# Einführung in die Programmierung für Physiker

SoSe 2024 – Marc Wagner

Michael Eichberg: eichberg@itp.uni-frankfurt.de

## Exercise sheet 2

*To be handed in on 03.05.2024.*

### Exercise 1 [*Compiler options*]

Programming languages can be divided into two classes: interpreted and compiled languages. `C` belongs to the latter class and you need a compiler in order to run the code you wrote. The compiler translates the input code into a form which can be understood by the machine and executed. This additional step allows to select a variety of options. In this exercise you will discover some features, which will be helpful throughout the semester. Use a `C`-program from the lecture to play with the `gcc` (or `g++`) compiler (using a different compiler is not prohibited, but compiler options are not universal).

(i) Open the manual page of your compiler and scroll to the end. This should scare you a bit. Be aware that compilers might be rather complex. The following steps will guide you in discovering what will be needed at the beginning.

(ii) Which is the minimal information you must provide to the compiler? Which is the default name of the created executable? How can you change it?

(iii) The compiler can warn you about some features of the language and **warnings should not be ignored**. However, by default, most of the warnings are not given and it is the user's responsibility to activate them. To read through the manual is tough, especially at this point of the lecture. Focus on the following options and try to understand why it is important to activate such warnings.

- `-Wunused`
- `-Wuninitialized`
- `-Wparentheses`

(iv) In general, it is easier to simply give `-Wall` and maybe `-Wextra` options than remembering many single options. In the worst case you activate some additional warning which you can also by hand deactivate at a later point. Check out these two options on the manual.

(v) Many compiler options refer to which freedom is granted to the compiler to optimize code. This is a very technical topic. Have a look to the `-O` options and try to get a rough understanding of them.

### Exercise 2 [*Printing to the screen*] `(2 + 2 + 1 = 5 pts.)`

The goal of this exercise is to become familiar with the `printf` function. A good starting point is its manual page, which can be obtained via the `man` command. In particular, you are interested in the *third* section of the manual. You can run `man 3 printf`.

(i) Read the manual and understand which format string you need to print

(a) an unsigned integer or a signed integer,

(b) an unsigned integer with leading zeros,

(c) a signed integer with a mandatory ± sign in front,

(d) a floating point number,

(e) a floating point number with a given number of digits after the comma,

(f) a floating point number with a given number of digits before and after the comma,

(g) a floating point number in decimal exponent notation,

(h) a percent symbol.

*Hint:* : What do `%.5f`, `%d`, `%e`, `%05u`, `%f`, `%+i`, `%8.3f`, `%u`, `%%` mean?

(ii) Write a program which uses the `printf` function to print the following numbers.

- $-35$ and $+42$ with explicit sign.
- $\frac{1}{4}$ with one decimal digit.
- $4 \cdot \text{atan}(1)$ with 20 decimal digits.
- $2^{(e^5)}$ in a way you think is easily readable.

Choose appropriate conversion specifiers. Do not forget to include the `math.h` library at the beginning of your program in order to have access to some mathematical functions (e.g. `exp`, `pow`, etc.) and constants.

(iii) Print the constant `M_PI` with 20 decimal digits and compare with what you obtained in (ii). In 1596 it was known that $\pi \approx$ `3.14159265358979323846`[1]. Is this the number you obtained? Are we wrong more than 4 centuries later?

## Exercise 3 [*Basic algebra manipulations in C*] (2 + 3 = 5 pts.)

Using `printf`, `scanf`, some arithmetic operators in C, i.e. `+`, `-`, `*`, `/`, `%`, and different types of variables (`float`, `int`), we will perform very basic algebraic manipulations on scalars, matrices and vectors. Write a program which

(i) reads nine integer numbers and stores them in variables to be called `a00`, `a01`, `a02`, `a10`, `a11`, `a12`, `a20`, `a21`, `a22`, as well as three integer numbers which are stored in variables `b0`, `b1`, `b2`,

(ii) for the matrix $A$ and the vector $\boldsymbol{b}$ defined as

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{b} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix},$$

(a) computes and prints the result of $A \cdot A$, as well as the result of $A \cdot \boldsymbol{b}$,

(b) computes and prints $\text{Tr}(A \cdot A)$ and $\|A \cdot \boldsymbol{b}\|$.

And now, time to **test your code**! You should *always* check that your code does what is expected, at least for simple cases for which results are known. For this exercise you can consider $A = \mathbb{1}$ and $\boldsymbol{b} = (1, 1, 1)^T, \boldsymbol{0}$. This being a case in which you might have to run your program multiple times (i.e. until the test succeeds) with a fixed input, use what you have learnt so far to avoid typing the same input every time.

## Exercise 4 [*Cyclic numbers*] (2 + 3 + 1 + 1 + 3 = 10 pts.)

As you know, the `printf` function in `C` accepts also mathematical operations as argument. In this exercise we will write a short program to discover some fancy numbers. Let us consider $N = 142857$, which has $n = 6$ digits. Store it in a variable with the most appropriate type.

(i) Use a `printf` statement with 4 arguments in order to print `"1*142857=142857"`. Add other statements to print analogue lines multiplying $N$ by $t \in \{2, 3, \ldots, n\}$. Do you see a pattern in the results? What about $(n + 1) \cdot N$?

[1] `https://en.wikipedia.org/wiki/Ludolph_van_Ceulen`

(ii) Multiply $N$ by any number you wish, which is not a multiple of $n+1$. Run your code and have a look at the outcome. Consider, from right to left, groups of $n$ digits of the result and add a `printf` statement to your program to sum up the resulting numbers. Repeat this step, if needed, until the result has not more than $n$ digits. Do it either *by hand*, copying from the output back into your code, or use a `for` loop, using one `printf` statement only. Have a look at the lecture material, to get inspired. What do you obtain?

(iii) Repeat task (ii) choosing a multiple of $n+1$.

(iv) Add to your code another line to print $\frac{1}{n+1}$ with at least $2n+1$ decimal digits. What is the decimal period? Cool, isn't it?

(v) Consider now $n = 13$ and $n = 17$. Repeat task (iv) to receive the value of $N$. Pay attention that now a leading zero has to be considered as part of the number! Repeat the other steps done previously and find a good candidate for a cyclic number.