## Design 0:Initial Design(mathlib-test.c not included)

### mathlib.c

- Define Epsilon
- Sin(x)
    - Initial numerator corresponds to starting domain value (x) from mathlib-test.c
    - The denominator will start as 1 since first value in series can be simplified to x/1
    - Sum = the total of all terms
    - Temp = current term
        - Loop:
        - The current numerator is multiplied by (-1)*x^2 to incrient its exponent value by 2
        - Since the series adds 0 every other term, we can't just increase i! By i+1, we need to take into account the missing i. To do this we multiply the current i by (i-1), or the previous i. Multiply this to the old denominator value from last loop to get the current terms denominator
        - Temp then stores the current numerator/denominator value and adds that to the series sum

- Cos(x) (will be simpler than sin to avoid redundancy)
    - Numerator is 1, because cos(0) is 1.
    - Denominator is 1
    - Total sum is 1 because the first term is 1
    - Temp = current term, the sum
    - Loop:
        - Same as sine because all we did was offset the starting point of the series. This is based on the derivatives of Cos and Sin being one another with slight alterations and a constant repetition every 4 derivatives, and the value of Sin(0) being 0, while cos(0) is 1.
- Tan(x)
    - Just have to do Sin/Cos
- Exp()
    - E^x is much easier to code because of its trivial taylor series and derivative sequence.
    - Like the document says new = previous * current

    - Since there is no skipping an i!, also makes it much easier
    - Set the first term to e^0/0! Which is 1
    - Loop:
        - Multiply the previous term by the current x/i and that will do (i-1 * i) which is the current i!, and multiply the numerators making for a smooth expression
        - Increase the total sum by the new term
    - Log(x)
        - We see that e^x = y, and ln of both sides will get us ln(y) = x, and using Newtons method to find roots we can subtract y from both sides to get e^x - y =0 and apply that to his formula.
        - Instead of writing Exp(x) in our loop, we can create a new variable to store Exp(x)
        - Loop:
            - Plug in values to newtons formula:
                - y-((x-e^y)/e^y)

## Design 1:Polished Design

- This program provides custom code for some provided library functions such as sin, cos, tan, exp, and log.
- Prints in an organized format that compares the differences in custom functions to those of the default functions.
    - **Mathlib.c**
        - Contains the code for the main functions, with a definition of epsilon.
        - Since using integers for the Taylor approximations will be insufficient in finding high precision numbers, doubles will be used for all variables such that Taylor series, P(x), terms can be at least 64 bits of precision, leaving small room for error.
        - Each Taylor series can be written with a while loop, iterating as long as the current term in the series is no smaller than our defined approximation of epsilon.
        - This file will not print, rather return the sum of each series in a variable called sum, which will be

used by mathlib-test.c to print the data in an orderly fashion

- **Mathlib-test.c**
    - Creates bools corresponding to command line arguments which will be altered depending on the command line argument
    - Prints an initial statement giving user instructions on how to interact with the program if no argument is given
    - A switch statement will be used to enable said bools
    - Implement various if statements that print depending on the values of the initialized bools.