Einführung in die Programmierung für Physiker
WS 2017/2018 – Marc Wagner

Francesca Cuteri: cuteri@th.physik.uni-frankfurt.de
Alessandro Sciarra: sciarra@th.physik.uni-frankfurt.de

Exercise sheet 3

To be corrected in tutorials in the week from 06/10/2017 to 10/11/2017

Exercise 1 [*Computers playing darts*]

Darts is a form of throwing sport in which small missiles are thrown at a circular dartboard fixed to a wall in order to gain points and win against the opponent.

You are requested to write a program in C simulating the game of darts with a single player (your computer). The computer is expected to play *blindly* in front of a square wall of side 10 (in some units) circumscribing the circular target of radius 5 (in the same units).

(i) First of all, we will need to read as input the `unsigned int` variable counting the total number `N` of darts to throw.

(ii) Given the assumption that the computer plays blindly, we will make use of the C library function `rand()` that returns a pseudo-random number in the range of 0 to `RAND_MAX`[1]. You should call the `rand()` function to initialize `N` times (in a loop) a pair of double variables `x` and `y` ranging from -5.0 to 5.0 that represents the 2D coordinates of the point on the wall that is being hit by a dart (exactly as if you were setting up a Cartesian coordinate system with its origin at the centre of the circular target). You will see later in the lecture how exactly functions work. For the moment, it is enough to know that, if a function returns a number, you can store it in a variable using a statement like `x=rand();`.

(iii) At this point we are able to determine, with the help of the Pythagorean theorem, what is the distance, from the centre of the circular target, of any point hit by a dart and verify whether this falls inside of the circular target.

(iv) Based on this condition you will need to increment the `unsigned int` variables `nPoints` counting how much we score. For each darts hitting the circular target you score 4 points. You score 0 if a dart hits the wall outside of the circular target.

(v) You are almost finished! The last thing you need to do is to print on the screen, as a floating point number with 8 decimal digits after the comma, the result of the floating point division between the number of points accumulated stored in `nPoints` and the number of thrown darts stored in `N`. And, how much do you get? Any special number? Can you explain why it is so?

Run your code multiple times without changing the `N` value in input. How do you explain the fact that the result is not changing from one run to another? We were supposed to get `N` random numbers, how comes they are always the same? Read on the web[2] about the function `srand(unsigned int seed)`.

Time to Test! Try out your code multiple times by changing the `N` value in input. Do you think that you are free to throw as many darts as you like? Said in other words, is there a limit to the total number of darts you can throw? Put a `printf` statement immediately after having thrown the last dart to print how many darts have been really thrown (i.e. print the last value of the loop index, not `N`) and see what happens increasing the input value for `N`.

Advanced: Can you check the value of `N` and stop the program in case this value is not acceptable? Use the `exit` function to terminate the program[3].

[1] `RAND_MAX` is a constant whose default value may vary between implementations, but it is granted to be at least 32767. Remember to put `#include <stdlib.h>` in your code to have access to the `rand()` function.
[2] `http://www.cplusplus.com/reference/cstdlib/srand/`
[3] `http://www.cplusplus.com/reference/cstdlib/exit/`

Exercise 2 [*4 ways to swap variables*]

To *swap* means to *exchange* and, for our purposes, it is just about

- having two variables initialized with own values;
- assigning to each of the two variables the value originally assigned to the other.

You can work with either `int` or `float` numbers and start with two variables, which we may call `a` and `b`. Your task is to write a program to achieve the swap in 4 different possible ways:

(i) The most naive way is

1. to make use of a third auxiliary variable `c`;

(ii) Somehow less naive ways, that is by **ONLY** using `a` and `b` and

2. Implementing a chain of addition/subtraction (and, of course, assignment) operations;
3. Implementing a chain of multiplication/division (and, of course, assignment) operations;
4. Implementing a so called "`XOR` swap algorithm" that relies on boolean algebra.

Time to Test! Can you identify any pitfall in the various implementations? Said in other words, can you make your swapping algorithm fail depending on the values of the variables to swap and find a reason for the failure?

Exercise 3 [*Quadratic equation*]

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 $t = 0$). Write a program that — given the quantities a, x_0, v_0 and $x(t)$ — prints to the screen how long the body takes to reach the position $x(t)$, if it is moving as described in Eq. (1). At first, ask the user for the parameters using `printf` and read them using `scanf`. Then read them using an input file (`./executable < input.txt`).

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

- How do you present to the user the result? How do you ask for the input? Each quantity has a unit of measure and your program should not be ambiguous.
- Think about the possibility that

$$v_0^2 - 2\,a\,\big[x_0 - x(t)\big] < 0\ .$$

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

Time to Test! Whenever you implement an algorithm, you should write tests for it. In this case, you can easily think of values of the input parameters for which you can analytically calculate the expected result. Each branch of your program should be tested. Create an input file for each test, calling them `test_input_1_output_XXX`, where `XXX` is the result you expect.