ECE 1305, Fall, 2015

Lab 4: Arrays

You'll need to use arrays for much of the rest of the semester, so this lab is intended to make sure you are comfortable working with arrays. To use arrays, you need to be comfortable with the following:

1. declare an array
2. declare an array and initialize it at the same time
3. access and modify individual elements inside an array
4. use loops to work through arrays

It would also be handy to know what sort of things arrays are useful for:

A. keeping track of a set of (column of) numbers
B. keeping track of two related columns of numbers
C. using an array as a look-up table
D. using an array as a set of counters

With those goals in mind, here are some exercises to help you get there.

Background:

Trig functions: C++ has a rich set of trig and other math functions available, all by #including <cmath> . For example, you can calculate the cosine of x with the following:

```
double x=0.5, y;
y = cos(x);
cout << "cosine of " << x << " is "<<y<<endl;
```

Note that the C++ trig functions assume the arguments are in radians, not in degrees. Similarly, the inverse functions (e.g., arc-sine, or asin(...)) return angles in radians.

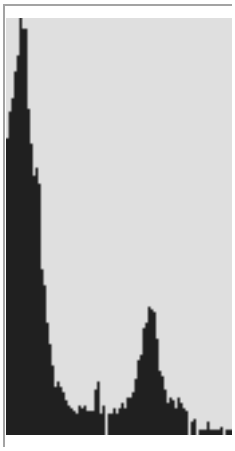
As an aside, here's an interesting way to get the value of PI as accurately as possible:

- recall that 360 degrees is 2 PI radians, and 180 degrees is PI radians;
- cosine(180 degrees), or the cosine(PI radians) is -1.0.
- Since the trig functions accept or return their results in radians, simply take the arc-cosine(-1.0).
- here's the code: `const double PI = acos(-1.0);`

Histograms:

A histogram is a table showing the frequency of occurrences of values in a data set. A histogram could be used to create a bar chart, for example, showing which values were the most prevalent in the data. Histogramming digital images is useful, because you can use the histogram to do a very effective contrast stretch to enhance the image. Here is an image of a locomotive (the one at the Ranching Heritage Museum right near campus), and here is the same image after histogram-equalization (a statistical process based on the histogram of the image, where the brightness values are redistributed based on how frequently they occur.) Here is a histogram (presented as a bar chart) of the brightness values in the image.

locomotive



histogram of brig

Array Basics (1-4 above)

Sample code to help you:

Study the following programs to see how to use arrays in various contexts. These will help you in the exercises for this lab.

1. a program to declare an array of strings (with room for 3 strings), prompt the user for (3) words and output them to the screen one by one
2. like above, but using a loop and loop counter to index the array
3. a program to demonstrate declaring and initializing arrays at the same time.
4. using 2 arrays to hold a set of related X & Y values. This program declares two arrays (x & y, such as you might use for a graph) with room for 100 values in each, uses a loop to assign them values, and outputs them to a file. (You can read the output file from this with excel and plot them if you wish.) The file is a

comma-separated values (.csv) file, a common way to make data easily readable by spreadsheets and other programs.

5. Reading an arbitrary number of data points, and using an array as a set of counters. This sample program reads integer data from a file called data.txt and counts how many times each value occurs. Data.txt can have any number of data points. Numbers should be between 0 and 29, inclusive.

Part 1: Basic use of arrays: declaring, accessing.

The first two sample programs (arrays_1.cpp, arrays_2.cpp) above should help you on this part.

1.a. Write a program which declares an array of 6 integers, then prompt the user for 6 values, reads them in from the keyboard, and stores them in an array. Then using a loop (different from the 1st one, assuming you used a loop to read in the numbers) write them back to the console. Compile, test and run your program.

1.b. Add another loop to the program in 1.a. to calculate the sum of the 6 values, then output the sum (integer) and average (floating point). Again, compile, test and run your program.

1.c. Just to see if you understand how to loop through the data in an array, add loops to the program in 1.b. to output the values in the array forwards and backwards.

At this point, assuming the user gave you the numbers 10 20 30 40 50 60, your output might look something like:

```
Your data was:  10 20 30 40 50 60
 sum, average are:  210 35.0
 forwards:  10 20 30 40 50 60
 backwards: 60 50 40 30  20 10
```

You only need to turn in part c. However, do it incrementally, as suggested above, so you only have a limited number of things to stumble over while figuring out how it all works.

Name your program six_numbers.cpp

Part II. Initializing arrays when declaring them:

The third sample program above will help with this part of the lab.

2.a. Write a program which declares an array of 12 doubles and initializes them to:
10.0 3.0 8.0 6.0 2.0 11.0 14.0 13.0 10.0 0.0 12.3 14.2
(without prompting for and reading them in.) Output the values of the array to make sure they are right. The first 3 sample programs above should help get you there.

2.b. Write a program to declare and initialize an array of 12 strings and initialize them to the months of the year. Add code to output them to the console.

2.c. Add an array for the number of days in each month (ignore leap years), and have your program print the month, the number of days in the month, and the number of days in the year preceding the first day of the month. For example:

```
Month      days days
                preceding

_________  __   __
January    31    0
February   28   31
March      31   59
...
```

2.d. Using a calculated value as an index into a lookup table: Prompt the user for a day of the year. Given the day of the year, print the day's month and day-of-month. (Again, you can assume the year is not a leap year, or, if you like the challenge, prompt for the year as well, and have it handle leap years.)
Hint: Modulus Operator!. If you're stumped on how to do that, click here.

You do not need to turn in your program from part 2.a. However, doing it will make 2.b – d easier.

Name your program jday.cpp

Part III. Columns of related data.

First: lay out the program on paper first before you start, so you're clear of the steps it needs to go through. The fourth sample program above should help you with this part of the lab.

Like the sample program, calculate a range of x-values, then calculate a set of y, z and w values based on the x values.

- Calculate X from –PI to PI in steps of 0.1 .
- calculate y=cos(x) for each value of x,
- calculate z=sin(x) for each value of x.
- calculate w=.37 cos(x) + .92 sin(x)

Output results to a file "sines.txt". You can verify your results fairly easily by hand-checking a few of the values with a calculator (be sure to check beginning and end of your data), then plot the entire sequence with excel or matlab to see if looks correct. (You do not need to turn in a plot of your data…that's to help you verify that you got it right.)

Name your program sine_cosine.cpp

Part IV: counting values.

The fifth sample program above should help you with this part of the lab. Here's data.txt for use with the sample program.

Use an array to count how many pixels are zero, how many are 1, how many are 2, etc., in a PGM image. Output a histogram table of the image. Prompt the user for the name of the PGM file to histogram. The fifth sample program does almost everything you need for this. You just need to figure out how to read (and make use of) the header info of the PGM, instead of reading exactly 30 values from a file. You can assume that all PGM images I give you have pixel values between 0 & 255 inclusive. Your table should have several columns of counts (so the user doesn't need to scroll through 250+ lines of output), but it may be helpful to start with a single column, till you get the counting part right. Then go back and get the table looking pretty.

Then add code so find some image statistics: the min, max, mean and median (middle value).

Finally, add code so summarize the histogram in groups of 10: print a table showing how many pixels are less than 10; between 10 and 19; 20 & 29, etc.

You can find the min, max & sum while reading the file initially, or you can reread the image in a second loop, or you can use a loop to go through your histogram to get most of the statistics. To find the median, the easiest way will be by summing up the histogram until you reach half the number of pixels.

You can test this with a small PGM file, like your initials from last week's lab. Careful: Make sure your counts are right. If you don't write your loop correctly, you'll either end up with an infinite loop, or count the last pixel twice by mistake when you hit the end of the file.

After that's working, Try it out on houset.pgm or peppers.pgm.

Note 0: Do NOT cut & paste from the sample program. Study it to understand how it works, then set it aside and write your own from scratch. If you get stuck, go back and study the sample program again, but write yours yourself. You won't get credit if we

find the same variable names we've used, or the same comments in the code, etc.. (You'll learn much more this way, I promise…)

Note 1: you do not need to store the image in an array for this assignment, but you will need an array of counters.

Silly Question: If the maxval for the image is 255, how big should the array be? (Hint: if you said 255, then you forgot something... er, I mean "nothing")

Name your program histogram.cpp

Challenge for the Week:

Read a PGM image, store it in an array, and output it to a new file (with a different name!). Once you get that working, see if you can get any of the following working:

- Create a photographic negative of it
    - (hint: if a pixel with value 32 is dark gray, and a pixel with a value of maxval (usually 255) is white, what gray level is a pixel of maxval – 32 ?)
- Create a thumbnail (max # rows or columns: 64)
- And here's a cool one: subtract each pixel from the one right before it. (For column 0, set it to zero.) Then add ½ maxval. Do it, look at it, and figure out why it looks the way it does. BTW, this is essentially doing a horizontal high-pass filter of the image. What features are left, and what features disappear?

Note 1: because we're not doing dynamic memory allocation yet, you'll need to declare an array large enough to hold "any" image. Assume a max size of 300 x 300 pixels, and simply output an error message if the user gives you an image larger than that. Easiest to have 2 arrays, one for reading into, and one to put the result in.

Note 2: You can do this with a 1-dimensional array; just keep reading pixels one after another in a loop from 1 to ncols * nrows.

Note 3: VERY IMPORTANT: For very obscure reasons, you may need to put the following at the top of your program. You'll know you need it if your program crashes when it reaches the statement which declares your array(s):

```
#pragma comment(linker, "/STACK:16777216")
```

(Why? This increases the stack size to 16 MB. Default stack size on MS C++ is 1 MB, and an integer array of 512x512 will overflow the stack. I told you it was obscure… BTW, dynamic memory allocation and pointers is the right way to do this, but we're not there yet.)

First one to send me a program that accomplishes any of the bullets above wins. (But you should do the required part of the lab first.)

Deliverables:

Turn in your programs for parts 1.c., 2.d, 3, and 4. (And the challenge, for extra credit, if you do it.)

You should be turning in:

- six_numbers.cpp
- jdays.cpp
- sine_cosine.cpp
- histogram.cpp
- (and the optional challenge, if you do it)