**CSCI 340** **Computer Assignment 3** **Spring 2019**
(10 points)

## Sieve of Eratosthenes

For this computer assignment, you are to write and implement a C++ program to find and print all prime numbers within a range `[lower upper]` using the algorithm known as the *Sieve of Eratosthenes*. The program will have basic interactive user interface to take input values of `lower` and `upper` and allow the user to continue or quit the game.

A prime number `p` is an integer greater than `1` that is divisible only by `1` and `p` (itself). The algorithm begins by initializing a set to contain all of the integers in the range `lower` to `upper`. Note that the smallest prime number is 2. If the value of `lower` is less than 2, you need to adjust that. A loop makes multiple passes over the elements in the set, using successive integer key values `2, 3, 4, …` Each pass "shakes free" nonprime numbers and lets them "filter through the sieve." At the end, only the prime numbers remain.

Begin with the integer `m = 2`, which is the smallest prime number. The pass scans the set and removes all multiples of `2`, having the form `2*k` for `k` ≥ `2`. The multiples cannot be prime numbers, because they are divisible by `2`. At the end of the pass, we have removed all of the even numbers except `2`. Next, look at the integer `m = 3`, which is a prime number. As with value `2`, remove all multiples of `3`, having the form `3*k` for `k` ≥ `3`. The multiples `12, 18,` and so forth, are even numbers, so they have already been removed from the set. The next key integer is `m = 4`, which is no longer in the set, because it was removed as a multiple of `2`. The pass takes no action. The process continues until it removes all multiples of prime numbers. In other words, for integer `m`, remove all multiples of `m`, having the form `m*k` for `k ≥ m`. The numbers that remain in the sieve are the prime numbers in the range `lower` to `upper`.

The algorithm uses an optimization feature by only looking at the key values for `m ≤ sqrt ( upper )`. To implement this, test all numbers `m` such that `m*m ≤ upper`, rather than computing the square root of `upper`.

assignment3.cc is provided to you at the directory:
`/home/turing/mhou/public/csci340spring2019.`

In this file, the main function is already implemented. You are required to implement the following functions:

- `void sieve ( set < int >& s, const int lower, const int upper )` : This routine can be used to apply the *Sieve of Eratosthenes* algorithm to remove all nonprime numbers from the integer set `s` = { `lower, …, upper` }.

- `void print_primes ( const set < int >& s, const int lower, const int upper )` : This routine can be used to print the

elements in the integer set `s` (`NO_ITEMS = 6` primes per line and `ITEM_W = 4` spaces allocated for each prime).

- `void run_game ( set<int>& s )` : This routine maintains a loop to take inputs from a user. The user will be asked for the range of the prime numbers. If the range is not valid, e.g. `lower` is greater than `upper`, the user will be asked to input again. The routine will invoke `sieve()` and `print_primes()`. Once the results are displayed, the user will be asked whether to continue or quit the game. In case of continuing the game, the user will be asked for values of the range again. The game continues until the user requests to stop.

Programming Notes:

- You are required to use the `set` container to store the prime numbers. In STL, a `set` is implemented as an associative container, and duplicate keys are not allowed.
- Include any necessary headers and add necessary global constants.
- You are not allowed to use any I/O functions from the C library, such as `scanf` or `printf.` Instead, use the I/O functions from the C++ library, such as `cin` or `cout.`
- In the final version of your assignment, you are not supposed to change existing code, including the main method, provided to you in the original source file assginment3.cc.
- To compile the source file, execute "`g++ -Wall assignment3.cc –o assignment3.exe`". This will create the executable file `assignment3.exe.` To test your program, execute "./`assignment3.exe < assignment3input.txt &> assignment3.out`", which will take redirected input from the text file `assignment3input.txt` and redirect the output (including any error messages) to the file `assignment3.out`. You can find the sample input file `assignment3input.txt` and the correct output file `assignment3.out` in the directory shown in the last page.
- Add documentation to your source file.
- Prepare your Makefile so that the TA only needs to invoke the command "`make`" to compile your source file and produce the executable file `assignment3.exe`. Make sure you use exactly the same file names specified here, i.e. assignment3.cc and assignment3.exe, in your Makefile. Otherwise your submission will get 0 points.
- When your program is ready, submit your source file assignment3.cc and Makefile to your TA by following the Assignment Submission Instructions.