Dmitrius Agoston
Prof. Darrell Long
10 October 2021

CSE 13S Fall 2021
Assignment 2: A Little Slice of Pi
Design Document

Description of Program:
The purpose of this program is to implement some mathematical functions and compare the accuracy of their results to the math.h library. The functions include approximating both pi and e as well as the square root function. There is also a test harness with the purpose of comparing the functions made to the equivalent ones in the default math library.

Layout/Pseudocode:

- e.c
  This file contains two different functions. The first being e() which gets the approximate value of e using the Taylor series. The second function, e_terms(), returns the number of terms that was used to get to the approximation of e

  Static variable for number of terms
  Define e
      Variable for e
      Variable for previous term
      Variable for previous e
      Keep approximating e until margin of difference is < epsilon
          Add 1 to count
          Multiply previous term by current term
          Set previous e to current e
          Add the current term to e
      Return e

  Define e_terms
      Return number of terms used

- madhava.c
  This file contains the functions pi_madhava() and pi_madhava_terms(). The first function will approximate the value of pi using the madhava series and the second will return the number of terms used to get the approximation.

Static variable for number of terms
Define pi_madhava
Previous pi variable
Current pi variable
Keep approximating pi until the margin of error is < epsilon
Add 1 to count
Previous pi approximation set equal to current approximation
Numerator variable set equal to -3
Factor variable set equal to -3
Loop current iteration number of times
Numerator multiplied by factor
Current pi set to (1 / numerator) / (2.0 * current iteration + 1)
Return current pi multiplied by the square root of 12

Define pi_madhava_terms
Return number of terms used

- euler.c

This file contains two functions called pi_euler() and pi_euler_terms. The first function will calculate the approximation of pi using euler's solution. The second will return the numbers of terms used to reach that approximation.

Static variable for number of terms
Define pi_euler
Previous pi variable
Current pi variable
Keep approximating pi until margin of error is < epsilon
Add 1 to count
Set previous pi to current pi
Temporary variable with value of current iteration squared
Set current pi equal to itself plus 1 divided by current term squared
Return approximation multiplied by 6 and square root

Define pi_euler_terms
Return number of terms used

- bbp.c

This file contains both functions called pi_bbp() and pi_bbp_terms(). The purpose of the first function is to approximate the value of pi using the Bailey-Borwein-Plouffe formula. The following function will return the number of terms used in the approximation.

Static variable for number of terms
Define pi_bbp
    Previous pi variable
    Current pi variable
    Keep approximating pi until margin of error is < epsilon
        Add 1 to count
        Set previous pi to current pi
        Variable for exponential value
        Loop current number of iterations
            Multiply exponential by 16
        Check if current iteration is first
            Set exponential value to 1
        Variable for numerator set equal to the math of the equations numerator
        Variable for denominator set equal to the math of the equations denominator
        Current pi set equal to itself plus (1/exponential) * multiplied by the numerator / denominator
    Return current pi approximation

Define pi_bbp_terms
    Return number of terms used

- viete.c
  The file contains the functions pi_viete() and pi_viete_factors().pi_viete() will approximate the value of pi using Viete's formula. pi_viete_factors() will return an int that represents the number of factors used to reach the approximation.

  Static variable for number of terms
  Define pi_viete
      Previous approximation
      Current approximation
      Variable for nested square root
      While the approximation margin of error is > epsilon keep going
          Add 1 to count

Square root the nested root variable
Set previous approximation equal to current approximation
Set current approximation equal to nested root variable divided by 2
Return 2 divided by current approximation

Define pi_viete_factors
Return number of terms used

- newton.c
  The file contains the functions sqrt_newton() and sqrt_newton_terms(). sqrt_newton will approximate the square root of the argument passed to it. The approximation will be made using the Newton-Raphson method. sqrt_newton_terms will return the number of terms used to make the approximation.

  Static variable for number of iterations
  Define sqrt_newton
  Previous approximation
  Current approximation
  While the approximation margin of error is > epsilon keep going
  Add 1 to count
  Set previous approximation equal to current approximation
  Set current approximation equal to 0.5 multiplied by
  (previous square root + number given / previous square root)
  Return current approximation

  Define sqrt_newton_terms
  Variable for current iterations equal to number of iterations
  Iterations set equal to 0
  Return number of iterations complete

- mathlib-test.c
  This file acts as the test harness for the previous files and supports command line options

  Define main
  Create boolean for each potential command
  While loop using getopt, != -1
  If getopt is equal to any command
  Set command equal to true

```
If -a is input
        Set all tests equal to true
If -e is input
        Run e test
        If -s is input
                Print statistics as well
If -b is input
        Run bbp test
        If -s is input
                Print statistics as well
If -m is input
        Run madhava test
        If -s is input
                Print statistics as well
If -r is input
        Run euler test
        If -s is input
                Print statistics as well
If -v is input
        Run viete test
        If -s is input
                Print statistics as well
If -n is input
        Run newton test
        If -s is input
                Print statistics as well
Any other input or no input
        Print out help instructions
```

Error Handling:

The main error that I had to deal with was based on the user's inputs. For example if they choose to run the executable with no other commands or if they ran an invalid command, I had the program print out the help statement. The other error handling that I had to deal with was running the correct tests only once and also printing the statistics only when need be. I basically restructured the whole test so it would work as shown in the example binary.

Credit:

When creating this code I largely used asgn2.pdf as my main source of reference for creating this program