Design Document - Assignment 2

Hsiang Yun Lu

January 30, 2023

1 Purpose of Programs

There are seven `C` programs. Six of them are implementation of different mathematical formulas which are used to compute the fundamental constants e and π. The `mathlib-test.c` is a test harness that compares the created implemented functions with that of the math library's.

2 Files To Be Included in Directory *asgn2*

- `newton.c`
  - This file contains two functions. One implements the square root approximation using Newton's method and the other returns the number of computed iterations
- `e.c`
  - This file contains two functions. One implements the Taylor series to approximate Euler's number e and the other returns the number of computed terms
- `madhava.c`
  - This file contains two functions. One implements the Madhava series to approximate π and the other returns the number of computed terms
- `euler.c`
  - This file contains two functions. One implements the Euler's solution to approximate π and the other returns the number of computed terms.
- `bbp.c`
  - This file contains two functions. One implements the Bailey-Borwein-Plouffe formula to approximate π and the other returns the number of computed terms.
- `viete.c`

- This file contains two functions. One implements the Viète's formula to approximate π and the other returns the number of computed factors.

- `mathlib-test.c`
  - The program with the main() function which tests each of created math library functions by allowing command-line options
- `mathlib.h`
  - The header file that contains the interface of the created math library (the above programs)
- `Makefile`
  - This file taht compiles the math library programs and builds the `mathlib-test` executable
- `README.md`
  - This file describes how to use all the programs and `Makefile`, including explanations on command-line options
- `DESIGN.pdf`
  - This file describes the design and design process for all the proigrams
  - Include pseudocode
- `WRITEUP.pdf`
  - Include the graphs that displays the difference between the values reported by the created math functions and that of the math library's
  - Include analysis and explanations for any discrepancies and findings on the testing of created math library

3 Design Process

Since there is a function to return the number of computed terms in each small program, I set the `static` counter in the beginning. According to each formula, different x or y variables are defined to represent k in the formulas or the previous value of power $k-1$. The calculation in the WHILE loop will continue until the difference between previous value (last) and current value (current) is smaller than EPSILON.

For the `mathlib-test.c` program, I first create function for each small approximation program. In each function, an integer k is used to control whether the function prints only the line of number of iterations. A *usage* function is created and used to print program synopsis and usage. The *main* function first checks the arguments. It prints program synopsis and usage if there is no argument. I then use a integer *present* to represent the existence of argument "-s". If "-s" exists, all the k will be set to 1 and will print the requested test results and their numbers of iterations. Otherwise, all the k will be set to 1 and print only the requested test result lines.

4 Design/Structure

(1) `newton.c`

SET counter to 0

function sqrt_newton(x)
  SET $s \leftarrow 1$, $y \leftarrow 1$, $z \leftarrow 0$
  WHILE $x > 4$ do
    $x \leftarrow x/4$
    $s \leftarrow s \times 2$
  END WHILE
  WHILE $|y - z| >$ EPSILON do
    $z \leftarrow y$
    $y \leftarrow \frac{1}{2}(y + \frac{x}{y})$
    counter ← counter + 1
  END WHILE
  RETURN s × y
END function

function sqrt_newton_iters()
  RETURN counter
END function

(2) `e.c`

SET counter to 0

function e()
  SET $x \leftarrow 1$, $y \leftarrow 1$, $last \leftarrow 0$, $current \leftarrow 1$
  WHILE $|current - last| >$ EPSILON do
    last ← current
    current ← current + $\frac{1}{x \times y}$

```
        y ← x × y
        x ← x + 1
        counter ← counter + 1
    END WHILE
    RETURN current
END function


function e_terms()
    RETURN counter
END function
```

(3) `madhava.c`

```
SET counter to 0

function pi_madhava()
    SET x ← 1 , y ← 1, last ← 0, current ← 1
    WHILE |current − last| > EPSILON do
        last ← current
        current ← current + 1/(2×x+1) × y × 1/(−3)
        y ← y × 1/(−3)
        x ← x + 1
        counter ← counter + 1
    END WHILE
    RETURN current × √12
END function


function pi_madhava_terms()
    RETURN counter
END function
```

(4) `euler.c`

```
SET counter to 0

function pi_euler()
    SET x ← 2 , last ← 0, current ← 1
    WHILE |current − last| > EPSILON do
```

last ← current
current ← current + $\frac{1}{x \times x}$
x ← x + 1
counter ← counter + 1
END WHILE
RETURN $\sqrt{current \times 6}$
END function

function pi_euler_terms()
RETURN counter
END function

(5) `bbp.c`

SET counter to 0

function pi_bbp()
SET $x \leftarrow 1$, $y \leftarrow 1$, $last \leftarrow 0$, $current \leftarrow 4 - \frac{1}{2} - \frac{1}{5} - \frac{1}{6}$
WHILE $|current - last|$ > EPSILON do
last ← current
current ← current + $y \times \frac{1}{16} \times (\frac{4}{8x+1} - \frac{2}{8x+4} - \frac{1}{8x+5} - \frac{1}{8x+6})$
y ← $y \times \frac{1}{16}$
x ← x + 1
counter ← counter + 1
END WHILE
RETURN current
END function

function pi_bbp_terms()
RETURN counter
END function

(6) `viete.c`

SET counter to 0

function pi_viete()
SET $x \leftarrow \sqrt{2}$, $last \leftarrow 0$, $current \leftarrow \frac{\sqrt{2}}{2}$

WHILE $|current - last|$ > EPSILON do
last ← current
current ← current × $\frac{\sqrt{2+x}}{2}$
x ← $\sqrt{2+x}$
counter ← counter + 1
END WHILE
RETURN $\frac{1}{\frac{current}{2}}$
END function

function pi_viete_factors()
RETURN counter
END function

(7) `mathlib-test.c`

DEFINE command-line options “aebmrvnsh”

function usage(program)
print(program synopsis and usage)
END function

M_PI ← π constant value in *math.h*
M_E ← e constant value in *math.h*

function get_e(k)
IF k = 1
PRINT e() output, M_E, difference
ELSE
PRINT e() output, M_E, difference
PRINT e_terms() output
END function

function get_m(k)
IF k = 0
PRINT madhava() output, M_PI, difference
ELSE
PRINT madhava() output, M_PI, difference
PRINT madhava_terms() output

```
END function


function get_r(k)
    IF k = 0
        PRINT euler() output, M_PI, difference
    ELSE
        PRINT euler() output, M_PI, difference
        PRINT euler_terms() output
END function


function get_r(k)
    IF k = 0
        PRINT euler() output, M_PI, difference
    ELSE
        PRINT euler() output, M_PI, difference
        PRINT euler_terms() output
END function


function get_b(k)
    IF k = 0
        PRINT bbp() output, M_PI, difference
    ELSE
        PRINT bbp() output, M_PI, difference
        PRINT bbp_terms() output
END function


function get_v(k)
    IF k = 0
        PRINT viete() output, M_PI, difference
    ELSE
        PRINT viete() output, M_PI, difference
        PRINT viete_terms() output
END function


function get_n(k)
    IF k = 0
        FOR i = 1 to 10, move 0.1 each time
```

```
            PRINT sqrt_newton() output, sqrt() output, difference
        END FOR
    ELSE
        FOR i = 1 to 10, move 0.1 each time
            PRINT sqrt_newton() output, sqrt() output, difference
            PRINT viete_terms() output
    END IF
END function


function main(argument)
    SET present ← 0

    IF no argument
        usage()
    END IF

    FOR each read-in argument
        IF argument == "-s"
            present ← 1
        END IF
    END FOR

    IF present == 1
        WHILE there is at least an argument
            SWITCH argument
                case argument "e" do get_e(1)
                case argument "r" do get_r(1)
                case argument "b" do get_b(1)
                case argument "m" do get_m(1)
                case argument "v" do get_v(1)
                case argument "n" do get_n(1)
                case argument "a" do
                    get_e(1)
                    get_r(1)
                    get_b(1)
                    get_m(1)
                    get_v(1)
                    get_n(1)
                case argument "s" do
                    IF there is only one argument
                        usage()
```

```
                ELSE
                    break
                END IF
            case argument “h” do usage()
            default do usage()
                return EXIT_FAILURE
        END SWITCH
    END WHILE
ELSE
    WHILE there is at least an argument
        SWITCH argument
            case argument “e” do get_e(0)
            case argument “r” do get_r(0)
            case argument “b” do get_b(0)
            case argument “m” do get_m(0)
            case argument “v” do get_v(0)
            case argument “n” do get_n(0)
            case argument “a” do
                get_e(0)
                get_r(0)
                get_b(0)
                get_m(0)
                get_v(0)
                get_n(0)
            case argument “s” do
                IF there is only one argument
                    usage()
                ELSE
                    break
                END IF
            case argument “h” do usage()
            default do usage()
                return EXIT_FAILURE
        END SWITCH
    END WHILE
END IF
END function
```

5 Credit

- Implementation in `newton.c` was inspired and rewrited from the python code in the assignment specification pdf, and the `indirect.c` program provided by Dr. Long
- The command-line option use took reference from `monte_carlo.c` program