EE231002 Introduction to Programming

Lab04. Balanced Prime Numbers

Due: Mar. 18, 2014

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 can be found to be

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

Note that the number 5 is equal to average of the neighboring prime numbers: 3 and 7. The next number has this property is 53. These numbers are called balanced prime numbers.

A balanced prime number is a prime number that is equal to the average of the nearest prime numbers above and below. For example, 3, 5, and 7 are three prime numbers. 3 is the nearest prime number smaller than 5, and 7 is the nearest prime number greater than 5. And, 5 is the average of 3 and 7. Thus, 5 is a balanced prime number.

In this assignment, please write a C program to find the first 1000 balanced prime numbers, and to print out the first 10 and the last 10 balanced prime numbers you find.

To make this lab more interesting, you are requested to write the program to run as fast as you can. The execution time of a program can be measured using unix `time` command. For example, after successful compilation of your program, you can execute it to get the following results.

```
$ time ./a.out
Balanced Prime Number #1: 5
Balanced Prime Number #2: 53
Balanced Prime Number #3: 157
Balanced Prime Number #4: 173
Balanced Prime Number #5: 211
Balanced Prime Number #6: 257
Balanced Prime Number #7: 263
Balanced Prime Number #8: 373
Balanced Prime Number #9: 563
Balanced Prime Number #10: 593
Balanced Prime Number #991: xxxxxx
Balanced Prime Number #992: xxxxxx
Balanced Prime Number #993: xxxxxx
Balanced Prime Number #994: xxxxxx
Balanced Prime Number #995: xxxxxx
Balanced Prime Number #996: xxxxxx
Balanced Prime Number #997: xxxxxx
Balanced Prime Number #998: xxxxxx
Balanced Prime Number #999: xxxxxx
Balanced Prime Number #1000: xxxxxx
14.266u 0.022s 0:14.33 99.6% 0+0k 0+0io 0pf+0w
```

The first number of the last row, 14.266u, is the CPU time used by this program. The postfix 'u' means user, ie., the time spent by this user, measured in seconds. This is the time that we need minimize by writing the program carefully.

Notes.

1. Create a directory **lab04** and use it as the working directory.
2. Name your program source file as **lab04.c**.
3. The first few lines of your program should be comments as the following.

```
/* EE231002 Lab04. Balanced Prime Numbers
   ID, Name
   Date:
*/
```

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

   ```
   $ gcc lab04.c
   ```

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

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

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

   ```
   $ ~ee231002/bin/submit lab04 lab04.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:

   ```
   $ ~ee231002/bin/subrec
   ```

   It will show the last few submission records.

6. You should try to write the program such that the CPU time used by your program is as small as possible. The CPU time is part of the grading criteria of this lab.
7. In case you want to use the square root function, you need to include an additional header file as the following.

   ```
   #include <math.h>
   ```

   Then you can call the the function `sqrt` as

   ```
   y=sqrt(x);
   ```

   where both `x` and `y` are of type `double`. And compilation should be done as

   ```
   $ gcc lab04.c -lm
   ```