Dirk Praetorius,
Michele Ruggeri

Wintersemester 2015/16
18.-20.11.2015

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Aufgabe 6.1. Write a function `lcm` that computes the *least common multiple* of two given natural numbers $a, b \in \mathbb{N}$. For the solution, you can either compute the prim factors of both numbers or use the relation $ab = \mathrm{gcd}(a,b) \cdot \mathrm{lcm}(a,b)$, where $\mathrm{gcd}(a,b)$ denotes the *greatest common divisor*. Save your source code as `lcm.c` into the directory `serie06`.

Aufgabe 6.2. An alternative root-finding algorithm (see also the Bisection method from the lecture) is the so called *secant method*. Let $f : [a,b] \to \mathbb{R}$. Given two initial guesses x_0 and x_1, the appromation x_{n+1} is obtained as the root of the line through $(x_{n-1}, f(x_{n-1}))$ and $(x_n, f(x_n))$, i.e.,

$$x_{n+1} := x_n - f(x_n)\,\frac{x_{n-1} - x_n}{f(x_{n-1}) - f(x_n)}.$$

Write a function `secant(x0,x1,tau)`, which performs the above iteration until either

$$|f(x_n) - f(x_{n-1})| \le \tau$$

or

$$|f(x_n)| \le \tau \quad \text{and} \quad |x_n - x_{n-1}| \le \begin{cases} \tau & \text{for } |x_n| \le \tau, \\ \tau|x_n| & \text{else.} \end{cases}$$

In the first case, print a warning to inform that the result is presumably wrong. The function returns x_n as the approximation of the root z_0 of f. Test your implementation with a suitable example.Then, write a main program, that reads x_0 and x_1 from the keyboard and displays x_n. Save your source code as `secant.c` into the directory `serie06`.

Aufgabe 6.3. Expand *MinSort* from the lecture by a parameter `type`, such that the functions sorts a vector $x \in \mathbb{R}^n$ in ascending ascending order (`type = 1`) or in descending order (`type = -1`). Moreover, write a main-programme, that reads in x and `type`, calls *MinSort* and prints the sorted vector. The length n should be a constant in the main-programme, but your implementation of *MinSort* should work for arbitrary lengths. Save your source code as `minsort.c` into the directory `serie06`.

Aufgabe 6.4. The *bubblesort* algorithm is an inefficient, but short sorting algorithm which works as follows: You run through the entries of a given vector $x \in \mathbb{R}^n$ several times. For every run, each entry x_j of x is compared to its successor x_{j+1}. If $x_j > x_{j+1}$, then the two entries x_j and x_{j+1} are swapped. After the first complete run through the vector, one knows that (at least) the last element is sorted correctly, i.e. the last element x_n is the maximum of the vector. Thus, in the next run one only has to go up-to the last-but-one entry of the vector. How many loops do you need for this algorithm? Write a function `bubblesort` which sorts a given vector $x \in \mathbb{R}^n$ with this algorithm. Additionally, write a main program that reads in $x \in \mathbb{R}^n$ and sorts it. The length n should be constant. However, your function `bubblesort` should be programmed for arbitrary lengths n. Save your source code as `bubblesort.c` into the directory `serie06`.

Aufgabe 6.5. Implement the *Quicksort*-Algorithm, which sorts a vector $x \in \mathbb{R}^n$: To do so *Quicksort* chooses an arbitrary Pivot-element form x, e.g. x_1. Then, x is split in to parts $x^{(<)}$ and $x^{(\ge)}$ and the Pivot-element x_1: $x^{(<)}$ contains all the elements $\le x_1$ and $x^{(\ge)}$ contains only elements $\ge x_1$. $x^{(<)}$ and $x^{(\ge)}$ are sorted recursively. Afterwards, the result is put together. The direct implementation of this algorithm, however, requires additional storage. To circumvent this, proceed as follows: Starting with $j = 2$ search for and element $x_j \ge x_1$, i.e. x_j belongs to $x^{(\ge)}$. Furthermore, starting with $k = n$ search an element $x_k < x_1$, i.e. x_k belongs to $x^{(<)}$. In that case, swap x_j and x_k. If j and k coincide then x has already the form $(x_1, x^{(<)}, x^{(\geq)})$. With one additional swap, the form $(x^{(<)}, x_1, x^{(\geq)})$ is obtained immediately. It remains to sort $x^{(<)}$ and $x^{(\geq)}$ recursively. Moreover, write a main-programme, that reads in x and calls Quicksort. The length n should be a constant in the main-programme, but your implementation of Quicksort should work for arbitrary lengths. Save your source code as `quicksort.c` into the directory `serie06`.

Aufgabe 6.6. Let the two series

$$a_N := \sum_{n=0}^{N} \frac{1}{(n+1)^2} \quad \text{und} \quad b_M := \sum_{m=0}^{M} \sum_{k=0}^{m} \frac{1}{(k+1)^2(m-k+1)^2}$$

be given. Write a program that measures the time used for the computation of a_N resp. b_M for different values of N resp. M. Print out the results tabularly. Do the results meet your expectations? Save your source code as `timing.c` into the directory `serie06`. *Hint:* Think of the computational complexity (Aufwand) for the computation of a_N resp. b_M.

Aufgabe 6.7. You place your money with your trusted bank for a fixed annual percentage rate. Write a function `capital` which computes your capital after $n \in \mathbb{N}$ years for a fixed annual percentage p (in percent %), and your starting capital $x \in \mathbb{R}_{\geq 0}$. The function should print out your money as follows

```
Year    Capital
====    =======
0      1000.00
1      1010.00
2      1020.10
3      1030.30
..      .......
10      1104.62
```

For this example holds $p = 1$, $n = 10$, and $x = 1000.00$. Furthermore, write a function `runtime` which computes how long (at least) you have to wait to increase your starting capital x to $x_{\max}$ for a fixed percentage p. The function reads in x, p, and $x_{\max}$. Additionally, write a main program that tests both functions. How long does it take to be a millionaire, if you invest $x = 1000$ with a fixed percentage $p = 4$? Save your source code as `capital.c` into the directory `serie06`.

Aufgabe 6.8. What is the best and worst case for the computational cost of the Bubblesort algorithm from Exercise 6.4? Explain your answer!