Computer Architecture
Homework no 2
Arrays and functions in C

In this homework, we carry on our exploration of « arrays » in the programming language C. Just as in the previous recitation, all the programs should be compiled with the option `-Wall` without producing any warning.

Exercise 1. Getting started

§1.1 Write a C function

```
void print_array(int a[], int n)
```

which prints the values of the array `a[]` sent as parameter and then breaks to the next line.

§1.2 Write a C function

```
void read_array(int a[], int n)
```

which reads n integers and stores them in the array `a[]`.

§1.3 Write a `main` function to test the two functions — and keep them available for future use in this recitation: we will heavily use them!

Exercise 2. Dichotomic search

§2.1 Write a function

```
linear_search (int a[], int length, int val)
```

which takes as parameter an integer `val` and an array `a[]` (together with its length) and returns the first index `i` such that `a[i]=val`. Write a `main` function to test the function.

§2.2 Write a function

```
is_sorted(int a[] , int length)
```

which takes an array (together with its length) as parameter and returns true (=1) precisely when the array is ordered (increasingly). Once again, write a `main` function to test the function.

Dichotomic search is a search algorithm which enables one to search in logarithmic time a value `val` in an ordered array. Dichotomic search manipulates three integer values

$$\texttt{s} \leq \texttt{m} \leq \texttt{b}$$

for « small », « medium » and « big ». At any time of the algorithm, one has

$$\texttt{a[s]} \leq \texttt{val} \leq \texttt{a[b]}$$

At the beginning of the algorithm, the integer `s` is equal to 0 while the integer `b` is equal to the length of the array minus one. At each step of the algorithm, the integer `m` is equal to

$$\texttt{m} = \lfloor \texttt{(s+b)/2} \rfloor$$

If `a[m]` is equal to `val` then the algorithm stops. Otherwise, two cases may happen:

- if `a[m]` < `val`, then `s` takes the value `m` + 1
- if `val` < `a[m]`, then `b` takes the value `m` – 1

When the algorithm reaches a state where `s` = `b` the algorithm stops and returns that the value `val` has not been found.

§2.3 Write a function

```
binary_search (int a[], int length, int val)
```

which takes the same parameters as the function `linear_search` but makes the assumption that the array is sorted, and applies a dichotomic search to it. What may happen when the function `binary_search` is called with an array which is not sorted ?

Exercise 3. Insertion sort

The insertion sort algorithm enables one to sort an array in-place, that is, without using any additional array. The insertion sort of an array of size n manipulates a variable m which splits the array in two parts: the part from 0 to $m-1$ of the array which is already sorted and the part from m to n of the array which remains to be sorted. The algorithm starts with the variable m equal to 1. At each step, one successively compares the element `a[`m`]` to any of the elements

	0	1	2	3	4
$m = 1$	5	**3**	1	2	4
$m = 2$	3	5	**1**	2	4
$m = 3$	1	3	5	**2**	4
$m = 4$	1	2	3	5	**4**
$m = 5$	1	2	3	4	5

Figure 1: Sorting by insertion: in the third line for instance, the variable m is equal to 3, and the three first elements of the array are sorted. As $3 > 2$ the index i is equal to 1, and after a circular permutation of $3;5;2$, one gets the array $1;2;3;5;4$ and the variable m is incremented to $m = 4$.

$$\texttt{a[0]}, \texttt{a[1]} \quad \text{up to} \quad \texttt{a}[m-1].$$

When one finds the smallest index i such that `a[`i`]` is larger than `a[`m`]`, one inserts `a[`m`]` before `a[`i`]` by applying a circular permutation of the elements

$$\texttt{a}[m], \texttt{a}[i], \ldots, \texttt{a}[m-1].$$

Once the circular permutation has been performed, one increments the variable m. The algorithm terminates when m is equal to n.

§3.1 Write a C function

```
void insertion_sort(int a[], int n)
```

which sorts in-place the array `a[]`.

§3.2 Write a function `main` which reads an array, sorts it, and prints the sorted array.

§3.3 Check that the program is correct and that it remains so when the same integer appears several times in the input array.

Exercise 4. Arrays: Horner's scheme

A polynomial function is a function of the form

$$f(x) \quad = \quad \sum_{i=0}^{i=n} \quad a_i \, x^i$$

where the coefficients a_i have arbitrary integer values. For example, the function

$$g(x) \quad = \quad 14\,x^2 + 13\,x + 32$$

is a polynomial function where $a_0 = 32$, $a_1 = 13$ and $a_2 = 14$. In this exercise, we represent the polynomial function by the array of its integer coefficients. For instance, the function g above will be represented by the array of integers $\{32, 13, 14\}$.

§4.1 Write a C function

```
eval(double x, double a[], int n)
```

which computes the polymonial function defined by the array `a[]` at the integer x.

§4.2 Modify your function such that it only performs n multiplications.

References and acknowledgments: among all the practice exercises in C which I found in the literature, I most enjoyed these exercises designed by my friend and colleague Juliusz Chroboczek, which I thus decided to translate and to adapt from French. Quite obviously, all mistakes are mine!