Exercise Sheet #4

Problem 1 (*Template functions*) 10 Pts

For the following tasks, implement the requested functions and call them in the `main` function.

(a) Implement a function with a template type for the arguments and the return value, which takes two numbers of the same template type (e. g. `int`, `double`) and returns the product, which is of the same type. (2 Pts)

(b) Write a function that has a templated integer parameter and takes as an argument an array of template type `T` and length `N`. The function shall print all entries of the array and the size of the array. (2 Pts)

Hint: The function declaration then looks like
`template<typename T, int N> func_name (T (&array)[N]) {...}`.

(c) Check what happens if the function is called with a dynamic allocated array, i.e. `int * a = new int[N]`. Why does this happen? (2 Pts)

Hint: The functions with the appropriate data types are generated from the templates at compile time, what about dynamically allocated arrays?

(d) Re-write the function such that you can pass a two dimensional $n \times m$ array. (2 Pts)

Note: In the case of a quadratic 2d array the function becomes a bit simpler (but thus more confusing) and can take the form
`template<typename T, int N> func_name (T (&array)[N][N]) {...}`.

Problem 2 (*Root Finding Function*) 10 Pts

A simple algorithm for finding the root of a function is **iterative bisection**: Aiming to find the point where $f(x) = 0$, start with an upper and lower bound, $sign(f(x_{lower})) = -sign(f(x_{upper}))$, then update one of the bounds to $x_{mid} = \frac{x_{lower}+x_{upper}}{2}$ replacing the bound with the same sign. Repeating this process approaches exponentially to the root, with a declining error range given by $|x_{lower} - x_{upper}|$.

Implement a root-finding function and pass some arbitrary function to it:

(a) Define a function `find_root` that takes as arguments a pointer to a function, starting values for `xMin` and `xMax`, the desired accuracy and a maximal iteration count.

(b) Implement the iterative bisection algorithm inside the function. it should run until either the desired accuracy or the maximal iteration count is reached, then print out the root that it found, the remaining error range and the number of iterations until convergence.

(c) If the function fails, an appropriate error message should be given. The algorithm fails when the given initial bounds do not satisfy the sign condition.

(d) Write a function `find_root_recursive` that runs the same algorithm, but does so using recursion. Validate that both functions yield the same results by testing them with a number of functions of your choice.

Problem 3 (*Bitwise Operations, advanced*) 10 Pts

C++ also allows he manipulation of data on the level of its binary representation. The six bitwise operations are:

Operator	Symbol	Form	Operation
left shift	`<<`	`x << y`	all bits in x shifted left y bits
right shift	`>>`	`x >> y`	all bits in x shifted right y bits
bitwise NOT	`~`	`~ x`	all bits in x flipped
bitwise AND	`&`	`x & y`	each bit in x AND each bit in y
bitwise OR	`|`	`x | y`	each bit in x OR each bit in y
bitwise XOR	`^`	`x ^ y`	each bit in x XOR each bit in y

(a) Write a function that takes two positive integers and returns their sum, not using +, -, * or /. You should be able to achieve this by manipulating the numbers on the bitwise level. Your are also allowed to use the conventional syntax to control the program flow, as well as comparators (e.g. `x > y`). For a start, consider how addition of numbers is done in binary arithmetic `https://en.wikipedia.org/wiki/Binary_number#Addition`. **Hint**: It might be necessary to perform the carrying, as described in the reference, multiple times. You need to add the carried bits to the result, so recursion could be useful. (5 Pts)

(b) Write a function that implements the multiplication of two positive integer numbers with the same restrictions to bitwise operations. See `https://en.wikipedia.org/wiki/Binary_number#Multiplication` for details. **Hint**: You should use the adding function from the first part of the exercise. (5 Pts)