Dirk Praetorius
Michele Ruggeri

Summer semester 2019
March 22, 2019

Scientific programming in mathematics

Exercise sheet 3

Arrays, `for` loop, and computational complexity

Exercise 3.1. Write two functions:

- the function `double scalarProduct(double u[3], double v[3])`, which computes and returns the scalar product $w = \mathbf{u} \cdot \mathbf{v} = ax + by + cz$ of two given three-dimensional vectors $\mathbf{u} = (a, b, c)$ and $\mathbf{v} = (x, y, z)$;
- the function `void vectorProduct(double u[3], double v[3], double w[3])`, which computes and prints to the screen the vector product $\mathbf{w} = \mathbf{u} \times \mathbf{v}$ of two given three-dimensional vectors $\mathbf{u} = (a, b, c)$ and $\mathbf{v} = (x, y, z)$, i.e.,

$$\begin{aligned} w_1 &= bz - cy, \\ w_2 &= cx - az, \\ w_3 &= ay - bx. \end{aligned}$$

Furthermore, write a main program, which reads the parameters a, b, c and x, y, z from the keyboard and prints to the screen the values of the two products of the vectors. Save your source code as `products.c` into the directory `series03`.

Exercise 3.2. Write the function `int lines(double u[3], double v[3], double s[2])`, which characterizes the mutual position of two lines: Given $\mathbf{u} = (a, b, c) \in \mathbb{R}^3$ and $\mathbf{v} = (d, e, f) \in \mathbb{R}^3$, the equations

$$ax + by = c \quad \text{and} \quad dx + ey = f$$

define two lines in the plane. The function `lines` determines whether the lines defined by the input parameters are *parallel* (return value `1`), *coincident* (return value `0`), or *intersecting* (return value `-1`). In the third case, the function computes and stores the coordinates of the intersection point (in the vector `s[2]`). Then, write a main program which reads the six parameters from the keyboard, calls the function `lines`, and prints to the screen a message with the mutual position of the lines. Save your source code as `lines.c` into the directory `series03`.

Exercise 3.3. One way (not the best way) to approximate the number π is based on the so-called *Leibniz formula*

$$\pi = \sum_{k=0}^{\infty} \frac{4(-1)^k}{2k+1}.$$

In particular, for any $n \in \mathbb{N}_0$, the n-th partial sum

$$S_n = \sum_{k=0}^{n} \frac{4(-1)^k}{2k+1}.$$

can be understood as an approximation of π (it holds that $\lim_{n\to\infty} S_n = \pi$). Write a function `double partialSum(int n)` that computes S_n for given $n \in \mathbb{N}_0$. Implement two different versions of the function: one is recursive, one computes the partial sum with a suitable loop. Moreover, write a main program that reads $n \in \mathbb{N}_0$ from the keyboard and prints the resulting approximation S_n of π to the screen. Save your source code as `piApproximation.c` into the directory `series03`.

Exercise 3.4. Implement the function `int = binomial(int n, int k, int type)` which computes and returns the binomial coefficient $\binom{n}{k}$ of two integers $n, k \in \mathbb{N}_0$ using three different approaches:

- If `type=1`, the computation is based on the formula $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. Note that, for this approach, a function which computes the factorial is needed.
- If `type=2`, the computation is based on the formula $\binom{n}{k} = \frac{n\cdot(n-1)\cdots(n-k+1)}{k\cdot(k-1)\cdots 1}$, which can be implemented using a suitable loop.
- If `type=3`, the computation is based on the formula $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$.

To ensure the correctness of your function, check that, for given $n, k \in \mathbb{N}_0$, the three approaches lead to the same result. Furthermore, write a main program, which reads n and k from the keyboard and prints the resulting binomial coefficient to the screen. Save your source code as `binomial.c` into the directory `series03`.

Exercise 3.5. The Fibonacci sequence is recursively defined by $x_0 := 0$, $x_1 := 1$ and $x_{n+1} := x_n + x_{n-1}$. Write a *non-recursive* function `int fibonacci(int n)`, which computes and returns the member x_n of the Fibonacci sequence for a given integer $n \in \mathbb{N}_0$. Then, write a main program, which reads n from the keyboard and prints to screen the corresponding value of x_n. Save your source code as `fibonacci.c` into the directory `series03`. What is the computational complexity of the computation of x_n? Justify accurately your answer. Compare your implementation with that of the recursive function `fibonacci` discussed in Exercise 2.7. Discuss the advantages and disadvantages of both implementations.

Exercise 3.6. Write a main program which reads $n \in \mathbb{N}$ from the keyboard and prints to the screen the first n lines of Pascal's triangle: Every line starts and ends with 1. The remaining entries are the sum of the two neighboring entries from the line above. For example, for $n = 5$, we obtain

```
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
```

For more details, see, e.g., `https://en.wikipedia.org/wiki/Pascal's_triangle`. Save your source code as `pascal.c` into the directory `series03`.

Exercise 3.7. Let x be a sequence of 10 integers (stored in a static array of type `int`). Let y be combination of 3 integers (also stored in a static array of type `int`). Write a function `int check(int x[10], int y[3])` which checks whether the combination y is contained in the vector x (return value 1) or not (return value -1). For example, for the vector $x = (1,2,3,4,5,6,7,8,9,10)$, the function returns 1 for $y = (3,4,5)$ and -1 for $y = (7,5,2)$. Furthermore, write a main program which reads the arrays x and y from the keyboard, calls the function and prints to the screen the result. Save your source code as `check.c` into the directory `series03`.

Exercise 3.8. Write a function `void minmaxmean(double x[], int dim)`, which computes and prints to the screen the minimum, the maximum, and the mean value $\frac{1}{n}\sum_{j=1}^{n} x_n$ of a given vector $x = (x_1, \dots, n_n) \in \mathbb{R}^n$. Additionally, write a main program that reads the vector $x \in \mathbb{R}^n$ from the keyboard and calls the function. The length of the vector should be constant in the main program, but the function `minmaxmean` should be programmed to work for arbitrary vector lengths. Save your source code as `minmaxmean.c` into the directory `series03`.