Einführung in die Programmierung für Physiker

WS 2017/2018 – Marc Wagner

Francesca Cuteri: cuteri@th.physik.uni-frankfurt.de
Alessandro Sciarra: sciarra@th.physik.uni-frankfurt.de

Exercise sheet 8

To be corrected in tutorials in the week from from 11/12/2017 to 15/12/2017

Exercise 1 [*The Sieve of Sundaram*]

The sieve of Sundaram is an algorithm to find all the prime numbers up to a specified integer. It was discovered by the Indian mathematician S. P. Sundaram in 1934.
Fixed $N \in \mathbb{N}$ and given the list of the integers between 1 and N, remove all the numbers of the form

$$i + j + 2\,i\,j \qquad \text{where} \qquad i \in \mathbb{N} \quad , \quad j \in \mathbb{N} \quad , \quad 1 \leq i \leq j \quad \text{and} \quad i + j + 2\,i\,j \leq N \quad .$$

All the remaining numbers, doubled and incremented by one, give all the prime numbers smaller than $M \equiv 2N + 2$ except 2.

(i) Implement the Sieve of Sundaram in a code which gets the value N as first command line option `argv[1]` and either prints all the prime numbers smaller than M or prints how many prime numbers exist up to M.

Time to Test! For $N = 500$, your code should tell that there are 168 prime numbers up to $M = 1002$.

(ii) Run your code with $N \in \{2 \cdot 10^4, 5 \cdot 10^4, 2 \cdot 10^5, 5 \cdot 10^5, 2 \cdot 10^6\}$ and measure the execution time, saving it in a file. Plot the obtained data as function of M on a bi-logarithmic scale. How does the algorithm compare to a naive prime numbers search?

Hint: Even if it could sound natural, do *not* use an array of `int` variables. Instead, think of the array index as your starting list for the sieve and use a smart data type to keep track of *deleted* numbers, which are indeed never deleted.

Exercise 2 [*Palindrome-ness of strings*]

A palindrome is a sequence of characters which reads the same backward as forward, such as *racecar*. Sentence-length palindromes may be written when allowances are made for adjustments to capital letters such as in *Madam*, and word dividers, such as in *Was it a car or a cat I saw*.

(i) You are requested to write a program that should achieve a case-sensitive and space-sensitive check on a maximally 100-characters-long string, telling the user whether the string is or is *not* palindrome. To read the input string you can use once again the `scanf` function, with the special conversion specifier `%[^\n]` matching all characters until a new line character `\n` is met. Can you devise an alternative to the usage of `scanf` in order to read the input string?

(ii) **Optional:** Your code can easily be extended to perform a *non* case-sensitive check. Just look into `ctype.h` for a function that can help you in this task.

(iii) **Optional:** As a further extension of your program you could have it neglecting (read deleting) spaces to get a *neither* case-sensitive *nor* space-sensitive check.

Time to Test! According to a correct implementation of (i) *racecar* is palindrome, while *Racecar* and *race car* are *not*. Also *Madam* and *Racecar* will instead be considered as palindromes by your program exteded according to (ii). Finally, if you were able to extend your program according to (iii), even *Was it a car or a cat I saw*, along with *race car* should be recognized as palindromes.