Assignment 2
A Little Slice of Pi

By Jiancheng (Jason) Xiong
CSE 13S Spring 2023

Due May 3rd 2023, 11:59 PM

1 Purpose

The purpose of this assignment is to create a program that calculates the value of pi and e using a variety of arithmetic sequences run by a main function.

2 Using the Program

In order to use the program, compile and run the file mathlib-test.c. In order to run the program, run ./mathlib-test and specify the commands in which you want to run. The available options are;

-a; running all tests
-e; runs e approximation test
-b; runs the Bailey Borwein Plouffe pi approximation test
-m; runs the Madhava pi approximation test
-r; runs the Euler pi approximation test
-v; runs the Viete pi approximation test
-w; runs the Wallis pi approximation test
-n; runs the Newton-Raphson square root approximation test
-s; enables printing of statistics to see computed terms and factors for all tested functions
-h; displays a help message detailing program usage.

3 Program Design

The main program will be in the file matlib-test.c, while each of the separate individual sequence tests will be in their respective files. Each separate sequence test will simply be a function which the main mathlib file runs.

4 Data Structures

The main data structures will be the usage of for loops to run the sequences an undefined number of times until the margin of error is lower than an epsilon we will define to be 10^-14. The sequence functions will take no input and output an integer that is within the epsilon margin of error from the actual value.

5 Algorithms

Main Routine:

Using getopt, iterate through the console command and retrieve all the requested commands

Instead of directly running the sequential function when the command is detected, instead use a temporary variable so that we can call them in the expected order.

Run each requested sequence until the margin of error is below 10^-14

Pseudocode for the sequences:

Initialize a variable k to keep track of iterations and a sum variable for total output

Do:

Plug in k into the algorithm and add it to the running total

Increment k by 1

while:

K plugged into the equation is greater than epsilon, otherwise break

If asked (-s), print the output number and the number of iterations they took.

6 Function Descriptions

Main function: takes its arguments from the command line, using getopt to know what sequential functions to run. Takes one (command line) input and outputs the value of the sequences requested.

For all sequence functions: takes no input and returns an output that is within the epsilon margin of error from the expected value. Prints nothing unless the command -s is called, then the value and number of iterations taken is printed.

7 Results

When the program is run without a command prompt, a help menu will be shown.

```
SYNOPSIS:
        A number of sequential tests for e and pi.
USAGE
        ./mathlib-test-x86 -[aebmrvnsh]
OPTIONS
        -a   Runs all tests.
        -e   Runs e test.
        -b   Runs BBP pi test.
        -m   Runs Madhava pi test.
        -r   Runs Euler pi test.
        -v   Runs Viete pi test.
        -w   Runs Wallis pi test.
        -n   Runs Newton square root tests.
        -s   Print verbose statistics.
        -h   Display program synopsis and usage.
```

The commands that can be specified are;

-a (all tests)

```
jasonx@cse13s:~/cse13s/asgn2$ ./mathlib-test -a
e() = 2.718281828459043, M_E = 2.718281828459045, diff = -0.000000000000000
pi_bbp() = 3.141592653589793, M_PI = 3.141592653589793, diff = 0.000000000000000
pi_madhava() = 3.141592653589800, M_PI = 3.141592653589793, diff = 0.00000000000
0002
pi_euler() = 3.141592558095903, M_PI = 3.141592653589793, diff = 0.0000000954938
81
pi_viete() = 3.141592653589789, M_PI = 3.141592653589793, diff = 0.0000000000000
00
pi_wallis() = 3.141592495717063, M_PI = 3.141592653589793, diff = 0.000000157872
698
sqrt_newton(0.00) = 0.000000000000007, sqrt(0.00) = 0.000000000000000, diff = -0
.000000000000007
sqrt_newton(0.10) = 0.316227766016838, sqrt(0.10) = 0.316227766016838, diff = 0.
000000000000000
sqrt_newton(0.20) = 0.447213595499958, sqrt(0.20) = 0.447213595499958, diff = -0
.000000000000000
sqrt_newton(0.30) = 0.547722557505166, sqrt(0.30) = 0.547722557505166, diff = 0.
000000000000000
sqrt_newton(0.40) = 0.632455532033676, sqrt(0.40) = 0.632455532033676, diff = 0.
000000000000000
sqrt_newton(0.50) = 0.707106781186547, sqrt(0.50) = 0.707106781186548, diff = 0.
000000000000000
```

-e, running just the e test

```
jasonx@cse13s:~/cse13s/asgn2$ ./mathlib-test -e
e() = 2.718281828459043, M_E = 2.718281828459045, diff = -0.000000000000000
```

-b, running just the Bbp pi test

```
jasonx@cse13s:~/cse13s/asgn2$ ./mathlib-test -b
pi_bbp() = 3.141592653589793, M_PI = 3.141592653589793, diff = 0.000000000000000
```

-m, running just the Madhava pi test

```
jasonx@cse13s:~/cse13s/asgn2$ ./mathlib-test -m
pi_madhava() = 3.141592653589800, M_PI = 3.141592653589793, diff = 0.00000000000
0002
```

-r, running just the Euler pi test

```
jasonx@cse13s:~/cse13s/asgn2$ ./mathlib-test -r
pi_euler() = 3.141592558095903, M_PI = 3.141592653589793, diff = 0.0000000954938
81
```

-v, running just the Viete pi test

```
jasonx@cse13s:~/cse13s/asgn2$ ./mathlib-test -v
pi_viete() = 3.141592653589789, M_PI = 3.141592653589793, diff = 0.0000000000000
00
```

-w, running just the Wallis pi test

```
jasonx@cse13s:~/cse13s/asgn2$ ./mathlib-test -w
pi_wallis() = 3.141592495717063, M_PI = 3.141592653589793, diff = 0.000000157872
698
```

-n, running just the Newton root test from a range of [0,10)

```
jasonx@cse13s:~/cse13s/asgn2$ ./mathlib-test -n
sqrt_newton(0.00) = 0.000000000000007, sqrt(0.00) = 0.000000000000000, diff = -0
.000000000000007
sqrt_newton(0.10) = 0.316227766016838, sqrt(0.10) = 0.316227766016838, diff = 0.
000000000000000
sqrt_newton(0.20) = 0.447213595499958, sqrt(0.20) = 0.447213595499958, diff = -0
.000000000000000
sqrt_newton(0.30) = 0.547722557505166, sqrt(0.30) = 0.547722557505166, diff = 0.
000000000000000
sqrt_newton(0.40) = 0.632455532033676, sqrt(0.40) = 0.632455532033676, diff = 0.
000000000000000
sqrt_newton(0.50) = 0.707106781186547, sqrt(0.50) = 0.707106781186548, diff = 0.
000000000000000
sqrt_newton(0.60) = 0.774596669241483, sqrt(0.60) = 0.774596669241483, diff = 0.
000000000000000
sqrt_newton(0.70) = 0.836660026534076, sqrt(0.70) = 0.836660026534076, diff = 0.
000000000000000
sqrt_newton(0.80) = 0.894427190999916, sqrt(0.80) = 0.894427190999916, diff = 0.
000000000000000
sqrt_newton(0.90) = 0.948683298050514, sqrt(0.90) = 0.948683298050514, diff = 0.
000000000000000
sqrt_newton(1.00) = 1.000000000000000, sqrt(1.00) = 1.000000000000000, diff = -0
```

Running the command -s in conjunction with any of the previous commands will allow for the number of terms taken to also be shown.

```
jasonx@cse13s:~/cse13s/asgn2$ ./mathlib-test -b -s
pi_bbp() = 3.141592653589793, M_PI = 3.141592653589793, diff = 0.000000000000000
pi_bbp() terms = 12
```

Running the command -h will reshow the help menu with the explanation of each command. The output of each sequential function is the calculated approximation of the value along with the value from the standard math library. The diff shows the difference between the two values.

8 References

"C - Switch Statement." *Tutorials Point*, https://www.tutorialspoint.com/cprogramming/switch_statement_in_c.htm.

https://pubs.opengroup.org/onlinepubs/009695399/basedefs/math.h.html.