# Synopsis Lab 10

## 10.1 Tasks

1. Skim lectures #8 and #9 and test all higher order functions.

2. (Haskell) Let us consider the following ordering relation among strings: `s1` is smaller than `s2` iff `s1` is shorter than `s2` or if they have the same lenght and `s1` is lexicographically smaller than `s2`. Write a function called `ltstr` with 2 arguments `x` and `y` which implements this order relationship.

   Write a function `qs` which gets 2 arguments: a list and a sorting criterion and which sorts the given list according to the given criterion, using the quicksort algorithm. E.g.:

   ```
   Main> qs ["Russia","Norway","Germany","Romania", "France",
   "Antigua and Barbuda","South Korea","Angola","Hungary"] ltstr
   ["Angola","France","Norway","Russia","Germany","Hungary",
   "Romania","South Korea","Antigua and Barbuda"]
   ```

   Then we will employ the `qs` to sort all students lexicographically by their family names. The students come from different countries, so their names might contain letters like **â, ă, ä** etc. For the sake of simplicity, we are going to consider all the characters above as equal to **a** (we don't want to set an order relationship between **â, ă, ä** as this won't scale easily).

   Write a function `strlt` which has 2 strings `s1` and `s2` as parameters and which returns `True` if `s1` is lexicographically lower than `s2` according to the convention above and `False` otherwise. Pass `strlt` to the previously defined `qs` in order to sort a list of names. E.g.:

   ```
   Main> qs  ["hässler", "hănescu", "hâldan", "hagi"] strlt

   ["hagi","h\226ldan","h\259nescu","h\228ssler"]
   ```

   For manipulating letters like **â, ă, ä**, you may want to use the module `Data.Char`. In order to do this, start your code with:

   ```
   \begin{code}
   import Data.Char
   ```

   Two useful Haskell functions are `ord`, which is given a character as argument and returns its ASCII code, and `chr`, which expects a number and returns the character whose ASCII code is that number.

3. (Haskell) Write a function called `leapYears` which lists all leap years between 1789 and 2018. A year is leap if its number is divisible by 4, except for the last year in a century. In this latter case, the year is leap only if it is divisible by 400. Thus, 1996, 2000, 2004 are leap years, while 1789 and 1900 are not.

   E.g.:

```
Main> leapYears
[1792,1796,1804,1808,1812,1816,1820,1824,1828,1832,1836,1840,
 1844,1848,1852,1856,1860,1864,1868,1872,1876,1880,1884,1888,
 1892,1896,1904,1908,1912,1916,1920,1924,1928,1932,1936,1940,
 1944,1948,1952,1956,1960,1964,1968,1972,1976,1980,1984,1988,
 1992,1996,2000,2004,2008,2012,2016]
```

4. (Haskell) Let us consider the following definition of a binary tree:

```
data Tree = Empty
          | Node Tree Int Tree
```

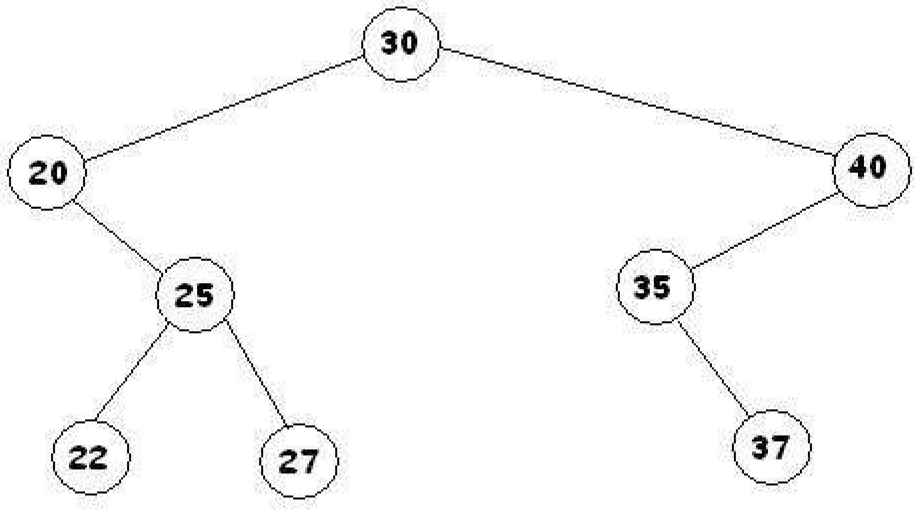
   Write a function for pretty printing a binary tree. It should print a dot (character `"."`) for every empty tree.

   Write a function `list2tree` with 2 lists as arguments: the preorder and inorder travesal of a tree and which builds back the original tree.

   E.g.(see picture):

```
Main> prettyPrintTree (list2tree [30,20,25,22,27,40,35,37]
                                 [20,22,25,27,30,35,37,40])

 30
    20
       .
       25
          22
             .
             .
          27
             .
             .
    40
       35
          .
          37
             .
             .
       .
```



5. (Haskell) Write a function called `stdDev` having as an argument a list of numbers `nrs`, which computes the standard deviation $\sigma$ of the numbers in `nrs` using the formula below:

$$\sigma = \sqrt{\frac{1}{N}(\sum_{i=1}^{N} x_i^2) - \bar{x}^2}$$

(here $N$ is the number of elements in `nrs`, while $\bar{x}$ is the average of the numbers in the list).

E.g.:

```
Main> stdDev [1,2,4,5,7,8]
2.5
```

Can you come up with a solution which makes only one traversal of `nrs`?

6. (Haskell,ML) Write a function called `combinations` with one argument `n` which returns a list containing all possible combinations of at most `n` elements.

E.g.:

```
Main> combinations 3
[[],[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3]]
```