Assg 09: The Sieve of Eratosthenes

CSci 515 Spring 2015

<2015-02-19 Thu>

Dates:

Due: Tuesday April 7, by Midnight

Objectives

- Practice using advanced array logic.
- More practice with passing arrays to functions.

Description

We have seen a brute force algorithm or two for testing whether or not a given integer is a prime or not. Recall that a prime number is any integer this is only evenly divisible by itself and 1. Also recall that it is handy to use the C language modulus operator `%` in order to test whether one number is divisible by another (the modulus operator returns the remainder of doing the division, and if the remainder is 0 then the numbers were evenly divisible).

The Sieve of Eratosthenes is a method of finding prime numbers that is much more efficient than the brute force method we have seen. The algorithm operates as follows:

1. Create an array of boolean variables with all elements initialized to true. Array elements with prime subscripts will remain true after the algorithm finishes. All other array elements will eventually be set to false. We will ignore the elements at index 0 and 1 in this exercise.

2. Starting with array subscript 2, every time an array element is found whose value is true, loop through the remainder of the array and set to false every element whose subscript is a multiple of the subscript for the element with a true value. For array subscript 2, all elements beyond 2 in the array that are multiples of 2 will be set to false (subscripts 4, 6, 8, 10, etc.); for array subscript 3, all elements beyond 3 in the array that are multiples of 3 will be set to false (subscripts 6, 9, 12, 15, etc.); and so on. After array subscript 3, when you check array subscript 4 you will find it is false (since it was set to false by the multiples of 2 pass), thus you won't set multiples of 4, etc.

When this process is complete, the array elements that are still set to true indicate that the subscript is a prime number. This algorithm is much more efficient at finding all prime numbers in some range from 2 up to N than checking each individual integer using a brute force method as we have done before. In fact, it is close to $O(N)$ to check for primes in the range $2...N$.

Perform the following tasks:

1. Create a function called `initValuesToTrue()`. This function should take an array of booleans, and the size of the array as its input parameters. It should initialize all of the values in the boolean array to `true` as a result of calling the function.

2. Create a function called `setMultiplesToFalse()`. This function performs the part in step 2 of the algorithm above. This function should take an array of boolean values, and the size of the array. It should also take a third parameter, and integer index or multiple. As we described above, for example, if the multiple is 2, it shouldn't set the value at index 2 to `false` in the array, but it should set all multiples in of 2 in the array to `false`, e.g. the values at index 4, 6, 8, ... up to the size of the boolean array.

3. Create a function called `sieveOfEratosthenes()`. This function will implement the above described algorithm. It should take an array of booleans, and the size of the array as its only 2 parameters. The function should first call the `initValuesToTrue()` function, to ensure that it starts out with all `true` values in the array in order to perform step 1. It should then iterate over all the indexes in the array, from 2 up to the size of the array, and as described, any index that it finds still marked `true`, it should use the `setMultiplesToFalse()` function to mark all subsequent multiples of that index to be `false`.

4. Create a function called `displayPrimeNumbers()`. This function will again take an array of boolean values, and the size of the array. It is assumed this is the resulting array obtained after calling your Sieve of Eratosthenes function. Display all of the values that were determined to be primes from 2 to the size to standard output. An example of the desired output is shown below.

5. In your main function, create a boolean array of size 10,000. Call your `sieveOfEratosthenes()` function to find the primes in the first 10,000 integers, and display the results using your `displayPrimeNumbers()` function.

NOTE: Now that our programs have more functions than just the `main()` function, the use of the function headers becomes meaningful and required. Make sure that all of your functions have function headers preceding them that document the purpose of the functions, and the input parameters and return value of the function.

Assignment Submission

An eCollege dropbox has been created for this assignment. You should upload your version of the assignment to the eCollege dropbox named `Assg 09 Sieve of Eratosthenes` created for this submission. Work submitted by the due date will be considered for evaluation.

Requirements and Grading Rubrics

Program Execution, Output and Functional Requirements

1. Your program must compile, run and produce some sort of output to be graded. 0 if not satisfied.

2. 60+ pts. For the correct implementation of the three helper functions.

3. 30+ pts. For implementing the Sieve of Eratosthenes algorithm and functions correctly, and using the functions as described above.

4. 10+ pts. Your main function should create the array of boolean values and demonstrate using your functions to find and display the primes in the first 10,000 integers.

Program Style

Your programs must conform to the style and formatting guidelines given for this course. The following is a list of the guidelines that are required for the assignment to be submitted this week.

1. The file header for the file with your name and program information and the function header for your main function must be present, and filled out correctly.
2. A function header must be present for all functions you define. You must document the purpose, input parameters and return values of all functions. Your function headers must be formatted exactly as shown in the style guidelines for the class.
3. You must indent your code correctly and have no embedded tabs in your source code. (Don't forget about the Visual Studio Format Selection command).
4. You must not have any statements that are hacks in order to keep your terminal from closing when your program exits (e.g. no calls to system()).
5. You must have a single space before and after each binary operator.
6. You must have a single blank line after the end of your declaration of variables at the top of a function, before the first code statement.
7. You must have a single blank space after , and ; operators used as a separator in lists of variables, parameters or other control structures.
8. You must have opening `{` and closing `}` for control statement blocks on their own line, indented correctly for the level of the control statement block.
9. All control statement blocks (if, for, while, etc.) must have `{ }` enclosing them, even when they are not strictly necessary (when there is only 1 statement in the block).
10. You should attempt to use meaningful variable and function names in your program, for program clarity. Of course, when required, you must name functions, parameters and variables as specified in the assignments. Variable and function names must conform to correct `camelCaseNameingConvention` .

Failure to conform to any of these formatting and programming practice guidelines for this assignment will result in at least 1/3 of the points (33) for the assignment being removed for each guideline that is not followed (up to 3 before getting a 0 for the assignment). Failure to follow other class/textbook programming guidelines may result in a loss of points, especially for those programming practices given in our Deitel textbook that have been in our required reading so far.