Specification

Assignment 2: A small numerical library

Files needed:

mathlib.h - this file contains the actual functions from the math library that we will be comparing our computations with
mathlib.c - this contains our implementation of the functions arccos(), arcsin(), arctan(), and log().
mathlib-test.c - this file is going to have the main function which will be running the test calling both mathlib.h and mathlib.c functions. It will format the output as expected.
Makefile: a make file to easily run the program by prompting `make` into the Linux terminal on our Ubuntu virtual machine.
README.md: describes how the program works and runs in addition to how to use the makefile
WRITEUP.pdf - this document provides the differences between our approximation versus the one in the math library. Essentially a mathematical explanation of why it works the way it does.

Description of the Program: This program is rather simple in that we are making our very own math library containing four separate functions. The four functions that will need to be made are the arcsin(), the arccos(), arctan(), and log(). We will be making these functions by using the four basic arithmetic operators: + , - , * , /. To make the arcsin and arccos, I intend to build it through a taylor series to get an approximation that is pretty close to the one from the math library. For arctan I intend to use arcsin or arccos to compute arctan. And lastly for log i intend to use Newton's method to find the approximation.

Functions Involved:
Mathlib-test.c main()
//SOURCE: Analogous to the parsing opt() example of Design 2 Document

```
main(int argc, char **argv)
While loop(opt=getopt(argc,argv,OPTIONS)) != -1
Switch statement(opt)
    Case -a:
        arcSin()
        Print chart
        arcCos()
        Print chart
        arcTan()
        Print chart
        Log()
        Print chart
    Case -s:
        arcSin()
        Print chart
    Case -c:
        arcCos()
        Print chart
    Case -T:
        arcTan()
        Print chart
    Case -l:
        Log()
        Print chart
```

The main function for this assignment is just an area where the functions can be called. By prompting in the "opt" we will be able to know which command will compute what function. The printing convention has a lot of formatting but that will be shown in the program itself. Also the opt commands are pretty self explanatory in that the first index of each word is being used to call the certain function. For example -a stands for word all and the index 0 is being pulled as the abbreviation. Similarly -s (arcSin), -c (arcCos), -t (arcTan), -l (Log).

mathlib.c arcSin()

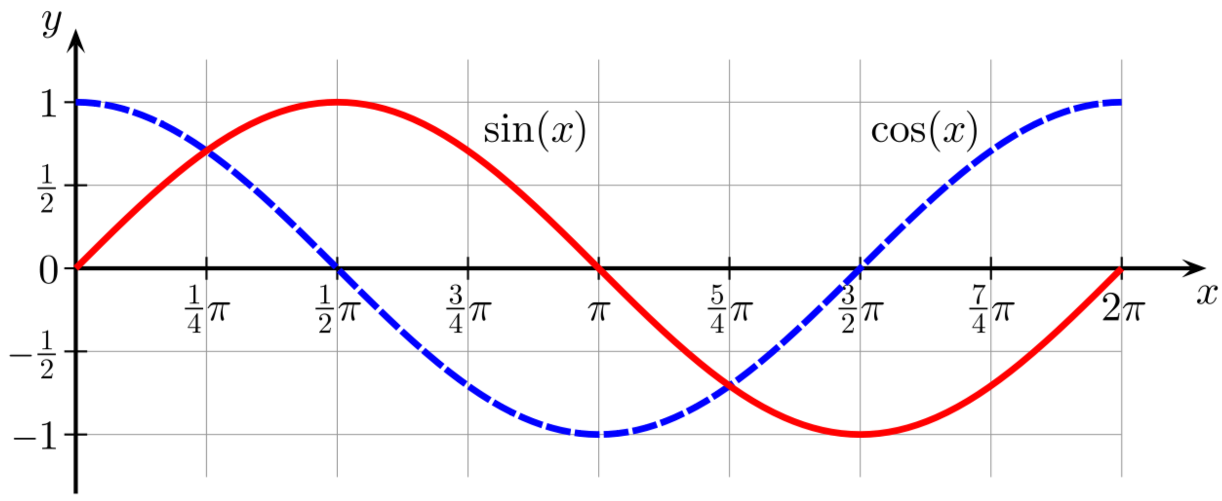
```
//Newton's Method
arcSin(number you want computed)
Epsilon = 10^-10
double guess = x
Double better_guess = 0.0
Double p = (sin y - x/cos y)
    // because sin y - x = f(x), and cos(y)= f'(x)
    while (Abs(better_guess - guess) > EPSILON)
        p = guess - p
        guess = p
```

Inside a computer's ALU, most arithmetic operations are done using the four arithmetic operators: +, -, *, /. In order to make this library we need to build the arcSin function by simply using these arithmetic operators. To do so it involves a little Calculus in what we call Newton's Method. Newton's Method is a process where we start with guessing a point anywhere on the sin graph minus local extremas. That will likely result in an error since the derivative of those are zero. $y_{n+1} = y_n - \frac{f(x)}{f'(x)}$, is the equation to resemble Newton's method where y_{n+1} is the `better_guess` in our function, and y_n is the `guess` of our function. By doing some algebra we can figure out that $f(x) = sin(y) = x$ since that is equivalent to $arcsin(x) = y$. If we take the derivative of that with respect to y we will have the resulting $f'(x)$ which is $cos(y)$. Now that we have the equation understood every iteration of this equation will result in a better approximation of the last. Once the approximation eclipses the `EPSILON` which in this case was 1e-10, we have found an approximation good enough for the assignment.

mathlib.c arcCos()

```
arcCos(number you want computed)
arcCos()
Epsilon = 10^-10
M_Pi_2 - arcsin(x)
```

Similarly to arcSin, arcCos is essentially the same. However we know from a sin and cos standard graph that cos is shifted to the left by a period($\frac{\pi}{2}$). Here are the graphs for a more visual representation:



Same gist as the arcsin function, we will need to use Newton's method to compute the approximation. However notice in the lab document that the initial $\frac{\pi}{2}$period is subtracted to account for the fact that it is different from arcsin. Other than that the equations are matching. We once again utilize that value epsilon to make sure the difference is within up to 10^{-10}likely showing that the difference is zero.

mathlib.c arcTan()

```
arcTan()
    Double p = x/sqrt(1+x)
    arcsin(p)
```

After attending Eugene's Section turns out you only really need to make the arcSin() and derive the rest of the values using this arcSin function. Therefore as given to us in the pdf arcTan is simply the arcSin of the following: $\frac{x}{\sqrt{1+x}}$. Therefore the function is a whole lot simpler than realized. Since we are using arcSin we will be computing using Newton's method.

mathlib.c Log()

//SOURCE FROM LECTURE

```
Log(parameter: number you want computed)
   Double y = 1.0
   Double p = Exp(y)
   while (abs(p-x) > EPSILON)
      y = y + (x - p) / p
      p = exp(y)

    Return y
```

Log on the other hand is using Newton's Method to get a linear approximation so close to what is expected. Using this equation $y_{n+1} = y_n + \frac{x-e^{y_n}}{e^{y_n}}$, we will continuously iterate till the approximation similar to the Taylor series is close enough to epsilon which we have denoted to be 10^{-10}. I guess I could also point out that the arcSin and arcCos could have been also made through Newton's Method.