Prime Number Generator

Description

In ancient Greece (approximately 200 BCE) there lived a man named, Eratosthenes, who was a Greek polymath: a mathematician, geographer, poet, astronomer, and music theorist; in short, a renaissance man some 18 centuries before the renaissance occurred. He is best known for being the first person to calculate the circumference of the Earth, which he did by using the extensive survey results he could access in his role as a chief librarian; his calculation was remarkably accurate.

He devised a technique for calculating prime numbers which was later called, "The sieve of Eratosthenes." If one eliminates all non-primes by generating all multiples of known primes and marking them as non-prime, only prime numbers remain.

A prime number is a natural number that has exactly two distinct natural number divisors: the number 1 and itself. To find all the prime numbers less than or equal to a given integer n, use the following algorithm:

1. Create a list of consecutive integers from 2 through n: (2, 3, 4, ..., n).
2. Initially, let p equal 2, the smallest prime number.
3. Compute the multiples of p by counting in increments of p from 2p to n, and mark them in the list (these will be 2p, 3p, 4p, ...; **NOTE: p itself should not be marked**). A shortcut to computing the first set of non-primes when p = 2 is simply to ignore all even numbers.
4. Find the smallest number in the list greater than p that is not marked. If there was no such number, stop. Otherwise, let p now equal this new number (which is the next prime), and repeat from step 3.
5. When the algorithm terminates, the numbers remaining not marked in the list are all the primes below n.

The main idea here is that every value that has p as a factor will not be prime. NOTE: some of the numbers may be marked more than once (e.g., 15 will be marked both for 3 and 5), let this happen. There are several methods to make the search more efficient. **You will implement only one of them.**

Since all even numbers have 2 (a prime number) as a factor, there is no need to consider them. This means that any number greater than 2 which is even should not be considered.

Requirements

- Write a program called ***primes.cpp***, that will calculate all the prime numbers between 2 and 1,000,000.
- You must use an array of bools called ***primes***.
- Your program will save these numbers in a file.
- Your program must be modularized.
- Prompt the user for the filename.
- There is no need for any other user interaction.
- You will have to test that the output file opened correctly. If it did not, let the user know and exit the program.
- **Very important: do not change the requirements of the program in any way.**

Program Algorithm

- Create an array of 1,000,000 **bools** (remember that the index 1,000,000 is even so it cannot be a prime).
- Write a function called ***init*** that initializes the element at index 2 and all odd elements of the array greater than 2 to be true (true means it is prime)

  - Each element's subscript represents the numbers that you are testing to be prime or not.
- The bool at 0 and 1 should be marked as false.
- Create a function called ***calcPrimes*** that takes into its parameters the array and the array size.
  - The function contains a loop that controls the value of p. it starts at the value of 3 and increments in steps of 2 (all odd numbers) up to n - 1 where n is the size of the array.
- The loop will locate the next prime and call a function called ***markNonPrimes***.
  - markNonPrimes has three parameters, the array, the size of the array, and the value of p.
  - markNonPrimes starts at 2p and marks it as false. It counts to k, where kp <= n, marking each element in the array with the index of a kp as false.
- When the markNonPrimes finishes and returns, calcPrimes should check each next element with an index greater than p (still stepping by 2) to see if it is true. If it is then that is the next value of p.
- Once you have processed the entire array, prompt the user for the name of the file to which to save the primes.
- Write a function called ***storePrimes*** that will write the primes to a file. This function has three parameters, the array, the size of the array, and the filename.
- When writing the prime numbers, be sure to include a new line character after each. Checkout the lecture for Topic 6, slides 21-23, to see how to write the values to the file.
- Open the output file with that filename gotten from the user and print all the indices of the array where the element is true to the file. **You will only have to check the odd indices but do not forget about 2!**
- **NOTE: there should be 78,498 primes between 2 and 1,000,000 including 2.**

What to Submit

- **primes.cpp**
- Run the program and save the primes that were generated to a file called, **primes.txt**. Submit this file.

Reminder

You are responsible to do your own work. **This is not a team project nor is this a practice in the use of copy and paste**. **Do not show anyone your code and do not look at, or copy, any code from any source (even the lecture notes or the book); create your own code.** The only code that you have my permission to mimic is the code that we used in our in-class exercises. Our lectures and the book contain all the information you need to complete this project. Any violation of the school's academic integrity policy or the policy of this class will result in a zero grade and an academic misconduct report filed with the school; no excuse will be accepted.

Your program must compile and run to receive any credit. If I cannot compile your program, then you will receive a zero for your score. Treat this like any other exam, start right away and put effort into it.

Rubric

This project/exam is worth 200 points. The points will be distributed according to the rubric below. If the program does not compile or is missing, then no points will be awarded.

Program 200
- **Requirements 70**
- **Correctness 70**
- **Code Quality 20**

Total 200