Version 1:

- Take in input from file
    - Find prime factors of input
    - Write these prime factors to file
    - Repeat for all lines of input

Version 2:

1. main:
    a. Take in input from stdin <- *input*
    b. If *isPrime(input)* create factorFrequencyList as (*input, 1*)
        i. *printPrimeFactors(factorFrequencyList)*
    c. *process(input) <- primeFactors*
    d. *makeNiceList(primeFactors) <- factorFrequencyList*
    e. *printPrimeFactors(input, factorFrequencyList)*
2. process(n)
    a. gets a list of prime factors of a number n by first getting the factors and then determining if these factors are prime. From there it will return this list of all prime factors
3. generateFactors(n)
    a. For all numbers (x) less than n, if n is divisibile by x, add x to a list. Then return that list
4. isPrime(n)
    a. Returns true if a number is prime and false if a number is not
5. makeNiceList(listOfPrimeFactors)
    a. takes a list in the form of (prime1, prime2, prime3… primen) where repeats are possible and returns a tuple of ((prime1, frequency1),(prime2,frequency2)…(primen,frequency))
6. printPrimeFactors(n, listOfTuples)
    - takes list of lists and prints them our in the form n = (factor1^frequency1)(factor2^frequency2)…(factorn^frequencyn) where when frequency is 1 the tuple is printed simply as (factorm)

Version 3

1. main
    a. open file and get all lines of input <- *content*
    b. for all elements of *content <- input*
        i. if *isPrime*(*input*)
            1. create list ((*input*,1)) <- *factorFrequencyList*
            2. *printPrimeFactors*(*factorFrequencyList*)
        ii. else
            1. *process*(*input*) <- *primeFactors*
            2. *makeNiceList(primeFactors) <- factorFrequencyList*
            3. *printPrimeFactors(input, factorFrequencyList)*
2. *process(n)*
    a. *[] <- primeFactors*
    b. *generateFactors*(*n*) <- *factors*

    c. Second and third elements of factors <- *actOnFactorList*
    d. Add first number of *actOnFactorList* to *primeFactors*
    e. if *isPrime(*second number of *actOnFactorList*)
        i. add to *primeFactors*
    f. otherwise
        i. add *process(*second number of *actOnFactor)* to *primeFactors*
    g. return *primeFactors*
3. generateFactors(n)
    a. [] <- *factors*
    b. 0 <- *count*
    c. for numbers in range [1, *n*] <- *x*
        i. if *n* % *x* == 0
            1. if the factor is the first or second
                a. add *x* to *factors*
            2. otherwise append quotient of n and last factor added to *factors*
    d. return *factors*
4. *makeNiceList(primeFactors)*
    a. *[]* <- *doneList*
    b. *[]* <- *factorFrequencyList*
    c. for each number in primeFactors <- *x*
        i. if *x* in *doneList*
            1. pass this loop (go to 4b)
        ii. 1 <- *count*
        iii. for each number in primeFactors <- *y*
            1. if *y* == *x*
                a. increment count
        iv. add (x, count-1) to *factorFrequencyList*
        v. add *x* to *doneList*
    d. return *factorFrequencyList*
5. *printPrimeFactors(input, factorFrequencyList)*
    a. write input = (factor1^frequency1)(factor2^frequency2)…(factorn^frequencyn) to file where when frequency is 1 the tuple is printed simply as (factorm) and both factor and frequency are pulled for each element in factorFrequencyList
6. *isPrime(n)*
    a. if number is 1 or 2, return true
    b. for numbers in range [2, sqrt(*n*)+1] <- *x*
        i. if *n* % *x* == 0
            1. return true
    c. return false

Test Case 1

Validates Software Requirement(s): 1-5.

Description: Validates that each line from standard input is read and that the program terminates after the last input integer is processed. Validates that the correct prime factors and their respective powers are determined. Validates that the output is correct.

Input Data (in a file named *euler003-test.in*)

```
2
3
4
12
100
1013
5000
13195
99999
1234567
56574433
600851475143
7817285266985093
```

Expected Output or Behavior (sent to a file named *euler003-test.out*)

```
2 = (2)
3 = (3)
4 = (2^2)
12 = (2^2)(3)
100 = (2^2)(5^2)
1013 = (1013)
5000 = (2^3)(5^4)
13195 = (5)(7)(13)(29)
99999 = (3^2)(41)(271)
1234567 = (127)(9721)
56574433 = (71)(463)(1721)
600851475143 = (71)(839)(1471)(6857)
7817285266985093 = (43^2)(53^3)(73^4)
```

Test Case 1

Validates software requirements: 1-6

Description: Has a variety of integers, and the algorithm is simple, so this one test case covers everything

2
3
4
12
100
1013
5000
13195
99999
1234567
56574433
600851475143
7817285266985093

Expected Output or Behavior (sent to a file named *euler003-test.out*)

2 = (2)
3 = (3)
4 = (2^2)
12 = (2^2)(3)
100 = (2^2)(5^2)
1013 = (1013)
5000 = (2^3)(5^4)
13195 = (5)(7)(13)(29)
99999 = (3^2)(41)(271)
1234567 = (127)(9721)
56574433 = (71)(463)(1721)
600851475143 = (71)(839)(1471)(6857)
7817285266985093 = (43^2)(53^3)(73^4)

Actual output (euler003-test.out contents):

2 = (2)
3 = (3)
4 = (2^2)
12 = (2^2)(3)
100 = (2^2)(5^2)
1013 = (1013)
5000 = (2^3)(5^4)
13195 = (5)(7)(13)(29)
99999 = (3^2)(41)(271)
1234567 = (127)(9721)
56574433 = (71)(463)(1721)

600851475143 = (71)(839)(1471)(6857)
7817285266985093 = (43^2)(53^3)(73^4)

Test Case Results: Passed