Programming 2

Ayman Makki (ayman.makki@uni.lu)
Semen Yurkov (semen.yurkov@uni.lu)

Lab 3 – Expressions & Arrays

Exercise 8 – Short-Circuit Evaluation #programming :ShortCircuitEvaluation #c :Operator #c :Expression

Write a C console application to prove the use of short-circuit evaluation for logical operators in the C programming language. Your expression must incorporate the use of `&&` and `||` as well as one call to a self-written function as part of the evaluation. For precedence and associativity of operators in C, refer to section *2.12 Precedence and Order of Evaluation* of *The C Programming Language* (which can be directly retrieved under #c :Operator).

1° Test the traditional arithmetic operators : +, -, *, /.Does their behavior fits their mathematical definition ?
2° Test the logical operators : &&, || and !.
3° Test the assignment operators : =, += , *= etc. Do they behave the same way as the previous operators ?
4° Test the unary operators : ++a, a++, a–, –a. Do you notice anything ?

Exercise 9 – Frequency Analysis (based on Exercise 1-14 in *The C Programming Language*) #c :ArrayType

Write a program that prints the frequencies of different characters in its input. You can gather the input character by character with the `getchar()` function until reaching `EOF`. Only consider the 26 characters of the latin alphabet, other characters can be skipped. Implement your program to be case-insensitive, i.e. do not distinguish between uppercase or lowercase letters.

> ✎ **Letters are numbers !**
> Please remember, as we discussed in previous labs, that following the ANSI character set specifications, all characters in C are encoded using a 8 bit integer.
> Therefore, what kind of operations might happen when using arithmetic operators on characters in C ?

To test your program, you may, e.g., send the content of a file via piping to your executable (cf. Operating Systems 1), or enter letters until hitting ctrl + D.

Exercise 10 – Merging Two Sorted Arrays

#c :ArrayType #c :PreprocessorMacroDefinition #c :RandomNumbers

Write a C console application performing the following tasks :

1° Create two arrays (t_1 and t_2) of length l_1 and l_2. l_1 and l_2 are constants that are defined in the program. Make sure that $l_1 \neq l_2$.
2° Write a function `void fill_array(int array[], int size)` that fills the array of given size with random numbers between 0 and 99. Use the function in your main program to fill both t_1 and t_2.

> ☞ **Say whaaat ?**
> You will notice that although you do not return the sorted array explicitly, the changes will be reflected outside the function.
> **Q :** Didn't you tell us in Lab 2 that *the C programming language employs call by value when passing arguments to functions* ?
> **A :** Yes, but arrays are different to primitive types. Stay tuned for the explanation in one of the next lectures !

Ayman Makki (ayman.makki@uni.lu)
Semen Yurkov (semen.yurkov@uni.lu)

3° Write a function `void sort_array(int array[], int size)` that sorts the given array (you may choose your favorite sorting algorithm).

4° Write a function `void print_array(int array[], int size)` that prints the given array.

5° In the main program, sort both t_1 and t_2 and print them to the console.

6° Merge both arrays and store the result in a new array t_3, which at the end shall also be sorted, but without calling the `sort` function on it. To do so, use a single loop that accesses all elements of both arrays. At each step, the program shall then choose the smallest element that has not yet been inserted into t_3 and insert it.

7° Print t_3 to the console.

Your solution must be able to handle arrays of arbitrary sizes, i.e. it shall be easily possible to change l_1 and l_2 without requiring lots of modifications throughout the code.

Exercise 11 – Mini-Minesweeper **#c :ArrayType** **#c :PreprocessorMacroDefinition** **#c :RandomNumbers**

Write a game similar to Minesweeper with a grid, represented by a two-dimensional array. The size of the grid is a constant value. Fill the grid with mines. The probability of encountering a mine in the grid is another constant value. The size of the grid as well as the probability of encountering a mine should be easily changeable in your code.

The user shall be constantly asked for x and y coordinates. The game goes on until he hits a field with a mine, in which case he loses, or he has uncovered all empty fields, in which case he wins.

☞ **Difficulty is not obvious**

In this exercise, as well as in programming in general, the difficulties might not be obvious when reading the guidelines. Most of the time when you code a program, you might realize the hardest part when you start coding them (or when you run into a bug you did not anticipate).
Therefore, it is a good practice to scrap your program on a paper before actually start coding for bigger programs. Write down the following elements :

— What are the important variables you need for the program ?
— Initialization : what has to be setup **before** the main program starts to run ? Constants ? Initializing arrays to 0 ?
— What are the important functions of your programs ? Their arguments and their return type ?
— A draft of the global processing flow of your programs (while and for loops, conditions etc.).

It would benefit you to try that for this exercise and to determine if this enabled you to anticipate unexpected behavior or bugs.