APS 105 — Computer Fundamentals

Lab #5: Functions and Arrays
Fall 2013

Important note: The course material necessary for completing this lab will be found in your lecture notes and in the Carter text up to the end of Chapter 6.
Another important note: For this lab, all `double` type variables should be printed with the `%.2f` conversion specifier. You are not required to round the values – just use the conversion specifier.

This is a 1.5-week lab. You must use the `submitaps105f` command to electronically submit your solution by 11:59pm on Wednesday, October 30, 2013.

Objective

In this lab, you will use functions and arrays to build a number of tools that are broadly useful in many areas of engineering: 1) using computers to do integration, 2) creating a histogram, and 3) approximating a transcendental function using a Taylor series with a finite number of terms.

Numerical Integration

You are learning about *differentiation* in your MAT196 calculus course, and you will soon learn about *integration*. *Integration* is the "opposite" of differentiation – it allows us to find the "area" under a function $f(x)$ within an interval (window) of x. For example, you are likely aware that, given the distance travelled as a function of time, one can differentiate to find velocity at a given time. In an analogous way, one can integrate velocity over a time window to find the distance travelled in that window.

The definite integral of a function $f(x)$ with limits from low to $high$ can be approximated by the following equation:

$$\int_{low}^{high} f(x)\mathrm{d}x \approx \frac{(high - low)}{n} \cdot [\frac{f(low)}{2} + \frac{f(high)}{2} + \sum_{i=1}^{n-1} f(low + i \cdot \frac{(high - low)}{n})] \qquad (1)$$

where n is an integer representing the number of subintervals. The intuition behind the above equation is that the domain from low to $high$ is divided into subintervals of width $(high - low)/n$. The function $f(x)$ is evaluated at each subinterval, as well as at the limits, low and $high$. The computed function values are then multiplied by the subinterval width to approximate the "area" of $f(x)$ within the limits. Using this approach, computers can approximate the integral of *any* function. You can Google "numerical integration" and check out the Wikipedia page to learn more about it.

For this part of the lab, you will write a *complete* C program that computes the definite integral for *any* fourth-order polynomial having the following form:

$$f(x) = a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e \qquad (2)$$

within the limits from low to $high$ for any number of subintervals.

Note that the limits and coefficients must be `double` type variables. If you already know how to integrate, you can use hand analysis to verify your program for simple-to-compute integrals (e.g. $\int x^2$). Below is a sample execution of the program that integrates $f(x) = x$ within the limits 0 and 1. Your output must conform to this format.

```
Enter the coefficients (a b c d e): 0 0 0 1.0 0
Enter the integral limits (low high): 0 1.0
Enter the number of subintervals (n): 10
The value of the integral is: 0.50
```

Here is another example execution of the program that integrates $f(x) = x^2$ within the limits 1 and 4:

```
Enter the coefficients (a b c d e): 0 0 1.0 0 0
Enter the integral limits (low high): 1.0 4.0
Enter the number of subintervals (n): 20
The value of the integral is: 21.01
```

Suggestion: First write a C function that computes the value of the polynomial $f(x)$ for a particular x given the coefficients provided by the user. Test this function and make sure it is working before moving on (i.e. call it from `main` and compare the results with hand/calculator analysis). Next, write a C function that implements the numerical integration equation above that *calls* your first function. To debug this, use a small number of subintervals, and print out each term of the summation above, comparing again with hand/calculator analysis.

Write your program in a file called `Lab5Part1.c`.

Histogram Generation

In the course of analyzing data, it is often useful to assess the *distribution* of the data, meaning, the number of data points that fall within certain ranges or "bins". A histogram is a graphical representation of the distribution of data. In this part, you will write a *complete* C program that takes in the data to be put into a histogram and the parameters of the histogram, and outputs a histogram in textual form.

The program should work as follows: it first prompts the user for the limits `low` and `high` (representing the limits of the domain of interest), and then prompts the user for the number of bins, `N`. `low` and `high` are `double` type variables; `N` is an `int` type variable. Then, the program repeatedly prompts the user for data points, which must be `double` type values. When the user enters a data point less than `low` or a data point greater than or equal to `high` then the data entry terminates and the program prints the histogram values for the data entered.

How to compute the histogram? The range from `low` to `high` should be divided into `N` equal-width bins. Your program should count the number of data points in each bin. Your program must correctly support the scenario where `low` and `high` are negative. Your program may assume that `low` will be less than `high`. When your program prints the histogram, it should print the count value for each bin the following format:

```
...
<centre of bin i> <count for bin i>
<centre of bin i+1> <count for bin i+1>
...
```

You may write your program with the built-in assumption that N will not exceed 100. That is, no *tester* or *marker* cases will require more than 100 bins.

Below is a sample execution of the program.

```
Enter the limits (low high): -5.0 5.0
Enter N: 10
Enter the values.
-4.5
4
3.0
4.9
4.75
2
-2
-2.0
5.1
The histogram values are:
-4.50 1
-3.50 0
-2.50 0
-1.50 2
-0.50 0
0.50 0
1.50 0
2.50 1
3.50 1
4.50 3
```

The bins in the example above are [-5:-4), [-4:-3), [-3:-2), and so on.

Write your program in a file called `Lab5Part2.c`.

Aside (this is not mandatory): you can copy and paste the histogram values into a separate file called `data.input` (i.e. the lines of the output after the message `The histogram values are:`). On an ECF computer, you can plot the histogram using `gnuplot`, which is a handy Linux data-plotting tool. Type `gnuplot` at the command line of an ECF computer. Then, when you see the command prompt within `gnuplot`, type:

```
set ylabel 'Count'
plot 'data.input' with boxes
```

You should see a plot like that shown in Figure 1.

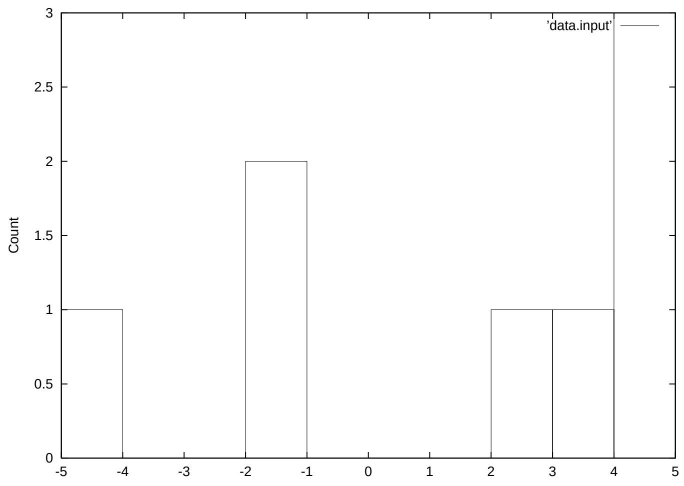

Figure 1: Plot of histogram with `gnuplot`.

Taylor Series

Transcendental functions (e.g. $sine$, $cosine$, e^x, etc.) can be expressed as infinite series, called Taylor series. You will learn more about Taylor series in your upcoming math courses. By summing a finite number of terms in the series, computers can approximate the values of such functions. In this part of the lab, we will implement a C function that uses a Taylor series to approximate the value of the $sine$ function. Check out the Wikipedia page on Taylor series to find the specific series for your favourite function.
The Taylor series for $sin(x)$ is:

$$sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots \tag{3}$$

Write a C function called `approxSine` that accepts a `double` type variable as a first parameter called `x`, and an `int` type variable as a second parameter called `N`. The function should compute an approximate value for $sin(x)$ using the first `N` terms of the Taylor series above. The prototype of the function is:

```
double approxSine(double x, int N);
```

Write a `main` function that uses the `approxSine` function to print an approximation of $sin(x)$ for different values of x as follows: The program first asks the user for an `int` type variable `N` representing the number of terms of the Talyor series to use. Then, the program asks the user for two `double` type variables `low` and `high` representing the minimum and maximum values of x at which to evaluate $sin(x)$, respectively. Finally, the program asks the user for a `double` type variable `step` representing the *step size* of x in the range from `low` to `high`, defining the values of x at which $sin(x)$ should be printed. For example, if `low` is 1.0 and `high` is 3.1 and `step` is 0.5, the the values of $sin(x)$ should be printed for $x = 1, x = 1.5, x = 2, x = 2.5$, and $x = 3$. Below is a sample execution of the program. Your implementation should conform to this output.

```
Enter N: 20
Enter low: 0
Enter high: 3.01
Enter step: 0.1
```

```
0.00 0.00
0.10 0.10
0.20 0.20
0.30 0.30
0.40 0.39
0.50 0.48
0.60 0.56
0.70 0.64
0.80 0.72
0.90 0.78
1.00 0.84
1.10 0.89
1.20 0.93
1.30 0.96
1.40 0.99
1.50 1.00
1.60 1.00
1.70 0.99
1.80 0.97
1.90 0.95
2.00 0.91
2.10 0.86
2.20 0.81
2.30 0.75
2.40 0.68
2.50 0.60
2.60 0.52
2.70 0.43
2.80 0.33
2.90 0.24
3.00 0.14
```

Each line of output above contains a value for x and $sin(x)$. You may wish to plot your output data using Excel or gnuplot to observe whether it corresponds to a $sine$ wave. Your program may assume that `N` will be at least 1.

Write your program in a file called `Lab5Part3.c`.

Hint: Compare successive terms of the Taylor series. What you will see is that each term need not be computed from scratch, but rather, may be computed by multiplying it's predecessor by a specific value.

Marking

This lab will be marked out of 10. Part 1 is worth 4 marks; Parts 2 and 3 are worth 3 marks each. A marking program will be used to automatically mark your lab. The marking program will use the last version of the lab files you submitted using the `submitaps105f` command.

Full marks are given if your program works correctly, fewer if not, and zero if it cannot be compiled. Late submissions or submissions with an incorrect filename will result in a mark of 0 for the entire lab. The deadline will be strictly enforced, so avoid last minute submissions.

You can run a testing program, called `tester`, yourself to test the correctness of your solution. At the command line in your ECF account, run:

```
/share/copy/aps105f/lab5/tester
```

in the same directory as your solution file. The testing program will use a number of test cases to test your solution, and report success if your solution produces output that is identical to the expected output. Some of these test cases will be used by the marking program as well, but the marking program will also be using other test cases that are not included in the testing program to test the correctness of your program. This implies that even though you do not have access to the marking program, you will obtain at least partial marks if all of the test cases in the testing program report success with your solution.

Your mark for this lab will contribute 3% to your mark in the course.

What To Submit

When you have completed the lab, use the command

```
submitaps105f 5 Lab5Part1.c Lab5Part2.c Lab5Part3.c
```

to submit your file. Make sure you name your file exactly as stated (including lower/upper case letters). Failure to do so will result in a mark of 0 being assigned. You may check the status of your submission using the command

```
submitaps105f -l 5
```

where `-l` is a hyphen followed by the letter *ell*. You can also download a copy of your submission by running the command:

```
/share/copy/aps105f/lab5/viewsubmitted
```

Good luck!!