Assignment 2 - *A Little Slice of pi* - DESIGN.pdf

About Program:

This program implements a small number of mathematical functions that mimics <math.h> then using them to compute the constants e and pi.

Files included in asgn2:

1. e.c -
    a. Contains the 2 functions, e() and e_terms(), which are used to approximate the value of e using the Taylor Series and tracks the number of computed terms with a static variable local to the file - returning this number
2. madhava.c
    a. Contains the 2 functions, pi_madhava() and pi_madhava_terms(), which are used to approximate the value of pi using the Madhava Series and tracks the number of computed terms with a static variable local to the file - returning this number
3. euler.c
    a. Contains the 2 functions, pi_euler() and pi_euler_terms(), which are used to approximate the value of pi using the Euler solution derived to the Basel problem and tracks the number of computed terms with a static variable local to the file - returning this number
4. bbp.c
    a. Contains the 2 functions, pi_bbp() and pi_bbp_terms(), which are used to approximate the value of pi using the Bailey-Borwein-Plouffe formula and tracks the number of computed terms with a static variable local to the file - returning this number
5. viete.c
    a. Contains the 2 functions, pi_viete() and pi_viete_factors(), which are used to approximate the value of pi using the viete formula and tracks the number of computed terms with a static variable local to the file - returning this number
6. newton.c -
    a. Contains the 2 functions, sqrt_newton() and sqrt_newton_iters(), which are used to approximate the sqrt of the argument passed to it using the Newton-Raphson method and tracks the number of iterations taken with a static variable local to the file - returning this number
7. mathlib-test.c
    a. Main test harness for the implemented math library with following command line options:
        i. -a: run all tests
        ii. -e: run e approximation test
        iii. -b: run bbp pi test
        iv. -m: run madhava pi test
        v. -r: run euler pi test
        vi. -v: run viete pi test
        vii. -n: run newton sqrt test
        viii. -s: enable printing of statistics to see computed terms and factors for each tested

function
    ix. -h: display help message detailing program usage
8. mathlib.h -
    a. Interface for math library
9. Makefile
    a. Directs program compilation, building mathlib-test executable
10. README.pdf
    a. Describes how to use program and Makefile, list and explains command line options, and notes any known bugs/errors
11. DESIGN.pdf
    a. Design process for the program
12. WRITEUP.pdf
    a. Graphs displaying the difference between values of the implemented function and math library functions
    b. Analysis and explanations for any discrepancies and findings from testing

Errors and Bugs

No known errors or bugs

Pseudocode/Structure

-- newton.c --

```
Static iters;   // static variable to return amount of iterations
sqrt_newton(x)
{  // approximates the sqrt of number passed - code equivalent in python given by professor in lab doc
        z=0.0;
        y=1.0;
        counter=0;
        while the absolute value of y - z is greater than epsilon { // while loop to calculate approximation
                z=y;
                y= ½ (z+x/z);
                ++counter;  // increment the counter to track iterations
        }
        Iters = counter;
        return y
}
sqrt_newton_iters() { // function to return the number of iterations
        return iters
}
```

-- e.c --

```
Static count; // static counter to track number of computed terms
e() {
	track=0; // counter for terms
	For (k = 0; till 1/k! reaches epsilon; k++) {  // for loop that loops until we get under epsilon
		// changed to while current - prev term > EPSILON - used in almost all functions
		factor = 1;   // our "factorial" variable
		While (k > 1) {  // calculate k! And save it into factor to use for the next iteration and keep
doing till 1 is reached
			factor = factor * k
			k--;    // decrement k
		}
		x += (1/factor);  // add each factorial and save to x
		track++  // increment terms
	}
	Count = track // save to static variable
}

e_terms() {
	Return count
}
```

-- mathlib-test.c --

Using example from lab doc "parsing options using getopt()"
Include getopt thing, mathlib.h, math.h for comparison,

```
Define options to aebmrvnsh
main(arg, argv things like in example) {
	Int opt = 0;
	Bools for 9 variable to use in switch
	while((opt = getopt(argc,argv, options))  != 1)
		Switch (opt)
			9 cases, one for each option
				Set bool variables to true
				Break
	If a bool variable is true for an option:
		Do the test (print out according to example binary)
```

-- bbp.c --

```
double pi_bbp(void)
```

```
        Define initial variables
        Track = 1, base = 1, pie = 47/15, k = 1
        While current - prev term > epsilon
                Base *= 16
                Prev = current pi
                Pie += k * (120.0 * k + 151.0) + 47.0 /( k * (k * (k * (512.0 * k + 1024.0) + 712.0) + 194.0) + 15.0)
* base
                tracker ++
int pi_bbp_terms(void)
  return tracker;
```

-- *euler.c* --

```
double pi_euler(void)
        While current - prev term > epsilon
                Sum += 1/k^2
                Prev = current pi
                Pi = sqrt(6*sum)
                Tracker ++

int pi_euler_terms(void)
        Return tracker
```

-- *madhava.c* --

```
double pi_madhava(void)
        Power = 1
        While current - prev term > epsilon
                Power = power * 3
                pi += 1.0 / (power * 2k+1);
        Pi = sqrt(12) * pi

int pi_madhava_terms(void)
        Return tracker
```

-- *viete.c* --

```
double pi_viete(void)
        A_n = sqrt(2) // first term , a_1
        Pi = a_n
```

While current - prev > epsilon
A_n = sqrt(2 + a_n)
Prev = pi
Pi *= a_n / 2
Pi = 2 / (pi / 2)

int pi_viete_factors(void)
Return tracker

Brainstorming/working through code

```
2.71.
① e(void) {
      x = 0
      factor = 1
      for (k = 0 , 1/factor > ep, k++)
          factor = 1
          while (k > 1) {
              factor = factor * k
              k-- }
          x += 1/ factor }
```

1st for round   k = 0
no while
x = 1

2nd for round   k = 1
x = 2

3rd for round   k = 2
while: factor = 2

x = 2.5

4th for round   k = 3
while: factor = 3
factor = 6.

x = 2.5 + 1/6 = 2.667

5th for round   k = 4
while: factor = 4
factor = 12
factor = 24
x = 2.708

bbp

$$\sum_{k=0}^{n} 16^{-k} \times \frac{(k(120k+151)+47)}{k(k(k(512k+1024)+712)+194)+15}$$

$$\hookrightarrow \quad \frac{1}{16^k} \times$$

$$\frac{47}{15}$$

main

```
~~sw~~switchaemlon . . . {
  case e : do e = true
     ⋮
  case s3 : do terms = true

if do e {
     do e . . .
   if do terms
       print terms
```

or . . .

```
switch {
   case s3 : do terms = true
switch {
   ccase e :
        print e
        if do terms = true
            print terms
```

e = 2 , prev

for k = 2 ; (e - (1.0/factor)) > eps ; k++)

    factor = 1
    while k > 1
        factor = factor * k
        k --
    prev = e
    e += 1.0/factor

1 < R = 0 ✓

~~1st round k = 2~~

~~e = 1~~

~~2nd round k = 1~~

e = 2

1 > eps

1st round k = 2

k2

~~factor = 2~~

factor = 2

k1

e = 2.5

2 > eps

2nd round k = 3

k3

factor = 3

k2

factor = 6

k1

e = 2.667

2.5 > eps

3rd round k = 4

Madhava

$$\sqrt{12}\sum_{k=0}^{n}\frac{(-3)^{-k}}{2k+1} \qquad \frac{-\frac{1}{3}}{3} \qquad \frac{-\frac{1}{9}}{5}$$

$$-\frac{1}{1} + -\frac{1}{9} + -\frac{1}{45} \cdots$$

$$\sqrt{12}\,(\rightarrow)$$

$$(-3)^{-k} \rightarrow \cancel{1/3}\ \frac{-1}{\ }, \cancel{\frac{1}{3}}, -\frac{1}{9}$$

pow = 3 , p = ~~1/3~~

pow *= 3 → 9, 27, ...

p += (-1.0 / pow → $-\frac{1}{9}$, $-\frac{1}{27}$ ...

Viete

$$\pi = \frac{2}{\prod_{k=1}^{\infty}\frac{a_i}{2}} \qquad a_1 = \sqrt{2}$$

$$a_k = \sqrt{2 + a_{k-1}} \quad \text{for } k > 1$$

$$= \frac{2}{\frac{\sqrt{2}}{2}\cdot\frac{\sqrt{2+\sqrt{2}}}{2}}$$

Eulers

$$\pi = \sqrt{6\sum_{k=1}^{\infty} \frac{1}{k^2}}$$

$$\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} \quad \ldots$$

$$\sqrt{6(\quad)}$$

$$\frac{1}{k^2} = \frac{1}{(k \cdot k)}$$

```
k=1
while   abs(pie-prev) > eps
     sum += 1.0/(k·k)
     k++
     prev = pie
     pie += sqrt(6·sum)
```

Round 1   k=1

sum = 1.0

k = 2

1.) 1011

4.) 1111

10.) 1100

Assignment 2 Lab Document - Notes

1. Introduction
   a. when you see Σ you should generally think of a for loop.
   b. Use infinite series
2. Fundamental Constants
   a. Euler's Identity
      i. $e^{i\pi} + 1 = 0$
   b. From this can find pi
      i. Pi = 2log(i^-i)
3. Calculating e
   a. $$e = \sum_{k=0}^{\infty} \frac{1}{k!} = 1 + \frac{1}{1} + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \frac{1}{720} + \frac{1}{5040} + \frac{1}{40320} + \frac{1}{362880} + \frac{1}{3628800} + \cdots$$
   b. How many terms must you compute? Fewer than you might expect, since k! grows very fast. You will be determining that experimentally as part of this assignment.
      i. $$\frac{x^k}{k!} = \frac{x^{k-1}}{(k-1)!} \times \frac{x}{k}.$$
      ii. At each step compute x/k starting with k=0! (0! = 1) and multiply by previous value and add it into the total.
         1. Simple for or while loop
4. Calculating pi
   a. Madhava Series
      i. $$p(n) = \sqrt{12}\sum_{k=0}^{n} \frac{(-3)^{-k}}{2k+1} = \sqrt{12}\left[\frac{1}{2}3^{-n-1}\left((-1)^n\Phi\left(-\frac{1}{3}, 1, n+\frac{3}{2}\right) + \pi 3^{n+\frac{1}{2}}\right)\right].$$
   b. Euler's Solution
      i. $$p(n) = \sqrt{6\sum_{k=1}^{n} \frac{1}{k^2}}$$
   c. BBC Formula
      i. The formula that they discovered is remarkably simple:

         $$p(n) = \sum_{k=0}^{n} 16^{-k}\left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6}\right).$$

         And if you desire to reduce it to the least number of multiplications, you can rewrite it in *Horner normal form*:

         $$p(n) = \sum_{k=0}^{n} 16^{-k} \times \frac{(k(120k+151)+47)}{k(k(k(512k+1024)+712)+194)+15}.$$
   d. Viete's Formula

Viète's formula can be written as follows:

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \times \frac{\sqrt{2+\sqrt{2}}}{2} \times \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \cdots$$

Or more simply,

$$\frac{2}{\pi} = \prod_{k=1}^{\infty} \frac{a_i}{2}$$

i. here $a_1 = \sqrt{2}$ and $a_k = \sqrt{2 + a_{k-1}}$ for all $k > 1$.

5. The Problem of Irrationality

mations. A *Newton iterate* is defined as:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

Each guess x_{k+1} gives a successive improvement over the previous guess x_k. In essence, we are using the *slope of the line* at the evaluation point to guide the next guess. The function begins with an initial guess $x_0 = 1.0$ that it uses to compute better approximations. `sqrt()` is sufficiently calculated once the value converges, *i.e.* when the difference between consecutive approximations is sufficiently small. In this case, $f(x) = x^2 - y$, so you can see that $f(x) = 0$ when $x = \sqrt{y}$.

a.

```python
def sqrt(x):
    z = 0.0
    y = 1.0
    while abs(y - z) > epsilon:
        z = y
        y = 0.5 * (z + x / z)
    return y
```

6. Your Task
    a. Implement math functions to mimic <math.h>
    b. Files:
        i. e.c
            1. 2 functions, e() and e_terms() - first computes e using taylor series in #3 while second returns number of computed items
        ii. madhava.c
            1. 2 functions, pi_madhava() and pi_madhava_terms() - find pi with madhava series + track computed terms with static variables like in e.c
        iii. euler.c
            1. pi_euler() and pi_euler_terms() - find pi with solution to Basel problem + return computed terms
        iv. bbp.c
            1. pi_bbp() and pi_bbp_terms() - find pi with bbp formula + computed terms
        v. viete.c
            1. pi_viete() and pi_viete_factors() - find pi with viete + computed factors
        vi. newton.c
            1. sqrt_newton() and sqrt_newton_iters() - use python code for sqrt function + track iterations and return
        vii. mathlib-test.c
            1. Main test harness for implemented math library