04JCJLZ - COMPUTER SCIENCES - 2015/2016

Laboratory 4

Objectives:

- Solve problems involving logic decisions
- Develop programs using iterations

Technical content:

- Use of *if-then-else* and *switch* constructs
- Use of iterative constructs: *while, do-while*, and *for*
- Introduction to the *cast* and *sizeof* operators.

Preferably to be solved in the laboratory:

Exercise 1. Write a C program to determine if a quadratic equation ($ax^2 + bx + c = 0$) has two real roots. Write it using the following guidelines:

a. Define three variables, called *a*, *b*, and *c*, which represents the three coefficients of the equation
b. Read from keyboard the values of *a*, *b*, and *c*
c. Evaluate the *discriminant* (Δ) of the equation:
   i. If Δ is positive, then show the following message: "The equation has two REAL distinct roots"
   ii. If Δ is zero, then show the following message: "The equation has two REAL coincident roots"
   iii. If Δ is negative, then show a message to let the user know that the equation has no real roots.

Exercise 2. Write a C program that, given an integer number between 1 and 12 representing the current month, is able to display the extended name of the month, using a *switch* construct (1→"January", 2→"February", 3→"March", …, 12→"December").

The program must also handle wrong inputs from the user (lower than 1 and greater than 12).

Further insight: modify the program in order to allow it to accept a data in the format *dd/mm/yyyy* (e.g., 15/04/2015): the program has to print the data with the extended name (e.g., 15 April 2015) of the month using a *switch* construct.

Exercise 3. Write a C program that reads in input integer numbers from the keyboard until the user inserts the value 0.

Hint: use the *while* (or *do-while*) loop construct.

Further insight: modify the program by accumulating during the acquisition process into the variable *sum* the values inserted before placing the number 0; at the end of the acquisition, the program will print on the screen the calculated value (*sum*).

To be solved at home

Exercise 4. Write a C program that reads in input from the keyboard a positive integer value N <= 40 corresponding to the base of a right and isosceles triangle, and represents the triangle on the screen by using of '*' characters.

Example: if the value inserted by the user is 3, the following sequence of characters has to be displayed:

```
*
**
***
```

Further insight: Write a C program that reads an odd integer number N and represents alternative geometrical figures, such as isosceles triangle, square, etc.

For example, try to represent the following geometrical figure:

```
*********
 *******
  *****
   ***
    *
```

In this case, N = 9.

Exercise 5. Write a C program to display on the screen the first 20 values of the Fibonacci series.
Hint: the first values of the series are: 0 1 1 2 3 5 8 …
Formally, the series is implemented by using the following relation:
$X_i = X_{i-1} + X_{i-2}$, with $X_0 = 0$ and $X_1 = 1$.
Further insight: modify the series as follows:
$X_i = X_{i-1} * X_{i-2}$, with $X_0 = 1$ and $X_1 = 2$; how many elements of this series can be represented with integer variables?

Exercise 6. [1]Write a C program that reads in input from the keyboard a decimal number N and then acquires from the keyboard a sequence of integer numbers until the following conditions are fulfilled:

a. The average of the inserted values is greater than *N*
b. 10 numbers have been acquired.

[1] This exercise will be solved using a multimedia format, and its solution will be provided in the course site during the following weeks.

Exercise 7. Write a C program to evaluate the maximum value that can be stored in variables of types *int*, *long* and *unsigned int*.

Hint: following the path shown below, use the step-by-step debugging mode and analyze the results of the various assignments.

a) Verify that there is not a practicable way to try to assign values progressively larger: for example, if you write the instruction *value = 3000000000*, the compiler does not report an error, but (maybe) only a warning. What do you see if you observe *value* with the Watch debugging feature after the execution of the instruction?

b) Try to get these values in an "empirical" way, i.e., by acquiring and printing them through *scanf* and *printf* functions. Verify that also this is not a correct procedure: the behavior of *scanf* in case of error is not controllable by the programmer.

c) At this point implement an algorithm that, taking into account the binary representation of unsigned and two's complement numbers, allows to detect the maximum value: for signed numbers, you can initialize *value* to 0, then increases it repeatedly. It is known that if you increase by 1 the maximum positive value you get an overflow and the value becomes negative; so the searched value is the value that precedes the first negative value found. Translate this procedure into a program and test it. How can you modify the algorithm (and the program) to work with unsigned numbers?