Tutors: C. Czaban, A. Dromard, J. Glesaaen, M. Neuman, D. Palao, C. Schäfer, A. Sciarra

Tutorial II

October 24

Excercise 1 [*printf*] The goal of this exercise is to work with the `printf` *function*, to understand its usage and to learn how the different data types can be processed by it. Start by having a look at its `man` page and read the sections about *precision* and *conversion specifier*; then you should be in a position to understand, for example, what `%d`, `%g` or `%.6f` mean. Note that you may be interested in the 3rd section of the `printf` manual, which is accessed through `man 3 printf` instead of simply `man printf`. Run `man man` to understand why ;)

(i) As it was explained in the lecture, by including the `math.h` header in our codes, via

```
#include <math.h>
```

we can get access to several mathematical functions and constants. In particular, in this exercise we will make use of the constant `M_PI`, which is equal to the first digits of the number π.
Compare the output of `printf` for the cases of `float` and `double` conversion specifiers (a `float` is another data type, similar to `double` but less precise).

Give enough precision to each field to contain 32 decimal digits. Compare the output with the actual 32 first digits of π. What is going on? Is the computer wrong?

(ii) In C, a *definition* is a statement where the properties of a variable are announced. The following code is an example

```
int i;
```

On the other hand, an *assignment* is a statement which sets a variable to a value, for instance

```
i = 23;
```

Write a program where you *define*, but not *assign* to any value, the following type of variables: integer, unsigned integer, single precision floating point, and double precision floating point. Then print them out with the proper *conversion specifier*. Compile it and run it. What values are you getting? Does the program print out the same values every time you execute it? Are the values the same when the program runs in different computers?

(iii) Copy the following code into a file

```
#include <stdio.h>

int main() {
  int a = -1;
  printf("a=%u\n", a);
  return 0;
}
```

compile it and run it. In this code, we are telling the computer to print out the number "−1", but in the screen another number appears. Can you explain why? The purpose of this exercise is to make you understand the code above, ideally by yourself alone. Remember that computers are stupid but strictly obedient.

Excercise 2 [*Motion of a classical particle in one dimension*] Consider a point-like body moving linearly with constant acceleration a. From Newtonian mechanics we know that its equation of motion is

$$x(t) = x_0 + v_0\, t + \frac{1}{2} a\, t^2 \,, \tag{1}$$

where x_0 is the initial position and v_0 is the initial speed (both at $t = 0$).

(i) Write a first program that — given the quantities a, x_0, v_0, t and $x(t)$ — checks whether the body is moving according to Eq. (1), i.e. check whether Eq. (1) is fulfilled. At first ask the user for these parameters using `printf` and read them using `scanf`. Then read them using an input file (`./executable < input.txt`).

(ii) Modify the previous code in order to get only a, x_0, v_0 and $x(t)$. Let the program determine how long it takes the body to reach the position $x(t)$, if it is moving as described in Eq. (1).

(iii) Make another version of the program which, by using `scanf`, reads in 6 numbers that are going to be interpreted as x_0, v_0, a, $t_{\rm min}$, $t_{\rm max}$ and N (*number of time intervals*). The program should output $N+1$ lines, each of them having t and $x(t)$, where $t = \{t_{\rm min}, t_{\rm min} + \Delta t, t_{\rm min} + 2\Delta t, \dots t_{\rm min} + (N-1)\Delta t, t_{\rm max}\}$, and $\Delta t = (t_{\rm max} - t_{\rm min})/N$. Use `gnuplot` to plot the results.

Make sure to have thought about the following questions before writing your code.

- How do you make the check in point (i)? What does it mean to compare two `float` or `double` variables with the operator `==`? Modify your first code. Now, do not ask the user for parameters, but set them as follows

  ```
  x0 = 1.0; v0 = 0.0; a = 1./17; t = sqrt(2.); xt = 1.05882353;
  ```

  what happens if you use `float` variables? And what with `double` variables? You should now know what to expect (compare with *the previous exercise*).
  Advanced: What happens if `x0 = 0.0; v0 = 0.0; a = 1./17; t = sqrt(2.); xt = 0.0588235294;`? Can you explain the difference with the previous case?

- How do you present to the user the result of point (ii)? Think about the possibility that

$$v_0^2 - 2\, a \left(x_0 - x(t)\right) < 0 \,.$$

- What happens if the acceleration a is equal to 0?

Excercise 3 [*Inside the unit sphere*] Write a program that reads in, with `scanf`, the following structure of input

```
n
x1 y1 z1
x2 y2 z2
...
xn yn zn
```

interpreats each line after the first one as the coordinates of a point in a 3-dim space. The program should print out a message like

```
xi yi zi -> r=... (in!)
```

if the coordinates correspond to a point inside the sphere of radius unit. At the end, your program should tell how many points are inside the sphere.
For example, given the following input

```
4
0.5 23 -8
0.1 -0.5 0.9
100 -2e40 0.006
0.23 -0.15 -0.05
```

the program should return something like

```
0.23 -0.15 -0.05 -> r=0.2791 (in!)
There are 1 points inside the unit sphere.
```