Tutorial n°2 : Functions, arrays, and pointers

1 Preliminaries

Create and add to your project two files called **Labs2.h** and **Labs2.cpp**.(in code::blocks, you can add "virtual folders" to separate headers and cpp files ...) **Labs2.h** file is your first personal header file and is going to contain all the signatures of the procedures and functions that will be implemented in the **Labs2.cpp** file. It may also contain variable definitions. For instance:

Within the .h file	Within the .cpp

`int MyFunction1 (int, int);`
`void MyFunction2 (float);`
`int dumvar = 3;`
	`#include <iostream>`
	`using namespace std;`
	`#include "my.h file"`
	
	`MyFunction1 (int a, int b) {`
	`  return (a+b+dumvar);`
	`}`
	
	`MyFunction2(float x) {`
	`  if (x == 0) cout <<"x is null"<<endl;`
	`  else cout <<"x is not null"<<endl;`
	`}`

Notice the differences:

1. In the .h file: for the function signatures (also known as prototypes) the variable name is not mandatory, only the type identifiers matter
2. In the .cpp file: do not forget to include the corresponding header file, and include other headers if needed (iostream or cmath today). Personal header files are included with " ", standard header with <>.

Include your header within the main.cpp file.
Now, from the **main.cpp** file, you can call any function declared in **Labs2.h** and implemented in **Labs2.cpp**, without taking care of the order of implementation.

2 Exercices

2.1 Input and output in the console

Declare and implement a function called `ExampleInputOutput( )` to illustrate, at the least, the following C++ standard objects and operators to manage input and output in the console : `cin`, `»`, `cout`, `«`, `endl`, and `getline`.

2.2 How to pass parameters to a function

2.2.1 On passing parameters by value or by reference

Implement two functions called $swap\_1(int, int)$ and $swap\_2(int\ \&, int\ \&)$ that are supposed to swap two values. Display the final values just before the end of each function, and display the results from the main function, before and after the call. What can you remark?

2.2.2 On passing parameters using pointers

Same question as before, but will now use pointers. Explain the differences with the previous question.

2.3 Multiple returned values

The keyword return can only return one value. You may sometimes need to return several values, sometimes an error code and several results as well. To do so, pass arguments by references to "return" multiple results. As an exercice, declare and implement a function `CartesianToPolar` which return the modulus $\rho = \sqrt{a^2 + b^2}$ and the angle $\theta = \arctan\left(\frac{b}{a}\right)$ of a complex number $z = a + ib$. Tips, have a look at the function `atan(double)`, and `atan2(double, double)`.

2.4 Default parameters

Implement a function $IsMultipleOf(\ldots)$ that determines if a number p is a multiple of number q. By default, q will be assigned to 2. In the main function, test this function with a given value for q, or without providing a value for q.

2.5 Recursive functions

A recursive functions is a function that calls itself during its execution. Declare and implement a recursive function called `Prime` which determines if a number p is a prime number (only multiple of 1 and p).

2.6 Monodimensional array

Declare and implement a function called `ArraysExample1`. Within this function, declare and allocate a static array of 10 integers. Declare and allocate a dynamic array of 10 integers. For both arrays, each element at index i contains the value i. Display both arrays.

2.7 Bidimensional array - Pascal's triangle revisited

the following exercice should be done for the following 2 cases:

1. Use a bidimensional array statically allocated, with [ ] operator.
2. Use a bidimensional array dynamically allocated, thus using pointers of pointers. In this case, try to avoid the [ ] operator, as much as possible, and use pointer arithmetic.

Declare a bidimensional array $Tab2$ that contains $n \times n$ integers (n being fixed for static allocation, or variable for dynamic allocation). Initialize the first value (`Tab2[0][0]`) to 1, and all the others to 0. In the Pascal triangle, we can remark that the following relations hold: $Tab[i][0] = 1$ and $Tab[i][j] = Tab[i-1][j] + Tab[i-1][j-1]$ if $j > 0$. Completely fill the array using the previous relationship, then display it. You should get the following result:

1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	2	1	0	0	0	0	0
1	3	3	1	0	0	0	0
1	4	6	4	1	0	0	0
1	5	10	10	5	1	0	0
1	6	15	20	15	6	1	0
1	7	21	35	35	21	7	1

2.8 Multidimensional arrays as functions parameters (read and write)

In the main function:
Declare and initialize two bidimensional arrays, A and B, with 3×3 elements. Values are up to you to verify the computations. Declare and initialize one bidimensional array, C, with 3×3 null elements. Declare and implement a function $MultMatrix(\ldots)$ that has A, B, C as parameters. After the function call, matrix C should contain the results of the matricial product of $A \times B$.