EECE 1080C / Programming for ECE

Summer 2019

Laboratory 4: C++ Functions

Plagiarism will not be tolerated:

all students who share files will receive a 100% penalty on this assignment

Topics covered:

- C++ program development practice.
- Arithmetic, comparative and logical operators.
- Loop and functions.

Objective:

- To practice program development by using loops in C++. The student will create several short programming assignments to create a solid understanding of functions in programming environments.
- The main goals of this laboratory are as follows:
    - Gain hands on experience with creation and usage of functions in C++.
    - Learn aspects of the language that are uniquely or nearly exclusively associated with functions. These include:
        - Syntax.
        - Return values.
        - Calling.
        - Prototypes/declarations.
        - Arguments and default values.
        - Overloading.
        - Variable scope.

Collaboration:

- As with most laboratory assignments in this course this laboratory assignment is to performed by an individual student. You can help each other learn by reviewing assignment materials, describing to each other how you are approaching the problem, and helping each other with syntax errors. You can get help from teaching assistants and instructors. Having slightly similar code for some assignments is expected but most assignments have multiple different solutions. Your code is expected to be different.

- Please document any help you receive from teaching assistants or instructors. Just add the names to the top of your source file.
- Having someone else code for you, sharing code with other students, or copy-pasting code from the internet or previous terms, is cheating. A helper should "teach you to fish, not feed you the fish". Laboratory assignments prepare you for exams so be smart.

Highlights:

- Please consider the following:
    - Review loops.
    - Review C/C++ functions.
    - Read Problem Specification.
    - Please avoid using magic numbers in these functions. Use constants when appropriate.
- To receive full credit for this laboratory please sign the attendance sheet.
- Please access the laboratory assignment via the canopy/blackboard link. The descriptions for each problem are contained within this document.
- **Submit on Blackboard using the assignment dropbox.**

Grading:

- Add a main comment section at the beginning of each program you submit, specifying your name in Pinyin, the date it was last updated and a brief description of what it does.
- Each part should be worked on separately. You will need a separate project for each part of this assignment when working within your IDE.

Rubric: 100 points

- Part A = 20
- Part B = 40
- Part C = 40

Tasks:

A. Comparing Delta Function

- Create a **function** called **bool** compareDelta() that takes a type **double** array with three parameters. The first two numbers are the numbers to compare. The last parameter is the "Delta". The function returns a **true** if the two parameters lay between the plus/minus of said third "Delta" parameter.
- **Example 1:**

  Number1: 1.205

  Number2: 1.305

  Delta: 0.100

  This set of numbers would return a true.

- **Example 2:**

  Number1: 1.3058

  Number2: 1.3059

  Delta: 0.00001

  This set of numbers would return a false.

B. Newton-Raphson Method

- Often in engineering you need to find the roots (zeroes) of polynomial functions. You should know how to do this for linear and quadratic functions, but there is no generalized analytic method for higher order polynomials. In general, the roots are computed numerically. A typical root finding algorithm is the Newton-Raphson (Newton) method.
- As an overview, for a given polynomial function, compute its first derivative analytically. To start the root finding process, specify a tolerance (ε) and an initial guess (x0).
- An improved estimate for the actual root can be computed as:

$$x_1 = x_0 + \Delta x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

  Where:
  - $f(x)$ is the polynomial.
  - $f'(x)$ is the first derivative of the polynomial.
- Store each improved estimate of the root as an element of a type **vector**
  - Hint: you will need to use the **pushback()** function to add new elements to the existing vector
- Continue to compute improved estimates for the root until the following condition is satisfied at the n^{th} iteration of the loop.

$$\varepsilon \geq |x_n - x_{n-1}|$$

- For this exercise, you need to implement the method to find the roots of the polynomial:

$$f(x) = x^4 + 2x^3 - 31x^2 - 32x + 60$$

- This polynomial has all real, distinct roots located in the range [-10, 10]. In addition to **main()**, you must write **three type double functions** that take in only one parameter, x.
    - Call the Newton method.
    - Compute the polynomial.
    - Compute its derivative.
- Its first derivative is:

$$f'(x) = 4x^3 + 6x^2 - 62x - 32$$

- Include and use the following global constant variable in your program for your tolerance value:

```
const double TOLERANCE = .0001;
```

- To find all the roots you will need to enter multiple guesses. In **main()**, allow the user to input as many initial guesses as the user wishes. For each guess call your **first function** to return a root of the equation. Output the value of the root from **main()**. Each guess should be in the range of -10 to 10. Keep guessing until you find all the roots. Allow the user to exit the program when the user has finished guessing.
- Output an error if:

$$f'(x_0) = 0$$
$$f'(x_n) = 0$$

- **Sample Calculations:**

For a guess of 2:
First loop iteration:

```
x0 equals guess                 = 2
fx equals f(2)                  = (24)+2(23)-31(22)-32(2)+60    = -96
fxprime equals fprime(2)        = 4(23)+6(22)-62(2)-32          = -100
deltax equals fx/fprime         = 0.96
x1 equals x0 minus deltax       = 1.04
```

deltax is greater than tolerance so loop again.

Second loop iteration:

```
x0 equals x1                    = 1.04
fx equals f(1.04)               = (1.044)+2(1.043)-31(1.042)-32(1.04)+60     =  -3.3900
fxprime equals fprime(1.04)     = 4(1.043)+6(1.042)-62(1.04)-32              = -85.4909
deltax equals fx/fprime         = -3.3900/-85.4909                           =   0.0397
x1 equals x0 minus deltax       = 1.04 - 0.0397                              =   1.00035
```

Is deltax greater than tolerance? If so keep looping. And so on. Keep iterating through the loop until deltax is less than the specified tolerance.

- **Example:**

```
Enter Guess: -7
                                                              Root: -6

Enter Another Guess: y/n? y
Enter Guess: -2.5
                                                              Root: -2

Enter Another Guess y/n? n
```

C. Electric field for an insulating sphere of charge

- You will soon learn that Gauss' Law requires that and electric field exists whenever there is net electric charge in a region of space or in a material such as a perfect insulator. Figure 1 shows the arrangement for an insulating sphere that contains a total charge, **Q** coulombs.

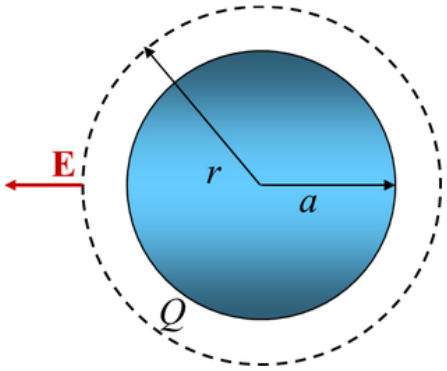

Figure 1. *Electric field for charged insulating sphere.*

- The electric field at any radial distance from the center of the sphere is given by:

$$E(r) = \begin{cases} k_e Q \dfrac{r}{a^3}; r < a \\ k_e Q \dfrac{1}{r^2}; r \geq a \end{cases}$$

- For this exercise, write a program that computes the electric field, $E(r)$, for values of r in the range [0, 100] meters (m).
- In your program, hardcode the following parameters as constant global variables of type double:

$$Q = 10^{-9}\ [C]$$

$$k_e = 9 \times 10^9\ [\frac{Nm^2}{C^2}]$$

- Create a function that computes the electric field for a specified value of **a** and **r** and returns the value to main(). You should then output the radial location and electric field values to the terminal.
- **Example 1:**

```
Enter r: 0.5
Enter a: 1
                                                    Efield: 4.5


Enter r: 2
Enter a: 1
                                                    Efield: 2.25


Enter r: 4
Enter a: 2
                                                    Efield: 0.5625
```