Assignment 2: A Little Slice of π Design

Thadeus Ballmer

8 October 2021

$$e^{i\pi} + 1 = 0$$

1 Introduction

This program implements various functions including e^x and $\sqrt{x}$ just like in <math.h>, and then uses these functions to approximate the values of e and π. There are several different approximations implemented in this program. Each approximation method has its own pros and cons, so the test harness **mathlib-test.c** compares the approximations to the <math.h> values of e and π.

2 Math Functions

2.1 e.c

This file contains functions **e()** and **e_terms()**.

2.1.1 e()

e() returns a **double** value of the approximation of e using the Taylor series:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}$$

We do not actually need to program a factorial function in order to calculate this series. By storing the value of the previous term (beginning at $0! = 1$) and multiplying the value by the current term, the factorial operator functions perfectly. In every approximation in this program, the calculations are halted at $\epsilon = 10^{-14}$. With these given parameters, the structure of the **e()** function should look like:

```
for(Loop until current value is less than or equal to EPSILON) {
    if (Current iteration == 0) {
        Factorial value = 1
        Current value = 1
    } else {
```

```
        Factorial value *= Current iteration
        Current value = 1 / Factorial value
    }
    Number of calculated terms += 1
    Total value += Current value
}
return Total value
```

2.1.2 e_terms()

e_terms() returns an **int** of calculated terms from **e()**.

2.2 madhava.c

This file contains functions **pi_madhava()** and **pi_madhava_terms()**.

2.2.1 pi_madhava()

This function returns a **double** value of the approximation of π using the Madhava series:

$$\pi = \sqrt{12} \sum_{k=0}^{\infty} \frac{(-3)^{-k}}{2k+1}$$

Functions in this assignment may not access the <math.h> library except for comparing approximations in **mathlib-test.c**, so the square root function needs to be implemented (implementation design appears later in this document). **pi_madhava()** structure should look like:

```
for(Loop until current value is less than or equal to EPSILON) {
    if (Current iteration == 0) {
        Exponent value = 1
        Current value = 1
    } else {
        Exponent value *= -3
        Current value =
            1 / (exponent value * (2 * current iteration + 1))
    }
    Number of calculated terms += 1
    Total value += Current value
}
Total value *= sqrt_newton(12)
return Total value
```

2.2.2 pi_madhava_terms()

pi_madhava_terms() returns an **int** of calculated terms from **pi_madhava()**.

2.3 euler.c

This file contains functions **pi_euler()** and **pi_euler_terms()**.

2.3.1 pi_euler()

This function returns a **double** value of the approximation of π using Euler's solution:

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

pi_euler() function structure:

```
for(Loop until current value is less than or equal to EPSILON) {
    current value = 1 / (current iteration * current iteration)
    Number of calculated terms += 1
    Total value += Current value
    Total value *= 6
}
Total value = sqrt_newton(total value)
return Total value
```

2.3.2 pi_euler_terms()

pi_euler_terms() returns an **int** of calculated terms from **pi_euler()**.

2.4 bbp.c

This file contains functions **pi_bbp()** and **pi_bbp_terms()**.

2.4.1 pi_bbp()

This function returns a **double** value of the approximation of π using the Bailey-Borwein-Plouffe fourmila in *Homer normal form*:

$$\pi = \sum_{k=0}^{\infty} 16^{-k} \left(\frac{(k(120k + 151) + 47)}{k(k(k(512k + 1024) + 712) + 194) + 15} \right)$$

pi_euler() function structure:

```
i = current iteration
for(Loop until current value is less than or equal to EPSILON) {
    fraction value =
        ((i*(120*i+151)+47)/(i*(i*(i*(512*i+1024)+712)+194)+15))
    if (current iteration == 0) {
        exponent value = 1
    } else {
        exponent value *= (1/16)
```

```
    }
    current value = exponent value * fraction value
    Number of calculated terms += 1
    Total value += Current value
}
return Total value
```

2.4.2 pi_bbp_terms()

pi_bbp_terms() returns an **int** of calculated terms from **pi_bbp()**.

2.5 viete.c

This file contains functions **pi_viete()** and **pi_viete_factors()**.

2.5.1 pi_viete()

This function returns a **double** value of the approximation of π using Viete's formula:

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

where $a_1 = \sqrt{2}$ and $a_k = \sqrt{2 + a_{k-1}}$ for all $k > 1$. Since the value of π is wanted by itself, the program implementation uses the reciprocal factors during computation. Once adequate computation is reached, the final approximation of τ is multiplied by 2 to reach π. The implementation looks like:

```
rec(x) {
    if (x == 1) {
        return sqrt_newton(2)
    } else {
        return sqrt_newton(2 + rec(x-1))
    }
}
pi_viete() {
    for(Loop until (previous value - current value) is less than EPSILON) {
        if (current iteration == 0) {
            current_value = 1
        } else {
            previous_value = current_value
        }
        current_value *= (rec(current iteration) / 2)
        num_factors += 1
    }
    pi = (2 / current_value)
    return pi
}
```

2.5.2 pi_viete_factors()

This function returns an **int** of calculated factors from **pi_viete()**.

2.6 newton.c

This file contains the functions **sqrt_newton()** and **sqrt_newton_items()**.

2.6.1 sqrt_newton()

This function returns a **double** value of the approximation of the $\sqrt{x}$ using the *Newton-Raphson method*:

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

Since we are using this method to calculate the square root of x, $f(x) = x^2 - y$ so that $f(x) = 0$ when $x = \sqrt{y}$. This pseudocode is based off of Professor Long's python sqrt function:

```
sqrt(x){
    z = 0.0
    y = 1.0
    items = 0
    while (EPSILON < absolute(y - z) > ) {
        z = y
        y = 0.5 * (z + x / z)
        items += 1
        number of iterations += 1
    }
    return y
}
```

The absolute function called in this code is defined in the provided mathlib.h file as such:

```
static inline double absolute(double x) { return x < 0.0 ? -x : x; }
```

2.6.2 sqrt_newton_items()

This function returns an **int** of calculated items from **sqrt_newton()**.

3 mathlib-test.c

This file is a test harness which supports the following command-line options:

- -a: Run 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 Viete π approximation test.
- -n: Runs Newton-Raphson square root approximation tests.
- -s: Enable printing of statistic to see computed terms and factors for each tested function.
- -h: Display a help message detailing program usage.

The output of this test harness compares the results of the **math.h** function to the assignment functions. The output should print the difference between these two results. This test harness pseudo-code is based off Professor's Long's pseudo-code in the assignment file:

```
main() {
    is_e = is_b = is_m = is_r = is_v = is_n = is_s = is_h = false
    is_null = true
    while(Loop through getopt OPTIONS) {
        switch(opt) {
        case 'a'
            is_e = true
            is_b = true
            is_m = true
            is_r = true
            is_v = true
            is_n = true
            is_null = false
        case 'e'
            is_e = true
            is_null = false
        case 'b'
            is_b = true
            is_null = false
        case 'm'
            is_m = true
            is_null = false
        case 'r'
            is_r = true
            is_null = false
        case 'v'
            is_v = true
            is_null = false
        case 'n'
            is_n = true
```

```
        is_null = false
    case 's'
        is_s = true
        is_null = false
    case 'h'
        is_h = true
        is_null = false
    }
}
if (is_e) {
    print(output of e() and diff with math_e)
    if (is_s) {
        print(output of e_terms())
    }
}
if (is_b) {
    print(output of pi_bbp() and diff with math_pi)
    if (is_s) {
        print(output of pi_bbp_terms())
    }
}
if (is_m) {
    print(output of pi_madhava() and diff with math_pi)
    if (is_s) {
        print(output of pi_madhava-terms())
    }
}
if (is_r) {
    print(output of pi_euler() and diff with math_pi)
    if (is_s) {
        print(output of pi_euler_terms())
    }
}
if (is_v) {
    print(output of pi_viete() and diff with math_pi)
    if (is_s) {
        print(output of pi_viete_terms())
    }
}
if (is_n) {
    print(output of sqrt_newton() and diff with math_pi)
    if (is_s) {
        print(output of sqrt_newton_terms())
    }
}
```

```
    if (is_h or is_null) {
        print(Information on using mathlib-test)
    }
}
```