of the first subscript are 0, 1, 2, 3 and the legal values of the second are 0, 1, 2, 3, 4

The values are stored in the following order:

```
x[0][0],  x[0][1], ... x[0][4],  x[1][0], .., x[3][4]
```

This is consistent with subscripting techniques commonly used in mathematics in which the first subscript commonly represents a *row* and the second represents a *column.*

The name *x[i]* represents a pointer to the *ith* row of the array.

To read a value into an element at row i and column j use:

```
howmany = fscanf(stdin, “%lf”, &x[i][j]);
```

Applying an image filters

- An image filter is an *n x n* array where *n* is an odd number. For our purposes *n* = 3.

- To apply the filter to a specific pixel of an image, envision superimposing the filter over the image with the center of the filter located above the specific pixel.

- Then compute the red, green, and blue products of the corresponding elements of the filter with the red, green, and blue components of the pixel the filter element overlays.

- The output value of the filter is a pixel whose red, green, and blue components are the sums of the 9 red, green, and blue products.

- For example suppose double filter[3][3] is applied to a pixel at location row = 4, col = 6. Then

```
rsum =   filter[0][0] * pixels[3][5].r +
         filter[0][1] * pixels[3][6].r +
         filter[0][2] * pixels[3][7].r +
         filter[1][0] * pixels[4][5].r +
         filter[1][1] * pixels[4][6].r +
         filter[1][2] * pixels[4][7].r +
         filter[2][0] * pixels[4][5].r +
         filter[2][1] * pixels[4][6].r +
         filter[2][2] * pixels[4][7].r;
```

- In your implementation the pixmap is physically a ONE DIMENSIONAL ARRAY. Therefore pixels[i][j] must actually be specified as image->pixels[ndx] where ndx is computed in the usual way: ndx = target_row * image->width + target_col.

Assignment:

Program lab13.c in the lab13 subdirectory should be used as a starting point for this assignment. Sample input and output are lab13.in and lab13.out.

Specific tasks include:

1. Complete the *read_filter()* function which should read in the 3 x 3 filter from the standard input. AFTER this is working correctly,

2. Uncomment the three lines in the *main()* function that are presently disabled and then,

3. Complete the *apply_filter()* function. For full credit you must use the nested for loop structure shown.

Turn In Work

Show your TA that you completed the assignment. Then turn in your lab13.c program using the command:

sendlab.101.*section_number* 13 lab13.c