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

Tutorial IX

January 16

Excercise 1 [*Iteration and recursion*] The purpose of this exercise is to practice two relatively common programming techniques: *iteration* and *recursion*. In particular, in this tutorial we suggest to write two functions that implement the integer exponentiation of a number.

(i) Write a function, `double pow_iter(double a, int n)`, that uses iteration to compute a^n. Using a loop, the function must multiply a by itself n times to produce the result.

(ii) Write another function, `double pow_rec(double a, int n)`, that computes again a^n but this time using recursion. You are allowed to use neither `for` nor `while` nor `do-while` loops.

Note: the standard mathematical library (linked with `-lm`) includes a function called `pow`. In this exercise you can use it to check your functions.

However, in real life we recommend you to use as much as possible the standard library: it comes with well tested and efficient functions that avoid the effort of reinventing the wheel.

Excercise 2 [*Linked list*] In the lectures, the concept of *(singly) linked list* has been introduced. It is a simple sequence-like data type. It consists of *nodes*, which in its simplest version, are objects that contain some data and a pointer to the next node. The pointer of the last element is a `NULL` pointer.

In this tutorial you should extend the basic functionality introduced in the lecture. For instance, by writing suitable functions, you could add the following features:

(i) Inserting an element at some position in the list:
`void insert(L_ELEM **p_list, const char name[], int index)`.

(ii) Determining the length of a list: `int length(L_ELEM *p_list)`.

Of course, if you wish, you could add much more features: reverse a list, concatenate two lists, copy a list, sort a list, ...