Euler 26

Version 1:

1. Get input from user
2. Each line has two numbers, first represents boundary that is denominator of largest fraction, second is boundary that is denominator of smallest fraction
3. Step through each integer denominator possibility between the two values and print the decimal representation, with repeating decimals represented as (the digit(s) repeated)
4. Go step step 1 until next input is not 0 0

Version 2:

1. main:
    a. takes all lines of input *x y*. sends the output of process(x, y) to out file
2. process(x, y):
    a. while x < y it determines the repitend of the number. it does this by calling getRepeat(x). for every iteration of x it will add the repeating portion to a string and return that string
3. getRepeat(x):
    a. takes in int x and finds the repeating decimal equivalent of 1/x. it does this by doing long division (modulo division) until the numbers are repeating or a whole divisor is found

Version 3:

1. main:
    a. get all lines of input as ints
    b. if the first element is not 0 the testing is not done and you can
        i. output result of sendind the next two elements to process(x, y) through opening output file and writing lines
2. process(x, y):
    a. stri = “”
    b. while x < y:
        i. append “1/x = getRepeat(x)\n” to stri
    c. return stri
3. getRepeat(x):
    a. lis = []
    b. stri = “0.”
    c. done = False
    d. base = 10
    e. count=0
    f. while not done:
        i. if base not in lis
            1. append base to lis
        ii. else
            1. break
        iii. temp = x

iv. count=0
v. while count <= base:
    1. count+=1
    2. temp+=x
vi. stri += count
vii. base = (base-(x*count))*10

g. count=0
h. while(count<len(lis)):
    i. if lis[count]=base
        1. break
    ii. count+=1
i. insert “(“ into stri at count + 2
j. insert “)” into stri at len(stri)
k. return stri

Test Case 1
Validates Software Requirements 1 - 7

Input Data (in a file named euler026-in.txt)
2 4
5 10
0 0

Expected Output or Behavior (sent to a file named euler026-out.txt)
Test Case 1: a = 2, b = 4
1/2 = 0.5
1/3 = 0.(3)
1/4 = 0.25
Longest Recurring Cycle: 1/3, 1 digit(s)
Test Case 2: a = 5, b = 10
1/5 = 0.2
1/6 = 0.1(6)
1/7 = 0.(142857)
1/8 = 0.125
1/9 = 0.(1)
1/10 = 0.1
Longest Recurring Cycle: 1/7, 6 digit(s)

Actual output (euler026-in.txt contents)
Test Case 1: a = 2, b = 4
1/2 = 0.5
1/3 = 0.(3)
1/4 = 0.25

Longest recurring cycle: 1/3, 1 digit(s)
Test Case 2: a = 5, b = 10
1/5 = 0.2
1/6 = 0.1(6)
1/7 = 0.(142857)
1/8 = 0.125
1/9 = 0.(1)
1/10 = 0.1
Longest recurring cycle: 1/7, 6 digit(s)
Test Case Results: Passed

Egyptian Hieroglyphics

Version 1:

1. Get input from user
2. Each line has two numbers, first is numerator and second is denominator
3. Find unit fractions that add up to the given fraction and print it out
4. Go to step 1 while next input is not 0 0

Version 2:

1. *main*:
    a. takes all lines of input *x y*, creates a fraction out of each with each being x/y and send these to *process*(*x*) which gives back a string of the unit factors. print this string out.
2. *process*:
    a. process will take in a fraction and create a new string that starts with that fraction and then adds unit factors as it gets them. to do this is will create a counter that starts at 2 for the largest possible fraction and continuall call *getUnitFraction* with the given fraction and the counter until it finds that the fraction – (all of the outputs of *getUnitFraction*) is 0. if any return from *getUnitFraction* is 0 then it will erase the last unit factor that was returned, and start from there instead with that counter. then it returns the cumulative string to main.
3. *getUnitFraction*
    a. will take in a fraction and a counter variable and find the next unit factor of that number. it returns the denominator, so if it is ½ it will return 2. if the next unit factor has a denominator greater than 1000000 it will return 0, which *process* will know is an issue and handle appropriately.

Version 3:

1. *main*:

a. open file and get all lines of content <- *content*
b. empty list <- *lis*
c. for numbers in range[0,length of content) <- *x*
    i. append *x* to *lis*
d. if x[0] is 0 return
e. *currentNum* -> *x*[0] / *x*[1]
f. *i* -> 2
g. while *True*
    i. print *process*(*currentNum*)
    ii. *currentNum1* -> x[i]
    iii. *currentNum2* -> x[i+1]
    iv. if *currentNum1* or *currentNum2* is 0 break from loop
    v. *currentNum* -> *currentNum1*/*currentNum2*
    vi. *i+=2*

2. *process(fraction):*
    a. count -> 2
    b. str -> “”+fraction+” =”
    c. done -> False
    d. unitFractionList -> []
    e. while not done
        i. *getUnitFraction* (fraction, count) -> tempFraction
        ii. if tempFraction is 0
            1. count = last element of list + 1
            2. delete last element of list
        iii. else
            1. add tempFraction to unitFractionList
            2. add “ + 1/tempFraction” to str
            3. one -> 1
            4. for elements in unitFractionList
                a. one = one – 1.0/element
                b. if one = 0
                    i. done = True
        iv. count+=1
    f. return str
3. getUnitFraction(fraction, count):
    a. tempFrac -> 1/count
    b. while True
        i. if fraction – tempFrac >= 0
            1. return count
        ii. if count > 1000000
            1. return 0
        iii. count+=1

Test Case 1
Validates Software Requirements 1 - 7

Input Data (in a file named egyptian-in.txt)
5 8
2 7
9 20
17 69
0 0

Expected Output or Behavior (sent to a file named egyptian-out.txt)
5/8 = 1/2 + 1/8
2/7 = 1/4 + 1/28
9/20 = 1/3 + 1/9 + 1/180
17/69 = 1/5 + 1/22 + 1/1086 + 1/686895

Actual output (Egyptian-out.txt contents)
5/8 = 1/2 + 1/8
2/7 = 1/4 + 1/28
9/20 = 1/3 + 1/9 + 1/180
17/69 = 1/5 + 1/22 + 1/1086 + 1/686895
Test Case Results: Passed