CSE13s Fall 2021
Assignment 2 :-A Little Slice of π
By: Ruhin Gharai

The assignment:-

The task of the assignment is to make mathematical functions that mimic <math.h> and using them to compute Fundamental concepts.There will be 6 C files for the 6 formulas given which are calculating e, calculating π using the madhava series,euler's solution,The Bailey-Borwein-Plouffe Formula ,Viète's Formula and square newton for the other functions in the other file.Each of these files will have two functions. The function with the code to make the formula and return the computed number that you get from the formula and the other function that returns the count of the loop. Each file will have the header file include "mathlib.h" so it will be able to be called by the test file.The mathlib-test.c file will be the main test file called mathlib-test.c to call out the other formula files to run using getopt GNU library to call each function from the 6 different files to run.

The files with the stated functions:-

- E.c:-
  This contains the implementation of the Taylor series to approximate Euler's number e and the function to return the number of computed terms.

  Pseudo code for e.c :-Make the variable s = 1 and pt = 1. Make a for loop to only break if pt is greater than epsilon. In the for loop pt is equal to pt divided by k. And then s is added to pt. Then return s which gives the value e.

- Madhava.c:- This contains the implementation of the Madhava series to approximate π and the function to return the number of computed terms.

  Pseudo code for madhava.c:- so make the variable k = 0, up = 1, mul = -3, pi_num = 0 and make_update = 1. Creating a while loop to loop but uptill 10000.Then make a for loop to iterate i with the condition of i < k -1 . When does -3 is multiplied to mul.But if make_update is equal to 1 then up = 1 and make_update = 0.Else up = 1/mul and make_update still equal to 0. Then up is multiplied after going through the all the loops to 1/((2*k) +1). Then pi_num is added to up and k is increased up one time by adding 1 to

it. Mul = 3. Then breaking from the while loop pi_num is multiplied to the square_newton function that has the number 12 in it. And return the pi_num to finally give the value.

- Euler.c:- This contains the implementation of Euler's solution used to approximate π and the function to return the number of computed terms.

Pseudo code for euler.c:-
Make the variables total = 0 and k = 1 . The while loop condition is that 1>(k*k) is greater than epsilon then it can stop looping.In the loop the total is added to 1(k*k) and k is added to 1. Then return the square_newton(6*total).

- Bbp.c:-
  This contains the implementation of the Bailey-Borwein-Plouffe formula to approximate π and th function to return the number of computed terms.

Pseudo code for bbp.c:-
Make the variable term = 1, sum = 0, num = 0.Then create a for loop which has the new variable x and the condition that absolute(term) > EPSILON) and iterates x. Then using the if and else statement if x = equal to 0 then num = 1 or else num is multiplied to 16. Breaking out of statement new variable f = 1/num and new variable term is given to make f * (((4) divided (8 * x +1)) -((2) divided (8 * x +4)) - ((1)/(8 *x +5)) - ((1)/(8 * x +6))). Then sum added to term.then return sum out of the for loop.

- Viete.c:-
  This contains the implementation of Viète's formula to approximate π and the function to return the number of computed factors.

  Pseudo code for viete.c:-
  This has two variable in that are v1 = 0 and v2 = 1.Make a while loop that has 1 in the Parenthesis .Then v1 equal to sqrt_newton (2 added to v1).
  Then V2 equals v1/2 multiplied by v2.
  A new variable tem which is equal to v1 divided 2.
  IF statement is if 1/tem - 1 is less than EPSILON. The double tem2 new variable equal to 2/v2 then return tem2.

- Newton.c:-

This contains the implementation of the square root approximation using Newton's method and the function to return the number of computed iterations.

Pseudo code for Newton.c:-
The function begins with an initial guess z = 0 and y = 1 used to compute better approximations
Using while loop when y-z is greater than epsilon,
the in the loop z now equals y
Y = 0.5 *(z+x/z)
To then return the value of y which gives the square root of the argument

- Mathlib-test.c

Using getopt GNU library , I will use the function in a while loop to run each function file. Using a switch statement , it will allow me to call upon the file just using a dash and a letter given below. This will initiate the file to run when called upon.

- -a : Runs all tests.
  Run all test by calling all cases
- -e : Runs e approximation test.
- Use printf statement and give the format required
  But compare with e number

- -b : Runs Bailey-Borwein-Plouffe π approximation test.
- Use printf statement and give the format required
- But compare with pi value

- -m : Runs Madhava π approximation test.
- Use printf statement and give the format required
- But compare with pi value

- -r : Runs Euler sequence π approximation test.
- Use printf statement and give the format required
- But compare with pi actually value

- -v : Runs Viète π approximation test.
- Use printf statement and give the format required
- But compare with pi actually value

- -n : Runs Newton-Raphson square root approximation tests.
- Use printf statement and give the format required
  Then compare it with the square root .
-
- -s : Enable printing of statistics to see computed terms and factors for each tested function.
  Talk about what each case is and how to call them

- -h : Display a help message detailing program usage.
  Talk about what each case is and how to call them

Also makefile was made so it's easier to compile each file to run each function rather than typing it out every time to compile and format.

Design Process:-
I first worked on the Makefile, so it would be easier to run each file. Then worked on the mathlib-test.c as I would be the longest and most difficult file to code as it uses a new confusing library. Then I work on Newton.c as the pseudo code is given in the assignment page and work on the other files from there. I made the code but some of my output is off by a few digits but I used a lot of math without the help of the math library. This took me a while as this required me to use 4 formulas to find one solution . The reason I got so many errors in the beginning was because my newton was completely wrong so it made all my numbers off. Figured out the correct one and submitted it late but the code actually worked.