# Design Document

Manikanta Illuri

**Final Design:**

**Objective:** The object for this assignment is to implement a small number of mathematical functions (e x and p x), mimicking , and using them to compute the fundamental constants e and π.

**E.c:**

For this file I will include the standard library and the mathlib.h to access the given functions. I will establish a static variable of the type double as a counter for the number of iterations. Counter stats from 1 because k value cannot start from 0 because it would still equal 1, this is to avoid an extra iteration.

Functions:

- Exponent
    - The exponent function was created according to a suggestion given by a member of the unofficial discord group. This function will be used to implement an exponent calculation. It will have two variable type doubles that will have an initial value of 1.0. The function will also take in one argument which will be defined as 1 in the e function. The function has a for loop that iterates while val is greater than Epsilon and will increase the counter variable every iteration. The for loop will multiply val value by the input /count and store it back in val. Then it will add ele to val to give the final exponent value. And will return ele.
- E
    - The e function will call the exponent function and passing 1 as the argument.
- E_terms
    - This function will return count as an int type which is the iteration count. It is types casted at this stage because having a double type for count in the exponent function will give us a more precise answer.

**Newton.c:**

For this file I will include the standard library and the mathlib.h to access the given functions. I will establish a static variable of the type double as a counter for the number of iterations. The counter is reset in the second function.

Functions:

- Sqrt_newton
    - This function will be of type double and take in one argument of type double. The purpose of this function is to approximate the square root of the argument using the Newton-Raphson method. This is done by first having two double type variables that have the values of 0.0 and 1.0. Then a while loop that runs with the

conduction |var1 and var2| > Epsilon, it will also increase counter every iteration. Within the loop var1 will be set to var2 to store the value. Then var2 will be give the value of ½ *(var1+(x/var1)). Then it will return the value of var2.

- Sqrt_newton_iters
    - This function will return count as an int type which is the iteration count. It is types casted at this stage because having a double type for count in the exponent function will give us a more precise answer.

**Madhava.c:**

For this file I will include the standard library and the mathlib.h to access the given functions. I will establish a static variable of the type double as a counter for the number of iterations. Counter stats from 1 because k value cannot start from 0 because it would still equal 1, this is to avoid an extra iteration.

Functions:

- Pi_madhava
    - This function's purpose is to approximate the value of π using the Madhava series. This is done using a for loop. First I created a variable with type double and value 1 to hold fnl calculated value. I also created a variable for the numerator with value 1 and denominator with no value as it will be modified. And a temp variable of the same type to store the value of the fraction.
    - The for loop runs when the absolute value of tmp is greater than Epsilon. And will increase the counter each time. The loop will first multiple num with -⅓ and store back in num. Then will calc den by 2*count +1. Then temp will be given the value num/dem and will be added to fnl value. Which is 1.
- Pi_madhava_terms
    - This function will return count as an int type which is the iteration count. It is types casted at this stage because having a double type for count in the exponent function will give us a more precise answer.

**Euler.c:**

For this file I will include the standard library and the mathlib.h to access the given functions. I will establish a static variable of the type double as a counter for the number of iterations. Counter stats from 2 because k value already is 1 so to avoid an extra iteration we start counter from 2.

Functions:

- Po
    - This po function will square the input value. I created this since in C “**” refers to a pointer.
- Pi_euler

    - The purpose of this function is to approximate the value of π using the formula derived from Euler's solution to the Basel problem. The function has 2 variables initiated of type double. The fnl will hold the final value and fraction and the count of squares done in the loop. The loo[p will first run with conduction while fraction is > Epsilon. And will increase count by 1. First the square count value will be set to the count squared. Then the fraction value will be set to 1/square count and then will be added to fnl. Outside the loop the fnl value will be multiplied by 6 and will be squares using sqrt__newton because the Euler solution requires the calculation of square root.
- Pi_euler_terms
    - This function will return count as an int type which is the iteration count. It is types casted at this stage because having a double type for count in the exponent function will give us a more precise answer.

**Bbp.c:**

For this file I will include the standard library and the mathlib.h to access the given functions.
I will establish a static variable of the type double as a counter for the number of iterations.
**I was unable to get 0.0 difference between the two pi values. I was unable to figure out what was causing a 0.008 difference between them.**
Functions:

- Pow
    - This function was implemented because a friend in section told me that having a seperate pow function will make keeping track of the exponent value simpler so I as well implemented it on my own. The only hit he gave me was to use a for loop and check for unusual circumstances with exponents. The pow function was created by me to calculate an exponent. The function takes in the base and exponent value. Then I will have a variable to hold the final value. In a for loop that iterates while i is less than or equal to exponent value. Then it checks for if the base is not 0 equal to 0 and if the exponent is 0 then it will return the value 1. Else it will calculate the power and return 1.0 / final value.
- Pi_bbp
    - This function has 7 double variables. 4 of them are to keep track of the terms of the equation. The fnl will hold the final value of all terms subtracted. The total var will hold the final value of pi.  In a for loop the power is calculated by the pow function and each of the terms are run through their respective equations as mentioned in the asgn doc. Then fnl is calculated as mentioned above. Then fnl is added to tot and returned outside the loop.
- Pi_bbp_terms

- This function will return count as an int type which is the iteration count. It is types casted at this stage because having a double type for count in the exponent function will give us a more precise answer.

**Viete.c:**

For this file I will include the standard library and the mathlib.h to access the given functions. I will establish a static variable of the type double as a counter for the number of iterations.
Functions:

- Pi_viete
  - This function will have a for loop that iterates over the conviction that absolute(1.0 - fraction) >= EPSILON, within the loop it will set the value of the previous to sqrt_newton(2 + previous term). Then it will divide it by 2 then it will set the value of tot to fraction * tot. Will return 2/tot
- Pi_viete_factors
  - This function will return count as an int type which is the iteration count. It is types casted at this stage because having a double type for count in the exponent function will give us a more precise answer.

**Mathlib-test.c:**

- This file will have a main function where each of the flags will be given a bool value. WHen each of the cases are called the bol value will change.
- Then through a series of if statements I will have two primary ones.
  - First if will be when verbose true
    - Withinit I will print all the flags using print statements and make it simile to the given executable file
  - The second statement will be when the verbose is false. I will print each of the flags in this section.

The Individual files are giving a similar output to the math library but when printing the difference I am getting a major difference. I have checked my mathlib-test file for any errors in my subtraction but I was not able to find any.

**ERROR**

As you can see below, my file output value and the library value are similar with a very minor difference at the end but when it prints the difference it says a very different value that doesn't seem mathematically correct.

```
e() = 2.718281828459046, M_E = 2.718281828459045, diff = 1.659648804812424
pi_euler() = 3.141592558095903, M_PI = 3.141592653589793, diff = 0.692102910806603
pi_bbp() = 3.133365596303085, M_PI = 3.141592653589793, diff = 3.141592653589793
pi_madhava() = 3.141592653589800, M_PI = 3.141592653589793, diff = 0.306622522521471
pi_viete() = 3.141592653589789, M_PI = 3.141592653589793, diff = 0.000000000000004
```

My subtraction  also seems correct but I am unsure what's wrong.

```
if (verbose == true) {
    if (all == true) {

        printf(
            "e() = %16.15lf, M_E = %16.15lf, diff = %16.15lf\n", e(), M_E, absolute(e() - M_E));
        printf("e() terms = %d\n", e_terms());

        printf("pi_euler() = %16.15lf, M_PI = %16.15lf, diff = %16.15lf\n", pi_euler(), M_PI,
            absolute(M_PI - pi_euler()));
        printf("pi_euler() terms = %d\n", pi_euler_terms());

        printf("pi_bbp() = %16.15lf, M_PI = %16.15lf, diff = %16.15lf\n", pi_bbp(), M_PI,
            absolute(pi_bbp() - M_PI));
        printf("pi_bbp() terms = %d\n", pi_bbp_terms());

        printf("pi_madhava() = %16.15lf, M_PI = %16.15lf, diff = %16.15lf\n", pi_madhava(),
            M_PI, absolute(pi_madhava() - M_PI));
        printf("pi_madhava() terms = %d\n", pi_madhava_terms());

        printf("pi_viete() = %16.15lf, M_PI = %16.15lf, diff = %16.15lf\n", pi_viete(), M_PI,
            absolute(pi_viete() - M_PI));
        printf("pi_viete() terms = %d\n", pi_viete_factors());

        for (double i = 0.0; i <= 10.0; i += 0.1) {
            printf("sqrt_newton(%0.5lf) = %16.15lf, sqrt(%0.5lf) = %16.15lf, diff = %16.15lf\n",
                i, sqrt_newton(i), i, sqrt(i), absolute(sqrt_newton(i) - sqrt(i)));
            printf("sqrt_newton_terms() = %d\n", sqrt_newton_iters());
        }
    }
```

**Initial Design:**

**Objective:** The object for this assignment is to implement a small number of mathematical functions (e x and p x), mimicking , and using them to compute the fundamental constants e and π.

**E.c -**

This file will have two functions.

The first being the e function which will use a while loop that compares a variable to the value of epsilon. Within in the loop the equation $\frac{x^k}{k!} = \frac{x^{k-1}}{(k-1)!} \times \frac{x}{k}.$ **.** I will start the variables at valuer 1 and and calculate the values based on the equation above. I will have a counter within the while loop to keep track of my iterations.

The next function will be returning the countervalue which will ve a static variable defined at the beginning of the file.

**Madhava.c -** I will include all the necessary files to calculate the madhava series. I will have a while loop till epsilon and within the loop I will have the value of the iterator raised to the power of 3 and then I will divide the iterator and multiply it by the value and add 1 to it.
The second function will be used to count the number of iterations the above function ran.

**Euler.c -** I will declare variables to keep as a counter and set that as a static variable. I will have a for loop and inside of the I will calculate the value by the square root divided by the index value iteration . Then multiply it by 10. The next function will store the value of iterations also known as the countervalue.

**Bbp.c -** The function will have a while loop that will run till epsilon and within the loop I will have the equation for the bbp calculation. This equation is This equation will have a for loop that will calculate summation and with in the loop I will raise the index value to the power of 16 and do the equation (k(120k +151)+47) k(k(k(512k +1024)+712)+194)+15 .

**Viete.c -** This will calculate the value based on the viete equation. I will have a nested for loop that will squarethe iterator value and add 2 +ak-1 to it. The nest function will calculate the iteration value to find pi.

**Newton.c -** I will use this function to calculate the value of pi using the Newton formula. This will have a for loop that will run based on the calculator using the 2 square root 2 divided by 9801 multiply it by 4k factorial. Then I will divide it by (k!)^4396^4k.
The second function , the second function will calculate the interactive value that we will use to calculate pi.

**Mathlib-test.c -**
-a : Runs all tests. • -e : Runs e approximation test. • -b : Runs Bailey-Borwein-Plouffe π approximation test. • -m : Runs Madhava π approximation test. • -r : Runs Euler sequence π approximation test. • -v : Runs Viète π approximation test. • -n : Runs Newton-Raphson square root approximation tests.

These are the flags that I will implement in the mathlib file. These can be called when the file is being run to produce the output pi value.