SEMINAR 1

LOOPS AND FUNCTIONS

Week 5 – Seminar 1

To be able to evaluate the performance of an algorithm, that is a solution to a given problem we need to make some simplification and abstraction of how a machine works. This will be the subject of the module SOF2 next term when we study algorithm complexity. Similarly, we need to formulate our algorithms in pseudocode that is an abstraction and simplification of imperative programming languages like C, Python, Java, etc. , combined with a liberal use of mathematical notation. There are many pseudo code style, but in this module as well as SOF2 we will be using Mehlhorn et al., from the book "Algorithms and Data Structures: The Basic Toolbox" (2008). Below you can find some of the main element of the style that you must applied for all the SOF1 seminars. A good exercise is then to try to translate the algorithm from pseudo code into Python.

- A variable declaration `v = x : T` introduces a variable `v` of type `T` and initialises it with value `x`.
- `a : Array [i..j] of T` introduces an array a consisting of $j - i + 1$ elements of type `T` stored in `a[i], a[i+1], …, a[j]`. At this stage of the module, a string is considered an array of characters and a list is an dynamic array where we can append value at the end of the list.
- The simplest statement is an assignment `x := E` where `x` is a variable and `E` is an expression.
- Flow-control structure are given below:

```
if condition then
    statements
else if condition then
    statements
else
    statements
endif

while condition do
    statements
endwhile

for i:= 1 to n do
    statements
endfor

foreach e in collection do
    statements
endforeach
```

Functions (returns a value) and Procedures (does not return a value) are written as follows:

```
Function name(param1: Type1, Param2:Type2, …): returnedType
    statements

Procedure name(param1: Type1, Param2:Type2, …)
   statements
```

Exercise 1:

You have given a **sorted** list of numbers with no duplicates, and you need to find the pairs of numbers in that list whose sum is equal to the given target value.

Numbers can be either positive, negative, or both.

a) Test Case 1:

```
Input: [-1, 1, 2, 4, 8], target = 7
Output: [(-1, 8)]
```

b) Test Case 2:

```
Input: [2, 4, 5, 7], target = 9
Output: [(2,7), (5, 4)]
```

c) Test Case 3:

```
Input: [2, 4, 5, 7], target = 8
Output: []
```

Write an algorithm to find all such pairs given a list and a target. Note there are two approaches to the problem, one called brute force where we try all possible combination of pairs and keep the correct one (computationally expensive), one cleverer that is far more efficient.

Exercise 2:

Write in pseudo code a function `merge(listA: List, listB: List)` that returns a sorted list containing the elements of both list where `listA` and `listB` are two **sorted** lists of integers. If an element exists in both lists, it must appear multiple times in the returned list. For example:

```
>>> merge([1,3,4,7],[2,3,5])
[1,2,3,3,4,5,7]
```

Exercise 3: *reinventing the wheel!*

For this question we are emulating the method `split()` from the type `str`. Write the algorithm for a function `splitText(text:String, delimiters:String)` which returns the list of the words by splitting the string `text` at each delimiters. The delimiters themselves are discarded. An example is given below:

```
>>> sampleText = "As Python's creator, I'd like to say a
few words about its origins."
>>> splitText(sampleText, ", '.")
['As', 'Python', 's', 'creator', 'I', 'd', 'like', 'to',
'say', 'a', 'few', 'words', 'about', 'its', 'origins']
```

You can assume that a string has a method `contains(Character)` that returns true if the character is in the string, false otherwise. This exercise is more challenging than it may look like.