# Exercise Sheet #4

Bulcsú Sándor <sandor@th.physik.uni-frankfurt.de>
Hendrik Wernecke <wernecke@th.physik.uni-frankfurt.de>
Laura Martin <lmartin@th.physik.uni-frankfurt.de>
Christopher Czaban <czaban@th.physik.uni-frankfurt.de>

**Problem 1** (*Pointers*) 6 Pts

(a) Define three pointers of the type `int`, `double`, `long double` that each point to an array of the corresponding type. Print the addresses of the arrays (i. e. the values of the pointers) and the value that the pointer represents. Then increment the pointers (e. g. by `++p` or `p=p+1`) and print both the value of the pointer and the element it points. What do you find for the different data types?

(b) Write a function `int myfunc(int &x, int &y)` that takes two integers by reference and returns their product. Rewrite the function such that it takes two pointers as arguments instead.

(c) Write a function `double scalarProduct(double *arr1, double *arr2)` that returns the scalar product of two vectors stored in `arr1`, `arr2`.

(d) Now implement the vector product of two vectors. Therefore you should create the function `void outerProduct(double *arr1, double *arr2, double *result)`, where the two vectors are stored in `arr1`, `arr2` and the result is to be stored in `result`. This structure is the typical way to return arrays from functions.

**Problem 2** (*Multi-dimensional arrays*) 6 Pts

In `C++` one can use multi-dimensional arrays like `int arr[m][n]` that correspond to $m \times n$ matrices. Create an array and write a function for each of the following tasks:

(i) filling the array with arbitrary values

(ii) multiply each element by a constant

(iii) printing out the values formatted.

The functions shall of course take a multi-dimensional array as an argument. Be aware of how to handle the dimensions (macros, global variables, constants).

Additionally write a function `void printArray(int arr[], int len)` that takes a one-dimensional array and its length. Then call the function giving one row of the previously defined matrix as an argument.

**Problem 3** (*Battleship game*) 8 Pts

The well-known battleship game is played by two players, each of them having two $10 \times 10$ square grids, which play the role of a battlefield and a tracking map. On one grid the player arranges ships and records the shots by the opponent. On the other, tracking grid the player records their own shots, trying to find all the battleships of the opponent. The rows and columns of the grids are labelled by capital letters and numbers, respectively. As a starting setup, both players place the ships of different sizes at random positions on their primary grid. The set of given ships is:

| # | Class of ship | Size |
|---|---|---|
| 1× | Carrier | 5 |
| 1× | Battleship | 4 |
| 1× | Cruiser | 3 |
| 2× | Destroyer | 2 |
| 2× | Submarine | 1 |

Write a `C++` code to implement the one-player (asymmetric) version of the game, i. e. a player against the computer, but the computer is not shooting back. The player shall see a tracking grid for the game, while the battlefield of the computer is not shown. The game proceeds in series of rounds. In each round the player launches a rocket targeting a certain position (a grid cell) of the battlefield, the tracking grid is then updated and refreshed, indicating whether the the shot was successful or not.

As a starting point you can use the code presented in the lecture following the instructions below:

- write a method to randomly place the ships on the battlefield at different positions and orientations (ships shall not overlap, and must be fully on the field)
- implement a function for checking whether the player's shot hit a ship of the computer
- mark the position of the hit on the tracking grid, placing on `o/x` corresponding to successful/unsuccessful trials
- update the screen after every round