CS120L – C Programming Language Labs
Fall 2014
Week 10: Functions, Arrays & Simple Problem Solving

In this lab, you are not allowed to use any libraries other than `stdio.h`, `math.h` and `stdlib.h`. You must not use array, "`%s`", "`%c`" or "`char *`" in Task 2. 20 marks will be deducted if you violate these requirements.

Task 1:

Part 1:
Write a function that will find all of the prime numbers between 0 and some number. (By definition, 0 and 1 are not prime, so you are really searching in the range of 2 to some number. We'll see why we consider 0 and 1 later.) The function will use an array to indicate the prime numbers in the range. For example, this is the list of all prime numbers between 2 and 100:

```
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
73, 79, 83, 89, 97
```

The method that you will use to find all of the primes in a range in this lab is called the **Sieve of Eratosthenes**. The idea is that you start with a list of all numbers in the range that you want to find prime numbers. So, if we want to find all of the primes between 2 and 100, we'd start with the list:

```
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, . . ., 96, 97, 98, 99, 100
```

Starting from 2, you mark 2 as being prime. Then, you cross out all multiples of 2, because any number that can be evenly divided by 2 can't be prime.

```
2, 3, 5, 7, 9, 11, 13, 15, . . ., 97, 99
```

Then, you go to the next number that isn't crossed out and mark that as being prime. (That number is 3). Then, you cross out all multiples of 3, because any number that can be evenly divided by 3 can't be prime.

```
2, 3, 5, 7, 11, 13, . . ., 97
```

Then, you go to the next number that isn't crossed out and mark that as being prime. (That number is 5). Then, you cross out all multiples of 5:

```
2, 3, 5, 7, 11, 13, . . ., 97
```

Those numbers are not shown in the list above, but they would be:

```
25, 35, 55, 65, 85, and 95.
```

To implement this method, you will use an array with 99 elements (or however large the range is). (Actually, to keep it simple, 101 elements because we are going to include the range of 0 to 100). You start by setting all elements to 1, indicating that every number in the range is prime:

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, . . ., 1, 1, 1
```

You can set element 0 and 1 to 0, indicating that the number 0 and the number 1 are not prime:

```
0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, . . ., 1, 1, 1
```

The number 2 is prime, it remains a 1. Crossing out all multiples of 2 means that all even-numbered indexes will be set to 0:

```
0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, . . ., 0, 1, 0
```

Continuing with 3 and crossing out multiples of 3:

```
0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, . . ., 0, 0, 0
```

and so on. You will continue up until, and including, the square root of the maximum number. In this case, the number is 100, so we only need to check multiples of 2, 3, 4, 5, 6, 7, 8, 9, and 10. Actually, we only have to check multiples of 2, 3, 5, and 7. The reason is that, since we've already crossed of multiples of 2, 3, and 5, we won't find any multiples of 4, 6, 8, 9, or 10. (Those multiples have already been crossed off.)

When you are done, the only elements in the array with a value of 1 will correspond to prime numbers. So, if the array was named primes, these indexes would have the value 1:

```
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
73, 79, 83, 89, 97
```

In code that means that primes[2], primes[3], primes[5], primes[7], primes[11], ... primes[89], and primes[97] have the value 1. Every other element has the value 0.

The prototype for the sieve function looks like this:

```
void sieve(int [], int);
```

• The first parameter is the array that you created in the sieve function.
• The second parameter is the positive size (number of elements) of the array. Hints:
• You need to set all of the values in array to TRUE before starting the loop.
• You will have a nested loop.
• Use the values TRUE and FALSE instead of 1 and 0.

Part 2:
A twin prime is a prime number that differs from another prime number by two. For example, here are some twin primes:

```
3 and 5
5 and 7
11 and 13
17 and 19
59 and 61
71 and 73
```

Like many primes, there are an infinite number of twin primes. Using the array that you created in the first task, print out all of the twin primes in a range. The prototype looks like this:

```
int twin_primes(const int [], int);
```

• The first parameter is the array that you created in the sieve function.
• The second parameter is the positive size (number of elements) of the array.
• To find the twin primes, you simply have to walk through the array looking for a pair and when you find one, print them out. The printf format is:

```
"twin prime #%4i: %4i and %4i\n"
```

• The return value is the number of twin primes (pairs) that you found.
• You'll notice that the first parameter is marked as `const`, meaning you are not changing it at all.

Part 3:
The sum of the reciprocals of all of the twin primes is known as Brun's Constant:

```
B = (1/3 + 1/5) + (1/5 + 1/7) + (1/11 + 1/13) + (1/17 + 1/19) + …
```

Using the array that you created in the first task, calculate Brun's Constant. The prototype looks like this:

```
double brun_constant(const int [], int);
```

• The first parameter is the array that you created in the sieve function.
• The second parameter is the positive size (number of elements) of the array.
• The return value is the value that you calculated in the function. The precision of the number will depend on how many twin primes you used in your calculation.
• You'll notice that the first parameter is marked as `const`, meaning you are not changing it at all.

If you've completed the twin_primes function, then calculating Brun's Constant is very simple.

Here are some values produced when running the program with different size arrays:

```
25 primes found between 0 and 100.
8 twin primes found between 0 and 100.
Brun's constant with 8 twin primes is 1.330990365719.
168 primes found between 0 and 1000.
35 twin primes found between 0 and 1000.
Brun's constant with 35 twin primes is 1.518032463560.
1229 primes found between 0 and 10000.
205 twin primes found between 0 and 10000.
Brun's constant with 205 twin primes is 1.616893557432.
9592 primes found between 0 and 100000.
1224 twin primes found between 0 and 100000.
Brun's constant with 1224 twin primes is 1.672799584828.
21336326 primes found between 0 and 400000000.
1507733 twin primes found between 0 and 400000000.
Brun's constant with 1507733 twin primes is 1.768841803744.
```

Implementation
You are provided with three files. “main.c” and “sieve.h” should not be changed. You need to implement the three functions in “q1-template.c” and then rename it to “q1.c” for submission.

Task 2:

In this task, we will search for the appearance of one number in another number. For example, 12 appears in 44<u>12</u>580 at index 2, but 28 does not appear in 7248256. You are **not** allowed to use array in this task.

Input: Two positive integers `a` and `b`. It is guaranteed that both are within the range of `int`.
Output: The index in `b` at which `a` appears, or `Not Found` if `a` does not appear in `b`.

You need to write a search function with the following prototype:

```
int search(int a, int b);
```

The function returns the index in `b` at which `a` appears, or `-1` if `a` does not appear in `b`.

Sample Run
Sample runs using interactive input.

```
12 4412580
2↲

28 7248256
Not Found↲
```

The ↲ denotes an invisible new line character.

Implementation
You should write a `main` function together with the `search` function. You should only do I/O inside the `main` function. Your printouts should match the sample outputs exactly.

Command line for compiling

```
/usr/bin/gcc -Wall -Wextra -ansi -pedantic q1.c main.c -o q1.exe
/usr/bin/gcc -Wall -Wextra -ansi -pedantic q2.c -o q2.exe
```

Grading scheme

- 100 marks in total (50 marks for task 1 and 50 marks for task 2).
- 10 marks for successful build.
- 10 marks for successful execution and correct output (2 marks * 5 hidden test cases).

  Note that your program should only print out the required output, ending immediately with an end-of-line character `\n`, and not any redundant character.

- 20 marks for correct implementation.

- 10 marks for good programming style. For the complete guide, please see the Assignment Guideline on CS120 page.

  In general, you should have clear variable names, reasonable comments to explain your code, and consistent indentation. Please use 2**-blank spaces** instead of tab spaces.

Output formatting
You should use the given sample input and output files to check your output format before submission. After compiling your program into **a.exe**, you can generate your output using:

```
./a.exe < q1.in > result.out
```

Then compare it with the sample output by typing:

```
diff result.out q2.out
```

You need to make sure that diff does not print out anything. Otherwise, your output is considered wrong and you will lose all the 10 marks for output correctness.

Submission

Please name your source code for task 1 as **q1.c**, task 2 as **q2.c**, put them in a folder named **cs120<session>_<your Digipen login id>_<labnumber>**, in which <**labnumber>** is **10** this week and **<session>** is either **a** or **b**, and zip them in **cs120<session>_<your Digipen login id>_<labnumber>.zip** for submission.

Wrong submission file/folder name will cause 10 marks deducted.

Note that the file and folder names must be **lowercase**.

The deadline of submission is 7th November 23:59 and late submission will receive zero mark.