Programming Assignment #1 (due 11:59pm, 2015, 10/09)

Problem: Prime Checker

RSA is one of the first practicable public-key cryptosystems and is widely used for secure data transmission. In such cryptosystems, primes play an important role. Hence, how to efficiently separate the primes from the composite numbers is a critical job in today's internet world.

In this programming assignment, you are asked to write a C++ program to perform prime checker through a defined integer interval, and returns the number of primes.

Provided files: (1)main.cpp, (2)PrimeChecker.cpp, (3)PrimeChecker.h, (4)example

- **main.cpp –** it checks the answer of **PrimeChecker** and can be changed if necessary for you to debug.
- **PrimeChecker.cpp &.PrimeChecker.h –** it's the program file you need to implement. In these two files, it includes the targeted function **long PrimeChecker(long, long);** The function **PrimeChecker** receives two long integer a and b, and returns the number of primes between the integer interval (a~b).
- **example** – it is an exemplary input test case, which can be used to test your program. You can modify the file for the use of debug.

A partial content of **example** is shown below:

a	b	Number of Primes
2	7	4
230266478	230266980	24
1815386264	1815387188	42

The numbers a and b are the input integers of the function **PrimeChecker**, and the number of primes are the golden answers of the returned value of the function **PrimeChecker()**, which will then be used in **main()** to compare the correctness of the function **PrimeChecker()**.
Note that the minimum and maximum of the input number is 2 and 2^64 (about 1.84467441e19) respectively. (1.84467441e19 is the maximum number of unsigned long integer in UNIX/Linux).

Language
C or C++.

Platform
You may develop your software on UNIX/Linux.
Compile: $ g++ main.cpp PrimeChecker.cpp
Execution: $ ./a.out

Submission
Please update the following materials to **E3** website by the deadline, specifying your account.

(1) PrimeChecker.h

(2) PrimeChecker.cpp

Grading

(1) example correct: 60%

(2) hidden cases correct: 10%

(3) hidden cases ranking: 30% (ranking: run time only)