Prime Number algorithm

The idea behind this algorithm:

1. Check if the input is lesser than 2. If yes, print: Not a Prime.
2. Check if the input is 2, 3, 5 or 7. If yes, print: Prime.
3. Check if the input is an even number. If yes, print: Not a Prime.
4. Find the square root of the input and check if the input is a perfect square. If yes, print: Not a Prime.
5. By now, we know that the integer part of the square root is an odd natural number greater than 8.
6. Test the divisibility of the input starting with 3 and going up using odd numbers till just below the square root of the input, since if the input is not prime, at least one of its factors will be an odd number lesser than its square root.

```
# Defining the variable to accept user input. The int datatype ensures that floats are not
accepted
int testNum

# Defining two variables to store the square root of the user input in float and int forms
float  sqrtflt
int sqrt

# Defining the variable to store the highest possible integer factor for the user input. This will be
an odd number at most as great as sqrt
int  highestFactor

# Accept user input for the number to test for primality
input testNum

# Checking for the input number being less than 2 (the smallest prime) and also filtering out
negative integers
if (testNum < 2)
{
        print “The number you entered is not a Prime”
        end
}

# Checking for the input number being 2, 3, 5 or 7 (the 4 smallest prime numbers)
# Filtering out these so that the optimized code below works without having to handle
exceptions caused due to having floor of square root lesser than 3. The optimization helps
minimize number of comparison iterations
if (testNum  == 2||3||5||7)
{
        print “The number you entered is a Prime”
        end
}

# Checking for the input number being an even number
if (testNum mod 2 == 0)
{
        print “The number you entered is not a Prime”
        end
}

# At this point, we have determined that testNum is an odd natural number greater than 7
# This means that it can be product of odd numbers only, at least one of which is lesser than
than the square root of testNum

# Finding the square root of testNum. Assuming squareroot() is a function available in the
system library
sqrtflt = squareroot(testNum) # Storing square root of testNum as a float
sqrt = floor(sqrtflt) # Storing the int part of square root of testNum as an int

# Checking if square root of testNum is an integer by comparing the values of its float and int
variables. If the square root is an integer, then testNum cannot be a prime
if (sqrt == sqrtflt)
{
        print “The number you entered is not a Prime”
        end
}

# If we are here, then it means that the square root of testNum was a float and therefore
testNum is not a square number
# The number sqrt can be an odd or an even number (ex: floor of square root of 17 is 4)

# Optimizing the highest number which needs to be tested for as a factor of testNum
# Since the testNum is not a square as determined above, we don’t need to test sqrt as its factor
and we limit ourselves to testing till the highest odd number less than sqrt

# Checking if sqrt is even and setting highestFactor
if (sqrt mod 2 == 0)
        highestFactor = sqrt – 1 # This will always be 3 or larger because we have already filtered
out all primes till 7, along with all even numbers. The lowest testNum to pass through this is 17
else
        highestFactor = sqrt # This will always be 3 or larger. The lowest testNum to pass
through this is 11
```

```
# Defining the integer variable to use as loop counter
int i

# The test condition limits the calculations to highestFactor
for (i = 3; i < = highestFactor; i = i + 2)
{
        if (testNum mod i == 0)
        {
                print “The number you entered is not a Prime”
                end
        }
}

# If the program did not end during the For loop, then testNum is prime
print “The number you entered is Prime”
end
```

Author: Nitin Bhatnagar 07/02/2020