EE231002 Introduction to Programming

Lab03. Consecutive Primes

Due: Oct. 10, 2018

How to find a prime number has been discussed in the class, and you are expected to be able to write a C program to find prime numbers quickly. The first few prime numbers are known to be

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, $\cdots$

Note that there are many sets of consecutive prime numbers with the difference of 2. Thus, it has been defined that a set of **consecutive primes** is a set of integer $\{N, N+2\}$ with both N and $N+2$ be prime numbers. The first few sets consecutive primes are

$\{3,5\}$, $\{5,7\}$, $\{11,13\}$, $\{17,19\}$, $\cdots$

In this lab, your assignment is to write a C program to find all the consecutive primes with $N, N+2 <$ **600,000**. The example of program output is shown below.


```
$ ./a.out
Consecutive primes #1: 3, 5
Consecutive primes #2: 5, 7
Consecutive primes #3: 11, 13
Consecutive primes #4: 17, 19
....
```

You are always encouraged to write your program efficiently. Though there may be very few loops needed for this assignment, but the loops need to be executed many times. Thus, how to write the program can have a big impact on the efficiency. To quantify your program execution time, the linux command **time** can be used. For example, to measure the CPU time of the `a.out` execution, one simply types

```
$ time ./a.out
```

Then there will be an additional line displayed after the normal program output as

```
0.646u 0.019s 0:00.77 84.4% 0+0k 0+0io 0pf+0w
```

The first number is the CPU time the program `./a.out` takes to execute. The postfix letter `u` means *user*. This is in contrast to the second number, which ends with `s`, that represents the time the operating system needs to handle this program. And, the third number is simply the total time, user plus system time. Of course, due to the fact that the operating system needs to take care many users at any given

time, switching time may not be accounted for accurately and thus the total might not be exactly user time plus system time. In this lab, or any other program execution, our main concern is the user time, the first number. Please write your program such that this number is as small as possible.

Notes.

1. Create a directory `lab03` and use it as the working directory.
2. Name your program source file as `lab03.c`.
3. The first few lines of your program should be comments as the following.

   ```
   /* EE231002 Lab03. Consecutive Primes
      ID, Name
      Date:
   */
   ```

4. After finishing editing your source file, you can execute the following command to compile it,

   ```
   $ gcc lab03.c
   ```

   If no compilation errors, the executable file, `a.out`, should be generated, and you can execute it by typing

   ```
   $ ./a.out
   ```

5. After you finish verifying your program, you can submit your source code by

   ```
   $ ~ee2310/bin/submit lab03 lab03.c
   ```

   If you see a "submitted successfully" message, then you are done. In case you want to check which file and at what time you submitted your labs, you can type in the following command:

   ```
   $ ~ee2310/bin/subrec lab03
   ```

   It will show all your submission records for lab03.

6. It is a good idea to define a macro in your program as

   ```
   #define max 600000
   ```

   And your program checks for all the integers less than `max` for consecutive primes. In the beginning, you should set `max` to be a small number such as 100 so program debugging can be done easily. Once you are satisfied with your program efficiency then change it to 600000 as required by the assignment.