Einführung in die Programmierung für Physiker

WS 2019/2020 – Marc Wagner

Alessandro Sciarra: `sciarra@itp.uni-frankfurt.de`

Exercise sheet 2

To be corrected in tutorials in the week from 28.10 to 01.11.2019

Exercise 1 [*Compiler options*]

Programming languages can be divided into two classes: interpreted and compiled ones. `C` belong to the later class and you need a compiler in order to run the code you wrote. Duty of the compiler is to translate the input code into a form which can be understood by the machine and hence run. This additional step allows also for lots of opportunities and it is important to know the compiler a bit. In this exercise you will discover some features, which will be helpful throughout the semester. Use a `C`-program taken from the lecture to play with the `gcc` (or `g++`) compiler[1].

(i) Open the manual page of your compiler and scroll it till the end. This should scare you a bit. Be aware that compiler are extremely powerful and hence also extremely 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 a 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 might also by hand deactivate at a later point. Check out these two options on the manual.

(v) Many compiler options are referred 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 grasp a rough understanding of them. In some weeks you will be able to touch with your hand their benefit.

Exercise 2 [*Printing to the screen*]

The goal of this exercise is to become friend with the `printf` function. A good starting point is its manual page, which can be obtained via the `man` command. Indeed, you are interested in the *third* section of the manual and you need to 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 $\pm$ sign in front;

(d) a floating point number;

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

[1] Using a different compiler is not forbidden, but options are not universal.

(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.

Said differently, 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. Which conversion specifiers is it better to use? 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 to some constants.

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

(iii) Print the constant `M_PI` with 20 decimal digits and compare with what you obtained in (ii). Apparently[2], already in 1596 it was known that $\pi \approx$ `3.14159265358979323846`. Is this the number you obtained? Are we wrong more than 4 centuries later. . . ?!

Exercise 3 [*Basic algebra manipulations in C*]

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.

(i) Start by writing a program which reads, via the `scanf` function, two integer numbers to be stored in the variables `x`, `y` and one floating point number to be stored in the variable `z`. To check that values were properly assigned to variables, you should

(a) Read input values from the keyboard and print `x`, `y` and `z` to the screen with the most appropriate formatting, according to your knowledge of the `printf` function.

(b) Suppose to have to run your program several times always with the same variables, how to avoid to always have to type them using the `<` shell operator?

(c) Now let us build a 3×3 matrix whose components are the results of arithmetic operations on `x`, `y` and `z`

$$\begin{pmatrix} x-1 & x+y & y*z \\ x*z & z*z & x/y \\ 1.*x/y & x\%y & z/3 \end{pmatrix} .$$

Print the above matrix to the screen for $x = 1$, $y = 2$ and $z = -1.2$, such that, by just using the conversion specifiers `%+5d` or `%+1.2f` when you expect a float or an integer respectively, the output will look like

```
|    +0    +3 -2.40 |
| -1.20 +1.44    +0 |
| +0.50    +1 -0.40 |
```

(d) Compute the trace of the above matrix and print out the result.

(e) Now print out also the vector

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

to the screen such that, again for $x = 1$, $y = 2$ and $z = -1.2$, the output looks like

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

```
|     +1  |
|     +2  |
|  -1.20  |
```

(f) Compute the norm of the vector as well as the sum of its elements and print out the results as a floating point number with explicit sign and 4 decimal digits after the comma.

(ii) Write yet another similar program which

(a) Reads nine integer numbers and stores them in variables to be called `a00`, `a01`, `a02`, `a10`, `a11`, `a12`, `a20`, `a21`, `a22`.

(b) Reads three integer numbers and stores them in variables to be called `b0`, `b1`, `b2`.

(c) For the matrix A and the vector $\mathbf{b}$ defined as

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

you are requested to

(1) Compute and print out the element in the first row and first column of the product of the matrix A with itself as well as the first element of the product of the matrix A with the vector $\mathbf{b}$.

(2) Compute and print out $\mathrm{Tr}(A \cdot A)$ and $\|A \cdot \mathbf{b}\|$.

(d) And now, time to **test your code**! You should *always* check that your code does what expected at least for the most simple cases for which results are easily predictable. For this simple exercise you can consider $A = \mathbb{1}$ and $\mathbf{b} = \mathbf{1}, \mathbf{0}$. Being this a case in which you might in principle 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 to type your input every time.

Exercise 4 [*Cyclic numbers*]

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 start 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"`.

  (ii) Add other statements to print analogue lines multiplying N by $t \in \{2, 3, \ldots, n\}$. Do you see any pattern in the results? What about $(n+1) \cdot N$?

  (iii) Multiply N by any number you wish, which is not multiple of $n+1$. Run your code and have a look to 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. Do it *by hand*, copying from the output back into your code. Repeat this step, if needed, until the result has not more than n digits. What do you obtain?

  (iv) Repeat item (iii) choosing a multiple of $n+1$.

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

- Consider now $n = 13$ and $n = 17$. Repeat item (v) to get out 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 out which is a good candidate to be a cyclic number.

One step further:
How would you do item (ii) using a `for` loop using one `printf` statements only? Have a look to the last lecture material to get inspired.