COP4520 HW1
Elijah Smith

## Design Summary

This program spawns eight threads to each compute a portion of the total $10^8$ primes. Each thread takes the next highest number from a counter that has not been checked for primality and increments that counter safely such that no other thread will check that same number. If that number is prime, that thread will safely increment the total primes found counter by one and the sum of primes counter by the value of the new prime. It will also safely store the new prime at the appropriate position in a list of the 10 largest primes, quickly bumping off the smallest prime on that list if there are now more than 10 stored. Once these operations have been performed, the thread releases all these resources at once for other threads to modify after finishing their own primes and waits to safely access the next highest prime. Once we check $10^8$, any threads that access the next highest prime that is larger than $10^8$ will shut down. Once all threads have shut down, we can safely output our pre-calculated results.

This approach eliminates synchronization issues between threads by locking the "get the next largest number" operations into a synchronized method and the "save this prime and increment things" operations into another. No two threads can be performing either set of operations at the same time, so only one thread will ever check any number and only one thread at a time can modify the top ten primes list and change result counters. Additionally, each thread grabs the next highest value to check since as these numbers get close to $10^8$ they take longer to evaluate. If we were to divide the total range into 8 linear parts, several threads with lower-value ranges would finish exponentially faster than the others. Finally, to verify the primality of any number a thread currently has, we only need to check that number for divisibility by 2, 3, and any integer $6k \pm 1$ for all integers $k \geq 1$ (proof here: https://en.wikipedia.org/wiki/Primality_test).

The initial implementation for checking primes checked the current number for divisibility by all numbers less than or equal to the square root of the current number. With only one thread, this program took ~91 seconds to execute. When increasing the thread count to eight, the program's execution was reduced to ~14 seconds (a 6.5x speedup). After optimizing the primality check by only checking divisibility by 2, 3, and $6k \pm 1$ as mentioned above, the execution time was reduced to ~12 seconds (for a 1.17x speedup).