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
arcSin flag = 0
arcCos flag = 0
arcTan flag = 0
Log flag = 0
Switch statement(opt)
    Case -a:
        arcSin flag = 1
        arcCos flag = 1
        arcTan flag = 1
        Log flag = 1
    Case -s:
        arcSin flag = 1
    Case -c:
        arcCos flag = 1
    Case -T:
        arcTan flag = 1
    Case -l:
        Log flag = 1
```

```
If arcsin flag = 1
   From range -1 <= x <= 1
   Print arcSin(), asin(), difference of arcSin and asin
If arcCos flag = 1
   From range -1 <= x <= 1
   Print arcCos, acos, difference of arcCos and acos
If arctan flag = 1
   From range 1 <= x <= 10
   Print arcTan, atan, difference of arcTan - atan
If Log flag = 1
   From range 1 <= x <= 10
   Print Log, log, difference of Log - log
```

The main function for this assignment is just an area where the functions can be called and printed. 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). These commands are called and printed through a flag system. At the beginning of the program the flags are all set to 0, but if one of the commands are entered when making the program, the flag is set to 1. If the -a is called then all the flags are set to 1. Once a flag has been set, they will go through the if statements below where the function will be called within their ranges and compute a value for every 0.1 increment. The printing format given to us in the lab document is responsible for how the program is outputted into the nice chart. The formatting print statement is listed below:

```
printf("%7.4lf%16.8lf%16.8lf%16.10lf\n", ...)
```

Once printed your program is essentially done.

mathlib.c arcSin()

```
//Taylor Series- logic was given in Eugene’s section
arcSin(parameter: number you want computed)
Epsilon = 10^-10
double summation = number you want computed
Double term = number you want computed
Double approximation = number you want computed
coefficient = 1.0
for (k= 3, Abs(approximation) > EPSILON, k+=2)
    Coefficient = coefficient * (k-2)/(k-3)
    // this is what is multiplied in every taylor series
    Term = term * number you want computed * number you want
computed
    // this will give you your $x^3, x^5, x^7$....
    Approximation = Taylor series put together = term/k *
coefficient
```

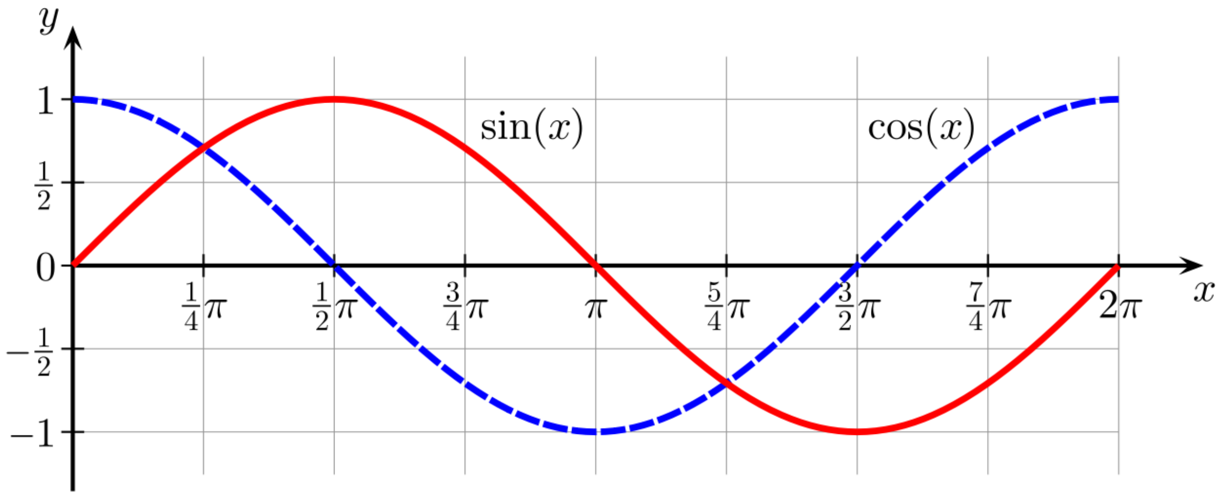
```
    Summation = summation + approximation

Return summation
```

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 a taylor Series. Taylor Series is the best way to find an approximation for this program because it involves the only basic operators. You can also use what's called Newton's method but I had trouble getting good approximations the farther they stemmed away from 0. Therefore in this taylor series we have the very daunting taylor series, but in its expansion listed in the lab document, you can break down the equation and see what is being added in each successive approximation in which we have made a small algorithm to compute each approximation within a for loop. Since Taylor series are summations however you must add each successive term to the total so then the approximation does indeed get closer to that value epsilon. Once the number is so small that it is less than epsilon, we know that the approximation is pretty well done. (*In my case I only had issues with the approximations at 1 and -1 for both arcSin and arcCos, but all other approximations should be acceptable).*

```
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 the same Taylor series that we made in the arcSin function 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 absolute value of the current approximation is less than 10^{-10}likely showing that the difference is zero. If you still question why you can use arcSin to construct arcCos just remember the trigonometric identity $\frac{\pi}{2} - arcSin(x)$. In fact you will see real soon that we also find arcTan simply by deriving its equation from arcSin as well. Obviously we could build out each function's Taylor Series, but it is far easier to just use one function to find the rest. Also notice that we use some constant `M_PI_2` which is equivalent to $\frac{\pi}{2}$. This is a constant straight from the math library that Eugene showed us in his section to show that we don't need to have an approximation of the value $\frac{\pi}{2}$.

mathlib.c arcTan()

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

As previously mentioned, arcTan, like arcCos can be found by our function arcSin() which is infact using a Taylor Series. Another trigonometric identity that we should recall is that:

$$arcTan \ = arcSin(\frac{x}{\sqrt{x^2+1}})$$

Therefore we can simply construct the formula inside the arcSin. Notice that we are also using a Sqrt() function in this function. We are not allowed to call Sqrt from the math library so we must build a Sqrt function ourselves. Luckily the professor has provided us his Sqrt, and Exponential functions to utilize for this assignment. I also forgot to mention that we had to construct an absolute value function that you saw in previous examples and that one is rather easy. If x < 0 we know that we must multiply whatever x is by -1 to get its positive value. So now that those questions are cleared, we are able to find arcTan simply by deriving it from arcSin once more.

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}. As previously mentioned you could make all the previous functions using Newton's Method in which I did initially, but I had a problem with getting accurate approximations the farther they got from 0 so that is why I have resorted to using the taylor series approximation. However I did not delete my code so you can see that I have attempted to make arcSin from using Newton's Method. However Newton's method is quite intuitive when building Log(). In Newton's method you have a current guess y_{n+1}, a previous guess y_n and a $f(x)$which you can derive from simply finding the inverse of Log, in this case it is $x - e^{y_n}$. If you were to take a derivative of the $f(x)$ you will notice that $f'(x)$is infact e^ysince the derivative of $e^y$is simply itself. So if you took the previous guess and added it to the $\frac{f(x)}{f'(x)}$ you would get an approximation that slowly and slowly gets closer to the answer. It iterates through the while loop until the values are once again less than Epsilon. Now I won't take credit for making this function myself because Professor Long sort of gave out his code during lecture. However instead of just copying it and citing it, I wanted to make sure I knew what the function was doing.

Updates:
-Initially, I made all my functions using Newton's method, but didn't quite know how to make the approximations better. I read all the piazza posts, and even post @357 where he talked about this issue, but what I didn't understand was how he was using inverse cosine to find a better approximation for 1 and -1, when my arcCos function relies on arcSin to find the correct answer
-In the end I decided to use Taylor Series, not because it made more sense but because my approximations just ended up being closer than that of Newton's Method. I also do want to say that I literally did give it a lot of thought on how I could continue to use Newton's method, but in the end I am likely turning in my taylor series version. However in my code you will see that I have left Newton's Method within my source code because I didn't want to simply just delete my work and instead just comment out that block of code.