Programming Concepts

Exercise 3

October 15, 2016

FH JOANNEUM
University of Applied Sciences

Institute of Internet Technologies & Applications

1 What do do

The *Sieve of Eratosthenes* is a simple but computationally relative efficient way to find (small) prime numbers. However it can require a large amount of memory in order to find larger primes, which makes a good use case for heap allocation.

2 The algorithm

I will not try to explain the algorithm in this exercise sheet! There are already some very good explanations available on the Web therefore will just give you some links to online resources:

- http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes
- https://www.youtube.com/watch?t=66&v=klcIklsWzrY

There are some optimization to the sieve of Eratosthenes for examp If you want you can implement them to but this is optional. However it is important that your implementation has no *memory leaks* or *invalid memory access*!

3 Implementation requirements

To guarantee that you practice all features of the C language that we have learned until now there are some restrictions on how to structure and implement your exercise.

3.1 sieve.h

The function that does the the actual sieving must be implemented according to the function prototype inside the `sieve.h` that is provided to you.

```
#define SIEVE_OK 0
#define SIEVE_INVALID_ARGUMENT 1
#define SIEVE_OUT_OF_MEMORY 2

uint64_t* sieve(uint64_t limit, int* status, size_t* primeCount);
```

In general the task of the function is to find all prime numbers up to value specified by the parameter `limit`. Remember we can use *call by reference* to 'return' multiple values from a function. The parameters `status` and `primeCount` are to be used this way.

- `status`: This parameter should return the status after the function has been called. The status codes are specified by the `#define` statements above.
  - `SIEVE_OK`: The function returned normally
  - `SIEVE_INVALID_ARGUMENT`: The function received invalid arguments (e.g. `limit < 2`)
  - `SIEVE_OUT_OF_MEMORY`: A memory allocation during the sieving algorithm failed.

- `primeCount:` If the functions finished successfully this parameter will store the number of primes found in the given range. If the functions fails the value is not changed by the function.

  Hint:

Return value: The function should return an array which contains all primes that were found. The number of array elements is stored in the `primeCount` parameter. If the functions fails `NULL` should be returned. The caller of the function is responsible for freeing this array.

3.2 Command line arguments

The program should receive **two** command line arguments:

- `-p`: Instructs the program to also print the found primes on standard out. (optional)
- The number up to which primes should be found (mandatory)

If no `-p` flag is specified on the command line the program should only return the **total number** of primes found.

`./sieve 100`: Gives the total number of primes between 2 and 100.

`./sieve -p 100`: Gives the total number of primes between 2 and 100 and prints all.

3.3 File structure

Your exercise should consist out of multiple source files

- **main.c**: Implements the command line parsing and calls the sieving algorithm.
- **sieve.c**: Contains the sieving function.
- **sieve.h**: Header file that is provided to you as part of this exercise.

4 Useful standard library functions

There are some standard library functions which might be useful to you in this exercise. As always read the documentation before using the these functions!

- `strcmp`: Checks two strings for equality.
- `memset`: Sets a memory range to a predefined value.
- `stroull`: Converts a string into an `unsigned long long`
  - **Example:** To convert a single string into an `uint64_t`: `strtoull("123456789",NULL,10)`

5 Testing

'Off by one' errors can occur very easily in this exercise, so test your program thoroughly! My suggestion is that you test your program by comparing your computed primes with those of Wolfram Alpha. (`http://www.wolframalpha.com/`)

You can also do the ultimate stress test by computing the number of primes between 2 and 4294967297. However since this requires a huge amount of memory (4+ GB) you can only do this in a 64 bit environment with more than 4 GB of RAM.

Hint: There are **203280221** primes between 2 and 4294967297 so it is probably a good idea to omit the `-p` flag if you perform this test. ☺

6 Submission

Since you exercise consists of multiple files zip your exercise and make sure that only `.c` and `.h` files are part of your upload. Please use the following naming convention: `exercise_3_<lastname>.zip`.

Any non compliance with the naming convention will be punished by the deduction of points!